University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Visualization of Classical Graph Theory Problems

Fatma Al-Sayegh

20 November 2022

Abstract

Whereas, formalism in mathematics gives the subject a structure and a level of abstraction which makes it applicable to the most general of the situations. Visualization of a concept on the other hand, restricts the theory to a particular example but it makes one to see a problem in a more concrete pattern. It may also allow the learner to see the same problem in a new light. The learner can then apply or extrapolate the learning to other instances of the problem in general. In this project we have tried to elucidate some classical problems in Graph Theory by the way of visualization on a Web application. The method of visualization are animations and user interaction with animations. Such methods, it is belived can help young students and self-learners to get the first brush with the subject of Graph theory.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Fatma Al-Sayegh Date: 20 November 2022

Contents

1 Introduction **1**

2 Background **2**

3 Analysis/Requirements **3**
3.1 Scope of The Project 3
3.2 Criteria for Selection of Problems 3
3.3 Methods of Elucidation 3
3.3.1 Animation 4
3.3.2 User Interaction 4
3.4 Technical Scope of The Project 4
3.5 Requirements 4

4 Design **6**
4.1 Wire-frame and Navigation 6
4.2 Animation Panel 6
4.2.1 Visual representation of graphs 6
4.3 Explanation Panel 7
4.4 User Stories for Elucidation of Topics 7
4.4.1 Graph Isomorphism 7
4.4.2 Max k Cut 8
4.4.3 Graph Coloring 9
4.4.4 Minimum Vertex Cover 9
4.4.5 Tree Width 9

5 Implementation **11**
5.1 Front End Development with The Elm Programming Language 11
5.1.1 Why Functional Programming? 11
5.1.2 The Elm Architecture 12
5.2 Implementation of Graphs 12
5.2.1 Grid 13
5.2.2 Using Linear algebra to Initialize Grids 13
5.2.3 Implementing Colors 13
5.2.4 Edges 13
5.3 Implementation of Animations 13
5.3.1 Morphing Geometry of a Graph 14
5.3.2 Re-formation of the Graphs 14
5.3.3 Drawing of Graphs 14
5.4 Explanation Panel 14
5.5 Implementation of Miscellaneous features 15
5.5.1 Navigation Bar 15
5.5.2 Keyboard Shortcuts 15

6 Evaluation **16**

7 Conclusion **17**

Appendices **18**

A Appendices **18**

Bibliography **19**

1 Introduction

2 Background

3 Analysis/Requirements

In this chapter, the scope of the project, the criterion of selection of the problems in graph theory, the thinking behind choosing the methods of elucidation of the selected topics will be discussed. Finally to conclude the analysis the requirements of the project are stated.

3.1 Scope of The Project

Understanding a problem is a necessary first step in trying to solve it. It also enables a student to abstract out a mathematical problem from a real life scenario present in the fields of science and engineering.

This idea has guided this project to be restricted to one which helps a learner to understand the problem itself. The solution, on the other hand, or suggesting an algorithm to solve the problem, if required, is the second important step which has been deliberately not touched upon to keep the scope of this project clear, precise and limited.

3.2 Criteria for Selection of Problems

One of the most important criteria for selection of the problems for the project was based upon the importance of the topic in the field of graph theory. There are several text books in graph theory, which discuss various theorems and problems in the subject. There are a few problems which occur commonly and frequently in them. The order of their inclusion in the text books is based logically. Building the concepts from the basics to advanced. Therefore the problems included in the project should represent all hardness levels.

Since imagining a graph theory problem is largely a visual exercise, there was no dearth of problems which could offer themselves as a subject of an interesting visualization. The additional criteria therefore for filtering the candidate problems was based on whether they could be elucidated in the form of a simple and meaningful example, employed for animation and user interaction. The simplicity of the example doesn't in anyway imply triviality of the problem. Indeed here the assumption is that a simple example problem can make a student reach to the heart of the concept in it's generality fairly quickly. From there on she can extrapolate the learning to more complicated examples.

It is important to mention here that a survey among peers in the field of software engineering and computer science was conducted to gather suggestions on the shortlisted topics and methods. The data from the survey had a role in determining topics and the methods chosen.

3.3 Methods of Elucidation

As it has been discussed in the previous section that feasible methods of elucidation/exposition played a prominent role in the selection of the problems in the first place. These methods can be broadly classified as animations and user interactions or a combination of both.

3.3.1 Animation

The visual medium has dimensions such as of color, position, shape and motion. Generating them with a computer program, scenes can be created attracting the users attention towards a particular aspect of it with the help of animations. The scene in this project consists of a graph, which undergoes transformations of color, shape and position to make a certain aspect of it emphasized for the purpose of explaining a concept. Animations are used in this project to make problems like Graph Isomorphism, Max Cut, and Tree Width. For instance, the example problem of Graph Isomorphism, was explained by morphing a graph to change it's shape to acquire another radically different shape. It will be explained later how it can be considered as a visual proof that the two graphs in the scene are thus isomorphic.

3.3.2 User Interaction

Although animations go a long way in terms of having a user's mind involved in the learning process they only offer a linear narrative. On the other hand, user interaction with an animation not only makes the experience more immersive, it also leads to a natural multiplicity of stories from a single program. This happens as a human input results in a novel path from one state to another in the program just like a video game.

3.4 Technical Scope of The Project

To achieve the above mentioned features The program has data structures to hold graphs and algorithms to visually and geometrically manipulate them.

It's important to note here the program does not contain algorithms to solve the listed problems. As the scope of the program was limited to the purposes of visualization and not coding the algorithms which can solve instances of the mentioned problems. Therefore in this project, although the data type of Graphs (Set of Vertices and Set of Edges) and the associated functions, are quite general and can support operations of various kinds, care is taken that we provide the solution to the visualization program before hand to give enough information to the various animations and user interactions.

3.5 Requirements

Based on synthesis of the above sections it is concluded that there should be a web application written for the desktop browser with user interactive animations which elucidate the following graph theory problems -

1. Graph Isomorphism
2. Max Cut
3. Graph Coloring
4. Minimum Vertex Cover
5. Tree Width

And to achieve all this by programming the following items -

1. Data structures which represent vertices, edges and graphs.
2. Display the above entities as Scalar Vector Graphics on screen.
3. Translation and shape transformation of the graphs for animation.
4. Generation and handling of events triggered by user interaction with the elements in the animation for user interaction.

5. Display appropriate text in synchronization with the animations and user interaction.

Doing so in a manner which fulfills the following subjective qualities -

1. Substantive learning impact
2. Ease of Use
3. Coherent Story telling
4. Pleasing Aesthetics

And finally, evaluating the application with the help of the peers on parameters which can be broadly classified into the following categories -

1. Quality of Elucidation
2. User Experience
3. Learning Impact

4 Design

This documents discusses the design choices made for the application. It progresses to how a graph are visually displayed and animated in the app without going any deep in the program implementaion. Finally, with the help of the above we shall discuss the intended experience of the user while going through the various topics explained in the application. We will also discuss the learning impact on the user as it forms an essential aspect of overall design.

4.1 Wire-frame and Navigation

The web application is a **Single Page Application**. Where the navigation from one topic to another occurs according to the user inputs. When the state moves from one graph theory to another topic, the data on the screen, that is the graphics and the text, change on the same page without changing the url.

The page is vertical divided into two parts, the left part of the page contains an instance of an animation, the right contains explanation of the topic and advice on how to interpret the animation along with navigation and control buttons.

The text in the explanation part of the page is dynamic in nature, if the animation has facility of user interaction with it's elements, the corresponding text on the right responds with advises on the state of affairs and what the user should do next.

There is a navigation bar at the bottom of the page, with left and right arrows, along with the names of the previous and the next topics, to hop from one topic to another.

4.2 Animation Panel

As mentioned in the preceding section, the left half of the page is for graphics, which contains an animation, a user-interaction session or a combination of both. The animation contains one or more than one graphs. These graphs undergo, according to needs of the topic in consideration transformations of appearance and annotation.

4.2.1 Visual representation of graphs

The graphs are represented as vertices and edges joining those vertices. Although the geometrical placement of the vertices is of no consequence in the subject of graph theory, for the purpose of visualization, vertices are assigned a 2 dimensional position. The edges, don't contain attributes such as length or positions of endpoints of a line segment, they are rather defined as a relation between a pair of vertices.

Appearance of Vertices The vertices of graphs in the animations are color filled circles of the same size save for certain exceptions. The color of the vertices have been allotted by varying mostly the hue and keeping saturation and lightness relatively same in the **HSL** (Hue Saturation Lightness) color space. The vertices contain a name inside the circle as an integer. The names of vertices were chosen as integers as it was assumed that such a representation would help the

developer and also the user to keep a track of them as order of integers is understood better by humans.

When a particular vertex is needed to be shown differently than the rest then it's size and color are displayed differently. For example, a user-selected vertex in some of the animations are shown bigger and it's color changed to golden. The gold color it was observed makes the vertex in consideration stand out differently from other colors in the animation.

Appearance of Edges Edges, although are defined very algebraically as a relationship between a pair of vertices it gets drawn by referring to the positions of the related vertices as a straight line segment white in color.

When an edge is supposed to shown differently than the rest of the edges, it's width is increased and color changed to the same value of golden as a selected vertex. Again it was found that this color and thickness, made the edge standout from the rest of the edges and helped in showing it distinguished without being unpleasantly distracting.

4.3 Explanation Panel

Whereas, the animation panel on the left page contains all the graphical components of the topic elucidation. The right half of the page is occupied by the Explanation Panel. It contains the title of the problem being explained, it's definition and instruction on how to go ahead with starting the animation or user-interactive tasks. It also contains control buttons for starting, re-starting and pausing animations. For user-interactive tasks it has buttons to reset the task.

As the animation or a user-interactive task progresses the explanation panel generates explanatory and instructive text.

4.4 User Stories for Elucidation of Topics

This section describes, what the user is intended to experience while interacting with the individual topics in the application with a special consideration towards learning impact and understandibility.

When the user opens the web application on a browser of her choice, the first topic she sees is that of Graph Isomorphism. She may stay there to interact with the example of the topic or navigate to other topics using navigation buttons at the bottom to have a bird's eye view of other topics.

Each graph theory problem has it's own character and require a different approach for elucidation. The following sections will explain these approaches with their intended experience on the user with learning outcomes which may be achieved.

4.4.1 Graph Isomorphism

The user is presented with a graph on the left and textual explanation on the right of the page. The textual explanation portion of the page also has some media buttons such as play/pause and reset to interact with the explanation. The text explanation briefly defines graph isomorphism and advises the user to press the play button.

Animation: When the user presses the play button, a new graph emerges out of the old one while the keeping the edges between any two vertices conserved. While keeping the connectivity between the vertices intact, the graph transforms into a completely new shape, almost giving a visual proof that the two graphs on the screen are isomorphic to each other.

User Interaction: After the two isomorphic graphs have separated from each other, the user is advised in a text panel to choose a vertex by either hovering over a vertex or pressing the corresponding number on the keyboard. Doing so, will change the visual appearance of the selected vertex, the edges incident on the selected vertex and the adjacent vertices to the selected vertex in both graphs. The selected vertex will be enlarged to a new radius and change its color from it's original color to golden color making it stand apart from the rest of the vertices. The edges incident on the vertex will change their colors to the same gold color. The adjacent vertices to the selected vertex will form a golden halo around them. This color transformation will distinguish a kind of a subset in the two displayed graphs. At this point of time, in the text panel the user is pointed out that the selected vertex has the same number of edges connecting to the same adjacent vertices in both the graphs. The user is also advised to inspect other vertices of the graphs and convince herself that each vertex of the graph has the same adjacent vertices in both the isomorphic counterparts.

Learning Impact: The transformation of a graph into a radically different looking graph but being essentially the same as far as the connectivity between the vertices go acts as a visual proof that the graphs are isomorphic. While individually inspecting each vertex will re-confirm this idea to the user. After having experienced the concept of graph isomorphism in this way it is assumed that the concept and definition of the term would be clear to her.

4.4.2 Max k Cut

Max Cut has two animations one after the other. The first animation is about Max 2 Cut and the second is about Max 3 Cut. It is assumed that the user will extrapolate the concept of the general Max k Cut after understanding the first two cases ($k = 2$, $k = 3$) and extrapolating it over greater values of k. The examples shown in the animations are a nearly bipartite and a tripartite graph for $k = 2$ and $k = 3$ respectively. A nearly bipartite and a tripartite graph is used to elucidate the topic as it is easier for the user to visualize how the two graphs can be segregated to sets of vertices such that the maximum number of edges pass between such sets.

In both the cases of $k = 2$ and $k = 3$, the weight of all the edges is taken to be equal to 1. This decision has been taken as with $w = 1$, the answers to both the max cut problems are more visual than unequal weights.

Max 2 Cut Animation It starts with an Original graph on the left and definition of Max K Cut and Max 2 Cut on the right of the web page. The right part of the page also contains media buttons to pause and play the animations. It also has a button for switching from Max 2 Cut example, to Max 3 Cut example. The Max 2 Cut animation starts with a graph, which starts when the user presses the play button. As the animation progresses a new graph emerges out of the original one and translates towards right changing it's shape to segregate it's vertices into two sets forming a Maximum 2 Cut. The two sets move vertically up and down and increase the distance between themselves, revealing the number of edges passing from one set to another which the user can intuitively tell is greater than the number of edges between any other two sets which may have been formed from the vertices of the graph.

The user is advised to put up a pre-defined horizontal line by pressing a button in the explanation panel. The line is drawn between the two sets of vertices. The intersection points between the edges and the max cut line is shown by blue dots. These user is advised to observe the number of intersection points which tell the user the number of edges passing from one set to another.

Max 3 Cut Animation Just like the animation for Max 2 Cut, this animation is started by the user by pressing the play button. The animation on the right starts with a tripartite graph arranged in a circular form. As the animation progresses the graph gets Divided into three sets of vertices in which the three sets translate in directions which are set 120° apart from each other. The graph transforms from a circularly arranged one to a triangular form. The user can draw the Max 3 Cut lines at any point in the progress of the animation. These are three lines

separating each set from the rest of the graph, with points of intersection shown in blue showing the number of edges passing from one set to the rest of the sets.

Learning Impact: Although the examples in the Max 2 Cut and Max 3 Cut can be seen as simple ones as the first one was nearly a bipartite graph and the second one was a tripartite graph, they do a good job at defining the problem well. Not just that such visualization explains the problem well it also may act as artwork especially in the case of Max 3 Cut, which may inspire an imaginative student to want to investigate the subject further.

4.4.3 Graph Coloring

Graph Coloring is explained to the user with the help of a user-interactive task. The user is presented with a graph which has all the vertices in color white. On the explanation panel to the right he is given the definition of the problem and advice on how to complete the task.

The task is to choose colors from a color palette of three colors namely red, green and blue, and color vertices in the graph such that no two adjacent vertices have the same color. Whenever the user colors two or more adjacent vertices the same color the text panel warns them to make amends. There is a reset button in the explanation panel to un-color all the vertices to start all over again if the user wants to start from the beginning.

The user is challenged to first challenged to color the graph successfully in just two colors but as it would be soon clear to the user it can only be done in three.

Learning Impact: The aim of the user task is to make the user understand the problem, and retain this in their memory in for a long period of time as user interactive task can create an impression of a problem for a very long period of time than just watching an animation as a spectator.

4.4.4 Minimum Vertex Cover

Minimum Vertex Cover, just by the nature of the problem is chosen to be elucidated by the help of a user-interactive task. The user is given a graph along with explanation of he Minimum Vertex Cover problem. He is also explained how to complete the task. The task is to select vertices successively either by clicking them or pressing a number corresponding to the vertex of choice on the keyboard. When he selects the vertex, the selected vertex and all the edges incident on it are displayed differently. He has to thus highlight all the edges by selecting the minimum number of vertices in the graph. If he has done this task effectively then he would not choose more vertices than required to cover all the edges in the graph. When the user covers all the edges by only selecting four vertices, he is given a congratulatory message for having done the task right. If he covers the graph by selecting more than four vertices then he is advised to do the same in just four.

Learning Impact: Just like Graph coloring, in the case of Minimum Vertex Cover too, it is assumed that a user-interactive task is effective not just in explanation of a topic but also, retention of the concept for a long period of time.

4.4.5 Tree Width

The topic Tree Width is explained in a multi-part animation. The first part begins with a graph in which the vertices are arranged in a circular pattern. The circular form conceals a tree like structure which can be abstracted out of the graph.

The user while reading the explanation is instructed to press the 'forward' button to move to the first part. The user hops from one part of the animation to another by pressing this 'forward' button. In the first part the graph which was hitherto arranged in a circular pattern transforms

into a regular lattice like pattern. The tree-like pattern is more apparent in the new visual form of the graph. This is also pointed out in the explanation panel.

The next part of the animation shows an example of a piece (a sub-graph) containing three vertices against the backdrop of the graph. The significance of pieces in the tree-width concept is discussed in the background chapter. The piece is also represented by a blue dot at the centroid of the three vertices. In the next part of the animation the whole of the graph is marked by it's constituent pieces by blue dots.

In the final part, the pieces form the nodes of a tree. The tree's edges (branches) are colored in golden color to make it stand out from the graph in the background. At this point, the definition of the tree width is given in the explanation panel.

Learning Impact:

5 Implementation

5.1 Front End Development with The Elm Programming Language

The project is developed using the Functional Programming paradigm. This is a paradigm which has been in development and practice since the days of infancy of computer science. Functional programming is based on a form of computation called lambda calculus proposed by Alonzo Church.

For most of the history of computing functional programming remained in the ivory towers of universities for purposes of exploring theoretical computer science and language research.

In the last decades however, programming languages such as Haskell and a few dialects of LISP have escaped the ivory towers to find application in the software industry.

In this section we will discuss, why functional programming was chosen as the programming paradigm of choice. How the functional programming language called Elm is used to write a well organized, maintainable, intuitive and understandable code to produce a dynamic front end.

5.1.1 Why Functional Programming?

Functional programming, makes the programmer think in a different way than what may be called imperative programming. In the functional paradigm, functions are first class citizens, which can be mashed up together with each other, by giving a function as input to another function, a function giving out another function as an output, function composed of two functions dove-tailed to each other, programming patterns being abstracted out as functions and so on.

For such Lego like usage of functions they must be dependable, such that for a particular input a function will give a particular output just like mathematical functions and has no business outside it's scope for side-effects. With such confidence in the functions, they can be fitted with each other to make them do complex computation.

Separation of Concerns It may be asked that if functions don't have side effects, how do they print output on the terminal or read file from the hard disk or accept inputs from a user. Functional programming environments have a way of separating the pure part of a program from the impure part, by introducing 'actions'. These 'actions' or side-effects are treated as a form of encapsulated data, which can be manipulated by pure functions, and the environment makes changes to the outside world by executing these actions.

Therefore the programmer has to himself a large part of the program where he deals with just pure functions. This allows him to exploit the perfectness of pure functional programming.

This separation of concerns of pure and impure code in the context of the Elm programming language is discussed in the next section called the Elm Architecture.

5.1.2 The Elm Architecture

The Elm Architecture is a pattern of writing Elm code for responsive web applications. The architecture separates the concerns of front-end development into the following categories:

1. Model
2. View
3. Update

The Model is a data structure which holds the state of a program. This state is used by the view function to render a webpage. The webpage, when rendered has elements, which may trigger events, such as user inputs by the way of clicking an HTML element. Such events are caught by the Elm runtime and sent to the update function. The update function takes these event messages and changes the state. The changed state is then rendered by the view function to a modified page. Therefore, the Model is changed by the update function, whereas it is used by the view function to render a webpage according to a formula set by the programmer.

The events described above are called messages in the Elm way of naming things. For this particular application they are defined as an Algebraic Data Type as:

```
type Msg
    = TimeDelta Float                   -- Clock Ticks for Animation
    | HoverOver Int                     -- Event when Mouse over a Vertex
    | MouseOut Int                      -- Event when Mouse out from a Vertex
    | VertexClicked Int                 -- Event when Vertex Clicked
    | AnimationToggle                   -- Pause or Play Animation
    | AnimationStartOver                -- Restart Animation
    | ToggleVertexStatus Int            -- Select/Unselect Animation
    | NextTopic                         -- Next Topic
    | PreviousTopic                     -- Previous Topic
```

The messages are not just generated by user interaction with this application, they are are also generated by the animation clock as well as can be seen in the first data constructor of the type *Msg*. The clock ticks and the key strokes are events which initiate the update function to act on Model. The animation clock and key presses need to be subscribed from the Elm runtime in the following way:

```
subscription : Model -> Sub Msg
subscription _ =
    Sub.batch
        [ E.onAnimationFrameDelta TimeDelta
        , E.onKeyPress keyDecoder
        ]
```

5.2 Implementation of Graphs

Inside the program, a graph exists as data structure which contains a list of vertices and a list of edges. The vertex which is a data type defined separately consists of a name (which is an integer), a color, a 2D position (it is actually implemented using a 3D vector, with z is always kept at zero). An edge on the other hand is defined as a combination of two vertices. Such Graphs are present in the Model in and are used by the view function to be drawn as SVG.

5.2.1 Grid

In the program a Grid is a list of 3D vectors, which can be taken in by certain functions to make graphs or change shapes of graphs. A list of Vertices, therefore can be formed by zipping together lists of names, colors and a grid. There are functions, especially for implementing animations, which takes two grids and outputs a grid which is geometrically in between the two grids.

5.2.2 Using Linear algebra to Initialize Grids

Linear algebra, in particular manipulation of vectors using Matrices has been used to create interesting grids for the placement of vertices in the scene. This includes rotation, scaling and translation of vectors to from polygonal patterns. Functions were created to form polygon with n geometric vertices which prove very handy in producing grids for various geometries like the one seen in Graph Ismoporphism and Max k Cut examples.

As a small example, here is a functional programming code to find the centroid of of three position vectors. You can observe how first two vectors are added on line 1, and then it is pipelined to addition with a third vector, which is in turn pipelined to being scaled by 0.33 (divided by 3.0). This could have been achieved in a single line of code, but Elm reserves operators like $+, -, *$ for only numbers and they can't be overloaded to work for vectors.

```
Math.Vector3.add v1.pos v2.pos
|> Math.Vector3.add v3.pos
|> Math.Vector3.scale 0.333
```

5.2.3 Implementing Colors

To have a list of neighboring colors acting as a color palette we work on the Hue Saturation Lightness color space (HSl), mostly varying the hue just pass a region in the spectrum of hues (First, Second or Third) and the number of colors needed as an Integer. On a scale from 0.00 to 1.00, the first region will produce hues ranging from 0.00 to 0.33, the second producing it from 0.33 to 0.66 and the third producing it between 0.66 to 1.00.

5.2.4 Edges

Edges are defined as a combination of two vertices. Since they are drawn as a straight line segment between the positions of the two vertices, they do not require positional data associated explicitly for them. It is drawn out from the vertices, they contain.

5.3 Implementation of Animations

In this section, it will be explained how various animations in the application are implemented. Though there are minor differences between animations for one topic to another, they follow a common pattern. The common pattern is this that events are generated by a quasi-regular clock. These events trigger the update function which transforms the current state of the program and changes the position of certain abstract entities. The view function while redrawing these entities takes the position information from the updated model to draw them as SVG (Scalable Vector Graphics).

5.3.1 Morphing Geometry of a Graph

In some of the animations in the application, the graph changes it's geometry to visually look different than the original. This is accomplished by a function which takes a graph and a grid to move the input graph incrementally towards the grid with every tick of the animation clock. The function calculates the displacement vector between a vertex and the respective final position given in the grid and finds a new position along the direction of the displacement vector.

5.3.2 Re-formation of the Graphs

At each tick of the animation clock the graph under transformation, is built again, with vertices having the same name and color as the original but new positions. The edges need to be re-constructed again as the vertex positions have been renewed. This is something which is expected in the functional programming paradigm where nothing is changed in place and new data structures are created with application of a function. This is true not just for animations, it is true for user-interaction or anything which requires visual (Geometric or Color) modification of the graph.

The re-formation of the vertices and the edges are quite explicitly shown in the Elm function below. The function takes a graph and a grid and produces a new graph situated at the new grid with new vertices and edges. The edges formed in the new graph are connected to the same vertices as the original ones.

```
morphGraph : Graph -> Grid -> Graph
morphGraph graph grid =
    let
        updatedVertices =
            List.map2 updatePositionVertex graph.vertices grid

        createEdge =
            updateEdge updatedVertices

        updatedEdges =
            List.map createEdge graph.edges
    in
    Graph updatedVertices updatedEdges
```

5.3.3 Drawing of Graphs

Drawing graphs is done using SVG elements. The vertices are drawn as color filled circles while the edges are drawn as straight line segments between the positions of the related vertices. The edges are drawn first and the vertices later so that vertices appear on top of the edges and the edges seem to be appearing out of the surface of the vertices.

5.4 Explanation Panel

The Explanation Panel consist of the title of the topic and suggestions on how to interpret and interact with the animations and user interactions. This panel is populated with text and buttons by functions which take the state (Model) of the program as input. According to the state of the program appropriate advises, instructions and buttons are spawned on the panel.

The functions responsible for Explanation Panel in the case of user interaction such as the ones in Graph Coloring and minimum vertex cover run a check on the state of the program to know if

the user is doing his task correctly. For example in graph coloring, if two adjacent vertices are colored the same color, there appears a message in the explanation panel warning about the same.

5.5 Implementation of Miscellaneous features

There are a few miscellaneous features which must be mentioned here.

5.5.1 Navigation Bar

The navigation bar consists of buttons to go to the previous and the next topics. When the message *PreviousTopic* or the message *NextTopic* is generated by the buttons, the update function catches it to change the state of the program to load it with the details of the previous/next topic.

5.5.2 Keyboard Shortcuts

Keyboard shortcuts provide a fast way to test various functionalities while developing the application. These functionalities have not been taken away even after development therefore they still can be used to trigger events in the application. Below is an example list of a few key-bindings.

- p: Toggle between pause and play animation. (Can be used instead of the Play/Pause button).
- r: Restart animation. (Can be used instead of Restart Button).
- n: Go to the next topic. (Can be used instead of the navigation button).
- N: Go to the previous topic. (Can be used instead of the navigation button).
- t: Next animation (Can be used in case of Max k Cut and Tree width to go to the next animation)

6 Evaluation

7 Conclusion

A Appendices

7 Bibliography