University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Visualization of Classical Graph Theory Problems

Fatma Al-Sayegh

20 March 2023

Abstract

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 this project an application to elucidate some classical problems in Graph Theory by the way of visualization is developed. The methods of visualization are animations and user interaction with such animations. These methods aim to help young students and self-learners as to get the first brush with the subject of Graph theory.

The details and experience of analysis, design, development, testing and evaluation of the application is discussed in this report.

The web application can be reached following this URL:
https://visualise-graph-problems-with-me.netlify.app/ .

Acknowledgements

I would like to thank my supervisor Sofiat for giving me a wonderful opportunity to explore the topics of Graph Theory. Her constant support and guidance throughout the project was essential in building the app. I would also like to thank my father for encouraging, believing and supporting me constantly in both the good and bad days. Finaly, I would like to thank my friend Shrey for introducing me to the Elm programming language. This project would not have been possible without the the help, aid and advice of friends and family.

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

1.1 Motivation

There are various kinds of phenomena in nature and concepts in computer science which can be modeled as graphs. Graphs can be thought of as points (called vertices or nodes) related to each other by edges. The points can be humans in a social network, genes in a gene expression network, proteins and metabolites interacting with each other in a cell, various living species in an ecology or a food-web. Graphs can also model the nature of relations between such objects, such as causation, interdependence, interaction. There is a new field emerging in computer science and statistical mechanics called *Complexity*, which promises to enlighten us about phenomenon such as life, ecology, intelligence, society and economy. Graphs stand as one of the fundamental tool to model and study such complexity. Gros (2015).

Reducing a real world problem to a graph may amount to losing details but as graph theory has developed a number of tools for analysis and insight, these tools became available immediately for a well formed inquiry. Therefore once a concept in Graph theory is learned then just like other concepts in Mathematics such as integers or vectors, it can be applied in a variety of fields of study.

Although the most dependable way to study Graph Theory should be mathematical formalism to state the terms, definitions and theorems; visualization of such concepts can act as a first stepping stone for the uninitiated. It can also act as an aid for a practitioner to enrich his understanding or to view the same concept the same concept in a different light.

Using a particular example to define a topic in mathematics may diminish it's generality. But a particular instance of a topic can give a concrete confidence to a student about his understanding of the topic in consideration. It is expected that a concrete example can be extrapolated by a student's mind for a variety of situations and also a general understanding.

1.2 Aim

The aim of this project is to create computer visualization of classical graph theory problems such as Graph Coloring, Graph Isomorphism, Max Cut etc. To that end graph theory problems which are important in the field should be shortlisted along with searching and designing of simple examples and user stories to elucidate the definition of such problems. The method of elucidation should be animation, interactive tasks and textual explanation on a modern front end web application. To achieve this, a program should be written to represent graphs and conduct basic operations on them; with functionality to spawn such graphs visually on screen and animate them using basic kinematics. Finally, the advice of peers in the field should be sought to evaluate the application in terms of usability and learning impact.

1.2.1 Link to the Application

For the reader's reference the application can be visited following this URL:
https://visualise-graph-problems-with-me.netlify.app/.

2 Background

This chapter introduces concepts which are essential to understand the later chapters in this report. First, I review relevant books and research papers referred and consulted for this project, then discuss Graph Theory definitions and its classical problems, which are useful to understand their implementation in the application. Discussion of prior work on visualization of computer science topics which have an influence over this project is also part of the research.

2.1 Literature Review

The books and the research papers cited in this project range from theoretical texts on Graph Theory, to programming books on functional programming and the web front end functional programming language called Elm.

2.1.1 Graph Theory and Algorithms Literature

Parts of the following literature was used to gain some basic familiarity with the basics of graph theory and understanding of definitions, theorems and explanations of classical graph theory problems. This literature also played a role in shortlisting the graph theory problems which will be implemented as examples in the developed application.

Networks: An Introduction. Newman (2010) was consulted for definitions, explanations and understanding of basics of Graph Theory and its problems. It discusses the mathematical and computer science perspectives of the subject in dedicated sections.

Algorithm Design. subsection 2.2.6 lent the description and the example of *Tree-width* problem incorporated for an animation in the application. Kleinberg and Tardos (2006) covers most kinds of data structures and has excellent portion on mathematical and computer science aspect of Graph theory.

2.1.2 Functional Programming and The Elm Programming Literature

In this section, literature covering functional programming and in particular Elm programming language which were helpful in this project is reviewed.

Why Functional Programming Matters. Hughes (1989) is a concise tutorial using a Haskell like programming language called Miranda, presenting the power of functional and modular thinking to glue functions together to create and manipulate data structures like lists and trees. It also discusses recurring patterns in programming and their abstraction in form of higher order functions such as maps, filters, and folds. In a steep learning curve it goes onto discuss some advanced concepts in Artificial Intelligence.

Elm in Action. Feldman (2020) may be used as a step by step tutorial introduction to the Elm language and its use in creating real world web applications. It starts with small applications and gradually moves to the management projects of considerable size. It acts as a bridge between functional programming and front-end design; it also covers into front-end design recipes; for example how to implement a single page application.

Model-View-Update-Communicate: Session Types meet the Elm Architecture. Fowler (2019) gives the first formal model of the Model - View - Update (MVU) programming pattern popularized by the Elm programming language. MVU architecture in Elm has directly inspired technologies such as *Redux* and *Flux* architecture used with popular *React* web framework. The paper lays down the Syntax and semantics of a calculus called λ_{MVU} which formalises the MVU programming pattern.

2.1.3 User Interface Aesthetics and Usability

The Beauty of Simplicity. Karvonen (2000) presents arguments for how optimal simplicity, minimalism and intuitiveness make a user interface more usable and trustworthy. This paper inspired me strongly to keep the application simple by minimising the number of complicated control and features.

2.2 Discussion of Classical Graph Theory Problems

This section formally discusses the concepts of Graph theory which are elucidated visually in the application. I will be referring to the subsections here for definitions and explanations and I recommend reading this section carefully for readers not yet familiar with this topic.

2.2.1 Definitions

A *graph* $\boldsymbol{G}$, can be understood as a collection of vertices which are connected to each other by edges. A *vertex* $\boldsymbol{v}$ can be understood as a point and an *edge* $\boldsymbol{e}$ is a pair of vertices. The *set of all the vertices* in a graph G is represented as $\boldsymbol{V(G)}$ and the *set of all the edges* in G is represented as $\boldsymbol{E(G)}$.

For a vertex v, its *degree* $\boldsymbol{deg(v)}$ is the number of edges connected to it. An *isolated vertex* v is such that $deg(v) = 0$. An *end vertex* is a vertex w such that $deg(w) = 1$. Two vertices are *adjacent* to each other if there is an edge that connects them.

A *bipartite* graph is a graph G such that its vertices $V(G)$ can be split into two disjoint sets A and B such that each edge of G joins a vertex of A and a vertex of B. Newman (2010)

2.2.2 Graph Isomporphism

Two graphs G_1 and G_2 are isomorphic if there is a one to one correspondence between the vertices of G_1 and G_2 such that the number of edges between any two vertices in G_1 is equal to the number of edges joining the corresponding vertices of G_2. Given two graphs, detecting if the graphs are isomorphic is a problem to solve as the graphs may appear to be different in appearance and in the labeling of the nodes and edges. Newman (2010)

Application The graph isomorphism problem finds application in the field of bioinformatics for finding network motifs (sub-graphs isomorphic to an input pattern) in a larger biological network. A network motif is a recurring pattern of connection of vertices in a large graph signifying their evolutionary selection over random patterns. Bonnici et al. (2013)

2.2.3 Max K Cut

A maximum cut is partioning the vertices of a graph in two groups such that the number of edges between these two groups is maximum. In a weighted graph, where the edges are weighted, the weights of the edges are also taken into consideration. A maximum k-cut is generalized version of maximum cut, where the graph is partitioned into k subsets, such that the number of edges between these groups is maximized.

It is important to note that a bipartite graph (refer to the Definitions section above) is a trivial example of Max Cut there are no edges among the vertices of a set A and no edges among the vertices of set B and all the edges are from the vertices in set A to vertices in set B.

2.2.4 Graph Colouring

Graph Colouring is optimization problem where the objective is to assign to the vertices of a graph a colour such that no two adjacent vertices have the same colour, while keeping the number of colours employed to a minimum. Here a colour can be thought of just any symbol from a finite set of symbols.

2.2.5 Minimum Vertex Cover

Minimum Vertex Cover of a graph is the minimum amount of vertices such that, all the edges in the graph must have one of such vertices as at least one of their endpoints. This is also a optimization problem in which the constraint is that all the edges must be covered while keeping the number of vertices in the set of Minimum Vertex Cover to the minimum.

2.2.6 Tree Width

This topic will be discussed in two parts. First a tree decomposition of a graph is described, then the tree width of a graph will be defined.

To decompose a Graph in a tree is to put nodes into sets called pieces, subject to certain conditions. The first condition is that all the vertices of G should belong to at least one piece. Every edge of G, must be present in at least one piece which contains both ends of the edge. And finally, in the tree decomposition, if there is a node n present in a walk from a node n_1 to n_2, and if both n_1 and n_2 have a vertex v in common, then the node n also contains that vertex v.

Any graph can be decomposed into a tree. Trivially, a graph can be tree decomposed by putting all of its vertices in just one node. But it will not be a very useful tree decomposition. Therefore a good tree decomposition of a tree is the one which has small pieces. Tree width is defined as the size of the biggest piece $V_t - 1$. A tree-width of a graph is the minimum of such possible tree widths across all possible tree-decompositions. Kleinberg and Tardos (2006)

2.3 Prior Work

This project takes subtle inspirations from some of the work which is available on the internet as web applications for visualization of popular algorithms. Although the works which are discussed in this section are focused on understanding algorithmic solutions of computer science problems, the visualization of graphs, trees and lists in these projects have been inspiring for depiction of graphs and their animation in this project.

2.3.1 Data Structure Visualizations

This tool was developed by David Galles, Associate Professor at University of San Francisco. Galles covers topics from various categories of computer science problems such as Dynamic Programming, Geometric Programming, Trees, Heaps and Graphs.

The design of the tool has several important features: The user has the facility to define their own data-structures rather than they being predefined or hard-coded. There are control buttons which allow the user to start pause and restart the animations. There is a slider to tune the speed of the animation as well.

I find this tool lacking in textual explanation of the algorithms while they run and I assume that the main purpose of this tool is as a classroom teaching aid such that the teacher first explains the topic and uses the tool as a visual demonstration to show his students the working of the algorithm on real data structures.

2.3.2 VisuAlgo

VisuAlgo was developed by Dr. Steven Halim of National University of Singapore. Halim covers topics from the subject of data structures and algorithms. Most relevant for this project are the topics related to graph theory; which are Maximum Flow, Minimum Vertex Cover, Traveling Salesman and Steiner Tree, although the emphasis is on algorithmic solutions to the problems and not problem visualization which is the emphasis of this project.

For most topics the user is able to construct their instances of datastructures. Unlike Data Structure Visualization application mentioned in subsection 2.3.1, there is an ample amount of textual information in terms of theory, tutorial and instructions. The explanation of the topics is done in text blocks which appear at appropriate places in a slide show like fashion. For organizing the textual information, it has drop down content menu for easy access to various sections. Although, different positions of the text blocks can be a little distracting. In this project, the text explanation is shown in one place.

2.3.3 Algmatch

Algmatch was developed as a final year individual project dissertation for Liam Lau under the supervision of Sofiat Olaosebikan, supervisor of this project as well. The application visualizes the matching algorithms such as Gale-Shapley Stable Matching and Extended Gale-Shapley Stable Matching algorithms applied to stable marriage and hospital/residents problem. Lau lends ideas about user friendliness and intuitive usage.

It has a panel which describes the algorithm steps while in an animation the matching algorithm works on an instance of the problem.

Aesthetically, the most noteworthy features of the app are, playback and speed controls which are as intuitive as media buttons on a media player and smooth page transitions resulting in fluid user interactions.

3 Analysis/Requirements

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

3.1 Scope of The Project

Understanding a problem in mathematics is a necessary first step towards the solution. Abstraction enables a student to obtain a formalized version of a problem from a real life scenario present in the fields of science and engineering.

This idea has guided this project to focus solely on aiding a learner to understand the problems in graph theory. The solution 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 and well defined.

3.2 Criteria for Selection of Problems

One of the most important criteria for selection of the problems for the project was their importance and relevance in the field of graph theory. There are several text books (Newman (2010), Kleinberg and Tardos (2006)) which discuss various theorems and problems in the subject. A few problems occur frequently in them and 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 levels of difficulty.

Since imagining a graph theory problem is largely a visual exercise, there are plenty 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 does not in any way 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 fairly quickly. From there on they can extrapolate what they learnt to more complex examples.

A survey among my 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 (see section A.3) had a role in determining topics and the methods chosen.

3.3 Methods of Elucidation

Methods of elucidation and exposition played a prominent role in the selection of the problems in the first place. It was decided that for this project, such methods can be broadly classified as animations and user interactions or a combination of both.

3.3.1 Animation

Humans tend to prefer to imagine even the most abstract concepts visually. Computer animations have been increasingly adopted to elucidate complex concepts in mathematics and science. Animations will be 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 its shape to take on a completely unrecognisable appearance. Later in this report I will cover the topic of 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 engaged and involved in the learning process they only offer a one-sided, 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, not dissimilar to a video game.

3.4 Technical Scope of The Project

To achieve the outlined features the program needs data structures to hold graphs and algorithms to visually and geometrically manipulate them.

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 I provide the solution to the visualization program before hand to give enough information to the various animations and user interactions.

3.5 Requirements

The requirements for the application are categorized into priority levels in accordance with *MoSCoW* analysis. This prioritization of requirements act a guidance for deciding which objectives are essential and make the minimum viable product and which ones should come in the category of extra desirable features. Hudaib et al. (2018)

The application must have the features mentioned in the *Must Have* section which is the minimum requirements necessary to have a functioning application without any extra decorative or convenience features. It should have the additional desirable features mentioned in the *Should Have* section relating to visual aesthetics, wider use and code extensibility. Finally to precisely define the boundries and discuss potential future development of the project there is section of *Will Not Have This Time*.

3.5.1 Must Have

The application must elucidate the following enumerated *Classical Graph Theory Problems* by employing user interactive animations of their respective examples.

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

5. Tree Width

And to achieve 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

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

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

3.5.2 Should Have

Although not essential for the basic utility and functioning of the application, a few desirable features should be included for wider use and additionally ease of contribution by others.

1. Pleasing, appropriate aesthetics
2. Device compatibility — the application must be usable on most common screen sizes.
3. Contribution friendliness, in the way of good code organization and documentation.

3.5.3 Will Not Have

For the reasons elaborated in section 3.1 and section 3.4 the project will not implement algoritmic solution of the the graph theory problems. Although it will have some algorithms implemented to check if a given solution is correct specially in user-interaction tasks. The algorithmic solutions to the problems can be taken up as future work for the application.

4 Design

In this section I design the design choices made for the application and the guiding principles behind them. It progresses to show how a graph are visually displayed and animated in the app without going in the technical details of implementation. Finally, the intended effect on the user while going through the various topics explained in the application will be discussed. I will also touch on the learning impact of each topic on the user as it forms an essential aspect of overall design.

4.1 Guiding Principles

4.1.1 Simplicity

For the purpose of elucidation of Mathematical concepts, which requires an undivided attention of the learner, it was decided that the user interface must have the minimum amount of clutter possible, without sacrificing in terms of the minimum amount of functionality required. The learner should not get distracted by an overpopulated user interface. Simplicity and elegance would also lead the user to stay longer on the application without getting visually exhausted. Furthermore the users on the web today are extremely goal driven and do not want any obstruction between them and their goal. Karvonen (2000)

4.1.2 Intuitiveness

The layout of the user interface should be such that it not only has utility, but should also help communicate the intention of the designer about the usage of the application. For example a play button, just like it was found on media devices for decades, invites the user to kick-start an animation even without going through the instruction which tells him to do so explicitly. The size and placement of the play button on the page, therefore becomes important. Right placement of user interactive elements guide the user through the story which is intended to be told.

4.1.3 Meaningfulness

For a substantial learning impact, the elucidation of the problems must reach at the heart of the topic. They must have a storyline which is meaningful and coherent. The learning outcomes of the animation or a user-interactive task must be well defined before an attempt is made to implement them.

4.2 Wire-frame and Navigation

The web application is implemented as a **Single Page Application**. The navigation from one topic to another occurs according to the user inputs, and 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 loading a new HTML page each time. The user however will notice the url change with navigation from one topic to another. This will also give the user the ability to use forward and backward buttons in the browser to navigate through the history of URLs visited. As a new HTML page is not loaded every time the user navigates from one topic to another the screen therefore does not turn briefly white and the transitions are imperceptible by default.

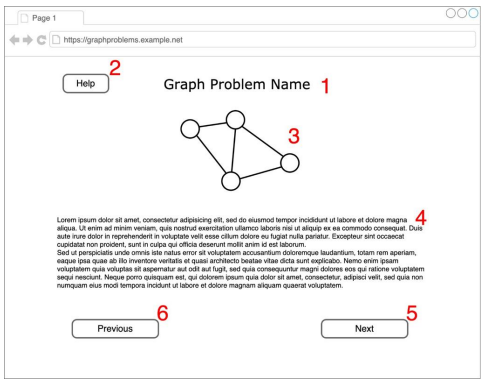

Figure 4.1: *An initial wireframe*

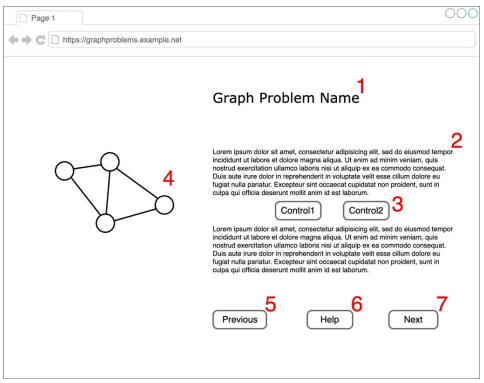

Figure 4.2: *The final wireframe*

There were several iterations made for the layout of the web page, one of them is shown in Figure 4.1. Finally a more suitable and visually balanced design was selected, see Figure 4.2. On this layout, the page is vertically 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 its 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 jump from one topic to another.

4.2.1 Home Page, About Page and Header Bar

The home page is the landing page of the application. It gives the user clickable icons for navigation to the topics. The icons must have the name of the topic and a miniature graph which best represents the topic.

The about page has a short description of the purpose of the application with a short write up introducing the developer and the supervisor of the project. It also has an acknowledgement section mentioning the people who have been important in the completion of the project.

Every page has a header bar, which has buttons to navigate to home and about pages respectively. The bar stays at its place even if the rest of this page is scrolled down for access to navigation.

4.3 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.

4.3.1 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 two-dimensional position. The edges do not 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 colour filled circles of the same size save for certain exceptions. The colour of the vertices have been allotted by varying mostly the hue and keeping saturation and lightness relatively same in the **HSL** (Hue Saturation Lightness) colour 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 may be understood better by users.

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

Appearance of Edges Edges are defined very algebraically as a relationship between a pair of vertices, but they are simply drawn by connecting the positions of the related vertices with a straight line segment white in colour.

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

4.4 Explanation Panel

The animation panel on the left page contains all the graphical components of the topic elucidation and the right half of the page is occupied by the Explanation Panel. It contains the title of the problem, its 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 also shows a button to reset the task.

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

The explanation panel also has navigation panel with buttons to navigate from one topic to another.

4.5 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 Graph Isomorphism. They 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 its 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.

The survey mentioned in the section 3.2 also gave quite a few suggestions on different ways to elucidate the short listed topics. The suggestions were very specific to topics, and included various ideas such as animations and games and the ability to construct user defined graphs. Some of the suggestions could be included in the project, some could not be accomodated as they did not fit the flow of narrative, while others while being brilliant ideas, act as inspiration for future work due to constraints of time.

4.5.1 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 control the state of the animation. 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 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 colour from its original colour to golden colour making it stand apart from the rest of the vertices. The edges incident on the vertex will change their colours to the same gold colour. The adjacent vertices to the selected vertex will form a golden halo around them. This colour transformation will distinguish a kind of a subset in the two displayed graphs. At this point, on the text panel the user is reminded 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 note that each vertex of the graph has the same adjacent vertices in both the isomorphic counterparts.

Quiz: The user can with a click of a button go to the quiz page which resides within the topic of Isomorphism. The application asks the user to select a graph (among more than one graphs) which the user thinks is isomorphic to a given reference graph. When they make their choice, they are presented with the right answer along with an animation on how the correct answer is isomorphic to the reference graph.

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, and individually inspecting each vertex will re-confirm this idea to the user. After having experienced the concept of graph isomorphism in this way, hopefully the concept and definition of the term would be clear to the user.

4.5.2 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: This animation 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 the right changing its shape to segregate its 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.

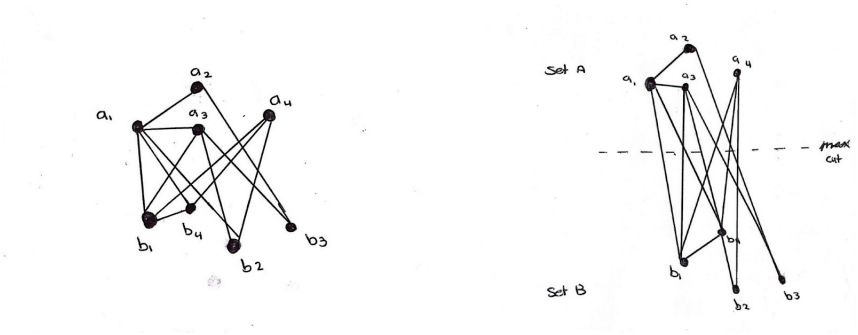

Figure 4.3: *Initial design of Max 2 Cut. The initial graph is shown in the left. It must go through a transformation such that the two sets of the Max Cut are separated form each other as shown in the right. Preferably there should be a line between the two sets for visual separation.*

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, I believe they will do a good job at defining the problem. Beyond the visualization explaining the problem, they may also have some artistic value in their aesthetic and satisfying appearances, especially in the case of Max 3 Cut, which may inspire an imaginative student to want to investigate the subject further.

4.5.3 Graph colouring

User Interaction: User-interactive task was chosen for explaining Graph colouring as the nature of the problem lends naturally for this implementation. The user is presented with a graph which

has all the vertices in colour 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 colours from a colour palette of three colours namely red, green and blue, and colour vertices in the graph such that no two adjacent vertices have the same colour. Whenever the user colours two or more adjacent vertices the same colour the text panel warns them to make amends. There is a reset button in the explanation panel to un-colour all the vertices to start all over again if the user wants to start from the beginning. The user may also complete this task in another form where there are only two colours available.

Learning Impact: I am hoping that introducing graph colouuring as an interactive task would help the user understand the concept in question faster and also gain a longer lasting memory than with the help of a non-interactive video or other less engaging formats.

4.5.4 Minimum Vertex Cover

User Interaction: Minimum Vertex Cover, 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 the 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 the user selects the vertex, the selected vertex and all the edges incident on it are displayed differently. The user has to highlight all the edges by selecting the minimum number of vertices in the graph. If they have done this task effectively then they will 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, they are given a congratulatory message for having done the task right. If they cover the graph by selecting more than four vertices then they are advised to do the same in just four. The user also has the option to do this task with greater number of vertices and edges to make the problem slightly more difficult.

Learning Impact: Just like Graph colouring, 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 longer period of time than a more static method of elucidation.

4.5.5 Tree Width

Animation: 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 in 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 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 its constituent pieces by blue dots.

In the third section of the animation, the pieces form the nodes of a tree. The tree's edges (branches) are coloured in golden colour to make them stand out from the graph in the background. At this point, the definition of the tree width is given in the explanation panel.

The final part gives a different tree decomposition to the one mentioned earlier with size of a piece increased to four. This shows that a graph can be decomposed in more than one way.

Hence the tree-width corresponding to this decomposition would be four. Ideally, though the tree decomposition of a graph should be performed in such way that the tree-width is kept at the minimum among all possible tree decompositions. Therefore the tree-width of three corresponding to the earlier tree decomposition was the correct value of tree-width for the graph.

Learning Impact: The user is expected to learn the concept of tree decomposition of a graph. By the help of animations, they will be inspired to learn abstract thinking: how a *form* of a tree can be derived from an unassuming, typical graph.

5 Implementation

This chapter discusses the platform and the programming techniques employed to implement the application. This is a web application meant to run on desktop and mobile browsers. Traditionally, Javascript or programing languages which transpile to Javascript are employed to execute such web apps.

In the implementation of this application a functional programming language called Elm which transpiles to Javascript is used. Therefore, the chapter starts with explaining the benefits of using a functional programming in general and introducing how Elm uses Model, View, Update architecture to implement a dynamic front end.

I will discuss how a Graph as a data-structure is defined and how it is drawn on screen as SVG (Scalable Vector Graphics), how a function generates a colour palette for the colouring of the vertices, how vertices are laid out in various geometrical patterns, and finally how animations are implemented and graphs are made to change shape and translate in 2D space among other things.

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. (Hudak et al. 2007).

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

Functional Programming, however, has recently found its way into the mainstream, for example through things like lambdas in C++ and streams in Java 8.

In this section I will discuss, why functional programming was chosen as the programming paradigm. How the functional programming language called Elm is used to write a well structured, 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 the may more popular imperative programming. In the functional paradigm, functions are first class citizens, which can be mashed up together in myriad different ways such as:

1. A function given as input to a another function,
2. A function producing another function as an output,
3. Composition of two functions dove-tailed to each other to produce another function,
4. Programming patterns being abstracted out as functions,

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 its scope for side-effects. With such confidence in the functions, they can be fitted with each other to make them do complex computation (Hughes 1989).

Since the functional paradigm is more reasonable and logical than imperative programming the runtime errors are substantially less frequent than imperative programming and is easier to maintain.

Separation of Concerns If functions do not have side effects, how do they print output on the terminal or read file from the hard disk or accept inputs from a user? Purely functional programming languages such as Haskell and Elm 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 write a large part of the program where he deals with just pure functions. This allows them 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 subsection 5.1.3 *the Elm Architecture*.

5.1.2 Why Elm Programming Language for This Project?

For the reasons in the previous sections, a functional programming language was chosen keeping in mind that the size of this project would be quite substantial. Unlike JavaScript, Elm does not require any external framework such as Angular or React. This makes the program easier to reason with and maintainable.

Friendly Compiler Errors It has a compiler which gives developer friendly error messages almost guiding the programmer for correct usage of the language. Refactoring code in Elm is easy as the friendly compiler errors guide the programmer to each line of the code which need modification while refactoring.

Elm-Ui The layout of the application on the screen can be done without writing HTML and CSS by using an Elm library called Elm-Ui. This package frees the programmer from using CSS styling and gives intuitive control of the page layout with the help of rows and columns (Pure functions). With Elm-Ui, multiple rows can be situated in a column and multiple columns can be situated in a row. When elements are put in a row they are stacked side by side horizontally. When the elements are put in a column they are stacked one below each other. The elements in such a row or a column can be put at a definite alignment and spacing from each other.

Libraries and Community Elm also has rich libraries for linear algebra, graphics and Scalar Vector Graphics which can aid in creating a *2D* graphics web app like this one. Elm also has a wonderful community help, advice and discussion on Slack and Discord for example.

5.1.3 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:

1. Model,
2. View,
3. and Update.

The Model is a data structure which holds the state of a program. Fairbank (2019) This state is used by the view function as an input to render the webpage. The webpage, when rendered has elements, which may trigger events, such as user inputs by the way of clicking an HTML element.

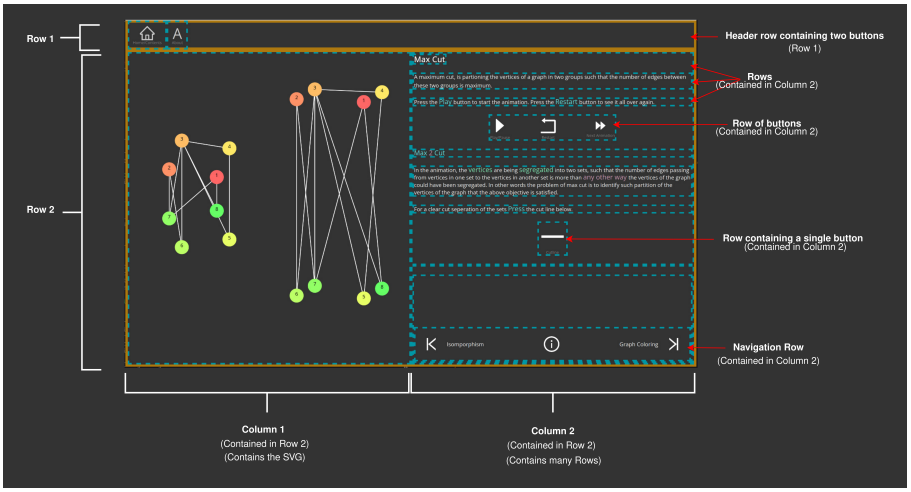

Figure 5.1: *Implementation of the layout using the Elm-Ui library. The library helps implement a layout without having to write HTML and CSS. The page contains of two parent rows. The second row contains two columns, the left column contains the graphical display (SVG) part of the app, the second column is made up of rows containing explanation and buttons. The last row in the second column acts as the navigation bar to hop between the topics.*

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, then it is used by the view function to render a webpage according to a formula set by the programmer. Hence the content of the webpage reflects the state of the program.

5.2 Events as Messages

The events described in section subsection 5.1.3 generated by animation clocks and clicks of the user on graph elements and buttons, are called messages in Elm. 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 Vertex
    | NextTopic                      -- Next Topic
    | PreviousTopic                  -- Previous Topic
```

Listing 5.1: *Abstract Data Type Msg with its Data Constructors. These messages are dispached by buttons, graph vertices system clocks and are received by the update function to change the* `Model`*, which carries the state of the program.*

The messages are not just generated by user interaction with this application, they are are also generated by the animation clock 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
        ]
```

Listing 5.2: *Subscription of Animation clock and Key presses services. Subscriptions are used to catch the events which are emanated outside the DOM. In this code, a system clock and key presses are subscribed to by the program.*

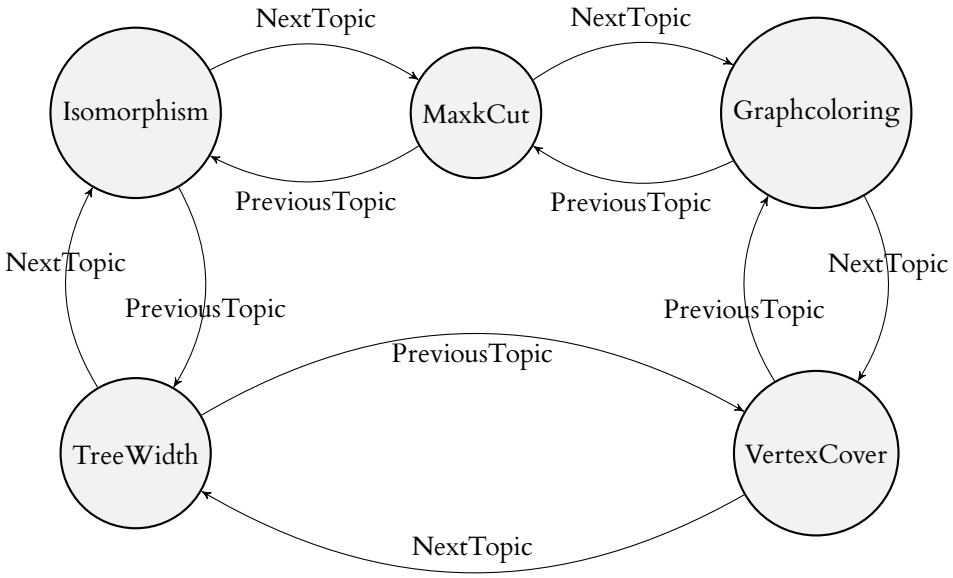

Figure 5.2: *This Finite State Machine (FSM) shows how the messages, NextTopic and PreviousTopic changes the state of the program (the Model) from one topic to another. This is a subset of a much larger FSM in which the number of states and inputs are greater in number.*

5.3 Implementation of Graphs

As part of the application state, a graph exists as data structure which contains a list of vertices and a list of edges. The vertex, as can be seen in Listing 5.3 which is a data type defined separately consists of a name (which is an integer), a colour, a 2D position (which had to be 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.

```
-- Definition Vertex
type alias Vertex =
```

```
3       { name : Int, pos : Vec3, color : color, glow : Bool }
4
5   -- Definition Edge
6   type alias Edge =
7       { vertexOne : Vertex, vertexTwo : Vertex }
8
9   -- Definition Graph
10  type alias Graph =
11      { vertices : List Vertex, edges : List Edge }
```

Listing 5.3: *Definition of Vertex, Edge and Graph. Vertex is defined as a data structure which has a name as an integer, a position as a 3 dimensional vector, colour, and a flag which depicts if it has been selected by the user. An Edge has two Vertices and a Graph contains a list of Vertices and a list of Edges.*

5.3.1 Grid

In the program a Grid is a list of 3D vectors or in other words a list of position for vertices, which can be taken as an input by certain functions to construct graphs or change shapes of graphs.

A list of Vertices, for example can be formed by combining together lists of names, colours and a grid.

Grids are important in implementing animations. In the animations a graph moves towards a grid of positions. To aid this, there is a function, which which take two grids and output a grid which is geometrically in between the two grids. The graph changes from one intermediate grid to the next until it reaches the final grid. This will be discussed in more depth in subsection 5.4.1.

5.3.2 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 and 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 Isomorphism and Max k Cut examples.

As a small example, here is a functional programming code in Listing 5.4 to find the centroid of three position vectors. You can observe how first two vectors are added on line 3, 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 cannot be overloaded to work for vectors.

```
1   findCentroidOfVertices : Vertex -> Vertex -> Vertex -> Vec3
2   findCentroidOfVertices v1 v2 v3 =
3      Math.Vector3.add v1.pos v2.pos
4      |> Math.Vector3.add v3.pos
5      |> Math.Vector3.scale 0.333
```

Listing 5.4: *Finding Centroid of Three Vertices. It logically starts with vertex the addition of positions of vertices v_1 and v_2. The output of which goes to the same function at line 4 albeit partially applied to position of v_3. The output, which is the summation of the position of the three vertices in turn is passed to a scale function which effectively divides its input vector by three. Hence the centroid emerges out of the other end.*

5.3.3 Creating Polygons

Two dimensional graphics were used to create polygons to form the Grids or subset of Grids for Graphs. Functional programming techniques such as mapping over a list was employed to create and manipulate such grids. The construction of a polygon proceeds like this:

1. A list of floats containing ones is created.
2. Each is converted to an angle in radians corresponding to the polar position of the vertex.
3. This list of angles is then presented to a function which maps a horizontal vector with unit length over this list such that the unit vector is rotated with an angle equal to the angle in the list giving rise to a list of rotated vectors.
4. This regular polygon with vertices at unit distance from its center is then scaled. The scaling factor for x is sometimes different from that of y. Which enables squeezing of the polygon along any 2 dimensional direction.
5. Mixing such polygons come handy to create more complicated shapes.
6. The polygon is then translated to an appropriate position on the SVG screen.

5.3.4 Implementing colours

To have a list of neighboring colours acting as a colour palette we work on the Hue Saturation Lightness colour space (HSL), mostly varying the hue just pass a region in the spectrum of hues (First, Second or Third) and the number of colours 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.

5.3.5 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 but are rather drawn out from the vertices they contain.

5.4 Implementation of Animations

In this section I will cover 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.

5.4.1 Morphing Geometry of a Graph

In some of the animations in the application, the graph changes its 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. When the animation is started, with each tick of the animation clock, the vertices of the second graph move towards the target grid points with a constant velocity. The velocity of a vertex is calculated by obtaining the displacement vector between the target position and the current position of the vertex and also the time available before the animation ends. The displacement is calculated as $\vec{d} = \vec{p_t} - \vec{p_v}$, where $\vec{p_t}$ is the target postion and $\vec{p_v}$ is the current position of the vertex. The time available is calculated as $t = t_{available} - t_{elapsed}$. The velocity is calculated as $\vec{v} = \vec{d}/t$. The distance the vertex must travel is hence calculated as $\Delta d = \vec{v} \times \Delta t$. Where Δt is the time elapsed since the last tick of the animation clock.

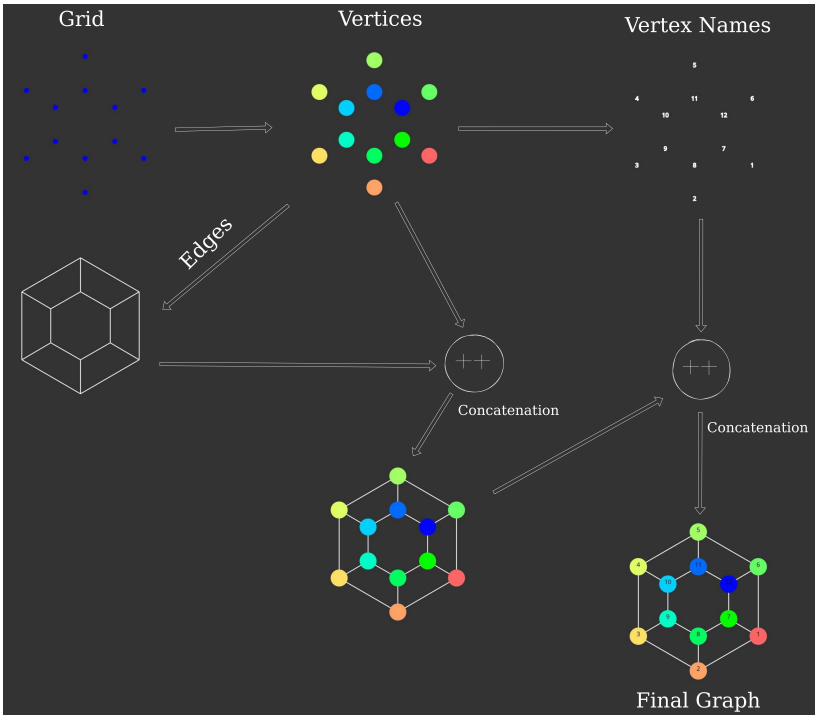

Figure 5.3: *Drawing a Graph. Starting with a grid of positions a set of vertices are created. Connecting the vertices, the edges are formed according to a list of tuples. The vertices is then laid on top of the edges by concatenating list of SVG elements representing edges with list of SVG elements of Vertices. This is followed by this new list being concatenated with a list of SVG text representing the vertex names.*

5.4.2 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 colour 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 colour) modification of the graph.

The re-formation of the vertices and the edges are shown in the Elm function in Listing 5.5. 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 (vertices with the same names) as the original ones.

```
1  morphGraph : Graph -> Grid -> Graph
2  morphGraph graph grid =
3      let
4          updatedVertices =
5              List.map2 updatePositionVertex graph.vertices grid
6
7          createEdge =
8              updateEdge updatedVertices
9
10         updatedEdges =
```

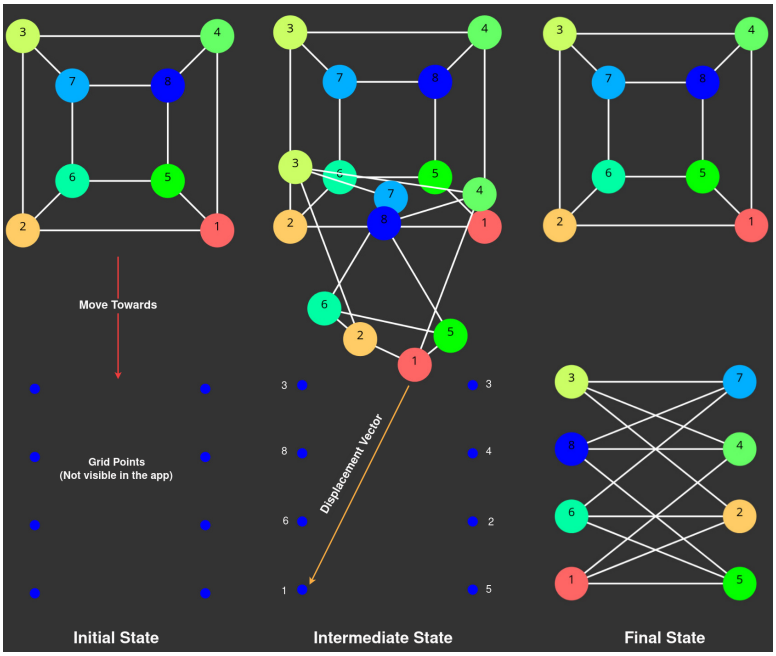

Figure 5.4: *Morphing a graph. This example from the Graph Isomorphism topic shows how shape transformation is done with the help of a targed grid. The target grid is shown as blue points. In the picture in the middle, a diplacement vector is shown between the current position of vertex 1 and its final position. Veloicty of this vertex is calculated by deviding the displacement vector with the available time left in the animation.*

```
11              List.map createEdge graph.edges
12      in
13      Graph updatedVertices updatedEdges
```

Listing 5.5: *Changing the shape of a graph for animation. The first line in the code excerpt describes the Type Signature of the function morphGraph. It says that the function takes a Graph and a Grid as input and produces a Graph as an output. The input Graph's vertices adopt the positions listed in the Grid.*

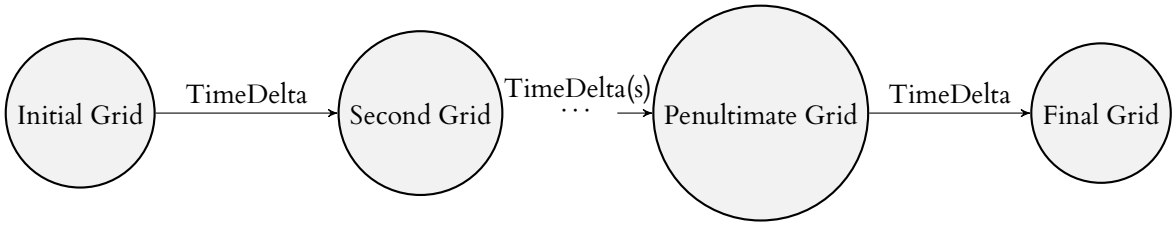

Figure 5.5: *A Finite State Machine shows how the animation progresses with each tick of the elm run-time clock, depicted as the message TimeDelta. With each such message each vertex takes their respective position in the following grid.*

5.4.3 Drawing of Graphs

Drawing graphs is done using SVG elements. The vertices are drawn as colour 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.

5.5 Explanation Panel

The Explanation Panel is always shown on the right and it consists of the title of the topic and suggestions on how to interpret and interact with the animations.

The content of this panel depends on the state of the animation or the user interaction associated with the topic.

The functions which populate the explanation panel with different elements have therefore a subset of the *Model* data structure as its input. According to the state of the program relevant advice, instructions and buttons are shown on the panel.

The functions responsible for Explanation Panel in the case of user interaction such as the ones in Graph colouring and minimum vertex cover run a check on the state of the program to know if the user is doing their task correctly. For example in graph colouring, if two adjacent vertices are coloured the same colour, a warning message appears in the explanation panel.

5.6 Navigation and Control

There are several buttons and keyboard shortcuts to go from one topic to the next and to play/pause and restart animations. These have been implemented by generating appropriate messages which are caught by the update function which in turn updates the model (state of the program). The updated model of the program is reflected on the screen by the view function.

5.6.1 Home Page

See the design of home page in subsection 4.2.1. The topic icons contain miniaturized version of the graphs for the corresponding topics. The miniaturized versions of the graph are borrowed from the implementation of the various animations and user interactive tasks in the app. The miniaturization is implemented by spawning the SVGs in small spaces . As the graphics is scalable, it adopts the size of its parent element. Clicking on the Graph Isomorphism icon, for example, triggers the *GoToIsomorphism* message which is used by the *update* function to change the state of the model to contain the *Isomorphism* topic.

5.6.2 URL Management

Although there is only one HTML page rendered (this application being a Single Page Application), when the user navigates from one topic to another they can see the URL path after the Website's name change to reflect the pseudo-page he is on. This change in the URL is brought by the function *pushUrl* at appropriate situations. The change in the URL is detected by the run-time to produce the message *UrlChanged* which in turn is caught by the Update function to re-populate the page with new topic.

5.6.3 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 respectively, the update function catches it to change the state of the program to load it with the details of the previous/next topic.

5.6.4 Keyboard Shortcuts

Keyboard shortcuts provided 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 without the use of a pointer device. The key presses messages are registered by the elm run-time which in turn triggers the update function to update the model. Information about the keyboard shortcuts can be availed on the screen by pressing the information icon present in the navigation bar. 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; Generates the *AnimationToggle* message).
- **r**: Restart animation. (Can be used instead of Restart Button; Generates the *AnimationStartOver* message).
- **n**: Go to the next topic. (Can be used instead of the navigation button; Generates the *NextTopic* message).
- **N**: Go to the previous topic. (Can be used instead of the navigation button; Generates the *PreviousTopic* message).
- **t**: Next animation (Can be used in case of Max k Cut and Tree width to go to the next animation; Generates the *NextAnimation* message)

5.6.5 Screen Compatibility

So that the various visual sections of the application occupy their appropriate place on screens of different sizes, such as screens of of various laptops and smart phones, the width and height of the screen is used to give the height and width to each section. The state of the program stores data of the type *DeviceType*, which makes the program aware of the screen size of the device being used. This data is used to give proper font size to various text elements used in the app. Such measures, have made the application friendly to mobile devices and changes in the browser window size caused by the user on a personal computer.

5.7 Implementation of Topics

The above sections have discussed the basic building blocks which can be used to implement the individual topics. The building blocks include graphs, animations page-layout, navigation and control.

5.7.1 Graph Isomorphism

This topic is elucidated by a user interactive animation and an animated quiz, discussed in detail in subsection 4.5.1.

Isomorphic Definition Animation The first animation consists of two Graphs which are the same. In the beginning the two graphs are superimposed with each other. Each graph is constructed by merging two polygons and connecting their vertices in a wheel like structure. As the animation begins one graph leaves its position and transforms to another shape. This is achieved by instantiating the two graphs with same position and form. An empty grid is also provided as the target set of positions for the vertices of the second graph.

When the user presses the play button the message *AnimationToggle* is generated. When the update function receives this message, it checks for the state of the program which is set at *IsomorphicTransition*. It starts the animation if it is not already in progress.

When the animation is started, with each tick of the animation clock, the vertices of the second graph move towards the target grid points with a constant velocity. For the kinematics involved in this transition see *Morphing Geometry of a Graph* in subsection 5.4.1.

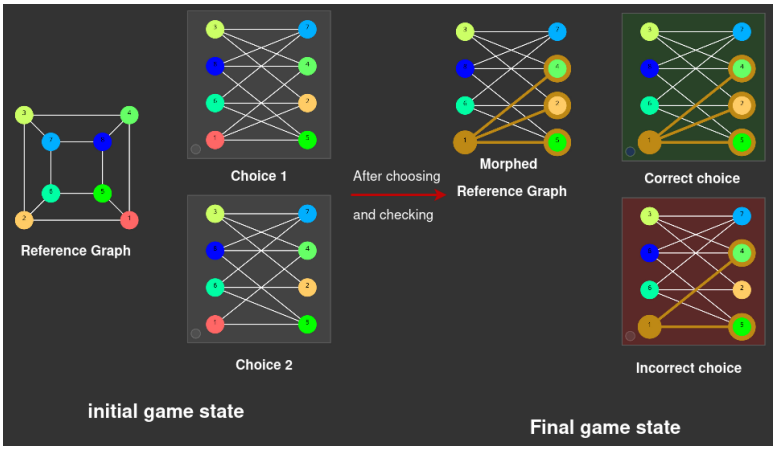

Figure 5.6: *Graph Isomorphism Quiz.* ***Left:*** *The user is presented with a reference graph on the left and two options on the right and asked which one of the two options are isomorphic to the reference one.* ***Right:*** *When the user makes his choice and checks the answer, then the right answer is shown in green background and the wrong in a red background. The edges which are essential in figuring which graphs are isomorphic to one-another are displayed differently in golden colour.*

The animation is also user interactive: when the user hovers over a vertex, or selects it by pressing the corresponding numerical key on the keyboard, a Boolean variable associated with the vertex is turned *True*. A function filters the list of vertices to find the selected vertex. Another function finds the edges incident on the vertex and also the vertices adjacent to the selected vertex by filtering over the list of edges in the graph. These are shown differently in a highlighted colour in both the graphs in the display section. The explanation section also uses this data to explain how the selected vertex is connected to the same adjacent vertices in both the graphs.

Animated Quiz for Graph Isomorphism The second activity gives the user three graphs. The first graph is a reference graph. The user is asked to choose which one out of the two remaining graphs is isomorphic to the reference graph. The right answer is hard coded, and when the user selects his answer, the program checks if the answer chosen is right one and changes the appearance of the display and the explanation panel. The state of the game consists of two variables: *Choice* and *CheckState*. When the user has not made any choice yet *Choice* is at *NoChoice* state. When the choice is made, *Choice* is set to *FirstGraph* or the *SecondGraph*. When the user presses the button to check the answer *CheckState* transforms from *NoCheck* to the *Check* state. There are case statements in the code which take the correct measure depending on the state of the game.

5.7.2 Max k Cut

The Max k Cut topic has two animations related to Max 2 Cut and then Max 3 Cut . See subsection 4.5.2 for explanation of the animations. The first animation has a graph constructed by merging two polygons. The edges are defined so that the graph is nearly bipartite (see subsection 2.2.1) save for one edge (see subsubsection 4.5.2). The target grid for the animation is

made of the combination of the same polygons but set apart vertically. The kinematics of the animation is executed the same way as is done for Graph Isomorphism.

The user can draw a predetermined line segment which separates the two sets. The line segment is superimposed by intersection points of the line segments and the edges between the two sets. An intersection point is calculated by finding intersection of two line segments. A linear algebra library function was used for this task.

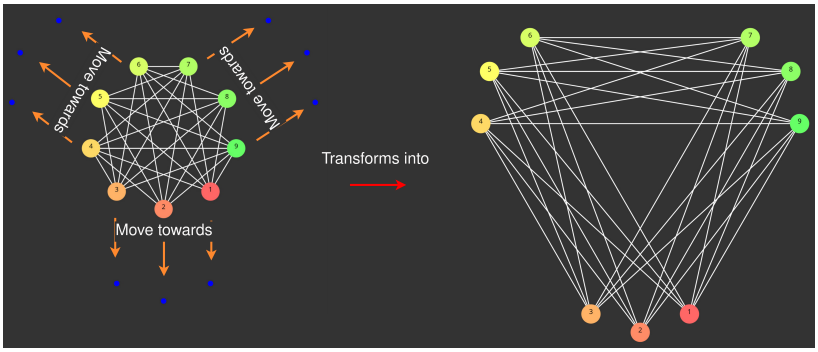

Figure 5.7: *Arrangement of animation of Max 3 Cut. This animation transforms a graph into a shape such that the three sets of vertices of the graph making a Max 3 cut are separated from each other distinctively. The three sets move away from each other in directions which are* 120° *apart.*

In the case of Max 3 Cut (see subsubsection 4.5.2), the animation begins with a tripartite graph. The graph is based upon a regular *nonagon* shaped grid. The destination grid, in this animation is shape of an equilateral triangle, with the grid points near the vertices of the said triangle. The graph consists of three sets of vertices which settle down at the vicinity of the three vertices of the equilateral triangle at the end of the animation. The kinematics of the motion of the vertices is explained in subsection 5.4.1.

5.7.3 Graph colouring

This section has two user interactive tasks to colour graphs in the way they should be coloured for a graph colouring problem. The graph in the first task is composed of two concentric square graphs such that each vertex has three adjacent vertices. The user task is to colour the graph using two different colours. See subsection 4.5.3 for further explanation of the task.

The graph in the second task is composed of two concentric pentagonal graphs such that each vertex has three adjacent vertices. The user task is to colour the graph using three different colours. See subsection 4.5.3 for further explanation of the task.

When the user chooses a colour from the colour palette, the chosen colour is updated in the state of the program. When the user clicks a vertex, the colour of the vertex is changed with the colour stored in the state of the program.

A function performs checks while the user is working on the task, whether two adjacent vertices have similar colour by doing a linear search on the edges. If they are, the information is used to display a warning to mark that edge differently and a warning appears on the explanation panel based on this search.

If in a linear search for finding adjacent vertices of the same colours does not have an output and if all the vertices have been coloured, then the function which populates text on the explanation panel shows a message that the task is complete.

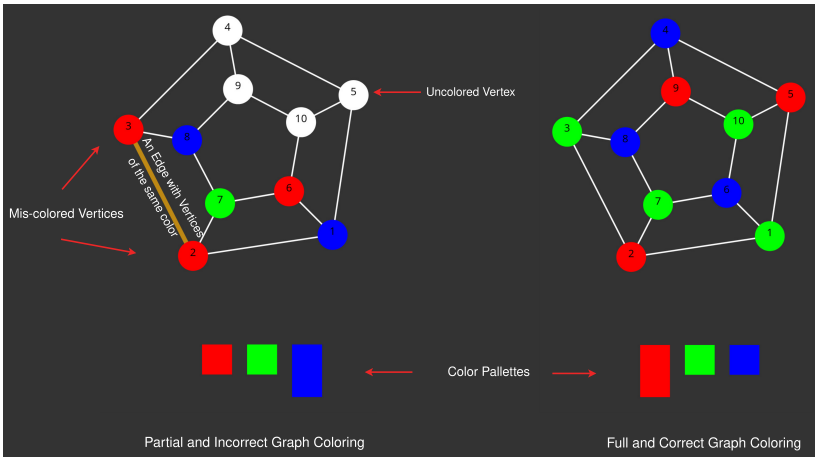

Figure 5.8: *Graph colouring Task. The graph on the left has a pair of incorrectly coloured vertices. On the right there is a fully and correctly coloured graph, where all the adjacent vertices are coloured differently from each other.*

5.7.4 Vertex Cover

The user interaction for this topic asks the user to choose the minimum number of vertices such that all the edges in the Graph belong to the set of vertices which are incident on the chosen vertices, as detailed in subsection 4.5.4.

The user is given two tasks similar in nature. In the first task the graph in the Vertex Cover is based on composition of two square grids. In the second task the graph is based on composition of two hexagonal grids. When the user clicks on a Vertex, there is a Boolean value associated with the Vertex which becomes *True*. There is a function which filters the list of edges to find which of them have at least one of their vertices *selected*. Such edges are illuminated, for the user to note that the edge has been covered.

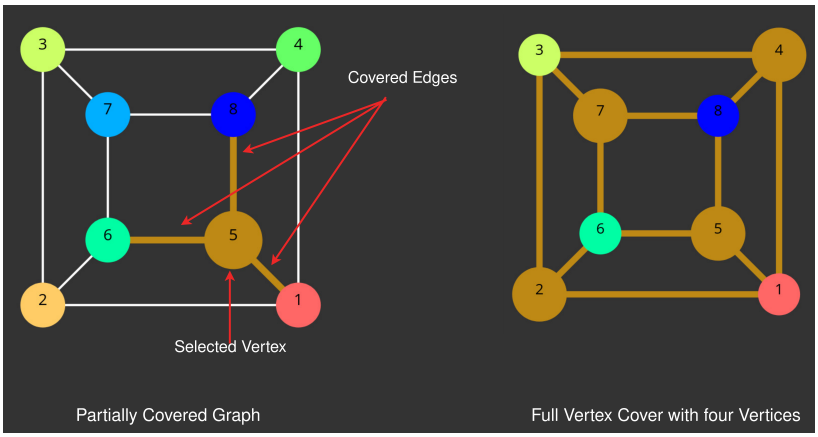

Figure 5.9: *Vertex Cover task. The graph on the left shows the selected vertex 5 and the edges incident on it in golden colour. A vertex can be selected by either clicking the vertex or pressing the corresponding number key on the keyboard. The edges shown in golden colour are the ones which are covered, they are coloured so automatically when a vertex related to them is selected. The graph on the right has all its edges covered.*

There is a function which filters out the edges whose one of the vertices have been selected, and

another which counts the number of vertices which have been selected. When the all the edges have been covered, the program checks for the number of vertices selected to do this by filtering the list of vertices. In the case of the particular example in this project, the graph can be covered using four vertices. Therefore if the number of vertices selected at the end of the session are greater than four, then the explanation panel is populated with a message that the task needs to be attempted again. Finally, next task button loads the program with the new task.

5.7.5 Tree Width

Tree width topic is implemented in a series of animations. In the first animation the vertices of the graph of this topic are arranged circularly albeit with edges such that there is a cellular structure in present inherently in the graph. This edges were set up in this graph by coding their connectivity in a list of tuples. This is also a graph-transition just like graph isomorphism. The target grid for this transition is a regular lattice which reveals the cellular nature of the graph. The kinematics of the transition is the same as that of graph isomorphism (see subsection 5.7.1) and Max k Cut (see subsection 5.7.2). For more information on the kinematics of such shape transitions see subsection 5.4.1.

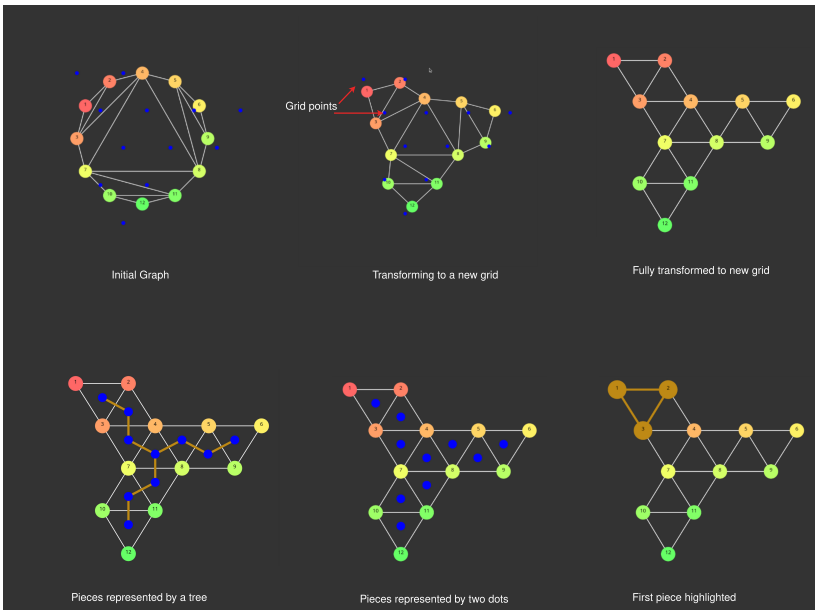

Figure 5.10: *Tree width Animations. From the top left to bottom left. Tree width is explained in a series of animations. The first animation turns the original graph to a cellular like structure by moving towards a pre-determined grid. After the transformation a piece is defined as a subset of a graph. The next graph shows how graph can be decomposed into several such pieces. The last picture shows how the pieces can be connected together by a tree.*

The second part of the animation highlights one piece of the cellular structure . The piece is also marked by drawing a blue dot at the center of it. The position of the blue dot is found by taking the centroid of the constituent vertices of the piece.

In the third part the centroid of all the pieces are found and marked as blue. These blue dots are connected by special lines in the last part of the animation to form the tree structure inherent in the original graph. The animation series is followed step by step by explanation panel which responds to the various stages of the user progressing through the task.

5.8 Modules

The program is divided into ten modules. *Main* holds a key role and acts as the launching pad for the program. It also serves as a connector, loading the features of specific graph theory topics as needed when navigation commands are issued.

The five topics included in the project own a module of their own. They are *Isomorphism*, *Maxkcut*, *Graphcoloring*, *VertexCover* and *TreeWidth* modules. These modules are imported by the main function to be used on the screen. Henceforth these will be collectively known as the *Topic Modules*.

Within the *Graph* module there are data types and functions for representing, constructing, drawing and animating the graphs. This module contains most of the code related to geometry. This module is depended on by all the *topic modules*.

The *Explanation* module contains the text data as a $0-$*nary* functions (functions which do not take input) used by topic modules for the explanation of the respective topics. These only contain the definitions of the concerned graph theory problems.

Finally, the *Messages* module contains the messages which are generated with system clock ticks or user interaction with the DOM elements. Such messages are defined as *Algebraical Data Types* (see section 5.2) and are imported by almost all other modules. This along with the *Main* and *Graph* modules form the central pillar of the program and are the minimum requirement of the program to function.

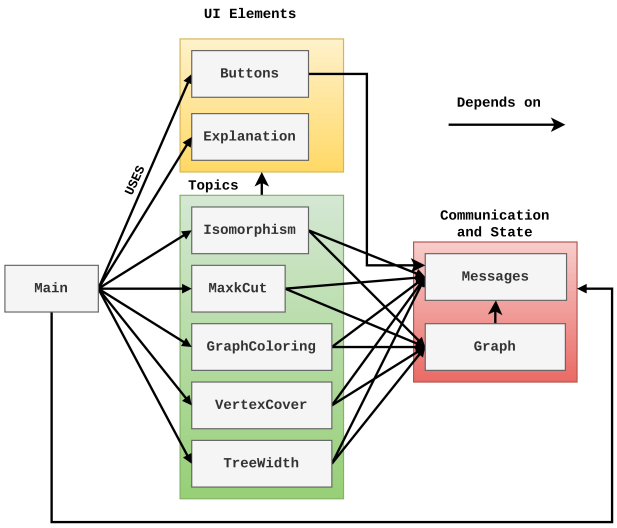

Figure 5.11: *Modules in the Program. Arrows from the blunt end to the pointy end depict dependency. The main module contains the main logic of the application which uses different modules. The most noteworthy of them are the modules related to the different graph theory topics.*

5.9 Software Engineering Practices

In this section I will point out some of the software engineering practices that I used throughout the design and development of this application.

5.9.1 Version Control

The program along with the documentation and the dissertation report were version controlled in a single GitHub repository. See section A.2. A git repository gave me more confidence for undertaking risky and complex refactoring and feature enhancement tasks by the way of creating separate branches for all issues. A different branch was created for editing documentation, and I made sure that modification in code files was not done on this branch. Similarly, a branch for code refactoring did not modify the documentation part hence separating concerns. Conveniently a branch could be simply discarded if an adventure in a radical change in code went wrong.

5.9.2 Continuous Deployment

The web application concerning the project is deployed on the public Internet using Netlify. This service lets one host a front end website and is connected to the master branch of the GitHub repository of the project. Every time a new version is pushed to the master branch, the Netlify service fetches the new version of the app from the repository, builds it according to the build script present in the GitHub repository and deploys the web application on a specified URL. The build and deployment can also be done manually by going to the dashboard of the Netlify website. The build and deployment process primarily consists of running a script to transpile Elm code to a JavaScript file and copy the output along with the boiler plate HTML to the published directory.

If it is needed to deploy the application anew instantaneously (something which is required to test the website for different screen sizes) a *curl* command with a specific *build hook* URL is executed on the local machine. It triggers the deployment of the website from an up-to date copy of code in the master branch in the repository.

5.9.3 Documentation

Documentation for the project was done in a continuous fashion in a variety of ways, such as a wiki for code implementation and a guide for future work, time logging for project management and report writing for the submission.

The wiki was supported heavily from the code comments made to explain the functions. Time logging brought a sense of discipline in the development efforts and it also helped in estimating time required to complete the tasks which lay in future and assess the learning curve.

6 Evaluation and Testing

6.1 Evaluation

For an application such as this one, which aims to explain mathematical or scientific concepts based on primarily animations, I believe the two most important apects to be evaluated are learning effectiveness and user interaction and user experience (UI/UX).

The evaluation of the application was done on the basis of feedback from peers mainly in the field of computer science and mathematics. There were two surveys conducted, the first was unsupervised and anonymous. The second survey was supervised in which the participants were interviewed while they were experiencing the application.

6.1.1 Unsupervised Survey

The unsupervised survey was conducted among peers mainly in the field of computer science, mathematics and education. The participants were given the URL of the application and an online feedback form. See section A.4 for the summary of the results of this survey. The users were asked to visit the application and answer a questionnaire.

Although the survey was anonymous, but details such as their academic qualification and familiarity with the subject of graph theory were obtained to put the responses in context.

The participants consisted of eleven PhDs, eight of which belonged to the field of computer science. There were four participants with master's degree all of whom belonged to computer science field. There were nine undergraduates mainly from the field of computer science and engineering.

Among the eight PhDs in computer science, six mentioned that they have done research in the field of graph theory, while the remaining two mentioned that they had a course either exclusively on the subject or a wider field. Therefore the feedback on the application brought a substantive quantity and quality of opinions, criticism and even some praise.

The feedback ranged from opinions on UI/UX to opinions on effectiveness of a particular topic.

6.1.2 Supervised Survey

The supervised survey consisted of the same questions the as unsupervised survey. See section A.5 for the summary of results. Unlike the former survey the web application was visited by the participants by sharing their computer screen. The participants this time belonged to a variety of education backgrounds like humanities, sciences and other fields. Therefore the participants were frequently assisted while understanding the animations and attempting the quizzes and tasks.

This gave the opportunity to see closely how the users navigate the app and which aspects of the UI and the learning material they find confusing.

6.1.3 Evaluation of UI/UX

The survey was divided into two sections. The first part dealt with the user interface and experience aspect of the application. There was both praise and criticism from the participants. Both were taken into consideration and analyzed in this section.

Praise for UI/UX: Positive feedback for the UI/UX include the following comments:

> *"Very fluid. I like the single-page-application feel, rather than needing to reload different pages."*
>
> *"The visuals and animation were pleasant. Quite easy to navigate."*
>
> *"[...] it was intuitive and clear how to interact with the application."*

The first *quote* about the application being a single page application (SPA) underlines an important aspect of the UX of the application. It removes distractions in the users learning experience. This is unlike ordinary applications which while loading a new page briefly shows a white screen even on fast connections.

Criticism of UI/UX: There was constructive criticism of the UI/UX of the application as well. Most of these criticisms were actionable while one criticism, given the constraints put by the requirement of the topics was not.

> *"One thing I found a bit frustrating was the varying levels and styles of interactivity between topics, such as clicking the cutline for max cut .. "*
>
> *"Maybe the topic words could be clickable to make navigation easier."*
>
> *"There are occasional spelling errors that could be corrected [...]"*

The first criticism in the list above points out that the different topics had different styles for interactivity. Although, it is agreed that all the pages should have consistency in the number and nature of interactive buttons but the different demands of each topic made it difficult. For example the max cut page has a cut-line button and others do not. This was unavoidable. However, there is no doubt that a variation in button layout and functionality is potentially quite distracting and in a future iteration of the application a drawer of tools which slides in and out of view can be made for each page, though it may have its own drawbacks in terms of visibility and accessibility.

However most criticism of the UI/UX like clickable names on the navigation bar were actionable. For example, the users tend to click the words on the navigation bar. Change in the implementation was made for the same. For spelling errors in the explanation panel, they were combed out and corrected.

6.1.4 Evaluation of Learning Effectiveness

The respondent feedback was also used to gain a better understanding of how effective my implementations were in teaching the users.

Most Effective Topic: To obtain what the respondents think about the learning effectiveness of the application they were asked which topic in the app they found the most effective for learning. The answers ranged over all the topics. This indicates that every topic had something to impress the respondents in terms of learning efficacy.

The respondents who are familiar with the field of graph theory mentioned *Tree Width*, and *Max k Cut* the most. Although they mentioned rest of the topics at least once. The two topics made to the top spot as they are two most difficult topics to understand.

They were also asked about the least effective topic and the reason of their choice. This revealed some of the weaknesses in topic explanation. Again Tree width faced the most criticism for lack of clarity in the explanation panel.

Graph Isomorphism: Graph Isomorphism was mentioned in the feedback positively for having both an animation and a quiz. The animation was appreciated for having the functionality where a user hovering over vertices causes the vertices and the edges light up.

There was a suggestion for the quiz that it would be nice if do a similar "hovering functionality" over the vertices because the quiz was more difficult without it.

Max k Cut: This was one of the most mentioned topic in the survey in a positive way, perhaps because the difficulty in understanding its definition. The cut-line functionality has been praised by multiple respondents. Here are a few comments on that from the respondents.

> *"Thought that max k cut was very well explained - the use of cutlines is rather nice."*
> *"I love the way the graph transforms then you add the cutlines it was creative."*

Graph Colouring: The topic received mentions for the user interactive task. Perhaps, user interaction is much favored way of learning than a passive animation. It was most praised by the respondents who were not much familiar with the field of Graph Theory. It should be so because it is a relatively easy topic to understand compared to the other topics.

Vertex Cover: Just like graph colouring, vertex cover found mention in the most effective topic for containing a user interactive task.

Tree Width: Tree width was in the special focus of the respondents who are researchers in the field. Tree width received both praise and criticism. It received praise for being at a different level of difficulty from the rest of the topics.

> *" Tree Width is a hard problem to understand , you have chosen an extraordinary way in visualising and explaining this problem, made it seem simple and easy to learn."*

Identification of the pieces and decomposition was pointed out by various respondents who are specialist in the field as significant in understanding the topic. For example the following comments point the same out.

> *" Tree width due to identification of blocks "*
> *" I liked the pieces and decomposition . I finally managed to start understanding Tree Width!"*

Credit should be given to the book Kleinberg and Tardos (2006) which lent a very clear example of tree width which was adopted for the animation for this topic.

Tree width also received some constructive **criticism**. There was ambiguity in the explanation of the possible tree decompositions of a graph and possible tree widths corresponding to such decompositions. Here the emphasis should be given to the word *possible* which was not put at the appropriate place in the explanation of the topic in the app as pointed out by two respondents.

> *" Treewidth wasn't very clear, unfortunately - I think it wasn't clear that the treewidth of a graph is the smallest maximum bag size - 1 across all tree decompositions. So it seems like a graph can have multiple different treewidths, when it actually has multiple possible widths based on tree decompositions, then a treewidth that doesn't depend on which tree decomposition you use [...]"*
> *" Tree Width was my favourite maybe you can try choosing better choice of words in the 'explanation panel' such as 'possible width decomposition'."*

Following their advice, a few more slides and the text on the animation panel was changed so that the concept of tree width is clearly and correctly stated removing ambiguities in the definition.

6.1.5 Extra Feedback from the Respondents

The application was commended for being clear and colourful and being potentially extensible to other concepts of graph theory by most participants. There were a few suggestions for extending the functionality such as:

1. Custom user graphs,
2. Detail on algorithms to compute these problems like vertex covers,
3. Animations showing non-optimal solutions.

6.1.6 Limitations

The application has several fundamental limitations some of which are specifically pointed out in the survey by the participants and others have been deduced independently.

Absence of Algorithmic Solutions: Since the objective of the application was to define and state the problems clearly and not to solve them algorithmically, the explanation of algorithms to solve the problems have been steered clear of. This limitation in the scope of the project remains a limitation of the application.

Easy Examples: The examples used to explain the topics are simple. For instance, the example chosen to explain the Max 3 Cut problem was a tripartite graph. Its solution is trivial. These examples were chosen to clarify the definitions of the problems, even though complicated examples could have given an uninitiated user a flavor of complexity and true difficulty of the problems.

Absence of Custom User Graphs Functionality: There are web applications on the internet, which allow the user to instantiate their own graphs, with user instantiated vertices and edges. Although such functionality has been avoided as the application's objective is to clarify the definition of the problems. For this purpose in the application there are only concrete examples which lend themselves to easy explanation of the problem at the hand. However if in the future, algorithmic solutions to such problems, is also included, then functionality of defining custom graphs can be added.

6.2 Testing

Automation of testing for this project was done by using a mix of Property based Testing and Unit Testing. Both paradigms of testing can be implemented for Elm projects using the library called *elm-test* and a CLI-tool which goes by the same name. Writing test files in Elm is fairly similar to other languages. The test functions can generate input test cases (specially in the case of property based testing) and use boolean predicates to generate the output in terms of *Pass* or *Fail*.

6.2.1 Property Based Testing

Testing of components of Graph module was done using Property Based Testing (PBT). This paradigm of testing was introduced for the first time for Haskell (a functional programming language). It checks if the output of a function satisfies certain prescribed properties. These properties are specified as boolean predicates.

For example, a function which constructs a fully connected graph (a graph in which all the vertices are connected to each other by one edge) with n vertices, will form $n(n-1)/2$ edges. The number of edges of the output graph are matched up against this mathematical reality. The function passed the test against a vast number of auto generated inputs.

Similarly, a cyclical graph i.e one in which the vertices are connected in a pattern such as this: A – B – C – D – A, with n vertices will have the number of edges equal to n. A function which constructs such a graph can be property based tested by comparing the number of edges it has with the number of vertices, which in this case should be equal. See section B.2 for examples of testing functions and section B.1 for the results.

6.2.2 Unit Testing

Unit testing was done on functionality of graph animations and functions responsible for navigation in the application against possible inputs and reference outputs. See section B.1 for the

results.

6.2.3 Manual System Testing

The system was continuously tested manually during the development of the project. Towards the end of the development a planned scheme for interacting with the system for the black box testing of the application was executed to search for unexpected behavior. See Appendix C for the details of the test and results. The application's behavior against all the tests was as expected.

6.3 Verification of Satisfaction of Requirements

Verification of Development of Interactive Animations: The requirements mentioned in section 3.5 have been satisfied by this application. The application has covered all the topics it had set out to achieve. As can be seen in the **application**.

Verification of Development of Data Structures and Functions: The data structures and the function to represent the graphs and its elements in the program; drawing of the graphs on screen; animating translation and shape transformation of graphs; handling of events generated by elements in the graph and explanation panel and synchronization of the explanation panel with the state of the graphs was achieved as dictated by the requirements. The code of the program can be found through section A.2.

Verification of Learning Impact: Although evaluating a learning method for its effectiveness is a subjective issue, the respondents of the evaluation survey have expressed their satisfaction in the quality of the animations and tasks. Several have commented that it could be a learning tool in reality.

Verification of the Quality of User Experience: The respondents and the first users have evaluated UI/UX of the application as simple and intuitive. The Single page application feel has been appreciated for being fluid and unobstructive.

Verification of Should Have Requirements: For satisfaction of *Device Compatibility* requirement, the application has been tested on various screen sizes of desktop and laptop computers and mobile devices and it neatly fits those screen sizes by properly placing the display and explanation divisions. For satisfaction of requirements regarding *Pleasing Aesthetics*, see *Verification of the Quality of User Experience* in this section. For satifaction of the *Contribution Friendliness* requirement, the code is published on GitHub and will invite public contribution after the completion of the dissertation. See section A.2. The program is well modularized and new sections can be added to it by adding another elm file for a new topic and making minor changes in *Main.elm*.

7 Conclusion

7.1 Summary

The application elucidates five graph theory problems by the help of animations and user interactive tasks. This app also has a dynamic explanation area which contains text appropriate to the state of the lesson at a particular time.

The end goal of the project was to have some effectiveness in learning outcomes. Therefore to elucidate the chosen topics, explanation strategies were made carefully and animation friendly examples were chosen (and also designed).

It was decided that the layout of the web pages will be divided equally for display of animations and*state dependent* on the *explanation panel*.

The program was written in a functional programming Elm and a functional type system was used to define graphs, vertices and edges. Vector and linear algebra was used to draw and animate graphs on screen as SVG elements. The program was divided into modules for the main program, graphs and the various topics.

The web application was hosted on the internet and tested to be adaptable to variable screen sizes by adjusting the size of the elements on the screen and font-size accordingly.

The application was evaluated by a survey among peers. This was useful in collecting feedback to improve the application and remove mistakes. It also informed about the limitations of the application specially lack of capability to draw custom graphs.

The application's capability is currently limited to display, animation and making simple queries. It can be extended for more advanced features like adjacency lists and traversals.

7.2 Future Work

The application can be extended according to different themes. It can be extended either by enhancing the graph module to have more algorithmic capabilities or enhancement of visualization methods or simply adding new topics to the present list of topics.

7.2.1 Enhancing Algorithmic Capabilities

Enhancing algorithmic capability of the application can be done in the following directions.

Explanation of Algorithmic Solutions to the Current Topics Implementation of algorithm to find solution for some problems discussed in the application. For example, a function to solve vertex cover using known algorithms such as brute force is achievable with present capabilities of the application.

Traversals: Implement traversal friendly graph type (with adjacency lists) and algorithms to traverse it. This will help explaining path traversal problems also their solutions.

Shortest Path Algorithms: Working of shortest path algorithms working of Dijkstra's and A-star algorithms can be added to the topics. This will require enhancement of algorithms in the graph module to execute the mentioned algorithms.

User Defined Graphs: Implement functionality to allow user instantiated graphs. Implement drawing facilities and a tool box to draw vertices and edges.

Compatibility with Graph Description Files: Graph description files should be able to be uploaded from or downloaded to the local device so that the graphs generated by the application can be exported to other applications as well. Or graph generated by the user somewhere else can be interpreted by the application.

7.2.2 Enhancement of Visualization

Clickable Words: The definitions in the explanation panel should be clickable and lead to a page in the application which has a more elaborated mathematical (formal) definition of the term than is possible to fit on the usual topic page.

3D Animations: Currently, the position and velocities are 2D vectors (3D vectors with the z always kept at zero). The animation can be made more engaging by using 3D vectors and rotating them relative to all the three axes.

Field of View Capability: For navigation inside large graphs, such as biological networks, zooming in and out of graphs may come handy. This can be achieved by viewing a graph using an abstract camera. Moving the camera with keyboard movements will change view of the graph on the display. This capability can also be used to hop from one graph motif to another or other explorations.

Virtual Reality Capability: For a more enhanced 3D experience, libraries in Elm which render stereo images for Virtual Reality (VR) devices can be employed for exploration of larger graphs. VR, if employed for right, may be a game changer in the field of online education.

7.2.3 Addition of topics

To add a topic to the application, a contributor has to define a new page, make use of core modules and integrate the module in the main file. The link to the program can be found in section A.2. Although addition of advanced topics may demand enhancing core functionality as well.

7.3 Reflections

The project was an important learning experience in design, implementation and evaluation of a project of the size. In this section I discuss some insights into this experience.

7.3.1 On Using Functional Programming

To model something as complex as the graph problems in this project, a programming paradigm which can model this complexity efficiently is needed.

An imperative programming language, however *high-level* it might be, is much closer to the Von Nuemann machine in terms of how a computer program is reasoned about by the programmer. There is a tape and it has to be read one step at a time. The programmer implements most of the logic in terms of control mechanisms and less in terms of logic itself. Therefore the program becomes far more complex than the actual complexity it has to represent.

Functional programming on the other hand felt much more efficient in modeling the domain. It implemented logic more in terms of logic and less in terms of control structures. Such that while reading the program one mostly saw logic and not boiler-plate code.

7.3.2 On Project Management

For a project of this size the adage *well begun is half done* is true.

During the development of the application it was observed that after acquiring basic drawing and animation techniques for implementing the first topic, the pace of development accelerated with the help of reusable data structures and animation functions.

7.3.3 On Learning Effectiveness

Graph Theory and mathematics in general are studied using formal language. It is left to the student for visualizing such concepts abstractly most of the times. Therefore computer visualization can be used as an aid in learning especially for the students who are new to the field.

Visualization of graph topics is limited to certain examples which lend themselves well for animation. Not all instances of graphs can do this. Therefore choosing the right examples was a crucial decision for each topic.

I found that oversimplification of the topic sometimes hides more than it reveals. Using a particular example of a topic has the potential to harm its generality. Therefore, one should be careful and give emphasis to formal methods in mathematics equally if not more.

7.4 Concluding Thoughts

Web applications have become today the primary interface between humans and computation and communication. Front end programming is therefore responsible for the interface between most humans and computer services. A web application can be an convenient tool for teaching visual concepts like maths and geometry. A program (and the programming paradigm used to develop it) must be robust enough to handle such complexity efficiently and elegantly.

Teaching such abstract mathematical and computing concepts is hard, and it may be even more difficult when it is done in absence of a teacher, through written and drawn material. Though a lesson should be simple, it should not be simpler than it ought to be and I feel this can give rise to complexity and non-linear stories.

While designing and implementing this project the user's intuition was my primary client. It was stepping into their shoes, that directed me to choose the material covered in the project and how it is presented. I am hoping that this project contributed to our understanding of the use of interactive learning environments for teaching graph theory, and that we see much more advancement in the field, leading us to more engaging and efficient teaching of science and technology.

A Links to External Resources

A.1 The Web Application

https://visualise-graph-problems-with-me.netlify.app/

A.2 Git Hub link of The Project Repository

Here is the Git Hub link for the Project. It consists of documentation, report and the code of the Web Application.

https://github.com/FatmaSayegh/Level4Report

A.3 Form for Feedback on Initial Concept

https://docs.google.com/forms/d/1MMN1H8YMCPlG9Modh5vv0Hy2ZHLsvvW769WPRKvuKF0/viewanalytics

A.4 Unsupervised Evaluation Survey Form

https://docs.google.com/forms/d/1G-RElMkFfn5TBZN5Os2lmySNeMe8DzWCH4nKqPKjRTw/viewanalytics

A.5 Supervised Evaluation Survey Form

https://docs.google.com/forms/d/1LzBP-KeheT2cCgGcfkuarrLV47omhg7ei6WIQbdvx-8/viewanalytics

B Unit and Propery Based Testing Information

B.1 Result of Running the Tests.

The unit and property based testing were done by running the cli command *elm-test*. This is a snapshot of the terminal.

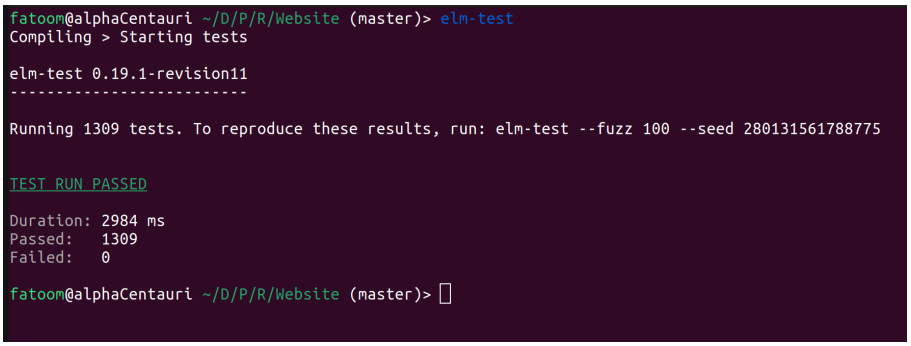

Figure B.1: *Results of unit and property based testing.*

B.2 Examples of Property Based Testing

The first example checks if a fully connected graph is constructed well by counting the number of edges which have been formed. The second example generalizes this to test many types of graphs. Here also the number of edges formed is compared against the mathematical number of edges which a particular kind of graph should have.

```
fullyConnectedGraphEdgeCountSuite : Test
fullyConnectedGraphEdgeCountSuite =
    describe "fully connected graph edge count suite " <|
    let
        polySize =
            List.range 3 100
    in
    List.map
        fullyConnectedGraphEdgeCountTest
        polySize

fullyConnectedGraphEdgeCountTest : Int -> Test
fullyConnectedGraphEdgeCountTest n =
    let
        size = (vec3 100 100 0)
```

```
16
17          pos = (vec3 100 100 0)
18
19          graph =
20             makeGraph
21                (PolygonFullyConnected n)
22                size
23                pos
24                0
25
26          noOfEdges =
27             graph.edges
28             |> List.length
29
30     in
31     test ("fully connected graph edge count " ++ (String.fromInt n)) <|
32      \_ ->
33          Expect.equal
34             (( n * ( n - 1) )//2)
35             noOfEdges
36  \label{listing: testA}
```

Listing B.1: *Property Based Testing of construction of a fully connected graph.*

```
1   makeGraphNoVerticesSuite : Test
2   makeGraphNoVerticesSuite =
3      describe "Checking if correct number of vertices is made correctly " <|
4          let
5             polygonSizes =
6                List.range 3 200
7
8             polygonCycles =
9                List.map PolygonCycle polygonSizes
10
11            fullyConnecteds =
12               List.map PolygonFullyConnected polygonSizes
13
14            dolls =
15               List.map PolygonCycleDoll polygonSizes
16
17            combinedGtypes =
18               polygonCycles ++ fullyConnecteds ++ dolls
19         in
20         List.map
21            makeGraphNoOfVerticesTest combinedGtypes
22
23  makeGraphNoOfVerticesTest : Gtype -> Test
24  makeGraphNoOfVerticesTest gtype =
25         let
26            sizeOfGraph =
27               (vec3 150 150 0)
28
29            positionOfGraph =
30               (vec3 150 150 0)
31
32            graph =
```

```
33              makeGraph gtype positionOfGraph sizeOfGraph 0
34
35          (expNoVer, gtypeStr) =
36              case gtype of
37                  PolygonCycle n ->
38                      (n, "Cycle")
39                  PolygonFullyConnected n ->
40                      (n, "Fully Connected")
41                  PolygonCycleDoll n ->
42                      (2*n, "Doll")
43       in
44       test
45          ("No. of vertices according to type of graph "
46              ++ String.fromInt expNoVer
47              ++ gtypeStr)  <|
48          \_ ->
49              graph
50              |> .vertices
51              |> List.length
52              |> Expect.equal expNoVer
53  \label{listing: testB}
```

Listing B.2: *Property Based Testing of construction graphs based on their type. For example, the graph with a given number of vertices should have the number of edges dependent on the number of vertices.*

C Manual System Testing

In this appendix, the details of the manual system tests and their resuls are tabulated.

C.1 Home Page Testing

Test Id	Test Case	Note	Expected	Actual
1	URL of the home page works	Typing the URL on the navigation bar of the browser loads the home page of the web app	Home page of the Web app loads	As expected
2	Home page button works	Pressing the Home page button should not change the page shown on browser.	Home Page stays on the browser	As expected
3	About Page button works	Pressing the About Page button on the header bar puts the About Page on the browser with the relevant url on the navigation bar of the browser.	About page is displayed.	As expected.
4	Title of the application is horizontally centered.	Title of the application is at the center of the page on different screen sizes.	Title of the screen size is at horizontally centered.	As expected.
5	Proper display of icons of graph theory topics.	The two rows of icons are horizontally centered and contain minigraphs of corresponding topics.	Mini graphs appear in the icons and icons are aligned nicely.	As expected.
6	Clicking on topic icons	Clicking on the topic icons should take the user to the relevant page.	Clicking on the topic icons should take the user to the relevant page.	As expected.

C.2 About Page Testing

Test Id	Test Case	Note	Expected	Actual
1	Elements appear at the intended positions.	All the elements appear at the right positions. The topic and description appears at the center and rest of the elements are left and right aligned alternatively and the page is scrollable.	All elements are at their respective positions.	As expected.

C.3 Header Bar Testing

Test Id	Test Case	Note	Expected	Actual
1	Header bar appears at the top of every page.	Header bar gives user the navigation options to go to the Home page (which has the contents) and the about page.	All the pages visited on the application must have the header bar.	As expected.
2	Header bar stays at it's position.	Header bar stays at the top of the page, even if the rest of the page is scrolled. This must be true for all the pages on the application.	All pages must have the header bar at top even if rest of the page is scrolled.	As expected.

C.4 Isomorphism Page Testing

Test Id	Test Case	Note	Expected	Actual
1	Url of Isomorphism Page Works.	In an SPA ideally each page be independently reachable with it's own unique Url.	Typing Url of the isomorphism path conclatenated to the url of the app will load the isomorphism page	As expected.
2	All elements of the page are diplayed.	All the elements of the page, the display part, the explanation panel, along with the buttons are present on the page.	All intended elements page are present.	As expected.
3	Animation is working for the first part of the Isomorphism topic.	At the press of the play button, the animation starts. When the press of pause button is pressed animation is paused. When the restart button is pressed, the animated graph moves to the initial position.	All intended function of the animation and the buttons are working as intended.	As expected.
4	Next Task button.	Pressing the next task button from the first animation in the Isomorphic topic should take the user to the next task in Isomorphic topic.	The second task of the Isomorphism topic is displayed.	As expected.
5	All elements of the task are displayed.	On the task section of the Isomporphic page being displayed, all elements of the page should appear at their respective positions.	All elements of the page are at their intended positons.	As expected.
6	The quiz is functioning as intended.	Functionality, of choosing the right option, pressing of the check button and resulting animation on the display panel and resulting change in explanation panel, even if the task is carried out by the user out of order should not lead to the application showing unexpected behaviour.	All intended function of the task and the buttons are working as intended.	As expected.

C.5 Max k Cut Page Testing

Test Id	Test Case	Note	Expected	Actual
1	Url of Max k Cut Page Works.	In an SPA ideally each page be independently reachable with it's own unique Url.	Typing Url of the Max-Cut path conclate-nated to the url of the app will load the Max K Cut page. Showing the Max 2 Cut page of the topic.	As expected.
2	All elements of the page are diplayed.	All the elements of the page, the display part, the expla-nation panel, along with the buttons are present on the page.	All intended elements page are present.	As expected.
3	Animation is working for the first part of the Max k topic.	At the press of the play but-ton, the animation starts. When the press of pause button is pressed animation is paused. When the restart button is pressed, the ani-mated graph moves to the initial position. The cut-line button produces a cut line over the graph.	All intended function of the anima-tion and the buttons are working as intended.	As expected.
4	Next Anima-tion button.	Pressing the next animation button from the Max 2 Cut part of the topic should take the user to the Max 3 cut topic and all the elments of that topic must be present at the right position.	Max 3 Cut topic is dis-played on the browser.	As expected.
5	Animation is working for the second part of the Max 3 topic.	At the press of the play but-ton, the animation starts. When the press of pause button is pressed animation is paused. When the restart button is pressed, the ani-mated graph moves to the initial position. The cut-line button produces a cut line over the graph.	All intended functions of the anima-tion and the buttons are working as intended.	As expected.

C.6 Graph colouring Page Testing

Test Id	Test Case	Note	Expected	Actual
1	Url of Graph colouring Page Works.	Graph colouring page be independently reachable with it's own unique Url.	Typing Url of the Graph colouring path conclatenated to the url of the app will load the Graph colouring page.	As expected.
2	All elements of the first graph colouring task are displayed.	On the task section of the graph page being displayed, all elements of the page should appear at their respective positions.	All elements of the page are at their intended positons.	As expected.
3	The tasks are functioning as intended.	Functionality, of picking up the right colour from the colour palette, colouring the vertex, diplay of edges with wrongly coloured vertices, diplay of appropriate text depending on the state of the task and the uncolour button even if the tasks are carried out by the user out of order should not lead to the application showing unexpected behaviour.	All intended function of the tasks and the buttons are working as intended.	As expected.
4	Next Task button.	Pressing the next task button from the first task should take the user to the second task.	Next Task is diplayed on the browser on the press of the relevant button.	As expected.

C.7 Vertex Cover Page Testing

Test Id	Test Case	Note	Expected	Actual
1	Url of Vertex Cover Page Works.	Vertex Cover page be independently reachable with it's own unique Url.	Typing Url of the Vertex Cover path conclatenated to the url of the app will load the Vertex Cover page.	As expected.
2	All elements of the first vertex cover task are displayed.	On the task section of the graph page being displayed, all elements of the page should appear at their respective positions.	All elements of the page are at their intended positons.	As expected.
3	The tasks are functioning as intended.	Functionality, of the task of selecting the vertices, and display of the edges incident on such edges, and appereance of corresponding text even if the tasks are carried out by the user out of order should not lead to the application showing unexpected behaviour.	All intended function of the tasks are working as intended.	As expected.
4	Next Task button.	Pressing the next task button from the first task should take the user to the second task.	Next Task is diplayed on the browser on the press of the relevant button.	As expected.

C.8 Tree Width

Test Id	Test Case	Note	Expected	Actual
1	Url of Tree Widht page works.	Tree width must page be independently reachable with it's own unique Url.	Typing Url of the Max-Cut path conclatenated to the url of the app will load the Max K Cut page. Showing the Max 2 Cut page of the topic.	As expected.
2	All elements of the page are diplayed.	All the elements of the page, the display part, the explanation panel, along with the buttons are present on the page.	All intended elements page are present.	As expected.
4	Next Animation/Previous Animation buttons.	The next animation and previous animation buttons hop between the intended animations in series.	The Next and Previous Animation buttons work as intended.	As expected.
3	The series of Animations are working.	The animations in the series occur in right order and each work as intended. each is accompanied by the intended text in the explanation panel.	The animation and explanations work as intended.	As expected.

7 Bibliography

V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics*, 14(S7), Apr. 2013. doi: 10.1186/1471-2105-14-s7-s13. URL https://doi.org/10.1186/1471-2105-14-s7-s13.

J. Fairbank. *Programming Elm: Build Safe and Maintainable Front-End Applications*. Pragmatic Bookshelf. 2019. ISBN 1680502859. URL http://gen.lib.rus.ec/book/index.php?md5=597c62b902710b0f282678b951ce8638.

R. Feldman. *Elm in Action*. Manning Publications, 2020. ISBN 9781617294044. URL https://books.google.co.uk/books?id=KNt8vQEACAAJ.

S. Fowler. Model-view-update-communicate: Session types meet the elm architecture, 2019. URL https://arxiv.org/abs/1910.11108.

D. Galles. Data structure visualizations. https://www.cs.usfca.edu/galles/visualization.

C. Gros. *Complex and adaptive dynamical systems*. Springer International Publishing, Cham, Switzerland, 4 edition, Apr. 2015.

S. Halim. Visualgo. https://visualgo.net/en.

A. Hudaib, R. Masadeh, M. H. Qasem, and A. Alzaqebah. Requirements prioritization techniques comparison. *Modern Applied Science*, 12(2):62, Jan. 2018. doi: 10.5539/mas.v12n2p62. URL https://doi.org/10.5539/mas.v12n2p62.

P. Hudak, J. Hughes, S. P. Jones, and P. Wadler. A history of haskell. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*. ACM, June 2007. doi: 10.1145/1238844.1238856. URL https://doi.org/10.1145/1238844.1238856.

J. Hughes. Why Functional Programming Matters. *The Computer Journal*, 32(2):98–107, 01 1989. ISSN 0010-4620. doi: 10.1093/comjnl/32.2.98. URL https://doi.org/10.1093/comjnl/32.2.98.

K. Karvonen. The beauty of simplicity. In *Proceedings on the 2000 Conference on Universal Usability*, CUU '00, page 85–90, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581133146. doi: 10.1145/355460.355478. URL https://doi.org/10.1145/355460.355478.

J. Kleinberg and E. Tardos. *Algorithm Design*. Addison Wesley, 2006.

L. Lau. Algmatch. https://liamlau.github.io/individual-project/.

M. Newman. *Networks: An Introduction*. Oxford University Press, 03 2010. ISBN 9780199206650. doi: 10.1093/acprof:oso/9780199206650.001.0001. URL https://doi.org/10.1093/acprof:oso/9780199206650.001.0001.