william roebuck

Centre number: 48317

Candidate Number: 6773

VisuGraph

General purpose graph visualisation software development project

Software development project

William Roebuck

Contents

[Analysis 2](#_Toc114398723)

[The problem 2](#_Toc114398724)

[Computational methods 2](#_Toc114398725)

[Key features that make the program solvable by computational methods: 2](#_Toc114398726)

[Object oriented programming 2](#_Toc114398727)

[Functional and procedural programming 2](#_Toc114398728)

[Visualisation 2](#_Toc114398729)

[Heuristics and backtracking 3](#_Toc114398730)

[Stakeholders 3](#_Toc114398731)

[Esteban Garcia 3](#_Toc114398732)

[Paula Weston 3](#_Toc114398733)

[Harry Thorne 3](#_Toc114398734)

[Interview 3](#_Toc114398735)

[Research and existing solutions 5](#_Toc114398736)

[CS academy graph editor 5](#_Toc114398737)

[Graph online 6](#_Toc114398738)

[Gephi 6](#_Toc114398739)

[NetworkX 7](#_Toc114398740)

[vis.js 7](#_Toc114398741)

[graph 7](#_Toc114398742)

[Essential Features 8](#_Toc114398743)

[Limitations 8](#_Toc114398744)

[Hardware and software requirements 9](#_Toc114398745)

[Success Criteria 10](#_Toc114398746)

[Design 11](#_Toc114398747)

[Decomposition 11](#_Toc114398748)

[Structure 11](#_Toc114398749)

[Algorithms 11](#_Toc114398750)

[Usability Features 11](#_Toc114398751)

[Data Structures 11](#_Toc114398752)

[Test Data 11](#_Toc114398753)

[Post development 11](#_Toc114398754)

[Implementation 11](#_Toc114398755)

[Evaluation 11](#_Toc114398756)

[Testing for function 11](#_Toc114398757)

[Testing for robustness 11](#_Toc114398758)

[Testing for usability 11](#_Toc114398759)

[Success Criteria 11](#_Toc114398760)

[Further development 11](#_Toc114398761)

[Maintenance 11](#_Toc114398762)

# Analysis

## The problem

Graphs are useful for a wide variety of applications, including computer networking and urban planning, however there are very few general-purpose node programs that can be used without significant programming experience. The aim of my program is to be easy to set up and intuitive to use, while still maintaining a high level of functionality.

This program will allow the user to create, edit and view node networks, including algorithms such as Dijkstra’s, A\*, Kruskal’s, Floyd’s, brute force travelling salesman, and the planarity algorithm. The networks can be weighted and/or directional. The program should save graphs to a file and can load graphs from files.

## Computational methods

This problem is suitable for a computational solution mainly to exploit the benefits of abstraction. Since the computation will be done for the user, a high level of expertise is not needed to understand the outputs of the program and visualisation will aid this.

### Key features that make the program solvable by computational methods:

1. Algorithms can be performed on the graph
2. The inputs can be taken from the keyboard and mouse to generate outputs of a visualised graph or algorithm
3. There are points of decision throughout the program, such as what a mouse click should do, affected by its position.

### Object oriented programming

OOP will play a large part in this project’s development as I intend to implement a “Graph” class that will be used for almost all of the program’s features. This has the benefit that the code is reusable if I create a project in future that includes graphs but it is also a useful abstraction, in that it is more intuitive to access the parts of a graph (Graph.Count, Graph.Weight) as attributes rather than as standalone variables (graphCount, graphWeight) which will help with future development as well as making code easier to understand. Furthermore, the class will have many methods, at least one for each graph algorithm that can be used, so that the graph and its data do not have to be passed into a function outside of the class which improves security.

### Functional and procedural programming

Functional and procedural programming will also be important for this project for handling most of the user driven parts of the program, including the UI. These subprograms will allow for event driven code (through windows forms) and make code more human readable.

### Visualisation

Visualisation is essential to the program and is key to making the program user friendly. Graphs will be visualised as soon as they are created/loaded and will update to reflect changes made by the user. It will also be used to show the outputs of algorithms such as Dijkstra’s by highlighting the given route

### Heuristics and backtracking

Heuristics will be used as part of A\* (which improves performance) whereas Dijkstra’s algorithm will use backtracking.

### Initial decomposition

The program will consist of 4 main parts, each of these will be broken down further in the design section

1. User interface with buttons, icons and graph rendering ability
2. Graph data structures that can be saved and loaded to/from files
3. A collection of algorithms that can be performed on the graphs
4. Graph editing capability (click and drag/add nodes/edit edges/edit nodes)

## Stakeholders

### Esteban Garcia

Esteban Garcia is a 57-year-old urban planner who is not fluent in English and requires the program to allow accents such as á, é, í, ñ, ó, and ú as inputs. He is familiar with graph structures but usually uses paper to work with them. He needs the program to be flexible in what it can display and calculate. As he may use the program in his work, he needs it to be robust and save data in the case of a crash, which should be a rare occurrence if it does ever happen.

### Paula Weston

Paula Weston is a partially sighted 17-year-old further maths student who requires the program to have a dark colour palette and an option to increase font size for the whole program. She needs to be able to create graphs quickly and wants a clear output that highlights things such as the shortest path for Dijkstra’s

### Harry Thorne

Harry Thorne is a 34-year-old professional cyclist who requires the program to be easy to use and understand without studying computer science or further maths. He will use the program to plan his routes and needs the nodes and edges to be able to store and display long multi-word names such as “Barton-upon-Humber” or “Cammeringham Hill”. Harry also needs the program to be able to handle large, detailed networks without becoming unusable or unstable.

## Interview

**How will you use the program?**

**Garcia:** I intend to use it to speed up the initial planning phase of my work and reduce the amount of writing materials I go through

**Weston:** I want to use the program to create questions and answers so I can improve my understanding in further maths. For example: I want to be able to create a graph in the program, work out the shortest route by hand and then verify if my answer is correct using the program.

**Thorne:** I spend a lot of time planning long cycles around the UK that often involve visiting a number of key locations like B&Bs or landmarks so I want to use the program to not only speed this up but also make sure that I’m taking the best route

**Do you have any hardware limitations?**

**Garcia:** I have not upgraded my hardware since 2010 and I am used to it being slow. As long as the program is responsive enough so that I can use it, it will be fine.

**(After some discussion, we came to the conclusion that for his PC, with 4GB of RAM, the response time should be no more than 2 seconds for any graph manipulation)**

**Weston: My PC is built for the latest videogames, so it should be able to run most things, but I dislike when the fan starts making a lot of noise, so I hope the program isn’t too intensive**

**Thorne: I have a somewhat modern laptop; it can run many tabs of Google Chrome without issues but it sometimes slows down during intensive physics simulations**

**Do you have any accessibility requirements?**

**Garcia:** My English is not very good, but I will be able to use the program as long as there are icons that show what things do

**Weston:** I am partially sighted and can’t use bright screens for a long time so I need text to be quite large and the interface to be dark

**Thorne:** No

**How experienced are you with graphs?**

**Garcia:** I studied them in university and use them quite often for my job. I can do some of the algorithms by hand, such as Dijkstra’s and Floyd’s.

**Weston:** I’m currently learning about them in further maths so I have a good understanding of them but I’d rather let the computer carry out the algorithms for me.

**Thorne:** I’ve never really used them before but I know that they can be very useful for route planning and optimisation

**How familiar are you with computer interfaces and how do you like to interact with them?**

**Garcia:** I use my company’s software on my computer, and most of it is not very user friendly so it takes a long time to get used to. I would prefer if I could just click buttons and not have to find everything in verbose menus and sub-menus.

**Weston:** I’ve used a lot of different programs for installations and niche applications so I am quite used to confusing user interfaces. I like to use my keyboard to navigate around the interface as its more convenient.

**Thorne:** I mostly use my computer for YouTube and Google but have used some physics software. I prefer visual representations in the user interface since I find icons easier to remember than words.

**At what scale are you likely to use the program?**

**Garcia:** My node networks are often between 25 and 50 nodes however there always far less edges than there could be (more like 300 connections instead of 1225)

**Weston:** I’d only need the program for really small networks; less than 20 nodes and 30 edges

**Thorne:** I don’t think I’d use any more than 15 nodes, but there’d be a lot of connections between them, about 100

**Would you like to name every node and edge?**

**Garcia:** Naming the nodes will be essential but I would only rarely name edges

**Weston:** The names of nodes and edges don’t matter to me; I’d prefer if each node was automatically named A-Z and the edges left blank

**Thorne:** I might not name every edge, but certainly the important ones and each node must have a name

**Which algorithms are the most important to you?**

**Garcia:** It is hard to decide, but I currently use Dijkstra’s the most. Floyd’s will also be very useful and I would use the planarity algorithm if it is included

**Weston:** I think I’ll use all of them except maybe A\* and travelling salesman, but I’d say Dijkstra’s is the most important to me

**Thorne:** I’d have to test some of them out, but right now I think travelling salesman is the most important. I’d probably rather use A\* over Dijkstra’s because it doesn’t have to be 100% accurate for me, I’d prefer whichever’s quickest.

**What screen size do you use?**

**Garcia:** 640x480

**Weston:** 2560x1440

**Thorne:** 1280x720

## Research and existing solutions

### CS academy graph editor

<https://csacademy.com/app/graph_editor/> 

This web-based solution is an effective visualisation tool for node graphs, allowing the user to “draw” a graph by clicking and dragging on nodes. Adding nodes is efficient and intuitive as one must click an empty space and then simply click and drag to create a connection between them. Additional details such as weight and direction can be added later by clicking on the connection. The program is somewhat lacking in functionality in that it can only display a graph, not make any calculations using one.

The program also has a range of customisation features, including the ability to change node colour, size, and other options, however there is no way to increase font size.

Graphs can only be saved and loaded by copying the data textbox on the left into a file which makes managing a large group of graphs difficult.

From this I will incorporate the click and drag functionality and the elegant interface that becomes more complex as more information is added

### Graph online

https://graphonline.ru/en/



Graph online is another web-based solution for working with graphs but with a focus less on visualisation and more on algorithms. It offers 20 different useful algorithms that can be performed on your graph, selectable from a dropdown above the graph.

However, the program is hard to use in that to add a new vertex one must first select “Add vertex”, click on the graph, then “Connect vertices” and click on the two vertices. Then, a popup box appears that must be filled in each time a connection is made. In comparison to the CS academy graph editor, it is far slower and more tedious to use.

I will try to avoid using dialogue boxes in my program as they interrupt the graph creation process.

### Gephi

Gephi is a powerful graph visualisation tool that performs well with huge datasets, claiming to support up to 1 million nodes and edges at once. Its primary use is for visual data analysis and it offers a range of tools such as grouping, highlighting and naming to make this more effective. It also offers a zoom feature so that large networks can be closely examined

The UI makes use of icons for most tools, most of these make the function quite clear however some are more vague, such as the aeroplane icon. As shown in this screenshot, there is an option to change the font size, however this is only for the node names and not the UI controls.

As for algorithms, the program offers multiple layout algorithms to improve readability, a range of data analysis tools (Betweenness Centrality, Closeness, Diameter, Clustering Coefficient, and PageRank) and an unspecified shortest path algorithm.

Gephi can import the majority of graph file formats and can save and load projects to files. One drawback of this is that there are no tools inside the program to construct a graph visually by placing individual nodes.

### NetworkX

NetworkX is a collection of python modules that can be used to create, manipulate and analyse networks. It is not primarily a graph visualisation software but can be used with Matplotlib to draw its networks

Here is an example of how to construct a graph and find a shortest path between two nodes. While intuitive for a programmer, this text-based interface is not suitable for somebody with no experience. Furthermore, for very networks typing out each bracket, quotation mark, and comma would not be efficient.

A unique benefit of this solution is that “Nodes can be anything”, for example images, text or even XML records. Edges can also hold data such as weights

### vis.js

vis.js is a visualisation library for JavaScript, one of its many applications being network visualisation.

As shown here, it is quite versatile in that nodes can be any shape/colour and edges can have both weight (indicated by their thickness) and labels. It also claims to support up to 1000 nodes and 1000 edges

Furthermore, as it is a JavaScript library, it can be implemented in such a way that allows the creation of networks without code and, while it does not include graph algorithms, the data structures it offers could be utilised to implement as many as the programmer desired.

### graph

graph is a graph visualisation library for Go and, while offering many of the features of solutions shown above, it also offers a few unique ones. It offers breadth and depth first traversal algorithms as well as an option to prevent cycles during graph creation.

## Essential Features

From my research and discussions with stakeholders, I have decided on these essential features:

* UI accessible algorithms to be performed on user’s graphs. These are important to the functionality of the program so that it can be used in the wide range of applications desired by my stakeholders. I have chosen to include Dijkstra’s **and** A\* because for some applications such as city planning, a large number of nodes will be required and Dijkstra’s may be too slow to remain effective at these scales.
  + Dijkstra’s – For finding the shortest path between two nodes, useful for planning and route finding
  + A\* – Same function as Dijkstra’s but faster at the cost of accuracy
  + Kruskal’s – For finding the minimum spanning tree, can be useful in planning and finding an upper and lower bound for travelling salesman
  + Brute force travelling salesman – For visiting every node in the network using the minimum weight.
  + Floyd’s – For constructing a minimum distances table, useful for route planning
  + The planarity algorithm – For determining whether a graph is planar, meaning the graph can be drawn without any overlapping lines, useful for urban planning.
* Ability to create, save, name, rename, and delete graphs (storing this data on file(s)) – This is important so that users can work on multiple projects concurrently
* Graph manipulation capability: Adding, removing, moving, and naming nodes; connecting nodes, naming, removing, and automatic/manual weighting connections. – Important for visualisation and the function of the program
* Accessibility settings: Font size, colour scheme, keyboard controls

## Limitations

Some solutions that I researched offered a “gravity” or automatic layout feature, where the nodes in the network automatically arrange themselves in a geometric way so that they are evenly spaced out. While this is useful for abstract applications where the physical layout of the nodes is not important, my program is primarily concerned with problems in the real world so will not include this.

One accessibility option that I have not accounted for is a screen reader to explain what the program is doing for blind users. I have decided not to implement this because one of the main features of the program is visualisation and there are other solutions which would be better suited to a purely numeric implementation. Furthermore, Windows 10, the operating system the program is designed for, comes with a screen reader so an additional one is not necessary.

Another potential feature would be language options; however, these should not be necessary as the program’s main purpose is visualisation so there will be more symbols than English language. Furthermore, all the algorithms have names or are named after somebody so will not need to be localised. The program will support user inputs from other languages, so the user can type with their language if they wish

Given the time provided, a step-by-step explanation of each algorithm’s process cannot be implemented. While this would be invaluable in an educational environment, this is not important for most applications. Using the program to teach graph theory is not any of my stakeholder’s intended use cases.

## Hardware and software requirements

To compile the program, I will need:

* Microsoft Visual Studio (which requires windows 10+)

The user will need:

* Windows 10[[1]](#footnote-1)
* 4 GB or more of RAM[[2]](#footnote-2)
* 1 GHz or faster processor[[3]](#footnote-3)
* 16 GB for 32-bit OS or 20 GB for 64-bit OS[[4]](#footnote-4)
* Built in microsoft libraries
* A keyboard, 4:3 or 16:9 display[[5]](#footnote-5), and mouse

## Success Criteria

Note to self: Specific, measurable, achievable, realistic, timely

|  |  |  |
| --- | --- | --- |
| No. | Criteria | Justification |
| 1 | The user can create and manipulate graphs with GUI controls | This is an essential feature for the stakeholders’ use of the program |
| 2 | The user can run algorithms on their graphs that display an output |
| 2.1 | Dijkstra’s algorithm |
| 2.2 | A\* algorithm |
| 2.3 | Kruskal’s algorithm |
| 2.4 | Travelling salesman |
| 2.5 | Floyd’s algorithm |
| 2.6 | Planarity algorithm |
| 3 | Initially, nodes should be named A-Z but edges should be left blank |
| 4 | The user can save and load graphs to/from files | A stakeholder has specified that the program must be suitable for long term, multi-project use |
| 5 | The user can name edges and nodes | Stakeholders require this feature as it is important for the program’s use to model real situations |
| 6 | Users can input Unicode which will be displayed with the same font, colour, size and spacing | A stakeholder has specified that the program must support languages with extended alphabets such as Spanish |
| 7 | If an error occurs, the user should be shown what has caused it | A stakeholder has said that the program may not be perfect, but that the user should be able to see what has gone wrong to avoid future errors |
| 8 | The program must have accessibility options to ensure suitability for partially sighted users | A stakeholder requires these features to use the program |
| 9 | In the event of a crash, the program should be able to recover a recent copy of the most recent graph if it has not been saved | This is important to stakeholders as the program must be reliable so that it can be used in industry |
| 10 | The program must be sufficiently stable and responsive (no more than 2 second delay) with large networks up to 50 nodes and 300 edges in size. | Stakeholders may use the program for very large projects so it must be functional as scale grows |
| 11 | The program must be usable without any English language, (excluding algorithm names). I will ask my stakeholders to evaluate whether this criterion is met | Not all of my stakeholders are fluent in English, so I should ensure that the program can be used without relying on English |
| 12 | The program must have at least 3 zoom levels | My stakeholders will manage large networks so zooming is important for visualisation |
| 13 | If the user tries to perform an algorithm with invalid parameters (such as Dijkstra’s on a graph with loose nodes) they should be shown a popup explaining this | My users may make mistakes if they are not already familiar with the algorithms, the program should be robust in this case |
| 14 | All parts of the program accessible with the mouse should be accessible with the keyboard | Some people, including a stakeholder prefer to navigate programs using a keyboard |
| 15 | The program should be resizable to fit on a 4:3 or 16:9 screen | This will make the program more versatile and suit the stakeholders’ hardware |

# Design

## Decomposition

This diagram will be useful to me during development as it is arranged in such a way that each component is linked to what must be implemented before it and which must be implemented after. For example, to complete the graph data structure I must implement graph algorithms, which also require error handling and validation. I will also show this diagram to my stakeholders so they can understand the order in which I will create the program.

## Structure

To develop my program, I will use the spiral methodology and I have chosen it because it allows flexibility when my stakeholders requirements change. My stakeholders do not know exactly what they want the final product to look like so they will see it evolve as I develop it and will be able to suggest potential features that I can add in the next iteration. Furthermore, with this SDLC the final program will have excellent usability because my stakeholders can have an impact during development, giving me the flexibility to adapt features to their needs

Evaluation

Analysis phase

Implementation

Design

Start of project

1st prototype

2nd prototype

3rd prototype

Final prototype

### Iteration 1

The first iteration will focus on the user interface which I can then show to my stakeholders and ask whether they have any suggestions

Prototype 1.0 will only have the controls and no functionality

Prototype 1.1 will have a non-functional options menu that is shown when the options button is pressed

Prototype 1.2 will be able to toggle a dark colour palette

### Iteration 2

The next iteration will implement and display the graphs

Prototype 2.0 will be able to display graphs inputted by the programmer

Prototype 2.1 will allow the user to rename nodes and edges

Prototype 2.2 will allow the user to move, add and delete parts of the graph

When this is complete, I will talk with my stakeholders to ensure the graphs are what they are looking for

### Iteration 3

Iteration 3 will allow the user to save and load graphs

Prototype 3.0 will allow the user to save and load graphs to/from a file

Prototype 3.1 will autosave and ask the user to recover data in the case of a crash

### Iteration 4

The final iteration will implement the algorithms

Protype 4.0 will allow the user to perform and see the output of Dijkstra’s and A\*

Protype 4.1 will allow the user to perform and see the output of Kruskal’s, Floyd’s and the planarity algorithm

Prototype 4.2 will allow the user to perform and see the output of the travelling salesman algorithm

Prototype 4.3 will validate user input to each algorithm

After this the program will be complete so I will ask for any final feedback from stakeholders and then implement any changes required

## Algorithms

## Data Structures

## Usability Features

Draw.io UI (mockups)

## Test Data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | Description | Test data | Type | Expected result | Linked success criteria |
|  |  |  | Valid |  |  |
|  |  |  | Invalid |  |  |
|  |  |  | Boundary |  |  |

## Post development

# Implementation

# Evaluation

## Testing for function

## Testing for robustness

## Testing for usability

## Success Criteria

## Further development

## Maintenance

1. The program will be tested on Windows 10 so although the program will most likely work from Windows XP and up I cannot be sure without testing for each operating system [↑](#footnote-ref-1)
2. This is the smallest amount of total RAM out of my stakeholders [↑](#footnote-ref-2)
3. Source: <https://support.microsoft.com/en-us/windows/windows-10-system-requirements-6d4e9a79-66bf-7950-467c-795cf0386715> [↑](#footnote-ref-3)
4. Source: <https://support.microsoft.com/en-us/windows/windows-10-system-requirements-6d4e9a79-66bf-7950-467c-795cf0386715> [↑](#footnote-ref-4)
5. My stakeholders use both 4:3 and 16:9 displays [↑](#footnote-ref-5)