Visualisation of Concept Maps to Show Both Size and Affinity Data

Author: Matthew Stuart Reilly

Final Year Dissertation

Bachelor of Science in Computer Science

Supervisor: Prof. Mike Chantler

2nd Reader: Dr Ioannis Konstas

Declaration

I, Matthew Reilly confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: 

Date: 23/04/19

Abstract

Concept maps are incredibly useful for discussion and exploration however they are difficult to generate automatically such that they give an engaging and informative overview of their data. It is surprisingly difficult to generate concept maps that simultaneously show (a) affinity data (i.e. how similar one item is to another, e.g. the more similar, the closer they should be placed to one-another) and (b) size information.

This dissertation will investigate ways in which to represent both affinity data and size information simultaneously and then implement different ways of packing different sized circles together to satisfy both affinity (similarity) and size constraints including one new method. There will be an evaluation carried out also which will determine which implementation represents given data the most effectively.

Table of Contents

1. Introduction

A concept map is a 2D graphic of some imaginary space that shows the relationships between major entities within that space. It has always been difficult to produce a concept map which shows both affinity data and size information simultaneously. There are many examples of concepts maps which show either affinity data or size information but not many that show both.

The goal of this project is to build and evaluate a new algorithm for displaying a concept map which shows both affinity data and size information simultaneously. Figure 1 shows a two mocks layout of the new algorithm interface. Each will colour represents a different cluster of data (a group of similar data).

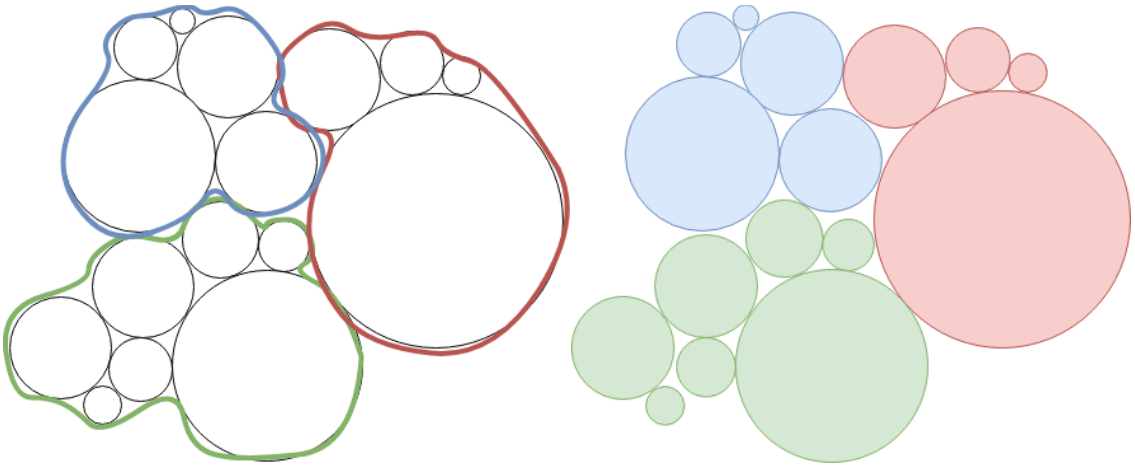


Figure 1 - Layout mocks

The motivation behind this dissertation is to create an easy to understand visualisation representation of a data set so that somebody wanting to look at a data set does not have to look at a database but can easily glace at this visualisation and get the data that need.

1.1 Document Structure

This document will be split up into 9 different chapters:

1. Introduction: Brief description of project with motivation and PLES.
2. Aims & Objectives: Details about the aim and objectives of the project.
3. Literature Review: A critical literature review about relevant research in the data visualisation.
4. Requirements Analysis: Functional and non-functional requirements of the system that is being developed.
5. Planning: Overview of the project with a timeline of the development and a list of software used in the project.
6. Implementation: How the algorithms were implemented and what problems were encountered.
7. Evaluation: Detailing approach taking to evaluate project and results that were found.
8. Conclusion: Reflecting on the overall project.
9. Future Work: What improvements could be made to the system.

1.3 Professional, Legal, Ethical and Social Issues

All code developed for this project shall be professionally written and will have detailed documentation. Doing this will allow future developers to easily understand the system.
All research will comply will the General Data Protection Regulation and all code and databases taking from online will have the relevant licenses.
There should be no Ethical and Social Issues involved with this project as there were no human participants, or no sensitive data used at any point during this project.

2. Aims & Objectives

In this chapter, the aims and objectives will be covered.

2.1 Aims

The aims of this project are to:

- Implement a new algorithm for displaying a concept map that shows both affinity data and size information simultaneously.
- Evaluate the new algorithm by comparing against other concept map algorithms that also show both affinity data and size information simultaneously

2.3 Objectives

The primary objectives of this project can be defined as followed:

- Investigate the different ways in which affinity data and size information have already been shown in a concept map.
- Investigate ways in which only affinity data can be shown in a concept map.
- Investigate ways in which only size information can be shown in a concept map.
- Investigate programming languages which would be able to implement a concept map.
- Investigate ways other people have used to get quantitative data out concept map to evaluate the algorithm.
- Investigate ways in which other people have got qualitative data from a concept map algorithm to help evaluate.
- Implement existing concept map algorithms that show both affinity data and size information.

- Implement a new concept map algorithm that both shows affinity data and size information.
- Evaluate each concept map algorithm by gathering quantitative data and then comparing each algorithm against each other to determine which algorithm is the most effective in displaying the data.

3. Literature Review

In the chapter of the document will be a review of related research in the field of data visualisation. Beginning by discussing and evaluating modelling algorithms that show both affinity data and size information. Then looking at modelling algorithms that only show affinity data and then look at modelling algorithms at only show size information. After this, there will be a discussion about different implementation technologies and then finally ways of assessing the concept maps.

The criteria for an algorithm must be:

- Good use of screen space
- Show both affinity data and size information
- Prevent overlapping of nodes
- Clearly show clusters
- Clearly show close and distant related data.

3.1 Algorithms

For each algorithm, there will be a critical analysis about if it fits the criteria that are needed for the project by looking at the algorithms use of screen space, whether it shows both affinity data and size information, if it correctly shows close and distant relations and if it prevents overlapping of circles.

3.1.1 Force-directed algorithm

For every force-directed algorithm, it relies on an attraction force and a repulsion force [1]. It uses these forces to determine the layout of the graph, attraction forces are only calculated between related nodes and a repulsion forces are calculated between every pair of nodes to keep them apart from each other [1] [2], see Figure 2.

Figure 2 - Clustered Force Layout III (https://bl.ocks.org/mbostock/7881887,2018)

The force-directed algorithm uses screen space well as the algorithm can contain a couple of constants k_a and k_r [1]. The attraction constant k_a will adjust the attraction force and k_r will adjust the repulsion force. If k_a is increased, it will reduce the size of the graph and decreasing it will increase the size of the graph. If k_r is increased, it will expand the size of the graph. These constants are useful as they can be easily adjusted to fit within the assigned screen space. Another method that is used by the algorithm to optimize screen space is the use of a 'gravity' force which is used to pull all nodes towards the centre of the space [1]. This helps prevent disconnected nodes or nodes with few links from becoming too far away from the rest of the graph and make the graph more compact.

Overlapping in force-directed algorithms is done by ensuring the centre points of two circles is greater than or equal to both radii added together [3]. Having the nodes in the graph not overlap helps the understandability of the graph.

Force-directed algorithms show affinity as attraction forces, which are calculated between nodes that are closely related making these nodes near each other on the graph. The algorithm can

also show size as the nodes can be increased or decreased in size to show the size information of each data node.

In conclusion, the force-directed algorithm is a viable algorithm that could be used for the project as it satisfies all the criteria that have been determined is needed.

3.1.2 Circle Pack Algorithm

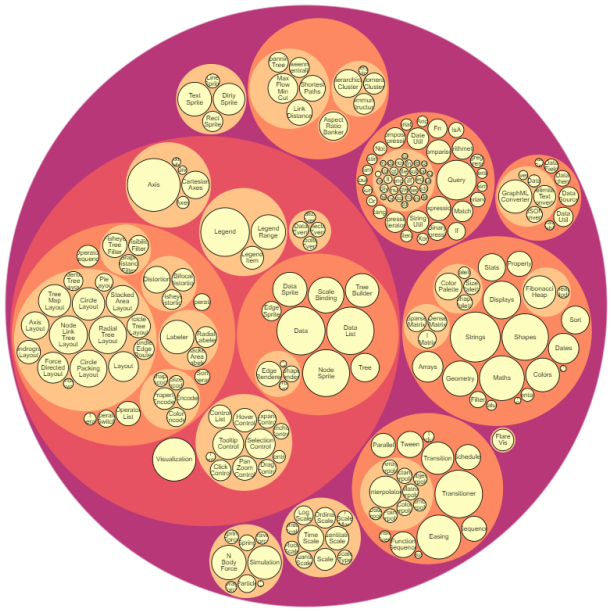


Figure 3 D3 Circle Packing (https://beta.observablehq.com/@mbostock/d3-circle-packing, 2018)

The circle pack algorithm uses hierarchical data to pack each node inside a cluster of the object [4] (as shown in Figure 3)

Inside each cluster, the circles are packed as close together as possible without overlapping to make each circle visible and save screen space [5]. That is not the case when packing the clusters together, as each cluster is a circle there is a lot of wasted screen space on the graph. The circle pack algorithm shows affinity by using the hierarchy to put the related data within the same parent circle. But within each parent circle, there are no similarities as all the data is added to use up as least about of space possible. This is done by first placing three circles together so that connecting the centres of the three circles will produce a closed loop (Figure 4 (a)) creating the front chain. Then a new circle is added to the external tangent of two circles on the front chain. The front chain will expand to include the new circle(Figure 4 (b)) [5]. This continues until all node brothers have been added. The circle pack algorithm shows close and distant neighbours by putting the near

neighbours within the same parent circle and the distant nodes will be in a different parent node on another part of the interface.

The circle pack algorithm is a viable algorithm that could be used for the project as it satisfies all the criteria that have been determined is needed.

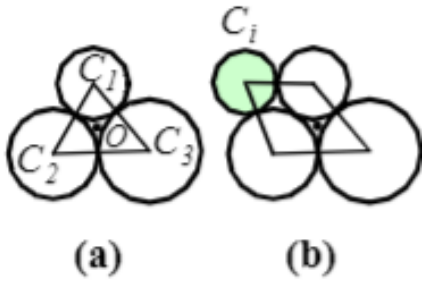


Figure 4 - Visualization of Large Hierarchical Data by Circle Packing

[5] (Page 518)

3.1.3 Agglomerative Clustering Algorithm

The Agglomerative clustering algorithm described in Le Bras Paper [6] only show affinity data between nodes.

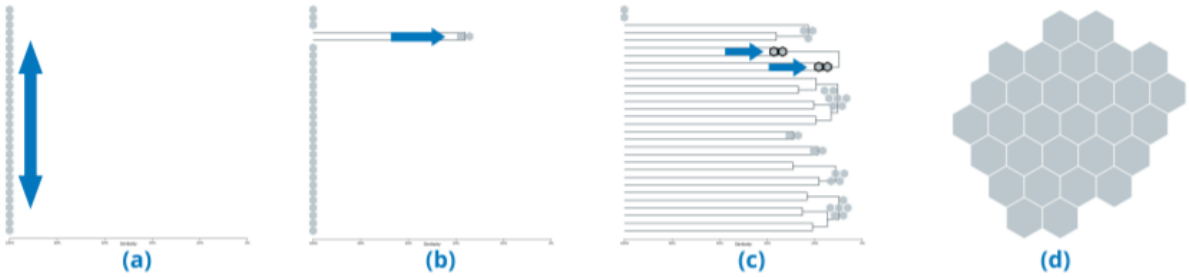


Figure 5 - An illustration of the explanation for the agglomerative layout method [6] (Page 4)

The algorithm works by first displaying all the concepts (hexagons) vertically in dendrogram order (Figure 5 (a)). The next step will be to find the two most similar sets of concepts, for the

first stage it will only be two concepts, will be merged together by positioning the hexagons together (Figure 5 (b)). This step will be repeated until the final map is created (Figure 5 (c)). Lastly, the dendrogram will be hidden and the final map will be left (Figure 5 (d)) [6].

This algorithm is very good for showing affinity data effectively since it joins the most similar concepts first meaning the most similar concepts will be closest to each other. The use of screen space is good as the hexagons are package together as close as possible with no unnecessary white space shown.

In terms of showing size information currently this algorithm does not show any but size information could be added by changing the hexagons to circles of different sizes. From the research done this method of adding size information onto an agglomerative clustering algorithm as not be done before therefore in this project an implementation will be done for this new agglomerative cluster algorithm that will also show size information.

3.2 Implementation Technology's

This section will cover different implementation technology's that the new agglomerative algorithm that was decided in the previous section will be able to be built in. The criteria needed an implementation technology will be:

- Able to perform Matrix Multiplication.
- Ability to display a graphical user interface (GUI).
- Able to clearly show clusters.
- Browser based language.

Matrix multiplication is needed for the transform the circles by performing translate (moving the circles in 2D space) and rotation.

3.2.1 Python

Python is a high-level general purpose programming language [7] created by Guido van Rossum and released in 1991. Since Python is open source [8] there are many third-party libraries that allow performing matrix multiplication one of which is NumPy [9]. Python also can plot vertices and draw circles using the open source library matplotlib which allows you to plot difference size circles (see Figure 6) by doing executing the line `matplotlib.pyplot.Circle((x, y), radius, color = "colour")` [10] [11]. matplotlib also can represent clusters as different colours by changing the colour attribute in the above line of code.

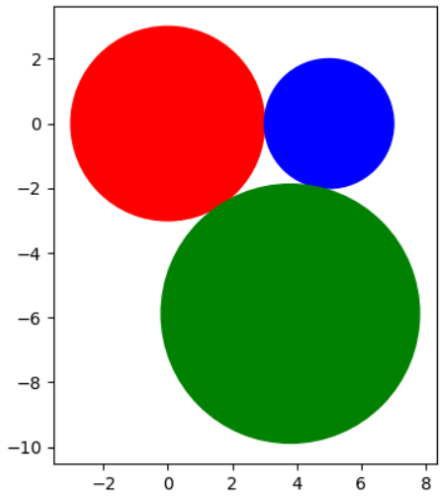


Figure 6 - pyplot drawing circles

3.2.2 D3.js

D3.js (or D3 for Data-Driven Documents) is a JavaScript library that takes a novel representation-transparent approach to visualisation for the web [12]. It makes uses of the widely implemented SVG, HTML and CSS standards. One of the developers of D3.js, Mike Bostock, has already created versions of the force-directed algorithm [13] and circle pack algorithm [14] which were released under the GNU General Public License, version 3.

3.3 Assessment Methods

This section will cover ways in which quantitative and qualitative can be gathered from concept map implementations by looking at other examples of how others have collected their data.

3.3.1 Quantitative Data

From the papers read during the research, there are many ways in which developers have gathered quantitative data. One way to gather quantitative data about an algorithm is by comparing the quality of an algorithm with other similar algorithms to it [1].

3.3.2 Qualitative Data

From the papers read during the research, there are many ways in which developers have gathered qualitative data. One way in qualitative data is collected is by performing interviews on participants [6]. This is done by letting the participant explore the concept map for as long as they want then after questions will be asked about their experience with the concept map. After using all the concept map methods then the participant will then be comparing the concept maps that they have used and decent which one they prefer.

3.4 Conclusion

In conclusion after this critical review, it has been decided that 3 different algorithms will be created. The first will be the force-directed algorithm which will be created in JavaScript using d3.js libraries as there is already a force-directed layout that can be used and modified to the

criteria. The next will be the circle pack algorithm which again will be created in JavaScript using the d3.js libraries as the is already a circle pack that can be used. The final will be the new agglomerative algorithm which will be first prototyped in Python using third party libraries like NumPy and Matplotlib then once the prototype has been complete implementation in JavaScript using d3.js libraries will try to be complete.

The assessment method that will be used is the collection of quantitative data that will be received by performing measurements on the accuracy of each algorithm and use of screen space. and comparing them.

4. Requirements Analysis

This chapter will list and explain the requirements needed to create the desired algorithms. The requirements are split into two section, Functional requirements, which describe what the algorithm should do while non-functional requirements describe how the algorithm should work. To assign priorities to each requirement MoSCoW Analysis will be used on each requirement to determine whether it is “Must have”, “Should have”, “Could have” or “Won’t Have”.

The acronyms that will be using here are:

- FR – Functional Requirement
- NFR – Non-functional Requirement

4.1 Functional Requirements

FR-1: The algorithm must show both affinity data and size information simultaneously.

Priority: Must have

FR-2: There must be no overlapping circles on the interface.

Priority: Must have

FR-3: Each circle must have a label describing what the circle represents.

Priority: Must have

FR-4: The algorithms should be able to run in a browser.

Priority: Should have

FR-5: The prototype must be built in Python.

Priority: Must have

FR-6: The final algorithm should be built using d3.js libraries.

Priority: Should have

FR-7: The algorithm must split up the map into clearly defined clusters.

Priority: Must have

FR-8: Closely related items (circles) are most close to each other and most distantly related items are furthest apart.

Priority: Should have.

FR-9: The aspect ratio of the concept map must be able to be changed.

Priority: Must have.

4.2 Non-functional Requirements

NFR-1: The algorithm must make good use of screen space.

Priority: Must have

NFR-2: The Layout of the interface should be readable and easy to understand.

Priority: Must have

NFR-2.1: Each cluster must have no more than 7 items (circles) inside it.

Priority: Must have

NFR-2.2: Labels should be contained within each circle to make it easy to read.

Priority: Should have

NFR-3: The final algorithm can be able to run in all browsers.

Priority: Can have

NFR-4: The final algorithm show run within 5 seconds.

Priority: Can have

5. Planning

Since this project has a very strict deadline it was imperative to have a plan in place so that time could be managed well. This chapter will outline the timeline of the project through a Gantt chart showing the project milestones. The software that had been chosen for this project will also be discussed in this chapter.

5.1 Project Plan

A Gantt chart has been created to show the breakdown of work over semester 2, see Figure 7. Week 1 starts on the 1st of January 2019. In addition to the activities in the Gantt Chart, there was weekly meetings with the supervisor of the project Mike Chantler.

ACTIVITY	START WEEK	DURATION
1. Circle Pack Algorithm	1	2
2. Force-Directed Algorithm	2	2
3. Start Agglomerative Algorithm	3	1
3.1 Martix creation	3	1
3.2 Joining 1-1	3	1
3.3 Joining 2-1	3	1
3.4 Joining 2-2	4	1
3.5 Joining n-1	4	2
3.6 Joining n-2	5	1
3.7 Joining n-m	6	1
3.8 Clustering	7	2
3.9 Labeling of nodes	7	2
3.10 Implementation in d3	9	4
4. Testing	11	4
5. Evaluation	15	2
6. Dissertation submission	17	1
7. Make Poster	17	2

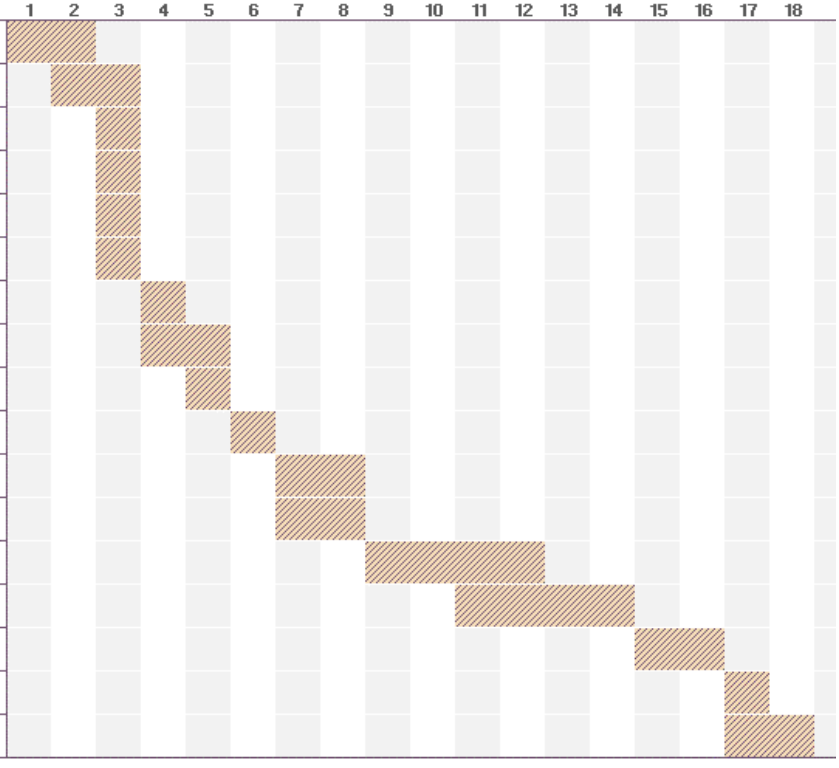


Figure 7 - Gantt Chart for project plan

Breakdown of each task:

1. Create the Circle Pack algorithm using d3.js libraries.
2. Create the Force-directed algorithm using d3.js libraries.
3. Start on the implantation of the new agglomerative method prototype in Python.
    3.1. Create a matrix to store the x, y coordinates and the radius of each circle in a group.
    3.2. Allow algorithm to join 2 circles together by using translate and rotation transform properties.
    3.3. Allow algorithm to join a circle to a group of 2 circles by using translate and rotation.
    3.4. Allow the algorithm to join a group of 2 circles to join another group of 2 circles using translate and rotation.
    3.5. Allow the algorithm to join a circle to a group of circles of size n using translate and rotation methods.
    3.6. Allow the algorithm to join a group of 2 circles to another group of circles of size n using translate and rotation methods.
    3.7. Allow the algorithm to join a group of circles of size n to another group of circles of size m using translate and rotation methods.
    3.8. Allow the algorithm to split up groups of circles no greater than 7 into clusters with a border around them.
    3.9. Give each circle an appropriate label that is contained within the size of the circle.
    3.10. Implement the algorithm using d3.js libraries.
4. Gather quantitative data by testing each of the algorithms by measuring the 6 closest neighbours and 6 most distant neighbours of a node to determine how accurate the algorithm is and measuring screen space used. Gather qualitative data by performing a user interface evaluation on students.
5. Evaluate and review finding and write up a report. Also, improve on other aspects of the dissertation.

6. Submit dissertation.
7. Design and make poster.

5.2 Chosen Software

The software chosen for this project were chosen based on previous use of the software, having a level of experience and if it was appropriate for the intended task. The following list of software's are what was decided for the project:

- Python [8] – used for implementing the new agglomerative clustering algorithm.
- NumPy [9] – Python library used for storing and manipulating matrices.
- Pandas [15]- Python library used for reading files into Python.
- Matplotlib.pyplot [10] – a 2D Python plotting library used for plotting the concept map.
- SciPy [16] - Python library used for creating the linkage table and dendrograms needed for showing affinity.
- Math - Python library used for mathematical functions.
- Random - Python library used for creating pseudo-random numbers.
- JavaScript – scripting language used for creating the visualisation for a webpage.
- D3.js [12] – JavaScript library that was used to visualise the different concept maps.
- HTML & CSS – used to show and style the visualisations on a webpage.
- Word – used for tying up all the documentation.
- Atom – the text editor used for the project.
- Google Drive – used to sync all project data between different computers.
- Excel – used for creating the data for the concept maps.
- Mendeley – used to store all references used in the project

6. Implementation

This chapter will detail the implementation stage of the project. This will include highlighting what different technology was used to complete the development as well as what approach was taken to implement the different algorithms. Problems that were encountered will also be discussed and how those problems were overcome.

6.1 Technology Used

The agglomerative method was implemented in Python making use of the different libraries available within Python including NumPy, Matplotlib and Pandas. Python was chosen due to haven previous experience and the broad variety of libraries available.

The implementation of the force-directed and circle pack algorithms were both completed using HTML, CSS and JavaScript. The visualisation of the concept maps was produced using the JavaScript Library D3.js. This library was chosen due to haven previous experience and it provided several different visualisation methods.

A full list of software used for this project can be found in section 6.2 – Chosen Software.

6.2 Approach to Implementing Agglomerative Algorithm

The approach that was taken for the implementation was to implement it incrementally. This approach meant new functionally was added with each version of the system, meaning if a version did not work correctly there were previous versions of the system to revert back to. When each new version was complete it was saved on Google Drive under a new version name (i.e. v0.2). This section will discuss the approach that was taking when implementing each of the

functionalities. A copy of the code for implementing the agglomerative cluster algorithm can be seen in Appendix 1 - Agglomerative clustering code.

6.2.1 Creating the initial functions

The first step that was taken for the implementation was creating the matrix in which the data will be stored and manipulated. For this, a two-dimensional list was created which stored the x and y coordinates, the radius of the circle and the label for each circle. An example of what the matrix looked like can be seen in Figure 8.

$$[x\ coordinate \quad y\ coordinate \quad radius \quad label] \begin{bmatrix} 0 & 0 & 9 & 8 \\ 12 & 0 & 3 & 12 \\ 18 & 0.014 & 3 & 7 \\ 15 & 14.7 & 12 & 18 \end{bmatrix}$$

Figure 8 - Initial matrix

During this version the mathplotlib.pyplot [10] library was also imported so that the visualisation could be displayed easily when testing the algorithm. PyPlot is a Matplotlib module which provides a MATLAB-like interface and allows simple graph plotting[https://matplotlib.org/index.html]. This allowed circles of variant radii to be easily plotted using an x and y coordinate onto as two-dimensional interface to be inspected. PyPlot also allowed labels and different colours to be added to each circle, which was good for displaying the names of the data sets and showing clustering easily.

6.2.2 Joining one to one

The task of adding the functionality of joining two circles together was fairly simple as all it involved was setting one of the circles to point 0,0 and then setting the second circles x coordinate to the radii of both the circles added together. A function was created for this which

would take the circle at point 0,0 and the radius of the second circle and return a two-dimensional list which contained the coordinates and radii for both the circle, so it can be displayed.

6.2.3 Joining one to two

For the task of joining one circle onto a pre-existing group of two circles the tangent between the two existing circles had to be found [17]. This is done by finding the two points of intersection between the two circles if the radius of the third circle was added onto their own radii as shown in Figure 9. This works by first finding the distance between the centres of the first two circles, which is d = $\|P_1 - P_0\|$ [18] as shown in Figure 10. This can be split this into two triangles $P_0P_2P_3$ and $P_1P_2P_3$ to create two Pythagoras equation $a^2 + h^2 = r_0^2$ and $b^2 + h^2 = r_1^2$. Next distance a, between P_0 and P_2, is calculated by using the equation $a = (r_0^2 - r_1^2 + d^2)/(2d)$. From this Pythagoras theorem is used to find h the final side in the triangle $P_0P_2P_3$ using the equation $h^2 = r_0^2 = a^2$. Finally, the coordinates of P_2 and P_3 can be calculated as the distances of r_0,a and h were needed first. To calculate the coordinates of P_2 the equation $P_2 = P_0 + a (P_1 - P_0) / d$ is used

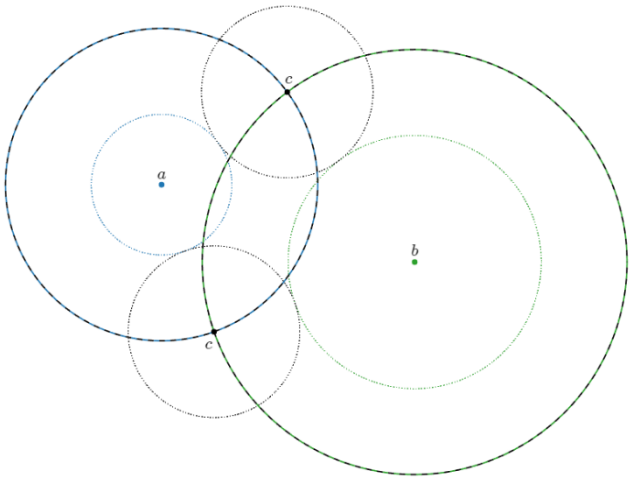


Figure 9 - Tangent of 2 Circles [17]

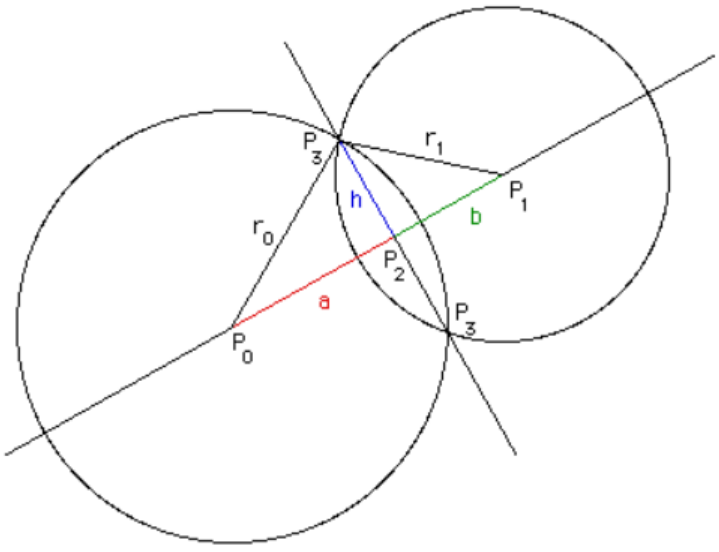


Figure 10 - Intersection of Circles [18]

where P_0 and P_1 are either the x or y values. Once P_2 is found P_3 can now be calculated by using the equations $x_3 = x_2 + h\,(y_1 - y_0)\,/\,d$ and $y_3 = y_2 - h\,(x_1 - x_0)\,/\,d$ where $P_3 = (x_3,y_3)$, $P_2 = (x_2,y_2)$ etc. Since the two intersection points are a mirror image of each other only one of the points had to be found. For joining one circle to two circles the process described above is repeated twice. During the second run-through, the two circles in the group are reversed so that circle 1 is now circle 2 and circle 2 is now circle 1 so that the other intersection point can be found. The implementation that has been created will always prefer the intersection that will take up the least screen space but for joining one to two this will not matter so the first intersection found will be chosen.

6.2.4 One to Many

Implementing the joining of one circle to a group of more than two uses the same algorithm for finding the intersections explained above in section 9.2.3, but the process is repeated for every pair in the group. After finding each intersection the algorithm will check whether the circle getting added is overlapping with any other circles in the group and if it is then the algorithm will skip that intersection and move onto the next one.

An example of the concept map looked like at this stage can be seen in Figure 11.

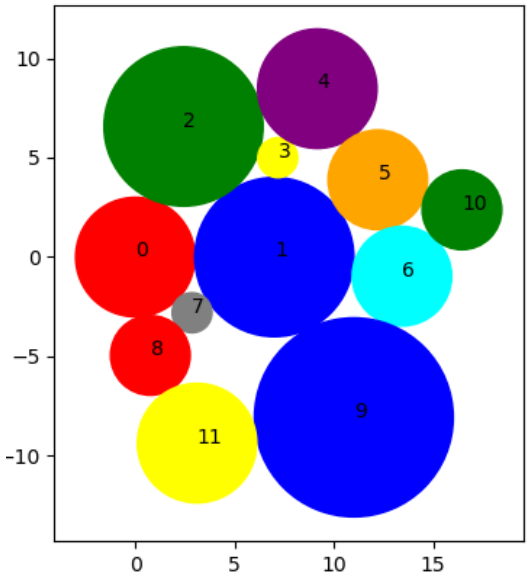


Figure 11 - Agglomerative Concept Map

6.2.5 Translation and Rotation multiplication

Adding the functionality to rotate and translate the groups of circles was needed to be able to combine the groups of circles together when joining many to many, which will be described in the next section (9.2.6).

Matrix multiplication was used for performing the rotation and translation on a group of circles. The NumPy [9] library was used for performing the rotations and translations as NumPy has the functions to perform matrix multiplication easily and efficiently. Matrix multiplication is a binary operation that produces a matrix from two other matrices. When doing matrix multiplication, the number of columns in the first matrix, which would be either the rotation or the translation matrix, must have the same, must be the same as the number of rows in the second matrix, which would be the groups of circles.

$$\begin{bmatrix} x\ coordinate \\ y\ coordinate \\ radius \\ label \\ homogeneous \end{bmatrix} \begin{bmatrix} 0 & 12 & 18 & 15 \\ 0 & 0 & 0.014 & 14.7 \\ 9 & 3 & 3 & 12 \\ 8 & 12 & 7 & 18 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 12 - Updated matrix

This meant that the way that the matrices storing the data in the implementation had to be changed as currently, they could have a different number of columns. This was fixed by making by rotating the matrices by 90 degrees so that the rows become the column and vice versa. While updating the matrix a homogeneous coordinate was added, as it is needed it perform matrix multiplication. An example of the updated matrix can be seen in Figure 12. Matrix multiplication works by multiplying each row in the first matrix by each column in the second matrix, see Figure 13.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} r & s & t \\ u & v & w \\ x & y & z \end{bmatrix} = \begin{bmatrix} ar+bu+cx & as+bv+cy & at+bw+cz \\ dr+eu+fx & ds+ev+fy & dt+ew+fz \\ gr+hu+ix & gs+hv+iy & gt+hw+iz \end{bmatrix}$$

Figure 13 - Matrix Multiplication

To use the rotation matrix correctly it must only change the x and y coordinate when performing the multiplication and not change the radius, label and homogeneous values.
The rotation matrix used can be shown in Figure 14. This rotation matrix will rotate the given values by θ degrees clockwise. The rotation matrix will rotate around point 0,0 so the group of circles must be moved to that position first. That is where translation is used, translation will move the circles proportionally to each other when given an x and y value, the matrix is shown in Figure 14.

$$rotation = \begin{bmatrix} cos\theta & sin\theta & 0 & 0 & 0 \\ -sin\theta & cos\theta & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad translation = \begin{bmatrix} 1 & 0 & 0 & 0 & x \\ 0 & 1 & 0 & 0 & y \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 14 - Rotation and Translation Matrices

The order of operation that the translation and rotation use is that it will first translate the group to point 0,0 then rotation will be used followed by another translation to return back to the starting location.

6.2.6 Joining many to many

Joining two groups of circles together was the most time consuming and difficult function of the algorithm to implement as it involved combining most of the previous functions together and working with greater about of geometry. Joining many to many worked by joining two pair together from different groups, this is done by first looping through every pair of circles in the first group of circles and the inside that loop, looping through every pair of circles in the second group of circles resulting in every combination of pairs connecting to pairs being found.

For each pair in the first group connected to each to each pair in the second group, there is a possible 8 different combination which the circles can be placed as shown in Figure 15. When each combination of pairs is found the second group uses translation and rotation, described in section 9.2.5, to align correctly with the first group then checks are performed to make sure that there are no overlapping circles between the two groups. If there were overlapping circles their combination of pairings will be ignored.

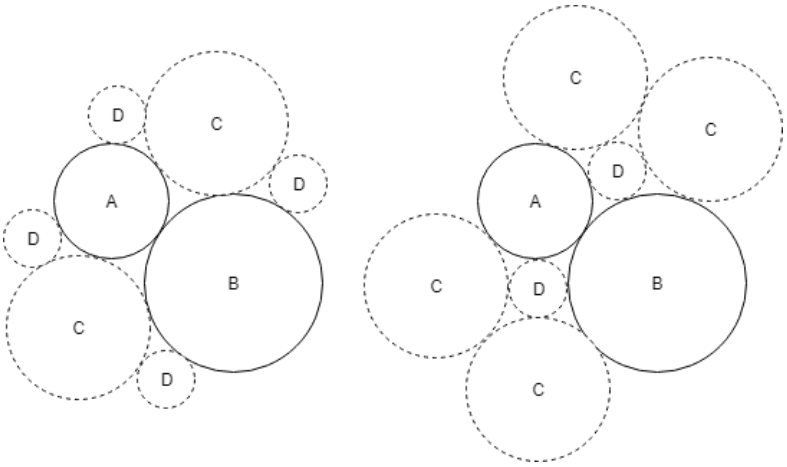


Figure 15 - Pairs connecting

The task of choosing what set of pairs to join together to use the least amount of screen space was done by first, for each combination of successful joining pairs finding the maximum distance between two circles in the group and comparing that against every other successful combination of pairs to find the group which had the minimum maximum distance.

6.2.8 Changing the aspect ratio

The aspect ratio is important as it allows the concept map to be displayed on screens of different sizes effectively. For this algorithm the aspect ratio cannot be defined but it can be swayed towards being at a certain aspect ratio. The aspect ratio of the concept map could be adjusted by changing where it chooses to join a circle, so if the aspect ratio was 16:9 and the concept map was becoming too tall it would try and place the new circle on the side of the concept map.

6.2.7 Using Imported Data

The functionality of being able to import data that that concept map could use was done at this stage. This involved importing a similarity matrix to be used to show affinity data and importing size information that will become the radius of the circles. As of this stage, all the data that was used for testing the implementation had just be created inside the program.

A CSV file was created which included similarity matrix which was randomly generated with numbers between 0.0001 and 0.9999, numbers closest to 0 the more related and numbers closer to 1 are the least related, this data was then imported into the implementation by using pandas [15], a Python library. The imported similarity matrix is then converted into a linkage matrix using SciPy [16] which shows what data is most closely related. For a more visual representation of this data a dendrogram can be produced using SciPy [16], this can be shown in Figure 16. The dendrogram is a tree structure at the bottom leaves of the tree are all the data points and each fork shows related data. Using the linkage matrix and all the previous functionality implemented a function was wrote to read through the linkage matrix and call the appropriate function to create the concept map.

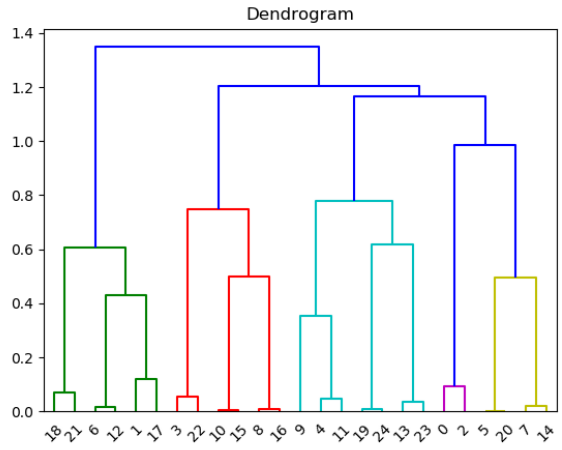


Figure 16 - Dendrogram

The linkage matrix also gives a similarity value, the lower the number the more similar the data is, this means clustering can be implemented by setting all the data points below a certain similarity value to be in the same cluster, for the implementation it was set to anything below 0.7 as this value consistently produced clusters no bigger than 6 or 7 circles. Clustering was

represented by having different colours for each cluster. The addition of clustering meant that the matrix that stored the data had to be updated to include a cluster value, this can be shown in Figure 17. The matrix for both the rotation and translation were also updated to work with the new matrix.

x coordinate	0	12	18	15
y coordinate	0	0	0.014	14.7
radius	9	3	3	12
label	8	12	7	18
cluster	3	3	3	3
homogeneous	1	1	1	1

Figure 17 - Final Matrix

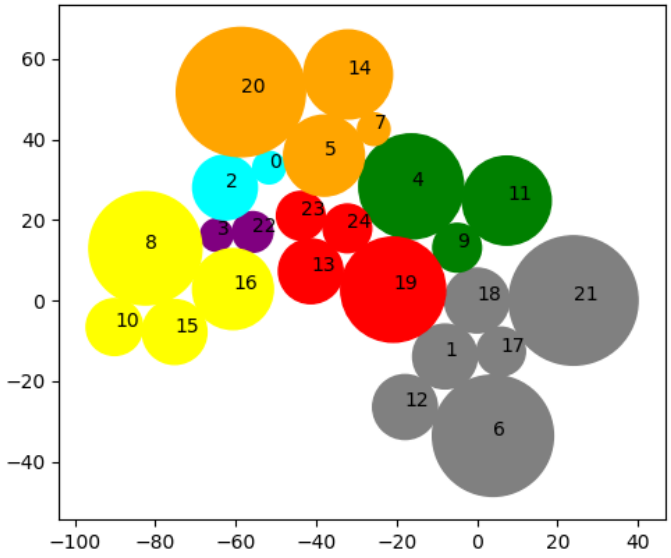


Figure 18 - Agglomerative Clustering in Python

To produce the size information needed for the circles, the Python library random [REFERNCE] was used to produce a pseudo-random list of numbers between 3 and 12 the same size as the number of circles in the similarity matrix which were stored in a text file that could be accessed by the algorithm. An example of a concept map generated with a similarity matrix and size information is shown in Figure 18, the number inside each of the circle is the label of the circle which can be changed to anything.

6.2.9 Exporting to D3.js

The final step for implementing the agglomerative clustering algorithm was to export the matrix storing the data to be used in D3.js [12]. The matrix was exported to a CSV file that was then accessed within d3.js and the concept map was then plotted, as shown in Figure 19. A label for each circle in the D3 version can be seen by hovering over a circle. This step was not

necessary to add functionality to the algorithm, but it was done so that the agglomerative algorithm could be fairly evaluated to the other algorithms that had been implemented in d3.js.

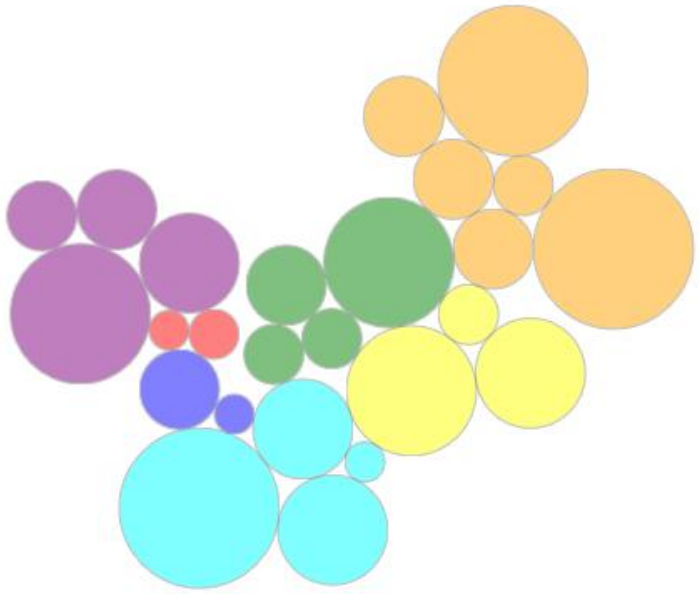


Figure 19 - Agglomerative Clustering in D3

6.3 Implementing Circle Pack Algorithm

The implantation of the circle pack algorithm completed by basing it on the zoomable circle packing algorithm created by Mike Bostock [14]. This was edited so that it could incorporate importing similarity data and size information. A python program was created to convert the similarity matrix and size information into a JSON file that the circle pack algorithm was able to read and convert into a concept map, an example can be shown in Figure 20. This visualisation allows the user to zoom in to certain parts of the concept map, this is good it allows labels for circles to only be displayed when zooming in on that particular circle meaning the overall concept map is not cluttered.

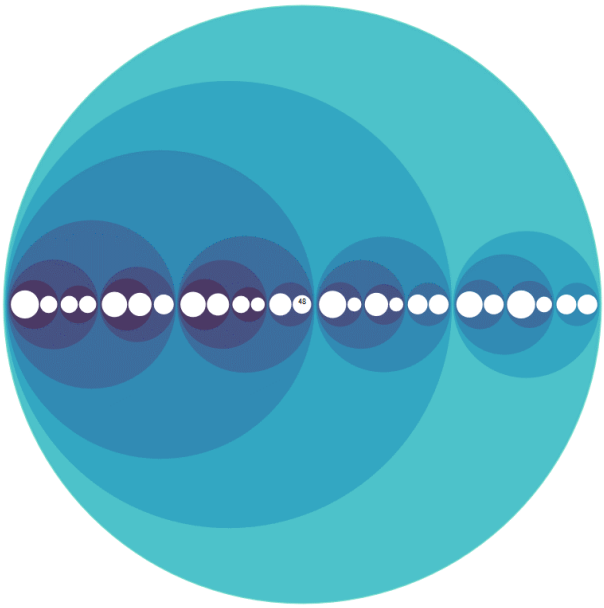


Figure 20 - Circle Packing Concept Map

6.4 Implementing Force-Directed Algorithm

When implementing the force directed concept map the Clustered Force Layout III [13] was used as a template to expand upon. This algorithm uses forces between the different circles and forces towards the centre of the display to create the layout of the concept map. The forces between the circles are created by using the linkage matrix, the more related circles will have a greater attracting force between them making them draw closer to each other. There is also a repulsion force between all the circle to keep them from becoming so close together that they are unable to more. To stop the circles from overlapping with each other there is a collide force that will stop any circles from becoming overlapped with each other. There are an *x* and *y* force that acts like a gravity applied to all of the circles to keep the circles close to the centre of the display area. The *x* and *y* force can also be used the give apply aspect ratios to the concept map, by setting the *x* force to a lower number than the *y* it will make the concept map wider. Each cluster is also defined as a different colour, the same as the agglomerative clustering algorithm. The force layout naturally is not a static layout and is constantly updating the

positions of the circles as the different forces interact with each other, but this was wanted as a static visualisation was need. To make the force layout static 2 seconds were given to allow the algorithm to work in the background before displaying the concept map in a static form. The example of the layout can be seen in Figure 21.

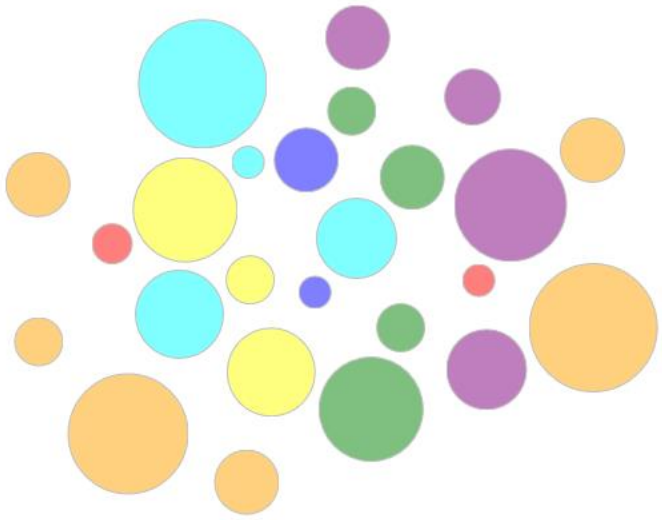


Figure 21 - Force Directed Concept Map

6.5 Implementing Force + Agglomerative Algorithm

When planning this project there was no intention of creating this algorithm but when creating the force directed algorithm it was discovered that it was possible to add a starting coordinate to the algorithm because of this a combination of the agglomerative and force-directed algorithms could be created. This new algorithm takes coordinates of the circles on the agglomerative and uses that as the starting locations for the circles in the force directed algorithm. The layout of this algorithm can be shown in Figure 22.

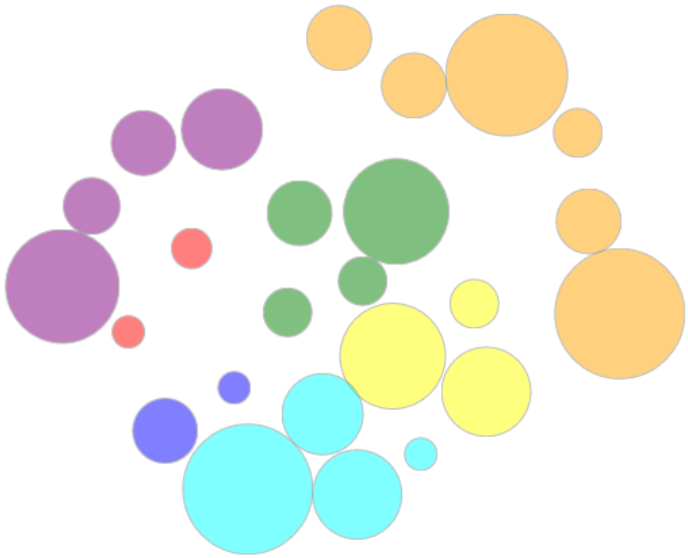


Figure 22 - Agglomerative + Force Concept Map

6.6 Problems Encountered

The biggest problem that was faced during the implantation was figuring out and implementing the calculations for manipulating geometry in 2D space. There was a lot of time spent on drawing how the algorithm should be manipulating the geometry and knowing what information the algorithm would need to be able to perform the manipulation. There would always be a mock-up built outside of the main build of each manipulation, to make sure it was implemented correctly before it was implemented into the final build which added a lot of time onto the development time.

6.7 Future Functionality

Some potential functionality that could be added to the agglomerative clustering algorithms would be the ability the check the mirror of a group when combing groups together as this could reduce the screen space because a better combination of pair could be found. Also, fully implementing the agglomerative clustering algorithm in JavaScript using D3.js in would be good as it would need it could be fully run within a browser.

7. Evaluation

In this chapter, there will be a description of the process of evaluating the algorithms created in this project and determine which algorithm displays the given data the most effectively. The algorithms that will be evaluated are:

- Circle Pack algorithm
- Force Directed algorithm
- Agglomerative algorithm
- Agglomerative + Force algorithm

The process of implementing these algorithms is explained in Chapter 9 – Implementation.
To measure which algorithm was most effectively measurements were collected from each of the visualisations of the algorithms. These measurements were:

- The use of screen space
- Aspect ratio
- Closest and distant neighbours

These measurements will be explained further in their relevant section below.

When evaluating each algorithm 9 data sets will be used to take the measurements from, these data sets will be made up of 3 data sets of size 10, 3 of size 25 and 3 of size 50. All the similarity data and sizes of the circles has been randomly generated. The data collected from this evaluation can be seen in Appendix 2 – Data Collected for Evaluation.

7.1 Use of Screen Space

To evaluate each algorithm's use of screen space, the total amount of space that the circles use will be compared against the overall required space of the visualisation, the greater the amount

of space the circles take up the better. This will be done by finding the total area of all the circles combined and the total area of the required display, then minus the total area of the circles from the total display area will give the amount of empty space in on the display. The total empty space is then divided by the number of circles in the concept map to give the average empty space per circle.

The results of testing each of the algorithms use of screen space can be seen in Figure 23. This show how the average empty space per circle from 9 data set is the agglomerative clustering algorithm with an average of 3043px^2 followed by the force + agglomerative with an average of 3567px^2. This result is not surprising as the way in which the force-directed algorithm works is that it the repulsion force between the circles will always be stronger than the attraction force meaning there will always be unnecessary space between the circles. The circle pack algorithm has a very high average of 8005px^2 as the intermediate and root nodes of the concept map take up an extremely high amount of screen real estate. The agglomerative clustering algorithm, on the other hand, makes great use of screen space as it tries to pack the circles as close together as possible and not leave gaps between the circles.

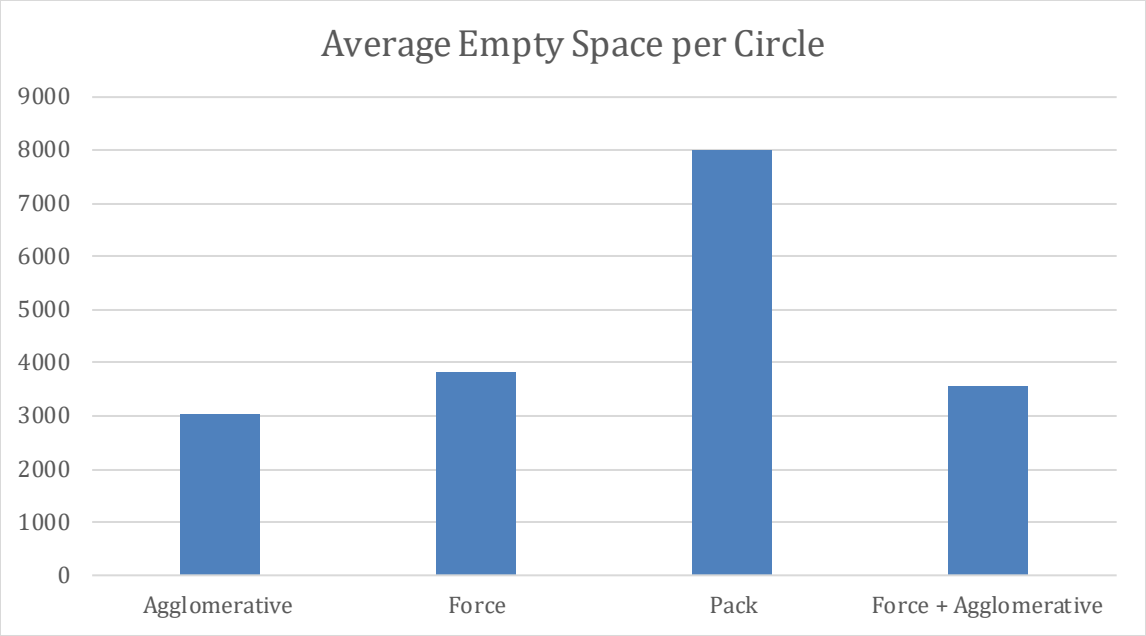


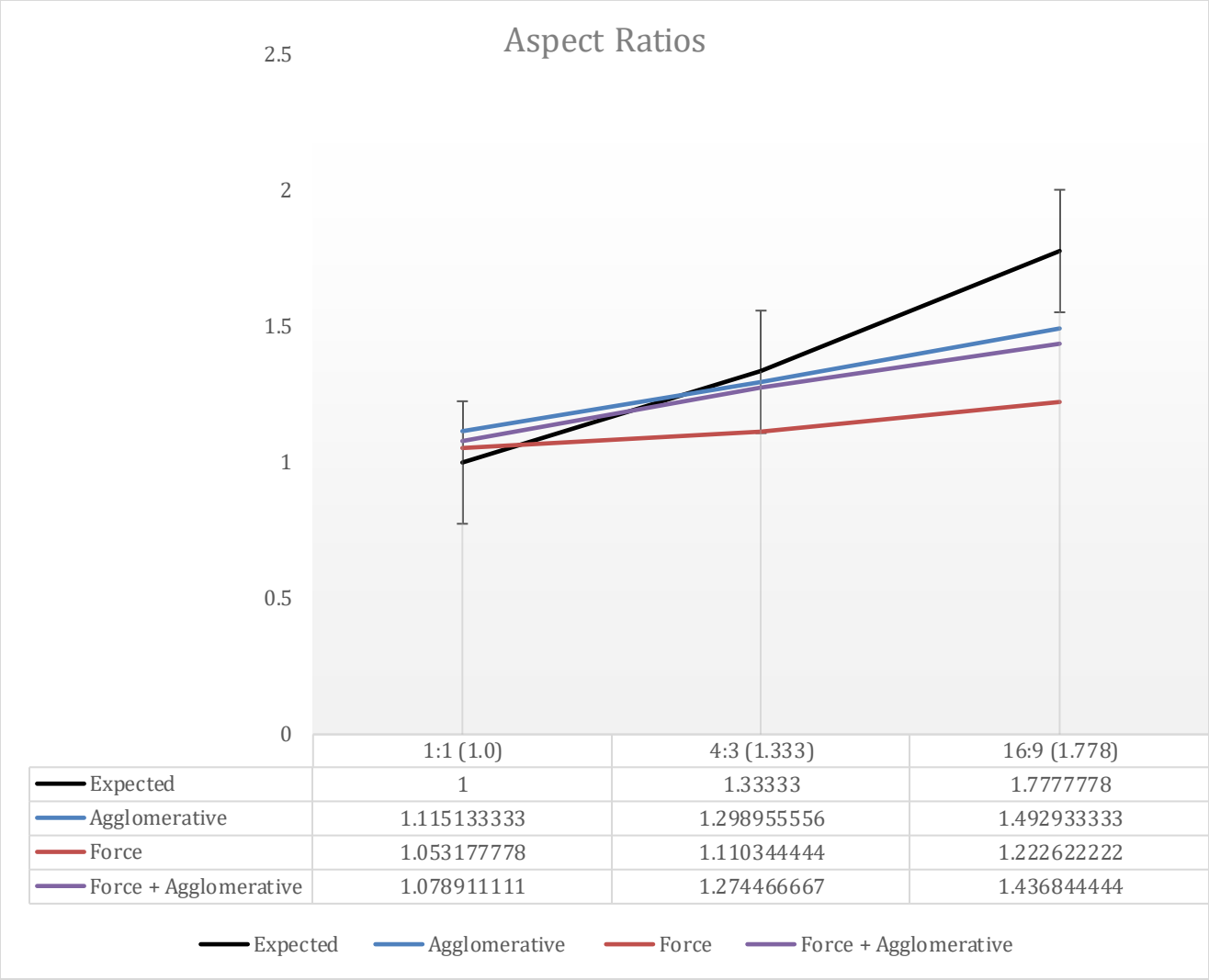
Figure 23 - Average Empty Space per Circle Graph

7.2 Aspect Ratio

To evaluate each algorithm's use of different aspect ratios the aspect ratio of the given visualisation will be compared to the requested aspect ratio to see how close the algorithm has kept to the requested aspect ratio. The aspect ratio is measured to determine if the outputted visualisation makes the best use of the aspect ratio of the screen it is being displayed on. For this evaluation 3 aspect ratios will be tested on the algorithms, these are 1:1, 4:3 and 16:9. As the circle pack algorithm can only be in an aspect ratio of 1:1 since all the data is stored inside one circle measurements cannot be taken for different aspect ratios, this will be taken into account when deciding of the most favourable algorithm after the evaluation is complete.

The results of performing the aspect ratio evaluation can be seen in Figure 24. The black line on the line chart shows the aspect ratio that was expected and the other 3 coloured inside show the 3 algorithms that allowed the aspect ratio to be changed. The line graph shows how for an aspect ratio of 1:1 the force-directed algorithm was the closest to the expected value, but the

other 2 algorithms are both inside the error bars. The force diagram produces the best aspect ratio 1:1 as the gravity forces going toward the centre of the screen are equal meaning a circle of circles naturally created. For an aspect ratio of 4:3 the agglomerative clustering produces the closest aspect ratio to the expected one with the force + agglomerative algorithm close behind and the force-directed algorithm producing a result outside the error bars. The force-directed graph is far off the target aspect ratio because the circles do not have enough time to settle in the correct place within the given 2 seconds. If the time was longer the aspect ratio would become more accurate. Even though the force + agglomerative algorithm uses the same forces as the force-directed algorithm it produces a better aspect ratio since it is taking the positions of the agglomerative clustering algorithm. For the 16:9 aspect ratio the all of the algorithms were out with the error bar of the expected aspect ratio, but the closest algorithm was the agglomerative clustering algorithm. There could be some improvements made in the future to the algorithms to help them to fit in a more widescreen format. This could be done for the agglomerative clustering algorithm by having the algorithm stop placing a circle in a certain place if it would harm the aspect ratio instead of just having it favour putting the circle in a certain place.



	1:1 (1.0)	4:3 (1.333)	16:9 (1.778)
Expected	1	1.33333	1.7777778
Agglomerative	1.115133333	1.298955556	1.492933333
Force	1.053177778	1.110344444	1.222622222
Force + Agglomerative	1.078911111	1.274466667	1.436844444

Figure 24 - Aspect Ratio Graph

7.3 Closest and Distant Neighbours

The measurement of closest and distant neighbours is used to determine how accurately the algorithm has displayed how similar or dissimilar the data is related to each other. This is done by looking at the similarity matrix for the data and selecting a random data point and looking at the closest related, which are the values on the matrix closest to 0, for this evaluation the 6 most closely related data will be found. These pairs will then be compared with their representation

on the visualisation to see if they are still shown as the most related data. The same process will be done for the most distant neighbours which are the values on the matrix closest to 1.

The result of this evaluation is that is that it shows the agglomerative clustering algorithm is the most effective algorithm in shown both close and distant neighbours as on average the agglomerative clustering algorithm get 5/6 of both the closest and distant neighbours correct when comparing with the similarity matrix. This is better than the agglomerative + force algorithm which only get an average of 4/6 for both close and distant neighbours.

7. Conclusion of Evaluation

After performing the whole evaluation, the conclusion has been met that the agglomerative clustering algorithm is the most effective algorithm for representing a concept map that shows both affinity data and size information. This is because the algorithm made the best use of screen space, it was able to represent aspect ratio the best, and it was able to show close and distant neighbours the most effectively. During this evaluation it was also found that the circle packing algorithm did not work as effectively as it was though before the implementation stage as since the all the algorithms that were created used a linkage table to show affinity data, this meant only two nodes are joined together at once, because of this the circle pack would not group the data effectively in terms of readability and screen space.

7.5 Reflecting on Project Requirements

In chapter 4 – Requirements Analysis, there was function and non-functional were stated for the project. To access whether the project was a success, it is important to look back at the requirement to see if they have been met. It is especially important that requirements that were prioritised 'must have' have been implemented.

The following table will list the requirements, a requirement with green beside it meant it has been implemented and requirements with red have not been implemented and orange means the requirement has been partly satisfied.

Requirement	Requirement Description	Priority	Was it Completed
FR-1	The algorithm must show both affinity data and size information simultaneously.	Must have	
FR-2	There must be no overlapping circles on the interface.	Must have	
FR-3	Each circle must have a label describing what the circle represents.	Must have	
FR-4	The algorithms should be able to run in a browser.	Should have	
FR-5	The prototype must be built in Python.	Must have	
FR-6	The final algorithm should be built using d3.js libraries.	Should have	
FR-7	The algorithm must split up the map into clearly defined clusters.	Must have	

FR-8	Closely related items (circles) are most close to each other and most distantly related items are furthest apart.	Should have.	
FR-9	The aspect ratio of the concept map must be able to be changed.	Must have.	
NFR-1	The algorithm must make good use of screen space	Must have	
NFR-2	The Layout of the interface should be readable and easy to understand	Must have	
NFR-2.1	Each cluster must have no more than 7 items (circles) inside it.	Must have	
NFR-2.2	Labels should be contained within each circle to make it easy to read	Should have	
NFR-3	The final algorithm can be able to run in all browsers	Can have	
NFR-4	The final algorithm show run within 5 seconds.	Can have	

Figure 25 - Refection on requirements

From Figure 25, requirements FR-6, NFR-2.2 and NFR-3 have not been implemented in this project. These requirements were not met due to a lack of time as the implementation of the agglomerative clustering algorithm took longer than expected to complete which push the rest of the development back.

Requirement FR-4 has only been partially completed as the 3 out of the 4 algorithms can be run within a browser, but the agglomerative clustering algorithm can only be displayed within the browser, but the algorithm is created within python. Requirement NFR-4 is also only partly

implement as the algorithm can run within 5 seconds if the number of circles is 25 or below but above 25 it will take longer than 5 seconds, this is because the time added for every new circle is exponential.

8. Conclusion

The overall goal of this project was to build and evaluate a new way of creating a concept map that shows both affinity data and size information. I can be concluded that this goal has been achieved as the agglomerative clustering algorithms has been implemented and been evaluated against other methods of displaying similar concept maps and it has been found that that the new algorithm implemented in this project is more effective than the methods that it was compared against.

The limitations of this project are that the final implementation was not created in JavaScript meaning that the algorithm cannot be run in a browser which is an import aspect of this project.

8.1 Reflecting on Project Aims & Objectives

The aims and objectives of the project were defined in chapter 2 – Aims & Objectives, this section will analyse each aim and objective to see if they have been completed.

8.1.1 Reflecting on Aims

Each aim will be analysis individually to see if they have been achieved:

Implement a new algorithm for displaying a concept map that shows both affinity data and size information simultaneously.

This aim was achieved as the new agglomerative clustering algorithm was created which shows both affinity data and size information the implantation of this is detailed in section 6.2 Approach to Implementing Agglomerative Algorithm.

Evaluate the new algorithm by comparing against other concept map algorithms that also show both affinity data and size information simultaneously.

This aim was achieved as an evaluation was conducted to compare 4 relevant algorithms against each other, this can be seen in chapter 7. Evaluation.

8.1.2 Reflecting on Objectives

Each object will be analysis individually to see if they have been completed:

Investigate the different ways in which affinity data and size information have already been shown in a concept map.

This objective was achieved as in section 3.1 Algorithms, where the force-directed algorithm and circle packing algorithms were both reached.

Investigate ways in which only affinity data can be shown in a concept map.

The hexagonal agglomerative clustering algorithm was investigated in section 3.1.3 Agglomerative Clustering Algorithm. This algorithm only showed affinity data, but the idea was used to create the new agglomerative clustering algorithm that was developed for this project

Investigate programming languages which would be able to implement a concept map.

Python and d3.js were both investigated in section 3.2 Implementation Technology's, both of these languages had the resources to be able to implement what was needed for this project, so they were both used to implement the different algorithms for this project.

Investigate ways other people have used to get quantitative data out concept map to evaluate the algorithm.

This objective was achieved in section 3.3.1 Quantitative Data, as one assessment method which someone else used to evaluate their concept map was to compare the quality of their concept map in terms of their use of screen space with other simple concept maps. This method was then used for the evaluation of this project.

Investigate ways in which other people have got qualitative data from a concept map algorithm to help evaluate.

This objective was achieved in section 3.3.2 Qualitative Data, as it was found that people conducted interviews with people and ask them to perform tasks using the concept map. Originally this project was going to conduct interviews on people to determine which algorithm they preferred but it was found not to be necessary as a quantitative evaluation was effective enough

Implement existing concept map algorithms that show both affinity data and size information.

The implement of the circle packing algorithm and force-directed algorithm are described in sections 6.3 Implementing Circle Pack Algorithm and 6.4 Implementing Force-Directed Algorithm.

Implement a new concept map algorithm that both shows affinity data and size information.

There were 2 new algorithms implemented for this project they were the new agglomerative clustering algorithm; the description of the implementation can be seen in section 6.2 Approach to Implementing Agglomerative Algorithm. The other new algorithm implemented is the combination algorithm of the force-directed algorithm and the agglomerative algorithm which is described in section 6.5 Implementing Force + Agglomerative Algorithm.

Evaluate each concept map algorithm by gathering quantitative data and then comparing each algorithm against each other to determine which algorithm is the most effective in displaying the data.

An evaluation of each of the algorithm that was implemented was carried out in chapter 7 – Evaluation. In this section, the algorithms are compared to decide what algorithm produces the most effective concept map. The concept map that was the most effective turned out the be the new agglomerative clustering algorithm that was implemented for this project.

9. Future Work

Some future work for this project would be to refine the agglomerative clustering algorithm by added mirror groups so more possibilities of combining groups can be found also allowing longer labels to be contained within a circle without it going over the edge would help make the concept map more readable. Also implementing the agglomerative clustering algorithm within JavaScript and using d3.js would be good ask it will allow the concept map to be run fully inside a web browser.

10. References

[1] M. Jacomy, T. Venturini, S. Heymann and M. Bastian, “ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software,” *PLoS ONE,* vol. 9, no. 6, p. e98679, 10 6 2014.

[2] T. M. J. Fruchterman and E. M. Reingold, “Graph Drawing by Force-directed Placement,” 1991.

[3] T. Dwyer, “Scalable, Versatile and Simple Constrained Graph Layout,” 2009.

[4] E. Meeks, D3.js in action.

[5] W. Wang, H. Wang, G. Dai and H. Wang, “Visualization of Large Hierarchical Data by Circle Packing,” 2006.

[6] P. Le Bras, D. A. Robb, T. S. Methven, S. Padilla and M. J. Chantler, “Improving User Confidence in Concept Maps,” in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*, New York, New York, USA, 2018.

[7] D. Kuhlman, “A Python Book: Beginning Python, Advanced Python, and Python Exercises,” 2009.

[8] “About Python™ | Python.org,” [Online]. Available: https://www.python.org/about/.

[9] “NumPy — NumPy,” [Online]. Available: http://www.numpy.org/.

[10] “pyplot — Matplotlib 2.0.2 documentation,” [Online]. Available: https://matplotlib.org/api/pyplot_api.html.

[11] “matplotlib.patches.Circle — Matplotlib 3.0.2 documentation,” [Online]. Available: https://matplotlib.org/api/_as_gen/matplotlib.patches.Circle.html#matplotlib.patches.Circle.

[12] M. Bostock, V. Ogievetsky and J. Heer, “D³ Data-Driven Documents,” *IEEE Transactions on Visualization and Computer Graphics,* vol. 17, no. 12, pp. 2301-2309, 12 2011.

[13] “Clustered Force Layout III - bl.ocks.org,” [Online]. Available: https://bl.ocks.org/mbostock/7881887.

[14] M. Bostock, “Zoomable Circle Packing - bl.ocks.org,” [Online]. Available: https://bl.ocks.org/mbostock/7607535.

[15] “Python Data Analysis Library — pandas: Python Data Analysis Library,” [Online]. Available: https://pandas.pydata.org/.

[16] “SciPy.org — SciPy.org,” [Online]. Available: https://www.scipy.org/.

[17] M. Bostock, “Tangent to Two Circles”.

[18] P. Bourke, “Circles and spheres,” 1992. [Online]. Available: http://paulbourke.net/geometry/circlesphere/.

11. Appendices

Appendix 1 - Agglomerative clustering code

```
##############################################
#Authors: Matthew Reilly
#16/04/2019
#What it does:
#agglomerative clustering algorithm

##############################################

#libaries
import matplotlib.pyplot as plt
import math
import numpy as np
from numpy.linalg import norm
import random
import pandas as pd
from functools import reduce
import scipy.cluster
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.spatial.distance import squareform
import json

#variales
PRECISION = 5
colour =
["red","purple","yellow","green","red","cyan","orange","gray","red","purple",
"yellow","green","red","cyan","orange","gray","red","purple","yellow","green"
,"red","cyan","orange","gray"]
cluster = 0

#define aspect ratio
aspectX = 16
aspectY = 9

#input the radii
f = open('radius.txt', 'r')
radius = f.read().split('\n')
f.close()

#import the linkage matrix
```

```
linkage_matrix = np.array(pd.read_csv("agglomerative/linkage.csv",header =
None))
linkage_matrix = np.delete(linkage_matrix, (0), axis=0)
cout=0

#create a new group
def new_group():
    group = []
    group.append([])
    group.append([])
    group.append([])
    group.append([])
    group.append([])
    group.append([])
    return group

#add circle to group of many
def Circle(group,x, y, radius, label):
    #0 = x coordinate
    #1 = y coordinate
    #2 = radius
    #3 = label
    #4 = cluster
    #5 = homogeneous coordinate
    group[0].append(x)
    group[1].append(y)
    group[2].append(radius)
    group[3].append(label)
    group[4].append(0)
    group[5].append(1)
    return group

#join one circle to another
def one_to_one(group,circle1, radius, label):
    Circle(group,circle1[2]+radius,0,radius, label)
    return group

#calculate where the circle interects with the group1
def circle_intersect(circle1, circle2,radius):


    X1, Y1 = circle1[0], circle1[1]
    X2, Y2 = circle2[0], circle2[1]
```

```
    R1, R2 = circle1[2]+radius, circle2[2]+radius

    Dx = X2-X1
    Dy = Y2-Y1
    D = round(math.sqrt(Dx**2 + Dy**2), PRECISION)
    if D > R1 + R2:
        return [0,0]
    elif D < math.fabs(R2 - R1):
        return [0,0]
    elif D == 0 and R1 == R2:
        return [0,0]
    else:
        a = (R1**2 - R2**2 + D**2)/(2*D)
        # distance from 1st circle's centre to the chord between intersects
        h = math.sqrt(R1**2 - a**2)
        midx = X1 + (a*Dx)/D
        midy = Y1 + (a*Dy)/D
        I1 = (round(midx + (h*Dy)/D, PRECISION),
              round(midy - (h*Dx)/D, PRECISION))

        #I2 = (round(midx - (h*Dy)/D, PRECISION),
        #      round(midy + (h*Dx)/D, PRECISION))


        return I1;

#check if there are any intersection when join one to many
def check_intersection(group,point,r):
    for x in range(len(group[0])):
        Dx = point[0]-group[0][x]
        Dy = point[1]-group[1][x]
        D = round(math.sqrt(Dx**2 + Dy**2), PRECISION)
        if D < r+group[2][x]:
            return bool(False)
    return bool(True)

#check if there are any intersection when join many to many
def check_intersection_many(group1,group2):
    for i in range(len(group1[0])):
        for j in range(len(group2[0])):
            Dx = group2[0][j]-group1[0][i]
            Dy = group2[1][j]-group1[1][i]
            D = round(math.sqrt(Dx**2 + Dy**2), PRECISION)
```

```
            if D < group1[2][i]+group2[2][j]:
                return bool(False)
    return bool(True)

#loop for all pairs
    #get the circle intersect
    #check for intersections
    #if no overlaps
        #get shortest distance between points
#pair with shortest distance between max join
#
#
#
def add_circle(group,r,label):
    minD = math.inf
    minI = 0
    minJ = 0
    for i in range(len(group[0])):
        for j in range(len(group[0])):
            if i != j:
                intersection = circle_intersect([row[i] for row in
group],[row[j] for row in group],r)
                if isinstance(intersection[0], float):
                    if check_intersection(group,intersection,r):
                        D = shortest_distance(group,intersection,r)
                        if(D<minD):
                            minD = D
                            minI = i
                            minJ = j
    intersection = circle_intersect([row[minI] for row in group],[row[minJ]
for row in group],r)
    #print(intersection)
    Circle(group,intersection[0],intersection[1],r, label)
    return group

#calulate the shortest distant many
def shortest_distance(group,point,r):
    maxD = 0
    for i in range(len(group[0])):
        for j in range(len(group[0])):
            if i != j:
                Dx = point[0]-group[0][i]*aspectY
                Dy = point[1]-group[1][i]*aspectX
```

```
                D = round(math.sqrt(Dx**2 + Dy**2), PRECISION)+r+group[2][i]
                if D>maxD:
                    maxD = D

                Dx = group[0][j]-group[0][i]*aspectY
                Dy = group[1][j]-group[1][i]*aspectX
                D = round(math.sqrt(Dx**2 + Dy**2),
PRECISION)+group[2][i]+group[2][i]
                if D>maxD:
                    maxD = D

    return maxD

#calulate the shortest distant many
def shortest_distance_many(group1,group2):
    group = np.concatenate((group1,group2),1)
    maxD = 0
    for i in range(len(group[0])):
        for j in range(len(group[0])):
            if i != j:

                Dx = group[0][j]-group[0][i]*aspectY
                Dy = group[1][j]-group[1][i]*aspectX
                D = round(math.sqrt(Dx**2 + Dy**2),
PRECISION)+group[2][i]+group[2][i]
                if D>maxD:
                    maxD = D
    return maxD
#rotate group
def rotate(group, radians):
    theta = np.radians(radians)
    c,s = np.cos(theta), np.sin(theta)
    rMatrix = [[c,s,0,0,0,0],[-
s,c,0,0,0,0],[0,0,1,0,0,0],[0,0,0,1,0,0],[0,0,0,0,1,0],[0,0,0,0,0,1]]

    group = np.dot(rMatrix,group)
    return group

#calulate the rotation angle needed to rotate group to correct place
def get_rotation_angle(p0,p1,p2):

    d = np.arctan2(p1[1],p1[0])
    e =  np.arctan2(p2[1],p2[0])
```

```
    angle = d-e
    return (180/np.pi)*angle
#translate a group to a new coordinate
def translate(group,x,y):
    tMatrix
=[[1,0,0,0,0,x],[0,1,0,0,0,y],[0,0,1,0,0,0],[0,0,0,1,0,0],[0,0,0,0,1,0],[0,0,
0,0,0,1]]
    group = np.dot(tMatrix,group)

    return group

#for each pair in group1
#   for each circle in group2
#       get circle intersect
#       rotate and translate
#       check for intersections
#       if no overlaps
#           get shortest distance between points
#pair with shortest distance between max join
#
def manyToMany(group1,group2):
    minDistance = math.inf
    minGroup = []

    for a in range(len(group1[0])):
        for b in range(len(group1[0])):
            if a != b:
                for c in range(len(group2[0])):
                    for d in range(len(group2[0])):
                        if c != d:
                            intersectionC = circle_intersect([row[a] for row
in group1],[row[b] for row in group1],group2[2][c])
                            if isinstance(intersectionC[0], float):
                                if
check_intersection(group1,intersectionC,group2[2][c]):
                                    intersectionD = circle_intersect([row[a]
for row in
group1],[intersectionC[0],intersectionC[1],group2[2][c],1],group2[2][d])
                                    if isinstance(intersectionD[0], float):
                                        if
check_intersection(group1,intersectionD,group2[2][d]):
                                            x = group2[0][c]
```

```
                                            y = group2[1][c]
                                            x1 = intersectionC[0]
                                            y1 = intersectionC[1]

                                            group3 = translate(group2,-x,-y)
                                            rotation_angle =
get_rotation_angle((group3[0][c],group3[1][c]),(group3[0][d],group3[1][d]),(i
ntersectionD[0]-x1,intersectionD[1]-y1))
                                            group3 =
rotate(group3,rotation_angle)
                                            group3 =
translate(group3,intersectionC[0],intersectionC[1])

                                            if
check_intersection_many(group1,group3):
                                                reverse = 0
                                                D =
shortest_distance_many(group1,group3)
                                                if D < 0:
                                                    D = D*-1
                                                    reverse = 1
                                                if(D<minDistance):
                                                    minDistance = D
                                                    minGroup = group3


    group3 = np.concatenate((group1,minGroup),1)

    return group3
#print concept map using pyplot
def printGroup(group):

    #print graph
    ax = plt.gca()
    for x in range(len(group[0])):
        ax.add_patch(plt.Circle((group[0][x], group[1][x]), group[2][x],
color=colour[int(group[4][x])]))
        plt.annotate(int(group[3][x]), (group[0][x],group[1][x]))



#Main loop what uses the linkage matrix to create the concept map
groups = []
```

```
#for every connection in linkage matrix
for x in range(len(linkage_matrix)):
    # if both a leaves then join one to one
    if(linkage_matrix[x][0] < len(linkage_matrix)+1 and linkage_matrix[x][1]
< len(linkage_matrix)+1):
        circles = new_group()
        circles = Circle(circles,0,0,int(radius[cout]), linkage_matrix[x][0])
        cout = cout+1
        circles = one_to_one(circles,[row[0] for row in
circles],int(radius[cout]), linkage_matrix[x][1])
        cout = cout+1
    #if one is a group and then join one to many
    elif(linkage_matrix[x][0] < len(linkage_matrix)+1 and
linkage_matrix[x][1] >= len(linkage_matrix)+1):
        circles = groups[int(linkage_matrix[x][1]-(len(linkage_matrix)+1))]

        circles = add_circle(circles, int(radius[cout]),
linkage_matrix[x][0])
        cout = cout+1
    #if one is a group and then join one to many
    elif(linkage_matrix[x][0] >= len(linkage_matrix)+1 and
linkage_matrix[x][1] < len(linkage_matrix)+1):
        circles = groups[int(linkage_matrix[x][0]-(len(linkage_matrix)+1))]

        circles = add_circle(circles, int(radius[cout]),
linkage_matrix[x][1])
        cout = cout+1
    #if both are groups then join many to many
    elif(linkage_matrix[x][0] >= len(linkage_matrix)+1 and
linkage_matrix[x][1] >= len(linkage_matrix)+1):


        group1 = groups[int(linkage_matrix[x][0]-(len(linkage_matrix)+1))]
        group2 = groups[int(linkage_matrix[x][1]-(len(linkage_matrix)+1))]
        #if the linkage matrix is above 0.7 similarity then make that a
cluster
        if(linkage_matrix[x][2] > 0.7):
            found1 = 0
            found2 = 0
            for y in range(len(group1[0])):
                if group1[4][y] == 0:
                    if(found1 == 0):
                        cluster = cluster + 1
```

```
                        found1 = 1
                    group1[4][y] = cluster

            for z in range(len(group2[0])):
                if (group2[4][z] == 0):
                    if(found2 == 0):
                        cluster = cluster + 1
                        found2 = 1
                    group2[4][z] = cluster


        circles = manyToMany(group1,group2)


    groups.append(circles)



printGroup(groups[len(groups)-1])



a = []
#format for output
for i in range(len(groups[len(groups)-1][1])):
    a.append(i)
np.set_printoptions(suppress=True)
groups[len(groups)-1] = np.insert(groups[len(groups)-1], 0, a, axis=0)



#output to files to use in d3.js
a = np.asarray(groups[len(groups)-1])
np.savetxt("agglomerative/array.csv", a, delimiter=",",fmt='%f')
np.savetxt("force/array.csv", a, delimiter=",",fmt='%f')
np.savetxt("agglomerative+force/array.csv", a, delimiter=",",fmt='%f')
plt.axis('scaled')
plt.show();
```

Appendix 2 – Data Collected for Evaluation

Test Data 1 - 15 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	1.2444	**1.2784**	1.0256	54371	99840	**45469**	**3031.266667**
Force	**0.9505**	0.9567	0.9699	54371	116864	62493	4166.2
Pack	N/A	N/A	N/A	694000	883600	189600	12640
Force + Agglomerative	1.3277	1.2539	**1.4212**	54371	105958	51587	3439.133333
Test Data 2 - 15 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	1.14	1.26	**1.32**	60424	101482	**41058**	**2737.2**
Force	**0.91**	1.04	1.05	60424	127120	66696	4446.4
Pack	N/A	N/A	N/A	694000	883600	189600	12640
Force + Agglomerative	1.25	**1.28**	1.24	60424	112846	52422	3494.8
Test Data 3 - 15 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	1.09	**1.38**	1.32	51734	91840	**40106**	**2673.733333**
Force	**1.02**	1.21	1.23	51734	101450	49716	3314.4
Pack	N/A	N/A	N/A	694000	883600	189600	12640
Force + Agglomerative	1.06	1.28	**1.37**	51734	99542	47808	3187.2
Average 25 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	1.158133	**1.306133**	1.221867	55509.6667	97720.6667	**42211**	**2814.066667**
Force	**0.960167**	1.0689	1.0833	55509.6667	115144.667	59635	3975.666667
Pack	#DIV/0!	#DIV/0!	#DIV/0!	694000	883600	189600	12640
Force + Agglomerative	1.212567	1.2713	**1.343733**	55509.6667	106115.333	50605.66667	3373.711111
Test Data 4 - 25 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	1.35	1.2422	1.334	72269.17	160530	88260.83	3530.4332
Force	1.1515	1.2164	1.3656	72269.17	160799	88529.83	3541.1932
Pack	N/A	N/A	N/A	694000	883600	189600	7584
Force + Agglomerative	**1.12**	**1.2784**	**1.3741**	72269.17	149473	**77203.83**	**3088.1532**
Test Data 5 - 25 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle

Agglomerative	**1.12**	**1.32**	**1.54**	68472.74	140582	**72109.26**	**2884.3704**
Force	1.32	1.15	1.37	68472.74	157890	89417.26	3576.6904
Pack	N/A	N/A	N/A	694000	883600	189600	7584
Force + Agglomerative	1.21	1.31	1.46	68472.74	148612	80139.26	3205.5704
Test Data 6 - 25 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	**1.08**	**1.29**	**1.43**	79456.62	178420	**98963.38**	**3958.5352**
Force	1.24	1.17	1.34	79456.62	181244	101787.38	4071.4952
Pack	N/A	N/A	N/A	694000	883600	189600	7584
Force + Agglomerative	0.91	1.25	1.37	79456.62	180482	101025.38	4041.0152
Average 25 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	1.183333	**1.284067**	**1.434667**	73399.51	159844	86444.49	3457.7796
Force	1.237167	1.1788	1.358533	73399.51	166644.333	93244.82333	3729.792933
Pack	#DIV/0!	#DIV/0!	#DIV/0!	694000	883600	189600	7584
Force + Agglomerative	**1.08**	1.279467	1.401367	73399.51	159522.333	**86122.82333**	**3444.912933**
Test Data 7 - 50 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	0.8118	**1.35**	**1.8868**	128450.33	272460	**144009.67**	**2880.1934**
Force	**0.8466**	0.91	0.9381	128450.33	318150	189699.67	3793.9934
Pack	N/A	N/A	N/A	694000	883600	189600	3792
Force + Agglomerative	0.7825	1.1679	1.3363	128450.33	374320	245869.67	4917.3934
Test Data 8 - 50 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	**0.96**	1.29	**1.84**	102604.67	248604	**145999.33**	**2919.9866**
Force	1.12	1.17	1.42	102604.67	295614	193009.33	3860.1866
Pack	N/A	N/A	N/A	694000	883600	189600	3792
Force + Agglomerative	0.91	**1.36**	1.54	102604.67	286540	183935.33	3678.7066
Test Data 9 - 50 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	1.24	1.28	1.74	118946.45	257418	**138471.55**	**2769.431**
Force	**0.92**	1.17	1.32	118946.45	294230	175283.55	3505.671
Pack	N/A	N/A	N/A	694000	883600	189600	3792

Force + Agglomerative	1.14	**1.29**	**1.82**	118946.45	271520	152573.55	3051.471
Average 50 circles	1:1	4:3	16:9	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	**1.003933**	**1.306667**	**1.822267**	116667.15	259494	**142826.85**	**2856.537**
Force	0.9622	1.083333	1.226033	116667.15	302664.667	185997.5167	3719.950333
Pack	#DIV/0!	#DIV/0!	#DIV/0!	694000	883600	189600	3792
Force + Agglomerative	0.944167	1.272633	1.565433	116667.15	310793.333	194126.1833	3882.523667
Average circles	1:1 (1.0)	4:3 (1.333)	16:9 (1.778)	Circle Space	Total Space	Empty Space	Empty Space per circle
Agglomerative	1.115133	**1.298956**	**1.492933**	81858.7756	172352.889	**90494.11333**	**3042.794422**
Force	**1.053178**	1.110344	1.222622	81858.7756	194817.889	112959.1133	3808.469978
Pack	#DIV/0!	#DIV/0!	#DIV/0!	694000	883600	189600	8005.333333
Force + Agglomerative	1.078911	1.274467	1.436844	81858.7756	192143.667	110284.8911	3567.049237