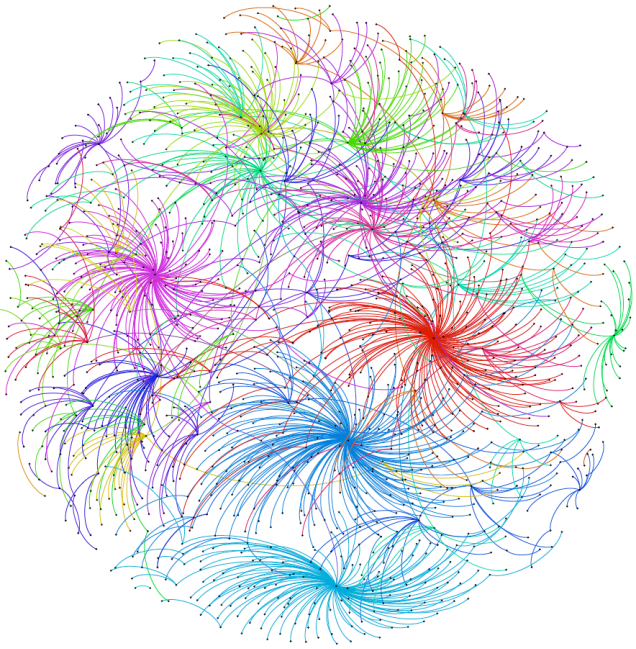
A NETWORK OF MATHEMATICAL THEOREMS

Avni Chotai, Christiana Mavroyiakoumou, Ioannis Papadopoulos, Xuening Wen

Supervised by Dr Nick Jones
Department of Mathematics
Imperial College London

June 17, 2015



Graph produced in *Gephi*

Abstract

We explore a network consisting of mathematical theorems as nodes and assume that two theorems share an edge if and only if they appear in the same publication. We use MATLAB to find potential bugs in our data and 'clean' our data set by tracking duplicate nodes, odd spellings, theorems with multiple names and other noise. Having mostly debugged the network, we show how to find surprising connections between mathematical theorems using network properties and give examples of our results. Furthermore, we use network analysis techniques such as community detection, spanning trees, betweenness centralities, and diameter of the network to find the most important theorems in our network.

Acknowledgements

We would like to thank our supervisor Dr Nick Jones for all his input and guidance in our project.

Contents

1. Introduction

The nodes in our undirected network are mathematical theorems and the edges are connections between mathematical theorems that have appeared in the same publication. The weight of the edge between node i and node j represents the number of publications in which theorem i and theorem j have appeared together. The initial data set consists of 1865 nodes and 24625 edges. Let us call the search algorithm used to collect our data set *Search Engine Algorithm*.

The aims of our project can be divided into three broad categories: debugging the given network, performing network analysis on a 'clean' network, and finding connections between seemingly unrelated theorems.

Debugging the given network consists of extracting the more essential parts of our original network by discarding the nodes and edges that carry non-meaningful data. Doing this had the added benefit of reducing our network size, making it simpler to work with. Let us call the irrelevant nodes and edges 'bugs'. Prior to debugging our network, we aim to understand the network given to us. This motivates sections 2–5 (inclusive) in our report.

In the third section of our report, we analyse the network on a level by exploring the connection between a node's degree and strength. We use this analysis to discover nodes that are peculiar, in that they have an abnormally high strength relative to their degree.

In the following two sections, we attempt to learn about the network by considering the number of publications in which a given pair of theorems appears, and how this value is significant to these two theorems. We do this by examining the weights of the edges in our network on a more local level, and subsequently calculating the mean squared errors between theorems. During this analysis, we find theorems that share similar connections as each other.

In sections 6 and 7 of our report, we look more closely at the data set itself, as opposed to the properties exhibited by the data set. We compare theorem names, searching for theorems that have been repeated for various reasons and theorems that are miscounted as a result of how *Search Engine Algorithm* works. Furthermore, we come across nodes that are not in fact theorems.

Sections 8–11 (inclusive) involve network analysis on our 'clean' network to help us find the most significant nodes in our network.

In section 9, we discuss the surprising theorems that we have come across whilst working on this project.

2. Preliminaries

Network theory is a subcategory of graph theory and so terminology from graph theory can be adapted to describe networks [New03].

A *network* is a *graph*, $G = (V, E)$, that consists of a set of nodes, V, and a set of edges, E.

A *node* (also called *vertex*), $v \in V$ represents the element under study in the network. An *edge* $e \in E$ is a line connecting two nodes.

A *directed* network consists of edges that have a specific direction, that is, it consists of edges of the form

$$E \subset V \times V = \{(v, w) \,|\, v, w \in V\}.$$

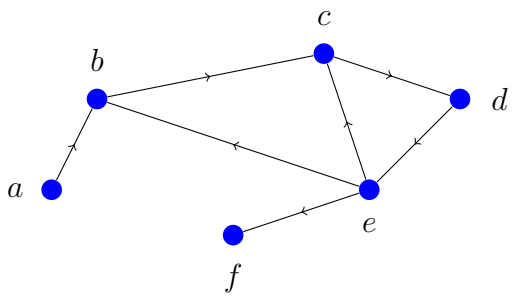


FIGURE 1: A directed graph, G

For example, Figure 1 represents the directed graph G, with vertex set $V(G) = \{a, b, c, d, e, f\}$ and edge set $E(G) = \{ab, bc, cd, de, ec, eb, ef\}$.

An *undirected* network consists of edges that run in both directions:

$$E \subset V^{(2)} = \{\{v, w\} \,|\, v, w \in V\}.$$

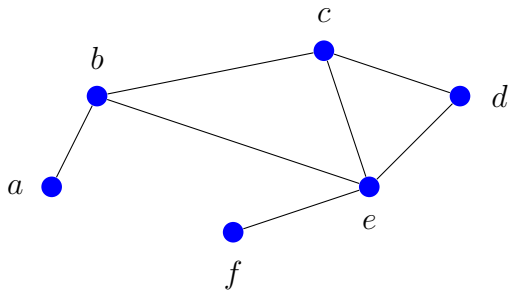


FIGURE 2: An undirected graph, G

The *weight* of an edge, $w(e)$, is the number associated with each edge. In our case, it is a function $w : E \to \mathbb{N}$. A *weighted* network has edges that carry a weight.

The *degree* of a node (also called *node order*) is the number of edges connected to the node. A directed network has both an *in-degree* and an *out-degree*. The *strength* of a node is the sum of the weights of all the edges connected to the node.

The *shortest path* between node v and node w is the path that minimises the sum of the weights of the edges that need to be traversed in order to go from v to w.

3. Degree and Strength

In this section, we explore the connection between a node's strength and degree. In particular, we are interested in nodes with high strength and low degree.

We begin by noting that the nodes with the highest degrees tend to have the highest strengths, thus dominating the network. In fact, the "strength [of a node well-connected to the rest of the network] tends to grow superlinearly with the degree." [SBV09].

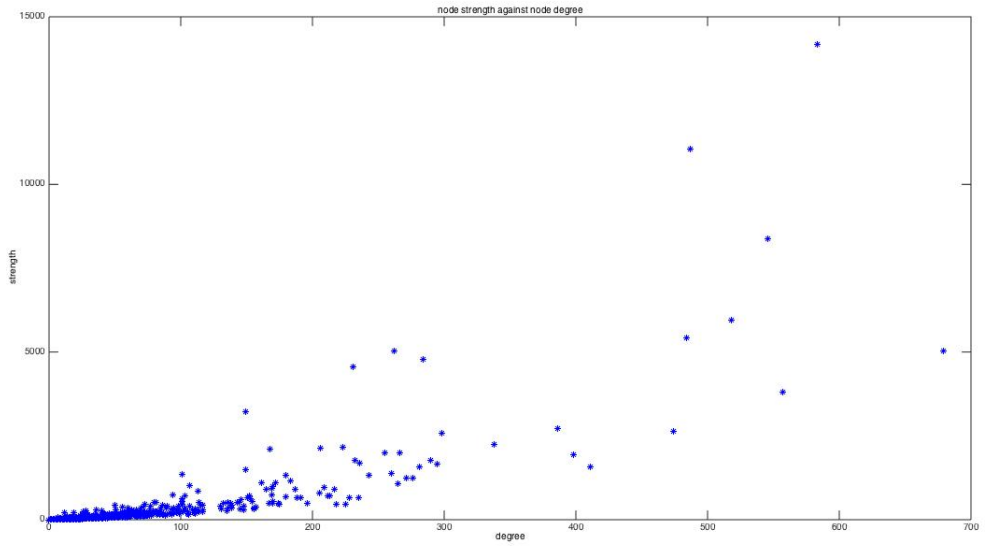


FIGURE 3: Node strength against node degree

However, the converse is not always true. That is, there exist many nodes with high strength and low degree. This is the case when one or more of the few edges connected to the node carry a large weight.

We are interested in nodes that have these properties because they correspond to mathematical theorems that do not appear in publications with many other theorems (low degree) but still have high strength, meaning that the theorem has appeared in many publications with the theorems to which it is connected.

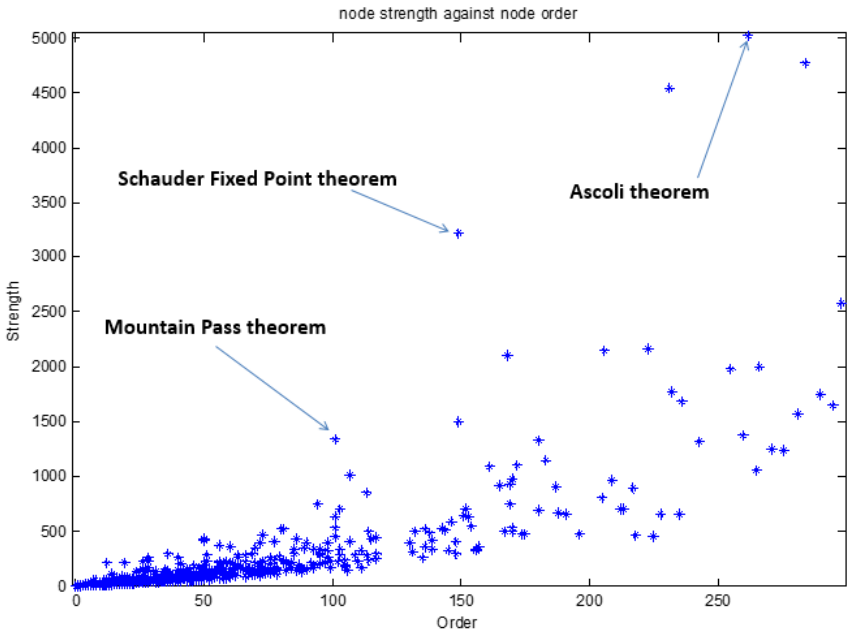


FIGURE 4: Node strength against node degree

We can see that there are some anomalous nodes (those with low degree and relatively high strength); a few examples are the *Mountain Pass theorem*, *Schauder fixed point theorem* and the *Ascoli theorem*.

4. Normalised Weights

In this section, we will discuss the importance of normalising the weights of each edge in our network. We will also explain the method we used to normalise weights.

To allow us to compare connections throughout the network, it is helpful to study the network on a more local level and allow each node to individually determine the importance of its incident edges. We do this by normalising the weights of each edge using the method outlined in [SBV09]: given a specific node, we divide the weights of its incident edges by the total node strength. Observe that this method results in two potentially different normalised weights for each edge in the network. These different normalised weights arise because we consider each edge twice. To illustrate this, we consider the edge (with weight w_{ij}) connecting node i and node j.

The *normalised weights* of an edge are calculated by

$$P_{ij} = \frac{w_{ij}}{s_i}$$

where w_{ij} is the weight of the edge connecting nodes i and j and s_i is the strength of node i.

We thus form a non-symmetric matrix P of normalised weights. This leads on to calculating mean squared errors.

5. Mean Squared Errors

In this section, we will analyse the mean squared errors between pairs of theorems.

We calculate the *mean squared error* between node i and node j in four steps:

1. We subtract the j^{th} row of P from the i^{th} row of P.
2. Next, we square each entry in the resulting row vector.
3. We then sum the elements of the resulting row.
4. Finally we divide this number by the total number of nodes.

We repeat this process for all pairs of rows in P, storing the mean squared errors in a 1865×1865 matrix $MSEs$, where the ij^{th} entry contains the mean squared error between node i and node j.

We observe that the matrix $MSEs$ is symmetric, as expected; the mean squared error between theorem i and theorem j is equal to the mean squared error between theorem j and theorem i. We additionally observe that this method results in the calculation of mean squared errors

between theorems that are not in fact connected. We thus eliminate the ij entries pertaining to pairs of unconnected theorems.

We analyse our network by finding pairs of connected theorems that have particularly large or small mean squared errors between them. To illustrate this, we use the example of *Green's theorem* (Theorem ID 18, Node 18).

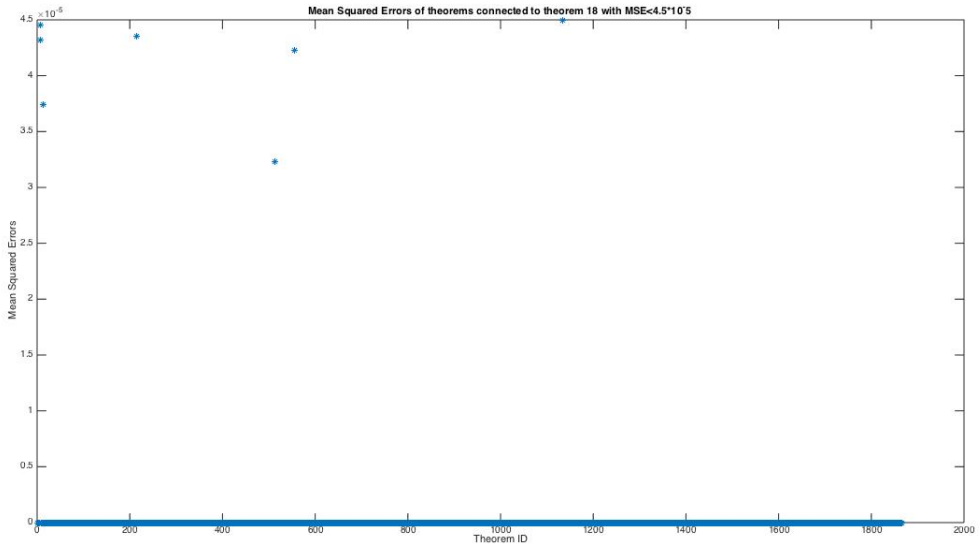


FIGURE 5: Mean squared errors of theorems connected to theorem 18 with $MSE \leq 4.5 \times 10^{-5}$

Figure 5 shows the theorems connected to *Green's theorem* that have a mean squared error of less than or equal to 4.5×10^{-5} with *Green's theorem*.

Theorem ID	**Theorem Name**	**Mean Squared Error with Green's theorem**
7	h theorem	0.000045
8	divergence theorem	0.000043
14	Stokes' theorem	0.000037
214	Liouville theorem	0.000044
513	Goursat theorem	0.000032
555	Alexandrov theorem	0.000042

Green's theorem is a special case of *Stokes' theorem* and the *divergence theorem*, so it is expected that *Green's theorem* is closely connected to these. It is also unsurprising that *Green's theorem* is related to the *Goursat theorem* as *Green's theorem* is used in the proof of the *Goursat theorem*. However, we do find it surprising that the *h theorem* and *Alexandrov theorem* are related to *Green's theorem*, because the *h theorem* appears in classical statistical mechanics and the *Alexandrov theorem* appears in analysis and algebra.

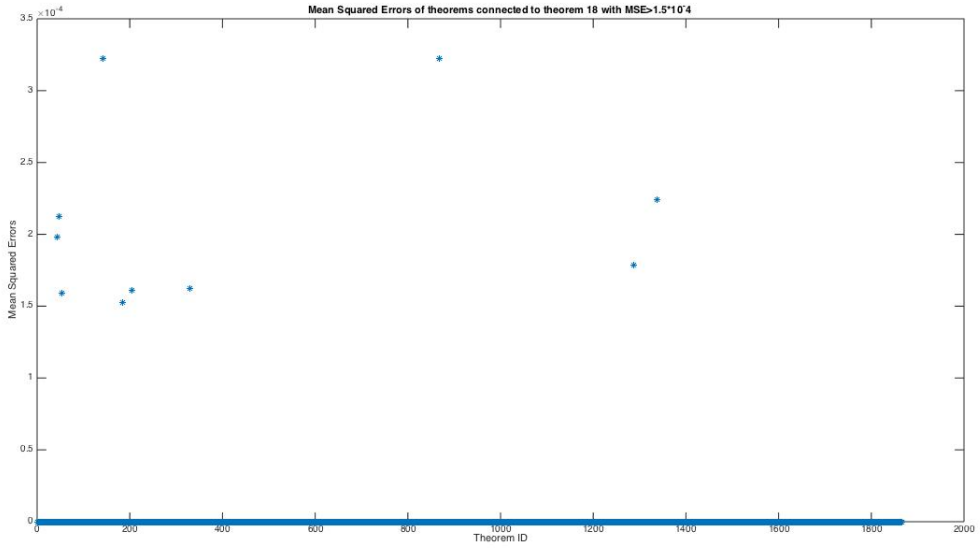


FIGURE 6: Mean squared errors of theorems connected to theorem 18 with $MSE \geq 1.5 \times 10^{-4}$

Figure 6 shows the theorems connected to *Green's theorem* that have a mean squared error of greater than or equal to 1.5×10^{-4} with *Green's theorem*.

Theorem ID	**Theorem Name**	**Mean Squared Error with Green's theorem**
43	Bonnet theorem	0.000198
48	Gauss Bonnet theorem	0.000212
54	Banach fixed point theorem	0.000159
143	Helmholtz theorem	0.000322
185	Lefschetz theorem	0.000152
204	Reynolds transport theorem	0.000161
329	Gauss Green theorem	0.000163
868	Helmholtz theorem	0.000322
1288	Rashevsky chow theorem	0.000179
1338	Gelfond theorem	0.000224

We note that *Helmholtz theorem* appears twice in this list, as theorem 143 and theorem 868. Theorem 143 and theorem 868 have the same mean squared error with *Green's theorem*. This is expected as they both refer to the same theorem. One possible solution to this issue is calculating the mean squared errors after removing this duplicate from our original data.

It is surprising that *Green's theorem* and the *Gauss Green theorem* have a relatively high mean squared error, as they both relate integration over an area to integration along the boundary of that area.

6. Comparing the initial n characters

In order to eliminate nodes that correspond to the same theorem, we use MATLAB to find theorems that have the same initial n consecutive characters. Carrying out this procedure gives

rise to a number of potential bugs in our data set of mathematical theorems.

When checking for matches in the data set by comparing the initial seven consecutive characters we came across different types of potential bugs.

Examples include:

- Theorems that have various spellings, for instance *Pythagorean theorem*, *Pythagoras theorem* and *Pythagoreas theorem*. (Note: This is expanded upon in the Bug Hunting section).
- Theorems that are the same but when mentioned in a publication often have one of the contributors' name omitted. An example of this is the *Ascoli theorem* and the *Ascoli Arzela theorem*.
- Nodes that describe a general class of theorems rather than being an actual theorem. These include *theorems on continuation*, *theorems on sums of squares* and *theorems of Euclid*.

This method motivates the bug hunting that follows in the next section.

7. Bug Hunting

A limitation we discover with the *Search Engine Algorithm* is that it produces a number of bugs. We classify these bugs into the following categories:

1. Substring bugs
2. Theorems that that appear under more than one name
3. Duplicate nodes
4. Nodes that are not theorems
5. Theorems that have been miscounted due to how the *Search Engine Algorithm* works
6. Other noise

A complete list of bugs discovered can be found in Appendix A.

Using a MATLAB script we search for the duplicates in our network. Twelve nodes were found to have duplicate names. Upon further examination we see that these duplicates also have the same frequency and are connected to the same theorems. The only difference they have is in their ID number. Thus we decide to simply remove these duplicates, keeping only one occurrence of the theorem.

The rest of these bugs are not as simple to discover. In an attempt to compare theorem names, we wrote a MATLAB script called *networkwithsimilarnames.m* (*listing 3*) that produces a new unweighted network. In this new network, the nodes are still the theorem names, however, now two nodes share an edge if and only if they share a word (not counting the words 'of', 'the' and 'theorem') in the theorem name. For example, *Banach* **fixed point** theorem and *Tarski* **fixed point** theorem would share an edge but *Banach fixed point theorem* and *implicit function theorem* would not. The following network is produced:

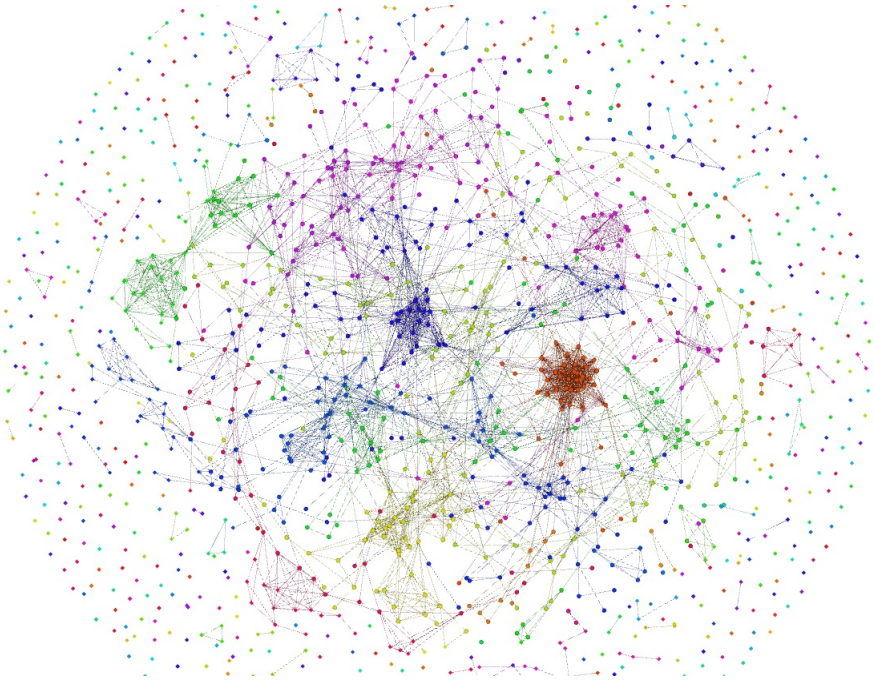


FIGURE 7: Partially same names network

When we rank the nodes of this network in order of decreasing degree it becomes easy to see certain bugs. Some nodes that have a degree of zero are theorems that have been spelt unusually. An example of this is '*Pythagoreas theorem*' (bug type 2). When looking at large communities we also discover a few instances of bug type 4. This includes Node 1558 '*fundamental theorems in mathematics.*' Thus we delete Node 1558 with no further action.

A trickier bug to resolve was an instance of bug type 2. The nodes in question were Node 37 *Pythagoras theorem* and Node 13 *Pythagorean theorem*. We wish to merge these two nodes without double counting. The following is an example of double counting. Suppose Node 37 and 13 both occurred in one paper together with Node Y. Thus, there exists an edge of weight 1 between Node 37 and Node Y and also an edge of weight 1 between Node 13 and Node Y. If we add the weights of these edges, we will have an edge between our merged nodes and Node Y of weight 2, whereas the edge should have a weight of 1.

Nodes 13 and 37 share an edge of weight 11 and we find there are 40 nodes to which they are both connected. As the edge between Node 13 and Node 37 has a weight of 11, there are eleven cases of double counting in the 40 nodes that they are both connected to. The weight of 11 accounts for 27.5% of these 40 connections. As we cannot be sure which edges this 27.5% refers to, we sum the weights of the 40 edges they both share, multiply by 0.725 and round the new weights up. If we were to round down we would be overcompensating for the double counting removal and therefore we would be losing information. We do this using MATLAB. We also readjust the frequency by adding their individual frequencies and subtracting 11 (the number of publications in which they appear together). Finally, we merge the two nodes together using the 40 reweighted edges and the remaining edges. A similar method is used to combine Node 429 *Cauchy mean value theorem* and Node 349 *generalized mean value theorem*. This method allows us to minimise the effect of double counting.

Node 1135 *fifteen theorem* is an example of bug type 5. Although there does exist a theorem called *fifteen theorem*, its huge frequency and number of edges in this network does not reflect its use in mathematics. We believe that the words 'fifteen' and 'theorem' come up together

frequently in literature even when not referring to the actual *fifteen theorem*. This produces the bug we observe. As we cannot tell which of the edges are bugs, we decide to remove Node 1135 entirely from the network.

Bug type 1 occurs very frequently in the network. Say the theorem *Brill Noether theorem* was mentioned in a paper. Then the *Search Engine Algorithm* would produce a hit for *Brill Noether theorem* and it will also produce a hit for *Noether theorem*. This occurs as *Noether theorem* is a substring from right to left of *Brill Noether theorem* even if *Noether theorem* was never mentioned in the paper. To resolve this *substring bug* we wrote a MATLAB script called *substringsearch.m* (*listing 4*). This script is used on the network that *networkwithsimilarnames.m* (*listing 3*) creates. This provides as an output all the theorems that are substrings and the theorems they are a substring of. In some cases the *substring bug* contributes very little to the overall frequency and edges of a substring's node. Thus we decide to only remove a substring node if the *substring bug* accounts for more than 5% of the frequency of the substring's node and otherwise leave the substring's node unchanged. For instance, the *Noether theorem* is removed but the *central limit theorem* is kept. We use a script named *substringtakeout.m* (*listing 5*) to remove the nodes for which the contribution of the *substring bug* is higher than 5%. This is the biggest bug removal, removing 78 nodes and approximately 5000 edges.

Once we had removed the bugs we observed, we renumbered the IDs of the nodes. This was to allow for quicker computations of network analysis we use later on. We call the new network H. From here on, we use the new IDs of the nodes — unless otherwise stated.

8. Betweenness centralities

There are two types of betweenness centrality, namely node betweenness centrality and edge betweenness centrality. In general, *betweenness centrality* is a measure of the significance of a node in the network. It is the number of shortest paths between two nodes that pass through a specific node, $n \in V$ or that go through a specific edge $e \in E$ [New03].

8.1 Node betweenness centrality

The *node betweenness centrality* is defined by

$$BC(n) = \sum_{v \neq n \neq w} \frac{\sigma_{vw}(n)}{\sigma_{vw}}$$

where σ_{vw} is the total number of shortest paths between nodes v and w and $\sigma_{vw}(n)$ is the total number of shortest paths between nodes v and w that pass through node $n \in V$.

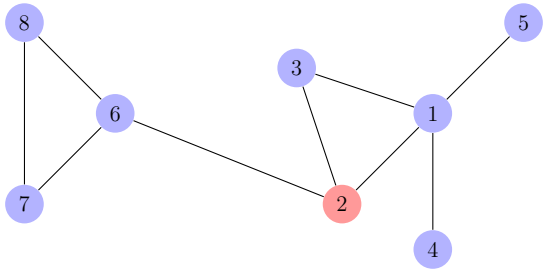


FIGURE 8: In this graph, node 2 has the highest node betweenness centrality.

A node has high node betweenness centrality if it is situated on many shortest paths, meaning that it is connecting different parts of the network together. Such a node has a greater impact on the entire network compared to other nodes.

In the table below all the values for the node betweenness centrality are normalised. This is achieved by using

$$BC(n) = \sum_{v \neq n \neq w} \frac{\sigma_{vw}(n)}{\sigma_{vw}} \div \frac{(N-1)(N-2)}{2}$$

[Bar04], where N is the total number of nodes. Here, $\frac{(N-1)(N-2)}{2}$ is the number of pairs of nodes excluding node $n \in V$.

Remark. The number of pairs of nodes excluding node n in an undirected network is found using combinatorics:

$$\binom{N-1}{2} = \frac{(N-1)(N-2)}{2}.$$

We normalise the node betweenness centrality by dividing by $\binom{N-1}{2}$ as this corresponds to dividing $BC(n)$ by the maximum value of $BC(n)$ [RKJ14]. The maximum value of the node betweenness centrality is achieved only in the case of a star graph by the central node, as this node lies on all the shortest paths between any node pair.

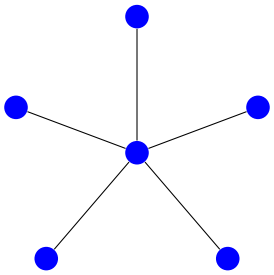


FIGURE 9: A star graph

Using the cleaned data set of mathematical theorems, we outline in the following table the theorems with the highest node betweenness centrality.

Theorem ID	Theorem Name	Node Betweenness Centrality
4	mean value theorem	0.1933
5	implicit function theorem	0.1728
1	central limit theorem	0.1640
7	h theorem	0.1561
87	characterization theorem	0.1415
19	Hahn Banach theorem	0.0788

In Figure 10 [1] the nodes are ranked according to their node betweenness centrality. So the *mean value theorem* appears as the biggest node.

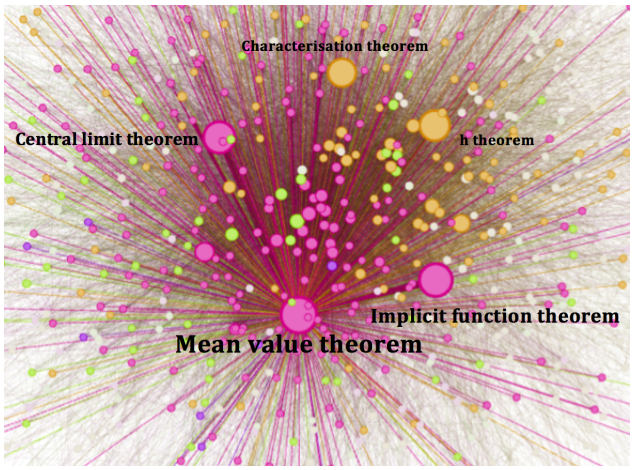


FIGURE 10: Theorems with highest node betweenness centrality

The following graph represents the node betweenness centrality in our network of mathematical theorems. The Count-axis represents the number of nodes with the respective node betweenness centrality values on the Value-axis.

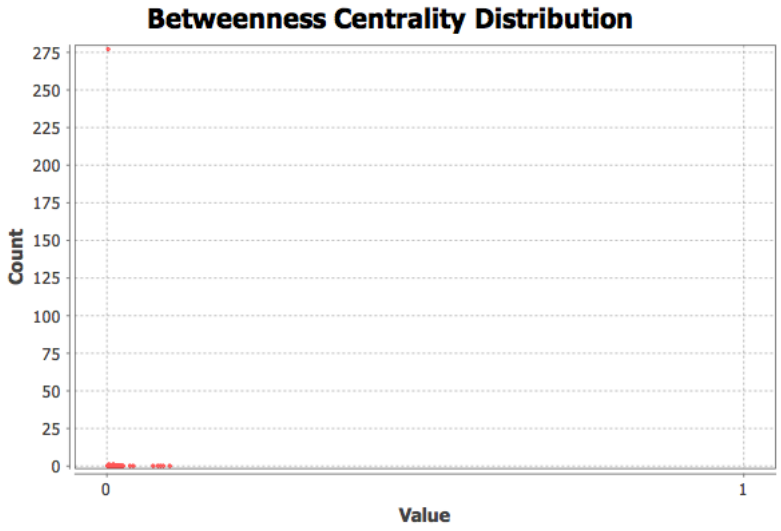


FIGURE 11: Graph of the Node Betweenness Centrality Distribution produced by *Gephi*

[1] The diagram was produced in *Gephi*

It can be observed from the graph that 278 nodes have zero node betweenness centrality. Therefore, there are 278 theorems in our data set that do not mediate between other theorems. These are theorems (nodes) that lie on the boundary of the network of mathematical theorems.

It is evident that the *mean value theorem* is the most important intermediary in our network as it has the highest node betweenness centrality. It is the theorem that if it is removed from the network it will cause the greatest disruption in the 'flow of information' between other theorems. Intuitively, you can think of this as follows: Say you encounter a theorem in a publication. To then come across another theorem, you are likely to see the *mean value theorem* first.

8.2 Edge betweenness centrality

The *edge betweenness centrality* [CPKM12] is defined by

$$BC(e) = \sum_{v \neq w} \frac{\sigma_{vw}(e)}{\sigma_{vw}}$$

where σ_{vw} is the total number of shortest paths between nodes v and w and $\sigma_{vw}(e)$ is the total number of shortest paths between nodes v and w that go through $e \in E$.

We use MATLAB to find the edge betweenness centrality [Git]. The output is a 1305×1305 symmetric matrix. The results of the edges with the highest edge betweenness centrality are given in the table below.

Theorem 1 ID	Theorem 1 name	Theorem 2 ID	Theorem 2 name	Edge Betweenness Centrality
3	mean value theorem	1001	Holditch theorem	0.0030
109	multiplicative ergodic theorem	1099	Ambrose Kakutani theorem	0.0030
5	h theorem	657	Nielsen Schreier theorem	0.0028
4	implicit function theorem	1112	Rashevsky Chow theorem	0.0020
1	central limit theorem	652	Frisch Waugh Lovell theorem	0.0018
251	q binomial theorem	1269	Gordon partition theorem	0.0018
130	cut elimination theorem	625	Beth definability theorem	0.0017

These are the normalised values of edge betweenness centrality [Nar05] using

$$BC(e) = \sum_{v \neq w} \frac{\sigma_{vw}(e)}{\sigma_{vw}} \div \frac{N(N-1)}{2}$$

where N is total number of nodes and $\binom{N}{2} = \frac{N(N-1)}{2}$ represents the total number of edges.

The edge connecting *mean value theorem* with *Holditch theorem* transfers the biggest 'flow of information' between the rest of the theorems. The weight of this edge is one. So the two theorems have only appeared together in one publication. However, the pair *mean value theorem* and *Holditch theorem* is still most 'in between' other theorems.

It is surprising that the pair of theorems with the highest edge betweenness centrality is the *mean value theorem* and the *Holditch theorem*, because although we expect the *mean value theorem* to have high edge betweenness centrality (since it is the theorem with the highest node betweenness centrality), this is not the case for the *Holditch theorem*.

Similar to the node betweenness centrality case, if an edge with a high edge betweenness centrality measure is affected, then so will the communication between pairs of nodes through the shortest paths between them.

An edge with high edge betweenness centrality connects many different parts of the network together. One way to obtain the community structure of the network is to delete such edges following the *Girvan – Newman algorithm* [Cal07].

Algorithm 1 Girvan – Newman Algorithm

Step 1: Calculate the edge betweenness centrality of all the edges in the network.
Step 2: Remove the edge with the highest edge betweenness centrality.
Step 3: Recalculate the edge betweenness centrality for the remaining edges in the network.
Step 4: Repeat step 2 and step 3 until all the edges have been deleted.

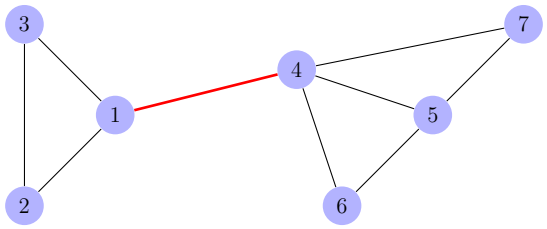


FIGURE 12: In this graph, the edge connecting 1 to 4 has the highest edge betweenness centrality.

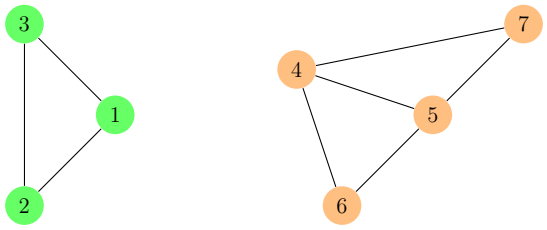


FIGURE 13: Division into communities after deleting the highest betweenness centrality edge.

In addition, a significant number of edges was randomly deleted from the network. Surprisingly, the edge betweenness centrality values only changed by a negligible amount. Even though removing edges would have been expected to greatly affect the distribution of the betweenness centralities amongst the remaining edges in the network, this was not the case.

9. Network Diameter

The *diameter* of a network, $diam(G)$, is the longest of its shortest paths. A network has $diam(G) = \infty$ if and only if it is not connected [CG84], that is the network can be divided into subsections (called *components*) which have no connection between them [New10].

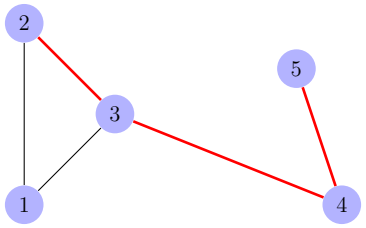


FIGURE 14: In this graph, the diameter is 3.

The cleaned data set in our project happens to fall into this category. More specifically, it has five completely isolated components. The five components consist of one major component, with most of the nodes (1297 out of 1305), and four components, each with only a pair of theorems connected. The diameter that we then calculate is the diameter of the major component, and this is found to be six.

These four pairs are interesting in that they are not connected to any other part of the literature. The four pairs are:

Theorem 1 ID	**Theorem 1 name**	**Theorem 2 ID**	**Theorem 2 name**
936	Karp Lipton theorem	1250	Sipser Lautemann theorem
1144	Lickorish Twist theorem	1253	Lickorish Wallace theorem
1214	Mohr Mascheroni theorem	1267	Poncelet Steiner theorem
1264	Mertens' First theorem	1288	Mertens' Second theorem

Having obtained the diameter for the major connected component, we then carried out several attempts to increase or decrease the diameter. However, we only succeeded to increase the diameter by 1, by deleting all the edges connected to *central limit theorem*, a genuinely important theorem that has both high strength and high degree. So it is safe to conclude that the network structure we have is already very stable, and a large amount of shortcut edges will be required to decrease the diameter.

Generally, smaller diameter is preferred for networks, especially for communication networks. In our case the motivation for minimising diameter might be that it minimises the largest possible time we need to understand the link between any two theorems.

10. Spanning trees

A *tree* is an undirected, connected graph containing no cycles. Let G be a connected graph consisting of n vertices. A spanning tree of G is a tree that is a subgraph of G that includes all the nodes of G [KG15]. A spanning tree of G has n nodes and $n-1$ edges. A minimum spanning tree of G is a spanning tree of weight less than or equal to the weight of every other spanning tree of G. Analogously, a maximum spanning tree of G is a spanning tree of weight greater than or equal to the weight of every other spanning tree of G. If each edge in G has a distinct weight, the spanning tree of G is unique. The edges in our network do not have distinct weights.

When referring to an unconnected graph H, we talk about spanning forests – a collection of the spanning trees of each connected component of H.

Let us call the graph of our network $H = (V, E)$. We note that H contains five separate components; four components consist of two nodes, and one component consists of 1297 nodes. We thus expect our minimum spanning forest to contain 1305 nodes and 1300 edges. Note that $4(2-1) + (1297-1) = 1300$.

Remark. If we make the strong assumption that theorems appear in publications in pairs (that is, a publication contains exactly two different theorems), then the number of edges in a spanning tree of H (1300) is the minimum number of publications one would have to read in order to come across all of the theorems in our network.

To obtain a minimum spanning forest of H, we apply *Kruskal's algorithm*[Pap]:

Algorithm 2 Kruskal's Algorithm

1. Say $H = (V, E)$ has n vertices. We begin by creating a graph G containing all the vertices in V, but no edges. That is, we begin with a graph of n single–node trees.
2. Arrange the set of edges E of H by their weights, in a non-decreasing order.
Note: if two edges have the same weight, the order of the two is arbitrary.
3. Consider each edge in E in turn, in the order specified in 2.
4. Repeat 3. until all edges in E have been considered.

As the choice between two edges of the same weight is arbitrary, we observe that the minimum spanning tree of a graph is not necessarily unique.

The result of applying Kruskal's algorithm to H is a graph G of 1300 edges and 1305 nodes, as expected. We use *Gephi* to visualise this minimum spanning forest, which has an overall weight of 1319.

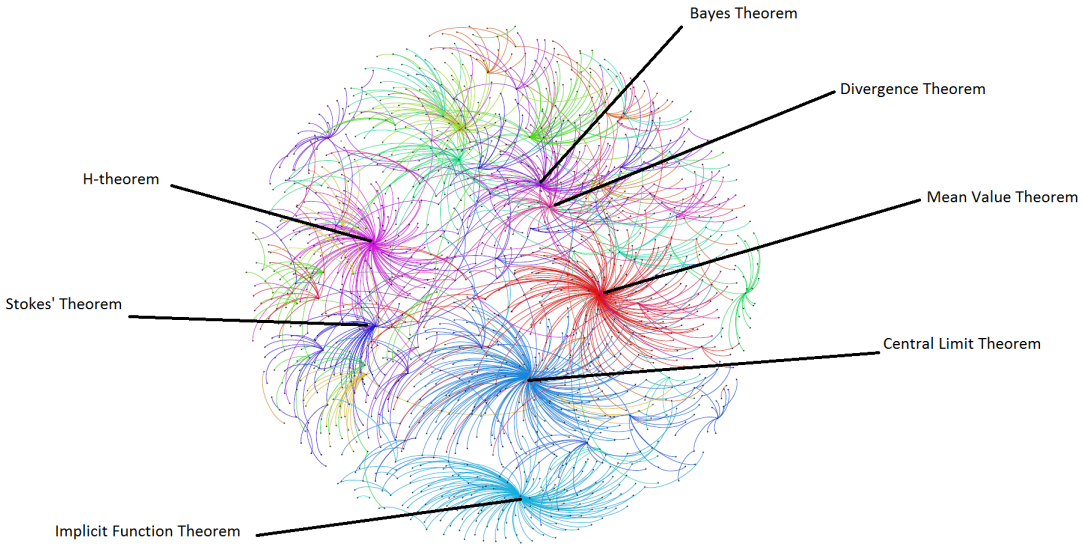


FIGURE 15: Minimum spanning forest of H

From the 'clusters' of edges in Figure 15, we observe that the minimum spanning forest is dominated by approximately four theorems: the *central limit theorem*, the *implicit function theorem*, the *h theorem* and the *mean value theorem*. This is expected as these theorems have the highest frequencies and degrees in our data.

We create a maximum spanning forest of our network by negating the weights of the edges and then applying *Kruskal's algorithm*; the resulting maximum spanning forest has a weight of 13810.

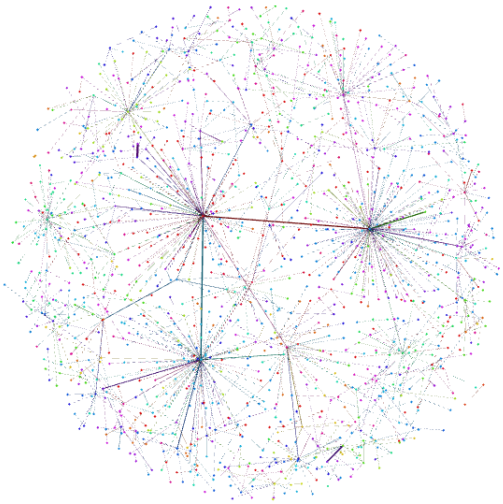


FIGURE 16: Maximum spanning forest of H

The three dominating theorems are the *mean value theorem*, *implicit function theorem*, and *central limit theorem*. We are aware that these theorems are extremely prevalent in mathematics, and so this is expected. To gain a better understanding of the rest of the network, we denote by H' the graph formed by eliminating the edges in H that emanate from these three nodes. We anticipate that the removal of these edges changes the structure of our network. We use MATLAB to determine that H' contains 22 separate components: 16 components of only one node, five components of two nodes, and one component of 1279 nodes. We create a maximum spanning forest of H' (shown in Figure 17) and expect that it consists of 1283 edges, which it does.

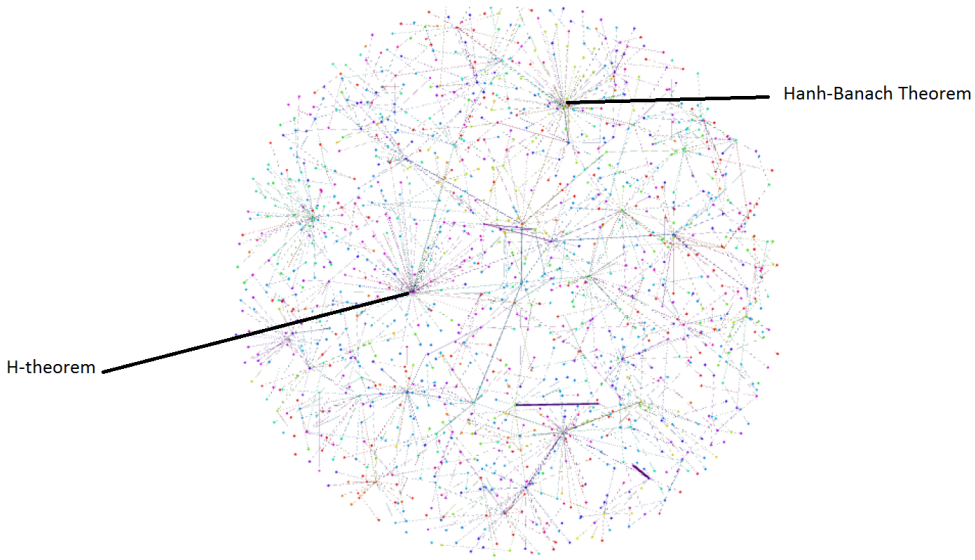


FIGURE 17: Maximum spanning forest of H'

The *h theorem* and *Hahn Banach theorem*, forming 'clusters' in the above maximum spanning tree, have high frequencies and degrees relative to other theorems in our data (excluding the *mean value theorem*, *implicit function theorem*, and *central limit theorem*). These 'clusters' appear to be less prominent than those in Figure 16, which we assume is due to the *h theorem* and

Hahn Banach having smaller degrees than the *mean value theorem*, *implicit function theorem* and *central limit theorem*.

We now further delete all the edges emanating from the *h theorem* and the *Hahn Banach theorem* from H', forming a new graph H''. We create a maximum spanning forest of H'' (shown in Figure 18). This brings to the surface 'clusters' caused by less commonly used theorems, in particular, the *Chinese remainder theorem* and *Lebesgue dominated convergence theorem*.

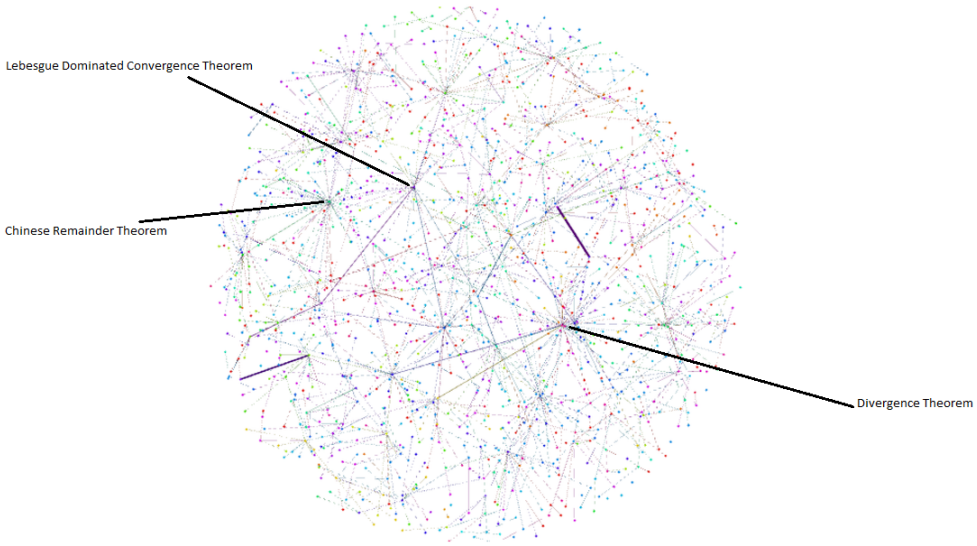


FIGURE 18: Maximum spanning forest of H''

Note that as the maximum spanning forest of H'' is not necessarily unique, we cannot be certain that 'clusters' form in the other maximum spanning trees, much less that they form around the *Chinese Remainder theorem* and *Lebesgue dominated convergence theorem*.

11. Community detection

Suppose we wish to discover theorems that are closely related but do not necessarily share an edge. We use the idea of a communities in a network to discover such theorems. A *community* is a collection of nodes that are more connected with each other than the rest of the network [DA05]. With regards to our network, finding communities would allow us to find theorems that are closely related together in literature. Two theorems might not share an edge but still be closely related and thus we expect these theorems to be a part of the same community.

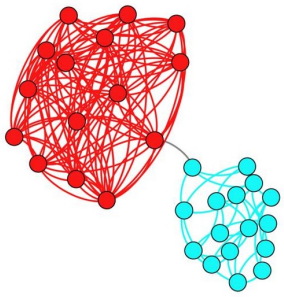


FIGURE 19: A network comprised of two communities - Produced in *Gephi*

As the size of the network is extremely large we turn to computational methods of finding communities. We favour using a measure called *stability* [DYB10].

This measure works akin to letting an ink-drop spread through the network. If there is no structure, an ink-drop would spread evenly over time. However, if there is structure, the ink would retain in some areas for longer than others until spreading to the next area. This is one way of discovering communities [DSYB13].

Adapting this idea to our network, we let random walkers jump from node to node. A random walker jumps from node i to node j with probability equal to the weight of the edge between node i and node j divided by the strength of node i. In the below Figure, if a random walker found herself at the red node, she would jump down the edge of weight 1 with probability $\frac{1}{4}$ and the edge of weight 3 with probability $\frac{3}{4}$.

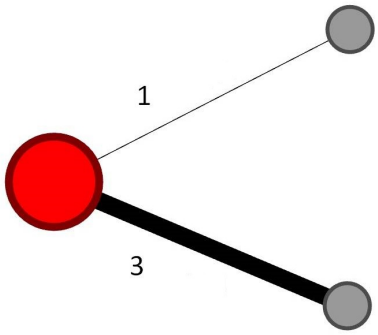


FIGURE 20: A Random Walk

The time at which the random walker will jump can vary. When we use normalised dynamics the waiting time is a random variable distributed exponentially with unit expectation [**?**]. The waiting time does not change depending on which node a random walker waits at. In contrast, we could change the waiting time to depend proportionally or inversely proportionally to the strength of the node the random walker is waiting at.

First we have a look at normalised dynamics [Sch]. We run the stability measure and look at the communities between 1 and 10 seconds. At $t = 8.71\,s$ we notice there is a particular good partition. (See the subsection for notes on variation of information).

Community (1.#)	**Count**	**Percentage**
0	463	35.48%
1	582	44.60%
2	235	18.01%
3	15	1.15%
4	2	0.15%
5	2	0.15%
6	2	0.15%
7	2	0.15%
8	2	0.15%

The communities of size two are all nodes which share an edge with the other node in the community but no other nodes. These communities are, therefore, not surprising and these theorems were discovered previously in the diameter analysis.

Community 1.3 contains the following 15 nodes:

Theorem ID	**Theorem name**	**Frequency**
13	Coase theorem	816
22	folk theorem	1164
63	Stolper samuelson theorem	359
79	median voter theorem	255
110	Heckscher Ohlin theorem	199
167	Rybczynski theorem	216
233	Price theorem	135
297	Shephard duality theorem	117
506	Gauss Lucas theorem	40
518	median theorem	43
535	Post theorem	13
562	Lerner symmetry theorem	18
790	Sonnenschein Santel Debreu theorem	25
1108	bisectors theorem	6
1169	perpendicular bisector theorem	2

The vast majority of these theorems are directly related to economics. Exceptions are *Gauss Lucas theorem*, *Post theorem* and the two bisector theorems.

The *Gauss Lucas theorem* gives a relation between the roots of a polynomial P and the roots of its derivative P'. A google search eventually finds a not immediately obvious connection between the *Gauss Lucas theorem* and economics [BLO04]. A similar story follows for *Post theorem* [Mih99].

We can immediately see that community detection can yield useful results, however this particular partition has too many large scale communities which are difficult to analyse. We notice that the three largest communities are centred on the high strength nodes *central limit theorem*, *h theorem* and *divergence theorem* respectively. These theorems are very commonly used and thus are adding noise to the network. In hope of obtaining more medium sized communities, we delete the *central limit theorem*, the *h theorem* and the *divergence theorem* from the network and running the community detection again. However, when run, we still find a similar spread of sizes of communities.

Community (2.#)	Count	Percentage
0	471	36.09%
1	559	42.84%
2	254	19.46%
3	15	1.00%
4	2	0.15%
5	2	0.15%
6	2	0.15%
7	2	0.15%

One of the communities of size 2 is absorbed into community 2.0. The other communities of size 2 remain the same.

Community 2.3 in this partition is almost identical to community 1.3 in the last partition. It no longer contains the two bisector theorems but contains the *Maskin theorem* which is related to economics. In this regard, we believe that this community is a better community than the community previously examined. Community 2.0 is dominated by the *mean value theorem*, community 2.1 by *characterisation theorem* and community 2.2 by *fluctuation dissipation theorem*. Thus we fail to eliminate the problem of large nodes dominating the communities.

We continue this procedure and delete the *mean value theorem*, *Bayes theorem*, *implicit function theorem*, *fluctuation dissipation theorem*, *Gauss theorem* and *characterisation theorem* from our network and rerun the community detection.

Community (3.#)	Count	Percentage
0	492	37.70%
1	735	56.32%
2	55	4.21%
3	2	0.15%
4	13	1.00%
5	2	0.15%
6	2	0.15%
7	2	0.15%
8	2	0.15%

This partition is found at $t = 10$. At first it looks very similar to our previous partitions. We would expect community 3.4 to be similar to community 2.3 or 1.3. However, this is not the case.

Theorem ID	**Theorem name**	**Frequency**
84	final value theorem	483
208	initial value theorem	165
240	Gershgorin theorem	111
369	Gershgorin circle theorem	90
424	Gerschgorin circle theorem	111
506	Gauss Lucas theorem	40
544	Bauer Fike theorem	65
736	Poincare separation theorem	29
944	Gershgorin disc theorem	30
1108	bisectors theorem	6
1132	Gerschgorin disc theorem	12
11169	perpendicular bisector theorem	2
1266	Sylvester determinant theorem	2

We first notice nodes 240, 369, 424, 944 and 1132 are in fact the same theorem. We missed this particular bug during our bug hunting before due to a lack of time to thoroughly search the network, but this exemplifies how community detection can also be used for bug detection.

In this particular community we find the link (which has a weight of three) between *Gauss Lucas theorem* and *Poincare separation theorem* of interest. We discuss this further in the section Surprising theorems. In fact at a later time of $t = 10.965$ the community of *Gauss Lucas theorem* and *Poincare separation theorem* persists.

Theorem ID	**Theorem name**	**Frequency**
506	Gauss Lucas theorem	40
736	Poincare separation theorem	29
1108	bisectors theorem	6
1169	perpendicular bisector theorem	2

This suggests a strong relationship between these two theorems.

In a change of tactic to produce more medium sized communities, we reduce the effect that nodes with high strength have on the network by changing the dynamics of the waiting time at each node. If each random walker waited at a time inversely proportional to the strength of the node, then a random walker would leave a high strength node more quickly. This reduces the effect the high strength node has on the formation of communities.

We implement this waiting time into the stability algorithm ourselves [DSYB13] and quickly find a partition with a low variation of information and multiple medium size communities.

Community (4.#)	Count	Percentage
0	66	5.06%
1	18	1.38%
2	49	3.75%
3	9	0.69%
4	10	0.77%
5	17	1.30%
6	18	1.38%
7	48	3.68%
8	7	0.54%
9	4	0.31%
10	973	74.56%
11	15	1.15%
12	19	0.54%
13	4	0.31%
14	23	1.76%
15	19	1.46%
16	10	0.77%
17	2	0.15%
18	2	0.15%
19	2	0.15%
20	2	0.15%

Community 4.9 is centred on the *Fubini theorem*.

Theorem ID	Theorem name	Frequency
9	Fubini theorem	1008
319	Sklar theorem	31
633	Savitch theorem	6
820	perpendicular bisector theorem	14

This provides an interesting link between computational complexity theory via the *Savitvh theorem* and probability via the *Sklar theorem*.

There are many other interesting results that can be obtained from this partition alone at one Markov time. We conclude that using inverse waiting time dynamics is more appropriate for discovering communities in this network.

11.1 Variation of information

To test how good a community might be we use another measure called the *variation of information*. The stability algorithm runs the *Louvain algorithm* 100 times at each Markov time and outputs the partition that comes up most frequently. If there is a high variation of information then many other partitions were found at the Markov time and we conclude the particular partition is not very robust. Conversely, if the variation of information is zero then the same partition was found all 100 times and we conclude the partition is very good. When we look for which communities we wish to explore further, we look for the ones with as low variation of information as possible [Str13].

12. Surprising theorems

When trying to find the diameter we found four pairs of theorems that shared no edges with the major components in the network.

The *Karp Lipton theorem* and *Sipper Lautemann theorem* pair belongs to the field of polynomial hierarchy in oracle machines so its focus on a very specific field explains its lack of connection to anything else.

The second pair, *Lickorish Twist theorem* and *Lickorish Wallace theorem*, shares the same reason with the first pair of theorems and is focused on the field of Dehn Surgery.

The *Mohr Mascheroni theorem* and the *Poncelet Steiner theorem* pair has a different reason for being isolated, because it is focused on Compass and Straight angle theories in Euclidean mathematics which is a rather ancient area of mathematics.

The pair *Mertens' first theorem* and *Mertens' second theorem*, however, has yet another reason. In number theory, when referring to both *Mertens' first and second theorem* together, we use the name *Mertens' theorem*. *Mertens' theorem* is connected to many other theorems.

When using normalised dynamics in community detection a surprising link was found between the *Gauss Lucas theorem* and the area of economics. Although the *Gauss Lucas theorem* did not share an edge with any of the theorems related to economics in the community we found it in, a google search does reveal an application for the *Gauss Lucas theorem* in economics. This is a result that would have not been able to be obtained via other methods.

After we removed some of the higher strength nodes we find a link (which has a weight of three) between *Gauss Lucas theorem* and *Poincare separation theorem* of interest. The *Poincare separation theorem* gives an upper and lower bound for eigenvalues under certain conditions. Here we seem to be relating complex analysis with algebra.

When using inverse dynamics in community detection we find a relationship between the *Savitvh theorem* (related to computational complexity theory) and *Sklar theorem* which concerns marginal distribution functions.

The *art gallery theorem* answers the question "how many guards (at minimum) do you need to supervise the whole gallery?" by modelling the gallery as a polygon and the guards as points in the polygon. The *Minkowski theorem* says that every convex body in $\mathbb{R}^n$ of volume greater than 2^n contains in its interior a non-zero point. It is surprising that these two theorems are linked because the *art gallery problem* is related to computational geometry and is a very 'visual' problem, whereas the *Minkowski theorem* relates to number theory. However, if we consider the *art gallery problem* in higher dimensions (so we would consider polyhedra instead of polygons) and then restrict ourselves to the case of convex body polyhedra, we can apply *Minkowski's theorem* to the problem [Vre05].

Interestingly, although the *Minkowski theorem* appears with the *art gallery theorem* in the paper *On Polygons and Polyhedra*[Vre05], the *art gallery theorem* and the *Minkowski theorem* do not share an edge in our network. Two potential reasons for this are as follows:

1. The paper *On Polygons and Polyhedra* was not part of the data set that the search algorithm was used on.

2. The paper refers to "Minkowski**'s** theorem," so the search algorithm did not recognise this as "Minkowski theorem."

13. Conclusions and Future Work

During this project we have cleaned the network removing approximately 500 nodes and performed a number of investigations into the network. Using network characteristics such as the diameter, minimum and maximum spanning forests, betweenness centralities and communities we have discovered which theorems are commonly used and connections between theorems that we would not necessarily expect. We can use this to develop some areas of mathematics that are interestingly connected to each other.

Whilst working on our project, we found areas that would like to further develop upon. One example of this is the *Search Engine Algorithm*. Presently, the search algorithm does not seem to detect punctuation, and so it picks up on erroneous phrases. For example, "*...contraction. Theorem ...*" would be incorrectly seen as the *contraction theorem*.

Another way we would like to improve *Search Engine Algorithm* is to minimise occurrences of the *substring bug*. For example, we would like to be able to search for "*Noether theorem*" whilst excluding results for "*Lasker Noether theorem*."

It may be worthy of further investigation to minimise the diameter of the network, using a limited set of shortcut edges.

It would also be interesting to create spanning forests using different algorithms, and noting whether 'clusters' still form, and, if so, whether they form around the same theorems as found in section 10.

For further work, we would vigorously look through the network for further bugs. Even towards the end we would find bugs that our algorithms overlooked for various reasons. We could also remove the big nodes such as *central limit theorem* which we already know a great deal about, thus creating noise, in order to investigate and find more hidden results.

The community detection also has more room for investigation. Given more time we could investigate more of the inverse waiting time dynamics communities, we could change the waiting time dynamics to be proportional to the strength of node, investigate later times and even look at communities of the line graph of the network.

Appendix A Nodes deleted

- Duplicates deleted were Node 547 *art gallery theorem*, Node 698 *Bertini theorem*, Node 1460 *Chasles theorem*, Node 1619 *Dirac theorem*, Node 167 *distortion theorem*, Node 749 *Euclid theorem*, Node 1580 *Foster theorem*, Node 531 *Fredholm theorem*, Node 363 *Godel incompleteness theorem*, Node 868 *Helmholtz theorem*, Node 1644 *Oka theorem*, Node 767 *Torelli theorem*.
- Some of these nodes are not theorems such as Node 1558 *'fundamental theorems in mathematics'*. Node 1558 was deleted.

- Node 718 *'fundamental theorem of symmetric'* was deleted as it either refers to Node 1071 *'fundamental theorem of symmetric polynomials'* or Node 970 *'fundamental theorem of symmetric functions'*. Every node that Node 718 is connected to is also either connected to Node 1071 or 970. We conclude that node 718 can be removed without further action.
- Node 319 *'fundamental theorem of projective'* was deleted as we assumed it was meant to be Node 333 *'fundamental theorem of projective geometry'*. The limitation of this deletion is that Node 319 has a frequency of 105 compared to Node 333 which has a frequency of 104. This means we are losing some information but we are not sure which additional paper Node 319 appeared in.
- Node 1488 *'algebra fundamental theorem of'* was assumed to be the same as Node 59 *'fundamental theorem of algebra'* as it is connected to Node 106 *extension theorem* which Node 59 is also connected to. Thus we assume Node 1488 was a bug. We simply delete Node 1488.
- Node 37 *Pythagoras* and Node 13 *Pythagorean* theorem. (Please see text for detailed explanation on how to resolve this issue.)
- Node 1135 *fifteen theorem* should not come up as frequently as it does. We believe the words 'fifteen' and 'theorem' come up often together in literature even if the paper is not referring to the actual *fifteen theorem*. As we cannot be sure which edges are genuine we remove the node from the network.
- Node 3 *fixed point theorem* is deleted as you are likely to state which fixed point theorem you are using. Simply adding up the edges would lead to double counting.
- Whenever the *first* or *second fundamental theorem of calculus* are mentioned they appear with the *fundamental theorem of calculus*. For all useful purposes they are the same, thus we delete node 942 and 944 together with their edges.
- Node 429 *Cauchy mean value theorem* and Node 349 *generalized mean value theorem* are the same. We merge the two by adding together the edges and normalising those that overlap similar to our Pythagoras theorems.
- In a rather bigger move, we tackle the bug that causes theorems to be counted when they should not be due to them being a substring of another theorem.
  - MATLAB code was written to discover all the substrings
    - networkswithsimilarnames.m
    - substringsearch.m
  - Then we removed all substring nodes such that this particular bug contributed more than 5% to the frequency of the node. This caused a big reduction to the network.
    - Use substringtakeout.m
- *Euler rotation theorem* and *rotation theorem* are the same theorem. Fortunately, all the edges of *Euler rotation theorem* were to *rotation theorem*. So we simply delete those edges. We keep the frequency of *rotation theorem* the same.

Appendix B MATLAB code

Listing 1: Comparing the initial n characters

```
% This function takes the first 'n' characters in the name of a theorem
% and finds all other theorems with same first 'n' consecutive characters.
% The output is of the form of a mx40 matrix, where each row number ...
   corresponds to
% the id of theorems with similar names.

function [bigIndex] = CompareInitialCharacters2(Graph, n)

% The input for Graph is 'nodes' generated by the MATLAB function
% [num txt nodes] = xlsread (['Nodes- Original Version.xlsx')
% first column of nodes is theoremID, second column is theorem name

% n is the number of consecutive characters we wish to compare.

% m is the length of the list

m=size(Graph,1);
bigIndex = zeros(40,m);

% Initialising bigIndex matrix

for i = 1:m


String = Graph(i,1);
index= strncmpi(String,Graph(:,1),n);

% This function compares n characters in the string with a list of strings

siz=size(find(index));

if isempty(find(index))
    bigIndex(1,i)=0;

else
    bigIndex(:,i)=([find(index)', zeros(40-siz(:,1),1)']);
end
end

bigIndex=bigIndex';
nonzero = find(bigIndex(:,2));
bigIndex = bigIndex(nonzero,:);
```

Listing 2: Nodes with low degree and high strength

```
% This function is used to discover nodes with a low order but high overall
% strength.
function [surprising]=loworderhighstrength(Graph, n, order, strength)
% Graph is a nx3 matrix. 1st column is the Id of the node. 2nd column
% is the order of each node and the 3rd column is the strength of the
% node. n is the length of the list.
j=1;
for i = 1:n
    if Graph(i,3)>strength && Graph(i,2)<order
        surprising(j,:) = [Graph(i,1) Graph(i,2) Graph(i,3)];
        j=j+1;
    end
end
```

Listing 3: Creating the Partially Same Name Netowork - networkwithsimilarnames.m

```
% Create txt by [num txt nodes] = xlsread ('filename.xlsx') using an excel
% file where each word in a theorem have been put in seperate tabs
% and then the words 'theorem', 'the' and 'of' have been removed.
[num txt nodes] = xlsread ('nodes.xlsx');
txt = char(txt{:,1},txt{:,2},txt{:,3},txt{:,4},txt{:,5},txt{:,6},txt{:,7},
      txt{:,8});
% Put all the words in one column. The 1st 1865 words are those already in
%the 1st column, the second 1865 words are from the 2nd column etc.
txt = cellstr(txt);
[un idx_last idx] = unique(txt(:,1));
unique_idx = accumarray(idx(:),(1:length(idx))',[],@(x) {sort(x)});
unique_idx{1,1}=[];
% The 1st entry refers to sharing a space which is not useful for our
% interests so is deleted.
unique_idx(cellfun(@numel,unique_idx)≤1)=[];
% unique_idx is now a cell array that lists all the positions that share
% an identical word.
for i = 1: size(unique_idx,1)
    [p,q] = meshgrid(unique_idx{i,1}, unique_idx{i,1});
    pairs = [p(:) q(:)];
    pairs(pairs(:,1) == pairs(:,2),:)=[];
    if i == 1
        allpairs = pairs;
    else
        allpairs = [allpairs; pairs];
    end
end
% this creates a 2xm matrix (m is decided on how many identical words there
% are). Where if two numbers are next to each other then the positions of
% those two numbers share an identical word.
while isempty(allpairs(allpairs>1865)) == 0
    allpairs1 = allpairs>1865;
    allpairs1 = 1865*allpairs1;
    allpairs=allpairs-allpairs1;
end
% Here we use a while loop to reduce the numbers above 1865. e.g. suppose
% 1866 is in the allpairs list. This word was originally from the 1st
% theorem on the list (1866-1865=1). As we are comparing theorem names and
% not just words we wish to reduce these numbers to by 1865 until the
% number is less than 1865.
clear allpairs1 un idx_last i idx nodes num p pairs q unique_idx
```

Listing 4: Finding Theorems that are Substrings - substringsearch.m

```
% You must have run networkwithsimilarnames.m first!
% nodes2 is a list of the nodes we currently still have. The first column
% contains the IDs of the theorems and the second column contains the
% names of the theorems.

[num1 txt1 nodes1] = xlsread ('nodes2.xlsx');

allpairsnames=cell(size(allpairs,1),2);
allpairsnames(:,1)=txt1(allpairs(:,1));
allpairsnames(:,2)=txt1(allpairs(:,2));

% allpairsnames is now a list of edges of the partially same network, where
% each element is the name of the theorem rather than the ID.

k=1;


for i = 1:size(allpairs,1)
    str1 = allpairsnames{i,1};
    str2 = allpairsnames{i,2};
    found=findstr(str1, str2);
    if isempty(found)==0
        allpairs1(k,1) = allpairs(i,1);
        allpairs1(k,2) = allpairs(i,2);
        k=k+1;
    end
end

% allpairs1 now contains the IDs of substrings and theorems they are
% substring of.

for j = 1:size(allpairs1,1)
    for r=1:size(allpairs1,1)

     if allpairs1(j,1) == allpairs1(r,2) && allpairs1(j,2) == allpairs1(r,1)
         allpairs1(r,:) = [0,0];
     end
    end
end

allpairs1(all(allpairs1==0,2),:)=[];

% We have now removed any rows in allpairs1 which are inverted
% duplicates that have already come up

substrings=cell(size(allpairs1,1),2);
substrings(:,1)=txt1(allpairs1(:,1));
substrings(:,2)=txt1(allpairs1(:,2));

% We convert these IDs into the names of the theorems.
% The final product substrings may contain duplicates but these are easily
% checked for and removed in Excel.

clear allpairsnames found i k nodes 1 num1 str1 str2 nodes1
```

Listing 5: Remove nodes that the substring bug contributes more than 5% to the frequency - substringtakeout.m

```
% In Matlab initialise a vector 'z' that contains in it the ID of the
% theorems we wish to remove. Here edges1 is the adjacency matrix of our
% 1865x1865 network.

for i = 1:size(z,1)
    edges1(z(i),:)=0;
    edges1(:,z(i))=0;
end
```

References

[Bar04] M. Barthelemy. Betweenness centrality in large complex networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):163–168, 2004.

[BLO04] J V Burke, A S Lewis, and M L Overton. Variational analysis of the abscissa mapping for polynomials via the gauss-lucas theorem. *Journal of Global Optimization*, 28(3-4):259–268, 2004.

[Cal07] G. Caldarelli. Scale-free networks: complex webs in nature and technology. *OUP Catalogue*, 2007.

[CG84] FRK Chung and MR Garey. Diameter bounds for altered graphs. *Journal of Graph Theory*, 8(4):511–534, 1984.

[CPKM12] A. Cuzzocrea, A. Papadimitriou, D. Katsaros, and Y. Manolopoulos. Edge betweenness centrality: A novel algorithm for qos-based topology control over wireless sensor networks. *Journal of Network and Computer Applications*, 35(4):1210–1217, 2012.

[DA05] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.

[DSYB13] J C Delvenne, M T Schaub, S N Yaliraki, and M. Barahona. The stability of a graph partition: A dynamics-based framework for community detection. In *Dynamics On and Of Complex Networks, Volume 2*, pages 221–242. Springer, 2013.

[DYB10] J-C Delvenne, S N Yaliraki, and M. Barahona. Stability of graph communities across time scales. *Proceedings of the National Academy of Sciences*, 107(29):12755–12760, 2010.

[Git] GitHub. Node and edge betweenness centrality. https://github.com/fieldtrip/fieldtrip/blob/master/external/bct/edge_betweenness_wei.m. Accessed: 2015–06–01.

[KG15] A. Kapanowski and Ł. Gałuszka. Weighted graph algorithms with python. *arXiv preprint arXiv:1504.07828*, 2015.

[Mih99] H R Mihara. Arrow's theorem, countably many agents, and more visible invisible dictators. *Journal of Mathematical Economics*, 32(3):267–287, 1999.

[Nar05] S. Narayanan. *The betweenness centrality of biological networks*. PhD thesis, Citeseer, 2005.

[New03] M. EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[New10] M. Newman. *Networks: an introduction*. Oxford University Press, 2010.

[Pap] G. Papachristoudis. Kruskal's algorithm. http://www.mathworks.com/matlabcentral/fileexchange/41963-kruskal-s-algorithm/content/kruskal.m. Accessed: 2015–06–03.

[RKJ14] K. Raghavan, U. Sunil, B. Kannan, and M. Jathavedan. Betweenness centrality in some classes of graphs. *International Journal of Combinatorics*, 2014, 2014.

[SBV09] M. Á. Serrano, M. Boguñá, and A. Vespignani. Extracting the multiscale backbone of complex weighted networks. *Proceedings of the National Academy of Sciences*, 106(16):6483–6488, 2009.

[Sch] M. Schaub. Markov stability code for normalised dynamics. http://wwwf.imperial.ac.uk/~mts09/code.html. Accessed: 2015–06–13.

[Str13] J. Stroud. Detecting community structure in genome-wide networks with an application to a case-control study of alzheimer's disease. 2013.

[Vre05] S. Vrecica. On polygons and polyhedra. *The Teaching of Mathematics, VIII (1)*, pages 1–14, 2005.