Patent Collaboration and Team Formation

Peter Keane `<peter.keane.2014@mumail.ie>`

M.Sc., Maynooth University, 2020

Declaration

I have read and understood the departmental policy on plagiarism. I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Signature:

Date:

Peter Keane (May 15, 2020).

Abstract

The team formation problem has existed for many years in various guises. One important problem in the team formation problem is to produce small teams that have a required set of skills. We propose a framework that incorporates machine learning to predict unobserved links between collaborators, alongside Steiner tree problem solutions to form small teams to cover given tasks. Our framework not only considers size of the team but also how likely team members are to collaborate with each other. The framework is tested on sets of data from two different companies. The results show that this model consistently returns smaller collaborative teams.

Acknowledgments

I would like to thank my family and friends for all their support throughout this research masters. Specifically my immediate family, my mam, Joyce, and my brother, Seán. Thanks to my sister, Jen, for letting me use her fancy camera for photographing an experimental setup. I'd also like to thank my girlfriend, Siobhán, for being my biggest fan, and giving me a kick up the arse when I needed it. I couldn't have done this without any of them.

I'd like to thank my supervisor, David Malone. For giving me this opportunity to begin with, for allowing me to pick his brain over the past two years, and for all the chats around the Hamilton lunch table about anything and everything.

Thank you, also, to all of the people in Hamilton. Rose and Kate, whose door was always open whenever I had any administrative issues at all. Thanks to everyone else who shared the Hamilton lunch table, postgrads and staff alike. I'd like to extend a special thanks to Dr. Charles Markham of the Hamilton Institute and his son Toby, for making my experimental acrylic pieces.

I'd also like to acknowledge all the guys in IBM's IIX team, and my first point of contact in IBM, Faisal Ghaffar, for welcoming me as one of their own, inviting me for lunches, games of table tennis and badminton, and for helping me when I hit coding walls.

Finally, and as is always the case with any milestone of mine, I'd like to dedicate this to my dad, Jim Keane. I hope he'd be proud.

"Life's a piece of shit, when you look at it.

Life's a laugh and death's a joke it's true.

You'll see it's all a show, keep 'em laughing as you go,

Just remember that the last laugh is on you."

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

The initial proposal was to analyse the previous social interactions of IBM employees on internal blog networks and various internal social sites. The goal was to see if there were particular interactions or relationships that seemed to encourage patent filing. From that info and analysis, potential collaborators could be recommended to a prospective inventor.

This very quickly morphed into a team formation problem based on analysis of existing IBM patents and their holders.

As such, the aim was to recommend teams of inventors, such that between them, the inventors possess the skills necessary to contribute to the proposed new invention. These recommendations are based on various parameters. Such parameters include graph theory measurements and machine learned link predictions.

1.2 Layout

In chapter 2 we will speak about the background of the project. Some of the basics of graph theory will be explained, along with some definitions.

The tools used over the course of the research will also be discussed. The programming language used is Python. Python is a high level language and has a huge amount of libraries (or packages) for performing certain tasks. There are packages for dealing with mathematical operations, graph theoretical operations, visualisation, database management etc. The tools and python packages which are used most in this research will be noted and a brief explanation given on them. Research already undertaken in this field by others will also be discussed.

Chapter 3 will focus on the data. Obviously to analyse patent collaboration, we will need patent data. In this chapter we will look at which source was used, why it was used, and what it looked like. There will be some focus on how the relevant data was gleaned. The chapter will also consist of some basic statistical analysis of that data source.

In chapter 4 the team formation problem will be discussed. Some previous work in the field will be referenced, and a framework for our scheme will be laid out. This chapter will include, among other things, descriptions of algorithms used for creating graphs, descriptions of the data used for analyses, snippets of code used for things like machine learning, and results.

Finally, in chapter 5, we will summarise and discuss the results of the experimentation. We will also discuss some potential areas for further research.

1.3 Outcomes

Much of the research from chapters 3 and 4 was published in the proceedings of the 2019 International Conference on Complex Networks and Their Applications [2]. We have also been invited to submit an extended version of same for possible publication in the special issue of Applied Network Science covering the conference.

Chapter 2

Background

2.1 Graph Theory

2.1.1 History

The history of graph theory lies in Leonhard Euler's solution to the problem of the seven bridges of Königsberg [3]. The city, in what is now Kalingrad, Russia, lay on what is now called the River Pregolya. It had two large islands in the river, Kneiphof and Lomse. These islands were connected to each other, and the city's mainland, by seven bridges (see Figure 2.1a). The problem was to find a walk through the city, which touched each landmass, and crossed each bridge only once. Landmasses could be revisited if needed.

In 1736, Euler approached this problem from a mathematical point of view. He created *vertices (or nodes)**, to represent the islands, and *edges**, to represent the bridges. The resulting *graph** can be seen in Figure 2.1b. Euler published a paper [4] in which he proposed that there was no solution to the problem.

Euler proposed that in order for there to be a solution to the problem, the graph had to be *connected** and there had to be exactly zero, or two vertices that had an odd *degree**.

*These terms will be explained in section 2.1.2

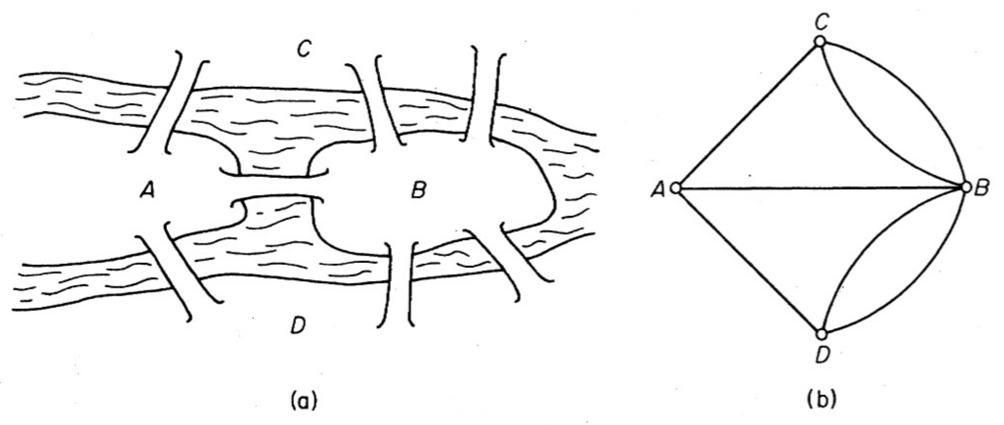


Figure 2.1: The seven bridges of Königsberg and their graph. [1]

This was later proven by Carl Hierholzer in a posthumously published paper [5]. The result of this proof gave rise to the idea of such a path, one which visits each edge exactly once, being named an *Eulerian Path*, in Leonhard Euler's honour.

Graph theory has evolved over the years and has many modern applications. It is used widely in logistics, for example in setting up supply chains [6], and indeed in assessing their vulnerabilities [7].

It has been used in social network analysis since the 1930s and has continued to be used into the modern era of social networks on websites such as Facebook, Twitter etc. [8, 9]. For example, in *Models and Methods in Social Network Analysis* [10] the authors gather the 1930s social network data of one Patrick Ashley Cooper, then a director of the Bank of England, based on his diaries, correspondence, and some film records.

The authors of *When Social Networks Cross Boundaries: A Case Study of Workplace Use of Facebook and Linkedin* [11] explicitly state that "the use of social networking software by professionals is increasing dramatically." As such, it could be considered somewhat easier in modern times to analyse the social network of a person. Rather than trawling through diaries and correspondence, one may simply look at the online social networks that the subject maintains, for example, on Facebook, LinkedIn, Twitter etc.

In *On The Influence of Social Bots in Online Protests* [8] the authors use graph theoretical measurements, in conjunction with textual analysis, to discern whether a particular twitter account is a bot or not. This gives valuable insight into the potential manipulation by foreign agitators in a country's protests, be they social or political.

Recently, graph theory has been used to analyse the 2016 U.S. presidential election. *Analyzing the Digital Traces of Political Manipulation: The 2016 Russian Interference Twitter Campaign* [12] found that "conservatives retweeted Russian trolls significantly more often than liberals" and helped to get Trump elected to the presidency. Graph theory came into the equation in this study when the authors constructed a retweet network, where nodes representing twitter users were linked directly if one user retweeted the other.

Graph theory has also been used in the set up and analysis of networks of information. For example, in *Network Information Flow* [13] the authors look at point to point communication of computer network applications, and in *Information Flow and Cooperative Control of Vehicle Formations* [14] the authors use graph theory to analyse the communications network between a collection of vehicles performing a shared task.

Interestingly, and distinct from the more obvious applications like logistics, graph theory has also been used in the study of disease. In *Graph Theoretical Analysis of Magnetoencephalographic Functional Connectivity in Alzheimer's Disease* [15], the authors use graph theory to create a map of the resting state brain network of patients with Alzheimer's disease. They then use graph theoretical measurements, for example, clustering coefficient, path length, and weighted edges[†], to analyse and model the risk of people developing Alzheimer's disease.

Another biological application of graph theory is to be found in Mason and Verwoerd's review paper, *Graph Theory and Networks in Biology* [16], where, among other things, the authors create a graph where every node is a person in a population, and each edge

[†]As before, these terms will be discussed in section 2.1.2

is a contact through which a particular disease can spread. They utilise numerous graph theory measurements over the course of the publication.

2.1.2 Definitions

Several definitions will be laid out in this section. The definitions will primarily be those that were used during the course of the research. Full details of these definitions can be found in *Modern Graph Theory* [17] by Bollobás, Béla or *Handbook of Graph Theory* [3] by Gross, Jonathan L and Yellen, Jay.

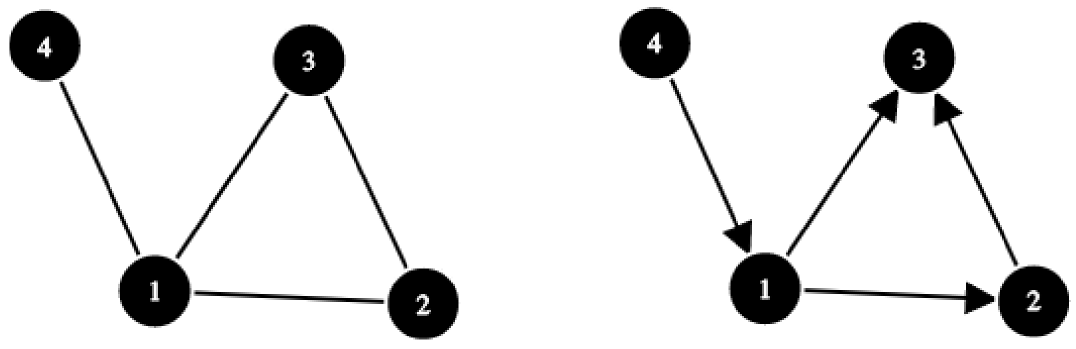
Definition 2.1. A ***graph*** G is an ordered pair of disjoint sets (V, E) such that E is a subset of V^2 of unordered pairs of elements in *V*.

- V is the set of *vertices* or *nodes* and $V(G)$ is the vertex set of G.
- E is the set of *edges* and $E(G)$ is the edge set of G.
- An edge {u,v} is said to *join* the vertices u and v, and if $\{u, v\} \in E(G)$ we say u and v are *adjacent.*

A graph is shown in Figure 2.2a. In this graph, $V(G) = \{1, 2, 3, 4\}$,
and $E(G) = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}\}$

Definition 2.2. A graph, G, is a ***directed*** graph if the edges are ordered pairs of vertices. An edge starting at u and ending at V is said to be ***edge directed*** from u to v and is denoted by $\overrightarrow{uv}$. That is to say that the edge can only be *traversed* in one direction; the direction of the arrow. A directed graph is shown in 2.2b. **Note:** this research will primarily utilise undirected graphs to represent interactions between inventors.

Definition 2.3. A ***weighted graph*** is a graph in which each edge is assigned a real number, called the ***weight*** of the edge. This can be considered the cost of traversing the



(a) A Graph (b) A Directed Graph

Figure 2.2: A Graph; Undirected and Directed

edge. In Figure 2.3, a weighted graph is shown. It can also be seen in this figure that, in some cases, it can cost less to traverse two edges, $1 \to \{1, 3\} \to 3 \to \{3, 2\} \to 2$, than to traverse one edge, $1 \to \{1, 2\} \to 2$.

Also worth noting is that since the weight of the edge could be entirely abstract, the weight of the edges shown do not necessarily correspond to the length of the edges of the visualisation. As such, the triangle inequality may not always hold.

Definition 2.4. We define $G'(V', E')$ as a ***subgraph*** of $G = (V, E)$ if $V' \subset V$ and $E' \subset E$. There exists subgraphs of both directed and undirected graphs.

Definition 2.5. In an undirected graph, the set of vertices adjacent to a vertex $x \in G$ is called the *neighborhood* of x. This is denoted by $\Gamma(x)$. We define the ***degree*** of x, as $d(x) =\mid \Gamma(x) \mid$.

For example, in Figure 2.2a, the degree of node one is 3, or $d(1) = 3$. The degree of the remaining nodes are, $d(2) = 2$, $d(3) = 2$, $d(4) = 1$. A simple way of looking at it is the

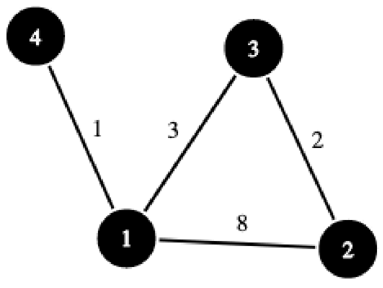


Figure 2.3: A Weighted Graph

number of edges that are incident to a node.

Definition 2.6. In a directed graph, nodes can have ***in-degree*** and ***out-degree***. This is the number of edges directed in to, and out of a node respectively. The in-degree of a node, x, is denoted as $d^-(x)$, and the out-degree is $d^+(x)$.

In Figure 2.2b, the in-degree and out-degree of each node is as follows: $d^-(1) = 1, d^+(1) = 2$; $d^-(2) = 1, d^+(2) = 1$; $d^-(3) = 2, d^+(3) = 0$; $d^-(4) = 0, d^+(4) = 1$;

Definition 2.7. A ***triangle graph*** is an undirected graph with three nodes and three edges that form the shape of a triangle, for example, Figure 2.4 is a triangle graph.

Definition 2.8. The number of ***triangles***, T_v of a node v is defined as the number of triangle graphs formed including that node. For example, in Figure 2.2a, $T_1 = 1$. If nodes 3 and 4 were joined by a single edge, then $T_1 = 2$. Likewise, $T_3 = 1$, however, as above, if nodes 3 and 4 were joined by an edge, $T_3 = 2$.

Definition 2.9. The ***clustering coefficient*** of a node or vertex v is defined as;

$$c_v = \frac{2T_v}{d(v)(d(v)-1)} \tag{2.1}$$

where T_v is the number of triangles formed through node v. In other words, it is the fraction of triangles that a node is part of against all possible triangles in its neighbourhood.

Looking to Figure 2.2a, it can be seen that the clustering coefficients for the nodes are as follows; $1 : 0.\dot{3}$, $2 : 1.0$, $4 : 0.0$, $3 : 1.0$. Taking node number 1 as an example, we can see there is one triangle through that node, involving node 2 and 3. $d(1) = 3$, so $c_1 = \frac{2}{(3)(2)} = 0.\dot{3}$.

Definition 2.10. A ***path*** is a graph P of the form

$$V(P) = \{x_0, x_1, ..., x_n\}, E(P) = \{x_0x_1, x_1x_2, ..., x_{n-1}x_n\} \tag{2.2}$$

Definition 2.11. Given two vertices, u and v, their ***distance***, $d(u, v)$ is defined as the length of the shortest path between u and v. If such a path doesn't exist, we say $d(u, v) = \infty$.

Distance can refer to weighted and unweighted graphs. For example, in Figure 2.2a, $d(4, 2) = 2$, as there are just two edges to traverse; $4 \to \{4, 1\} \to 1 \to \{1, 2\} \to 2$. However, in Figure 2.3, which is the same same shape as in Figure 2.2a, $d(4, 2) = 6$, as we usually consider the weight of the edges when calculating distance.

In an undirected graph $d(u, v) = d(v, u)$, but this is not always the case in a directed graph. Sometimes a node may not be reachable by edges. For example, in Figure 2.2b there is no path from node 3 to any other node. In this case, $d(3, 1) = \infty$, but $d(1, 3) = 1$.

Definition 2.12. The ***diameter*** of a graph is defined as $\max_{x,y} d(x, y)$.

Again, looking to Figure 2.2a, the diameter is 2.

Definition 2.13. The ***radius*** of a graph is defined as $\min_x \max_y d(x, y)$.

The radius of the graph shown in Figure 2.2a is 1.

Definition 2.14. A graph is ***connected*** if for every pair $\{u, v\}$ of distinct vertices there is a path from u to v. Or in other words, the diameter is finite.

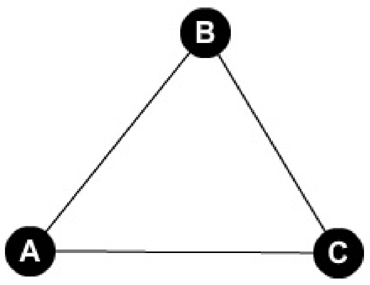


Figure 2.4: A graph with a cycle from $A \rightarrow B \rightarrow C \rightarrow A$

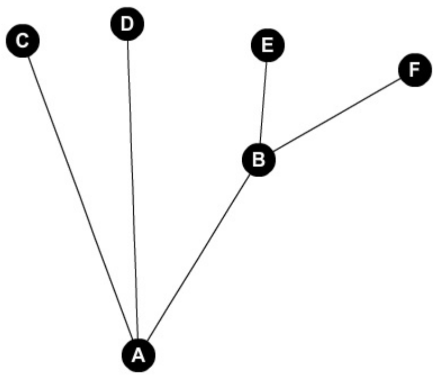


Figure 2.5: A tree

Definition 2.15. A ***cycle*** is a path in which the only vertex visited more than once is the first vertex. It ends at the same vertex it began at. Every other vertex in the cycle is visited only once.

Definition 2.16. A graph is a *forest* if it has no *cycles*. A ***tree*** is defined as a *connected forest.*

Definition 2.17. A ***spanning tree*** of a graph, G, is a subgraph that is a tree which includes all vertices or nodes of G.

Definition 2.18. A ***minimum spanning tree*** of a graph, G, is a spanning tree of G, with the smallest possible sum of edge weights. This can be very useful in the context of team formation.

Definition 2.19. An ***adjacency matrix*** of a simple graph is a square matrix, A, of size $\mid V \mid \times \mid V \mid$, where V is the vertex set of the graph, where the entry A_{ij} is 1 if there is an edge between vertices i and j, and A_{ij} is 0 if there is no edge between vertices i and j.

For example, the adjavency matrix of the graph shown in Figure 2.2a is as follows;

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

2.2 Dijkstra's Algorithm

Part of our research will consist of finding the shortest path between particular inventors, i.e. nodes of a graph. A well known algorithm, named after its creator, Edsger Wybe Dijkstra, will be employed in parts. It is used for finding the shortest distance between any two nodes in a weighted graph. It was first published in 1959 [18] and we have outlined its steps in algorithm 1. Dijkstra's algorithm has many practical applications. It is implemented in *open shortest path first*, which is the most common internet routing protocol in use [19]. This algorithm can be computationally expensive, however, so this will need to be addressed at a later stage in the research, as we may be working with large graphs.

If we use Figure 2.3 as an example, and try to calculate the shortest path from node 4 to node 2, it would go as follows.

Node 4's distance is marked as zero. Nodes 1, 3, and 2's distances are tentatively set to infinity. The distance from node 4 to node 1 is node 4's distance plus the weight of

Algorithm 1: Dijkstra's Algorithm

```
Result: The Shortest Path Between a Starting Node and a Goal Node
Initialisation: Mark all nodes as unvisited;
Set source node $s$ distance to zero, and all other node distances to infinity;
for Current node do
    for Each neighbour of current Node do
        Calculate distance to neighbour through current node;
        Compare this distance to currently assigned distance;
        Assign the smaller value of the two to the neighbour;
    Mark current node as visited, when all neighbours are considered;
    Move to neighbour with smallest distance, mark this neighbour as current node;
```

the edge. Node 4's distance is zero, so the distance is set to 1. This is checked against the current tentative distance, which is infinity. The smaller of the two is chosen. If 1 is smaller than infinity (which it is) the distance for node 1 is set to 1. Since all of node 4's neighbours have now been visited, node 4 is marked as visited, and will not be returned to.

Node 1 will now be considered. From the previous step, we have marked node 1's distance as 1. Node 1's unvisited neighbours are node 3 and 2, which are both still set to tentative distances of infinity. We go to the neighbour which currently has the lowest tentative distance assigned, but in this case they're both the same, so it is irrelevant which is picked. We will pick node 3. The distance from node 1 to node 3 is node 1's distance (1) plus the edge weight. This is 4. We check 4 against node 3's current tentative weight, which is still infinity. Since 4 is less than infinity, node 3's new tentative weight is set to 4.

We now consider node 1's other unvisited neighbour, node 2. Again, node 2's distance is node 1's tentative distance, 1, plus the edge weight, 8, 8+1=9. 9 is compared against node 2's tentative distance, which is still infinity. Since 9 is smaller than infinity, node 2's new tentative distance is 9.

All of node 1's neighbours are now considered, so node 1 is marked is visited, and its weight is set to 1. We will now go to node 1's neighbour with the lowest tentative distance.

Node 3 has a tentative distance of 4, and node 2 has a tentative distance of 9. As such, we will now consider node 3. Node 3's only neighbour which is marked as unvisited is node 2.

Node 2 has a currently has a tentative distance of 9. This will be compared against node 3's distance, 4, plus the edge weight, 2. This gives 6. Since 6 is smaller than 9, node 2's new tentative distance is 6. All of node 3's neighbours have no been considered, so node 3 is marked as visited and its distance is set to 4.

We will now move to node 3's unvisited neighbour which has the lowest tentative distance. In this case, there is only one neighbour left; node 2, with a tentative distance of 6. Node 2 has no unvisited neighbours to consider, so it is marked as visited and its distance is set to its current tentative distance, which is 6.

We have now visited all nodes on the graph in Figure 2.3, and can see that the shortest distance from node 4 to node 2 is 6, and is as follows; $4 \rightarrow \{4, 1\} \rightarrow 1 \rightarrow \{1, 3\} \rightarrow 3 \rightarrow \{3, 2\} \rightarrow 2$.

2.3 The Steiner Tree

Steiner Tree problems, named after Jakob Steiner, are a combinatorial optimisation problem. They typically marry the problems of finding minimal spanning trees with the shortest path problem. For example, in *The Steiner Tree Problem* [20] it is used to find "*a shortest network which spans a given set of points.*"

A very simplified version of the original Steiner tree problem, sometimes called the Euclidean, or geometric Steiner tree problem, arises when there are four towns, each represented by one of the nodes labelled 1-4, each on the corner of a square with side lengths of ten kilometers. We are just using a simple figure of ten kilometers for ease of imagination. Continuing with this experiment, we imagine that there is a limited budget with which to build roads to connect all four towns. As such, connecting the towns using the shortest

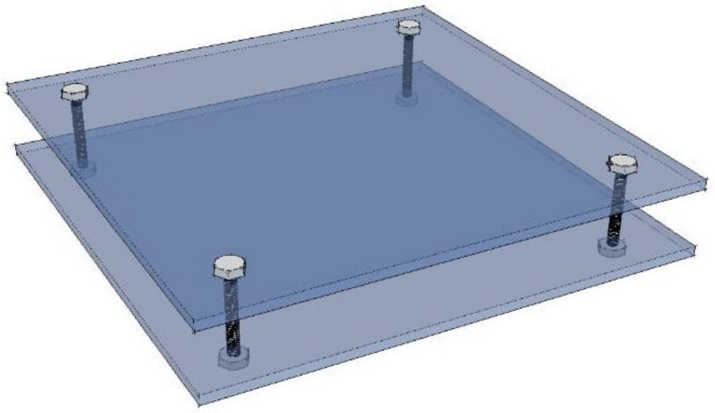


Figure 2.6: Steiner Tree Problem Experimental Setup

length of road possible would be the cheapest option for our builders.

The history of this problem is well described in *On The History of The Euclidean Steiner Tree Problem* [21], by Brazil, Graham, et al. As outlined in that paper, the Steiner tree problem, having been considered as early as 1811, seemed to vanish into obscurity until 1934, when Vojtěch Jarník, and Miloš Kössler, looked at solutions to Steiner tree problems, not only in the plane, but in higher dimensional Euclidean spaces [22]. The consideration of the Steiner tree being applied as mentioned above, to interconnect cities using minimum road length, appears to have arisen first in 1938. French mathematician, Gustave Choquet, wrote an extended abstract which considered this situation [23].

One interesting home experiment that shows a physical solution to this problem is to take two sheets of perspex, and put four screws, or machine bolts, in a square formation through the sheets of perspex such that there is a gap between the two sheets supported by the screws. An example of such a setup is shown in Figure 2.6. If our apparatus is submerged in soapy water and taken out, the soapy film left behind will converge to a locally shortest path between the screws. That path will look like the edges shown in Figure 2.7. The synthetic points that form, S1 and S2, are called Steiner points, or Steiner nodes. Repeated with three nodes, the resulting film should look very like Figure 2.8.

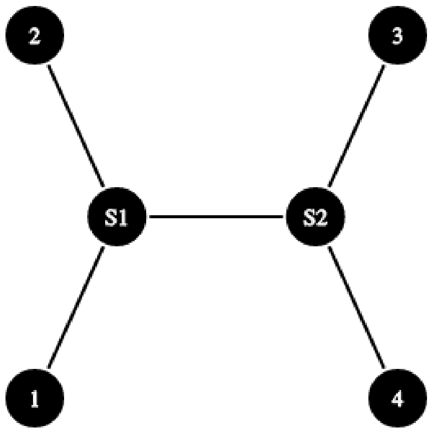


Figure 2.7: Steiner Tree Problem - Solution for Four Nodes With Steiner Nodes S1 and S2

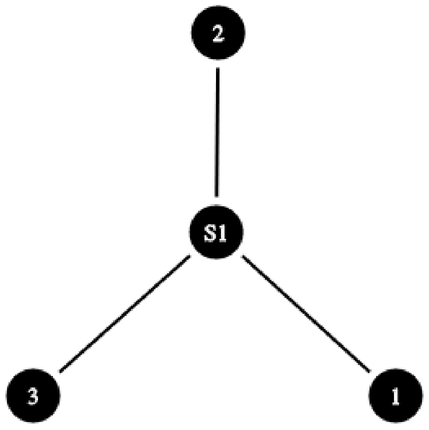


Figure 2.8: Steiner Tree Problem - Solution for Three Nodes With Steiner Node S1

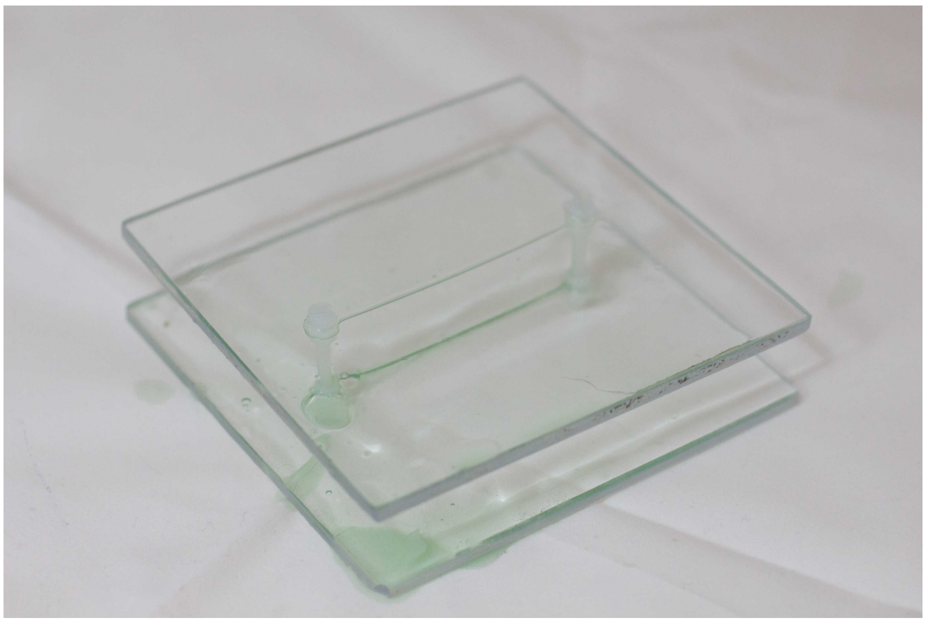


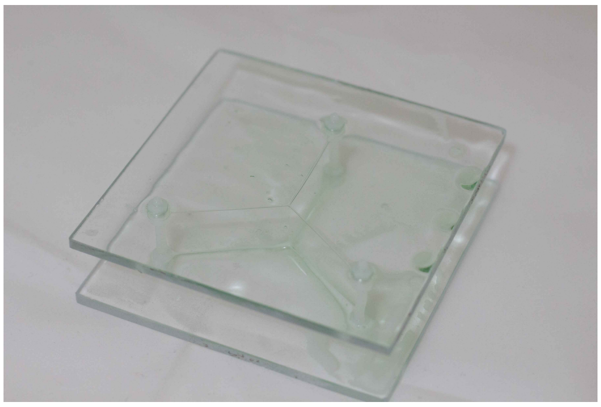
Figure 2.9: Steiner Tree Solution Experiment for Two Nodes

We performed this experiment using setups with two, three, four, five, and six nodes, in linear, triangular, square, pentagonal, and hexagonal arrangements respectively. We also performed one with nodes representing the cities of Ireland; Dublin, Derry, Cork, Galway, Limerick, Waterford, and Belfast.

Unsurprisingly, the solution for the shortest path between two nodes is a straight line, as shown in Figure 2.9. The solution for the shortest paths for three and four nodes are almost exactly as shown in Figure 2.7, and Figure 2.8. The experimental setup and solution for each configuration is shown in Figure 2.10, and Figure 2.11.

The resulting film pattern becomes harder to predict and less intuitive as we increase the number of nodes. The experimental results for five and six nodes are shown in Figures 2.12 and 2.13. It is worth noting that there may not always be a single shortest path solution at these greater number of nodes either.

Finally, we looked at the experimental setup involving nodes representing the cities of Ireland, and the experimental result is shown in Figure 2.14. So, Iarnród Éireann, if you're


(a) Angled View

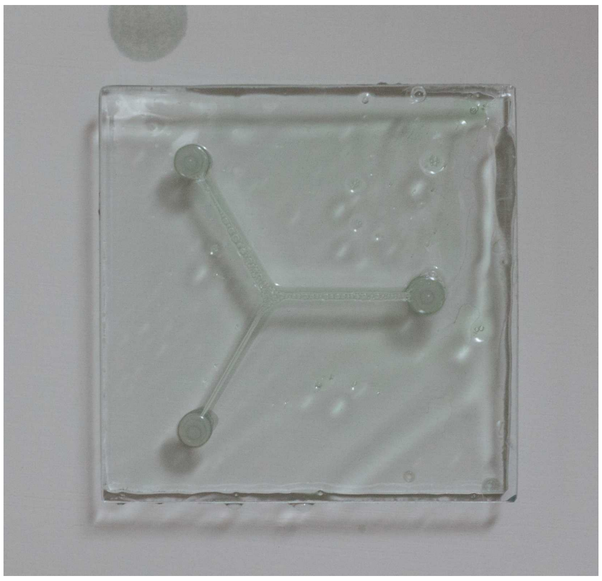
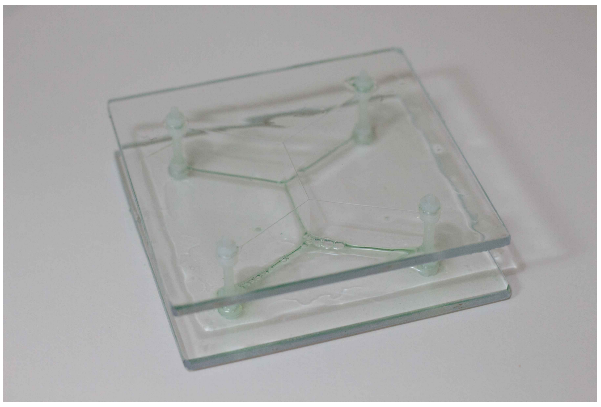

(b) Top View

Figure 2.10: Steiner Tree Solution Experiment for Three Nodes


(a) Angled View

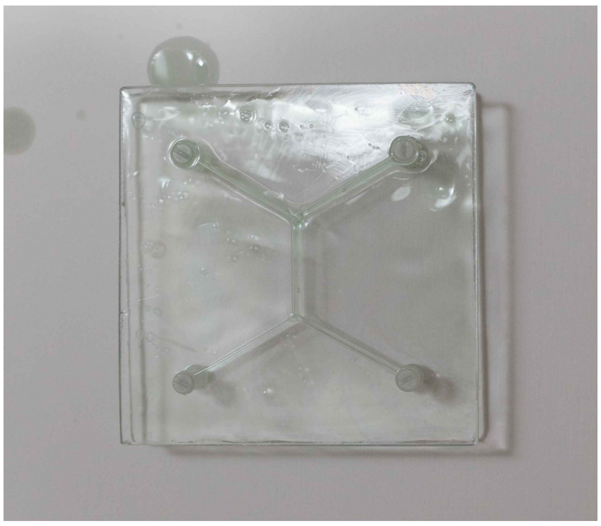
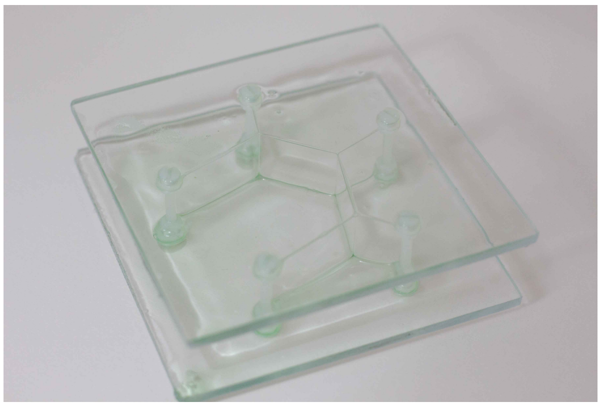

(b) Top View

Figure 2.11: Steiner Tree Solution Experiment for Four Nodes


(a) Angled View

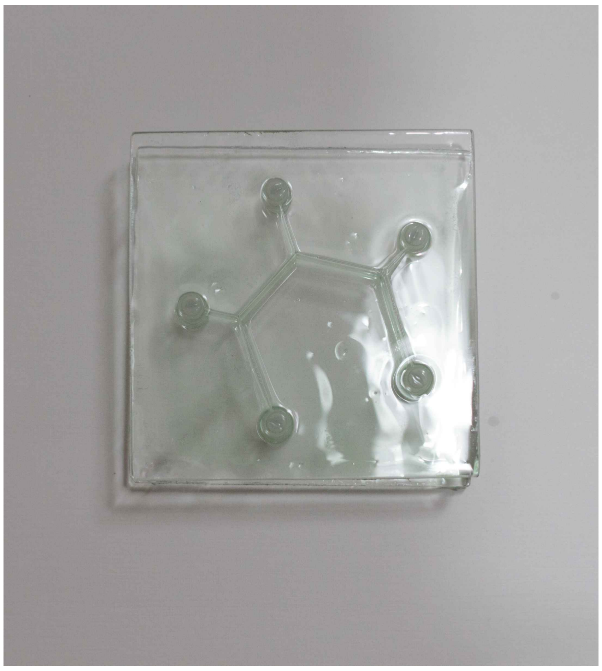
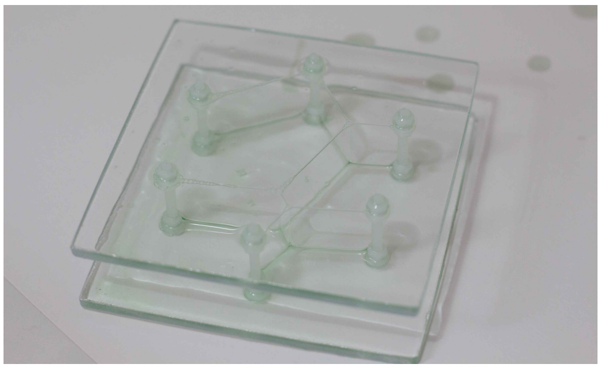

(b) Top View

Figure 2.12: Steiner Tree Solution Experiment for Five Nodes


(a) Angled View

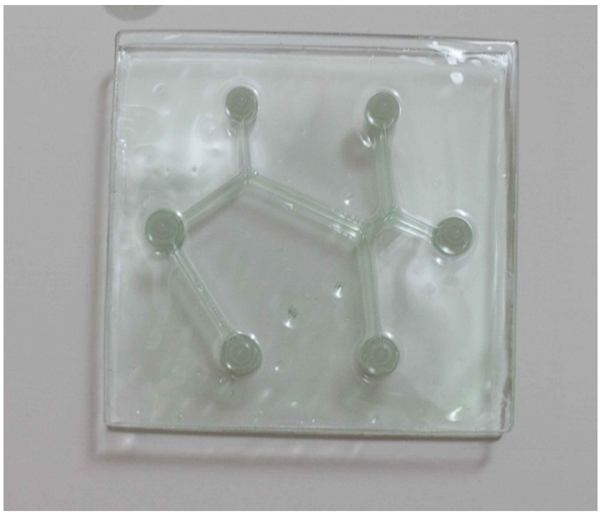
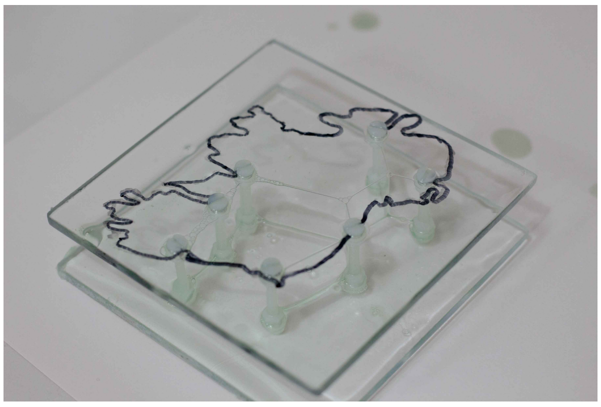
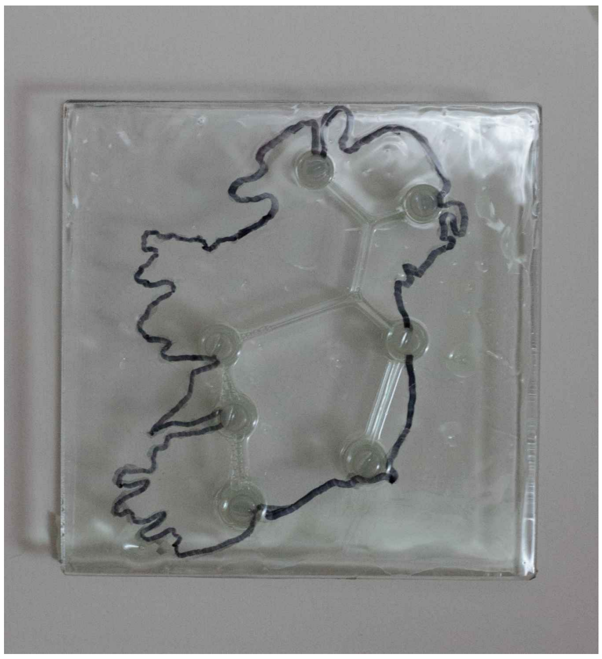

(b) Top View

Figure 2.13: Steiner Tree Solution Experiment for Six Nodes



(a) Angled View



(b) Top View

Figure 2.14: Steiner Tree Solution Experiment for Irish Cities

listening, contact us!

This converging of the soapy film to a shortest path which connects all nodes is due to the principle of minimum energy. The principle of minimum energy is essentially a restatement of the second law of thermodynamics. It states that in a closed system with entropy, and with all external parameters being constant, i.e. no more energy being put into the system, the energy of the system will decrease and approach a minimum value at equilibrium. The energy in a soap film comes from the force which holds the molecules together, the surface tension. As such, and in accordance with the principle of minimum energy, the system's energy will decrease until it is at equilibrium. Since the energy comes from the surface tension, the minimum energy state will be one such that the surface is minimal.

In our case, the Steiner Tree problem will be used to find the smallest number of inventors whose skills cover the skill requirements for an invention.

2.4 Collaboration and Team Formation

The subject of collaboration has been studied, in depth, by many great minds. A cursory Google search of collaboration or team formation will yield thousands of results of papers in journals of various calibres the world over.

There have been numerous methods applied to the problem of team formation across these publications. In this section we will review a selection of works relating to the team formation problem.

The renowned 1977 paper, *The Strength of Weak Ties* [24], by Mark Granovetter deserves particular attention. We will draw upon the strength of weak ties later in the research. In particular, the bridging ability of some inventors, or the ability of some inventors to create a connection between two otherwise unconnected components of a graph. Certain inventors could become Steiner nodes, as discussed in section 2.3. They could provide substantial bridging between communities within our graph structure, and these communities could possess a wealth of knowledge that is relevant to the potential invention.

In *Cross-Domain Collaboration Recommendation* [25], author matching and topic matching systems are proposed, via a random walk algorithm. This is then combined with cross domain topic matching and learning in order to recommend collaborators.

Various topological methods are explored and compared in *A Comparative Study of Team Formation in Social Networks* [26]. These include graph theory measurements and algorithms, such as radius, diameter, and communicative cost based on edge weight over a *Steiner tree*. Based on the criteria above, the authors don't find that any one algorithm performs best across all communicative cost functions. They do, however, find that algorithms based on communicative costs based on sum of distances appear to be able to effectively find teams with low cost.

People also, perhaps unsurprisingly, tend to interact differently on their private social networks and their business social networks. These differences are explored in *Enterprise Social Link Recommendation* [27], where they note that link prediction has been studied extensively on social media sites such as Facebook and Twitter, but not in enterprise social networks, like Yammer.

One of the primary motivations of our work is to form small collaborative teams. In *The relationship between software development team size and software development cost* [28] the authors observe that smaller teams lead to lower communicative and co-ordination costs. They also explicitly state that *"Most of the software development cost is related to the programmers salaries"*. To this end, it seems prudent to recommend the smallest possible teams to cover necessary skills required for given tasks.

More related work will be discussed throughout this thesis as it becomes pertinent to the particular topic being examined.

2.5 Tools

The vast majority of the work was done in Python. Python is a high level programming language. The reasons we used Python are many. Not only does Python have a substantial standard library with many tools for many different things, it also has a large range of importable modules, or packages, for performing specific tasks.

One such feature in the Python standard library allows basic mathematical calculations to be made. For example, the mean, maximum, minimum values of a list are all readily obtainable with a simple function call.

A number of packages were used over the course of the research to address specific problems or perform specific tasks. For example, certain packages exist for dealing with graph theory and graph theoretical calculations. Other packages exist for managing, manipulating, and accessing big data. Others still for training and testing machine learning models.

Some of the packages which featured heavily will be outlined in the coming subsections.

2.5.1 NetworkX

NetworkX[‡] is a Python library for creating, changing, analysing, and visualising graphs. It includes various functions within for computing graph measurements. These include functions to return many of the definitions outlined in section 2.1.2, for example, the degree of a node, subgraphs of the main graph, weights of edges, etc.

It also includes algorithms for finding things like the shortest path between two nodes, or for finding communities within the graph structure. The finding of shortest paths is done by implementing Dijkstra's algorithm, which we discussed in section 2.2. As also mentioned in section 2.2, the algorithm can be computationally expensive.

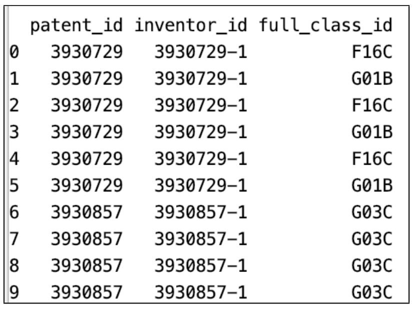
NetworkX will be one of the main packages used over the course of this research. It allows for creation of multigraphs, directed graphs, undirected graphs, weighted graphs etc.

2.5.2 Pandas

Pandas[§] is another Python library which was used extensively over the course of the research. It allows for the manipulation and analysis of data by creating **dataframes**, an example of which is shown in Figure 2.15. A dataframe is like a Microsoft Excel spreadsheet, in that it has rows and columns. Values within these columns can be accessed by referencing the location by position, like using D24 in Excel, or by column and row name, if you have given the columns and rows names.

These dataframes can then be examined in many ways, redundant columns can be dropped, unique values extracted, statistical analyses made, etc.

[‡]https://networkx.github.io/
[§]https://pandas.pydata.org/



```
  patent_id inventor_id full_class_id
0   3930729   3930729-1          F16C
1   3930729   3930729-1          G01B
2   3930729   3930729-1          F16C
3   3930729   3930729-1          G01B
4   3930729   3930729-1          F16C
5   3930729   3930729-1          G01B
6   3930857   3930857-1          G03C
7   3930857   3930857-1          G03C
8   3930857   3930857-1          G03C
9   3930857   3930857-1          G03C
```

Figure 2.15: An Example of a Pandas Dataframe

Importantly, data can be grouped by particular headings. For example, we can group data by inventor id, which would return all data in the dataframe associated with each particular inventor id. This will be very useful for dealing with patents that have multiple inventors, or inventors that hold multiple patents.

2.5.3 SciPy

SciPy¶ has a number of functions for returning various different calculations. It also has functions for returning matrices, and various matrix calculations. Additionally, SciPy is able to deal with large sparse matrices, which can come up a lot when dealing with networks. A sparse matrix is one in which a large amount of the entries are zero. For example, in section 2.1.2 we saw the adjacency matrix, and it is clear that even for a small graph, like that in Figure 2.2a, half of the entries are zero.

2.5.4 Sklearn

Sklearn, or scikit-learn‖ is a library for creating machine learning models. It includes many classification, regression, and clustering models. Its most useful feature over the course of this research was, perhaps, its metrics module. This allowed for the production of confusion

¶https://www.scipy.org/

‖https://scikit-learn.org/stable/

matrices and various other measures of accuracy of machine learning models.

2.5.5 XGBoost

XGBoost is another machine learning package[**]. This one focuses on gradient boosting, which will be used extensively in this research to produce models which will predict links between inventors. This will be discussed further in section 4.4.4.

2.5.6 SMOTE

SMOTE, or synthetic minority oversampling technique, is a method for dealing with skew in data classification, where one particular class appears far more than the other. This can impact the accuracy of a machine learning model. As before, like with XGboost, the concept of data skew, and how SMOTE counteracts it will be discussed further in section 4.4.4. This package will be imported from Python's imblearn[††] module.

2.6 Summary

To recap, we have explored some of the necessary tools and definitions that were used over the course of this research.

A brief history of graph theory was explored, beginning with its creation, when Leonhard Euler proved that the problem of the seven bridges of Königsberg couldn't be solved. We discussed the application of graph theory to more modern, sometimes infrastructural, problems like supply chains and logistics. Also discussed was the application of graph theory at the cutting edge of medicine, for example, in assessing Alzheimer's development risk in patients by mapping their brain networks to graphs.

[**]https://xgboost.readthedocs.io/en/latest/
[††]https://imbalanced-learn.readthedocs.io/en/stable/api.html

We also looked at how graph theory can be used to map how people interact through the years. From looking at diaries, correspondence, and video footage from the 1930s, right up to the modern day social network across various websites.

It can be seen, thus, that graph theory has many applications. These applications include, but are not limited to, logistics, infrastructure, medicine, biology, and in our case, collaboration.

Some of the more formal aspects were then discussed. The definition of a *graph*, the most basic building block of graph theory, was described. The mathematical definitions behind the measurements that we will make use of were given. Terms such as *clustering coefficient*, *triangles*, *adjacency matrix* and others were defined.

Two important aspects of graph theory, Dijkstra's algorithm for shortest paths, and the Steiner tree were also introduced and discussed in more detail. A step by step example of how Dijkstra's algorithm works was provided, as was a physical interpretation of the Steiner tree solution to shortest path problems.

The subject of collaboration and team formation were explored in section 2.4. References were made to previous studies and models that have incorporated graph theory in their approach to team formation.

Finally, having presented the history and applications of graph theory, the mathematical definitions, and some important algorithmic aspects, we looked at some of the tools required to carry out the research. This included a look at the programming language used, Python, and the libraries within Python which were used to address specific requirements.

Chapter 3

Data Source

In this chapter we will explore the patent data used to conduct this research. We will also look at some of the basic graph theory measurements of the inventor network we will create.

3.1 USPTO

The main data sources were the research datasets available on www.uspto.gov. There was also potential access to IBM's internal patent data as part of this research. This could have had the advantage of providing information on different levels of interaction between employees on IBM social network pages, for example blog posts, blog likes, and who follows who on IBM's blog pages. The USPTO datasets were chosen because they are publicly available. As such, using the USPTO data assuaged any concerns about using proprietary data that might arise from using the IBM specific data.

TSVs were downloaded from the USPTO website[*], specifically the files; *patent, ipcr, patent_inventor,* and TSVs relating to company assignments.

Using the company assignee data for IBM, all the IBM patents were extracted using

[*] `http://www.patentsview.org/download/`

regular expressions. They were cross referenced with the other other TSVs and Python's pandas package was used to merge the relevant data to produce a final working data which consisted of the following columns; *patent_id, inventor_id, full_class_id.* The full *full_class_id* used is the International Patent Classification[†] system.

3.2 Basic Statistics

In the *patent* dataset downloaded from the USPTO there are 7, 088, 733 unique patents. These patents are held by 3, 833, 204 different inventors. Mining this vast data was done using pandas (as mentioned in section 2.5.2).

The assignee data downloaded from the USPTO was searched for all text references to IBM. This included its abbreviation, its full name, and other variations of same. This includes, but isn't limited to "*IBM, I.B.M., International Business Machines, IBM Corp, International Business Machines Corporation.*"

From this, pandas was used to extract all unique patent IDs associated with IBM, and to correct any record errors. There are some input errors in the USPTO dataset. This gives rise to some NaN values. After removing these NaN values from the dataset there were 131, 143. These patents are held by 61, 707 different inventors.

The earliest patent date in the dataset was 06/01/1976, and the most recent was 20/08/2019. The number of patents IBM filed in each year is shown in Figure 3.1. This plot only goes to 2018. There are 2019 figures available. As mentioned, the most recent date we have is in August of 2019, but as there is not data for the whole of 2019, including 2019 on the plot might give an incorrect impression regarding the trend of number of patents. For clarity sake, there were 6, 996 IBM patents filed in 2019 up until 20/08/2019. The figure clearly shows an upwards trend. IBM's number of patents filed increased almost

[†]https://www.wipo.int/classifications/ipc/en/

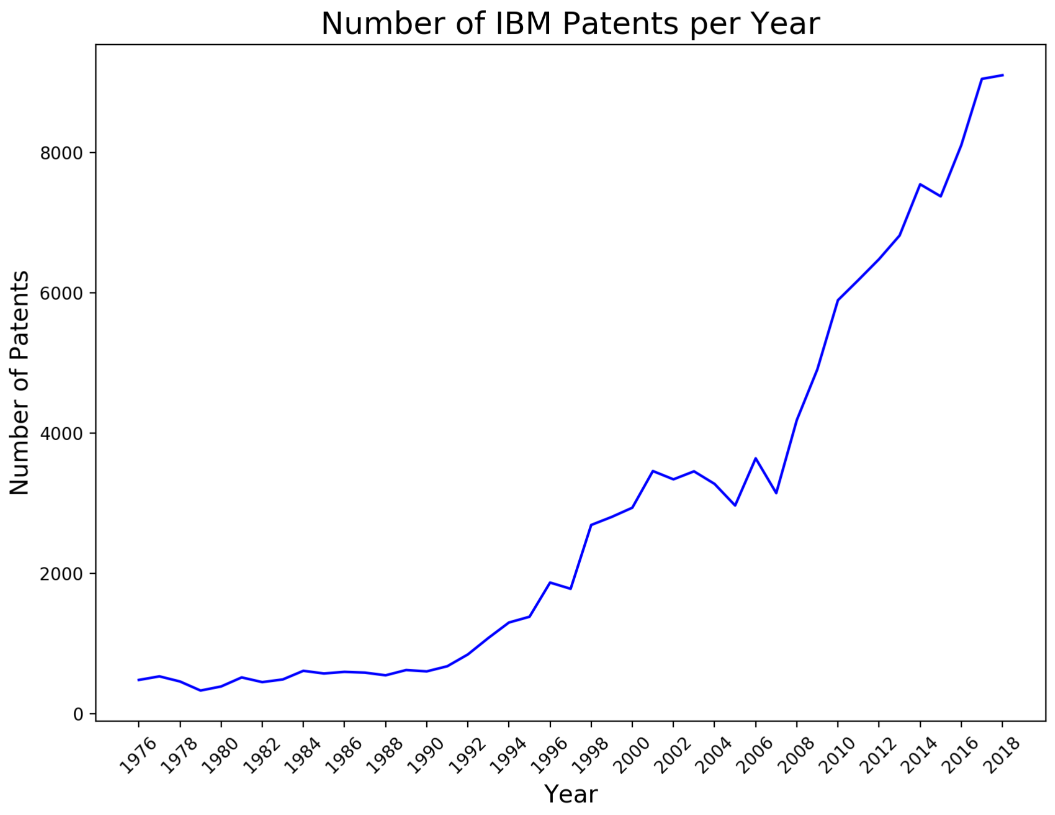


Figure 3.1: IBM Patents over Time

every year except for just a few minor downward dips. In 1976 IBM filed 485 patents. In 2018, IBM filed $9,097$ patents.

The average number of patents held per IBM inventor is 7.07. The median is 2, and the sample standard deviation is 20.81. In Figure 3.2, the distribution of the number of patents held by each inventor is shown. The bin width is 10. It can be seen in the figure that the vast majority of inventors hold a number of patents ranging between 1 and 10. The numbers drop off rapidly and it can be seen that by the time the number of patents nears 40 and above, the number of occurrences becomes negligible. Figure 3.3 shows the same data, but with a bin width of 1, and the number of patents restricted to values below 30. It can be seen that the majority of occurrences are of inventors holding just 1 patent.

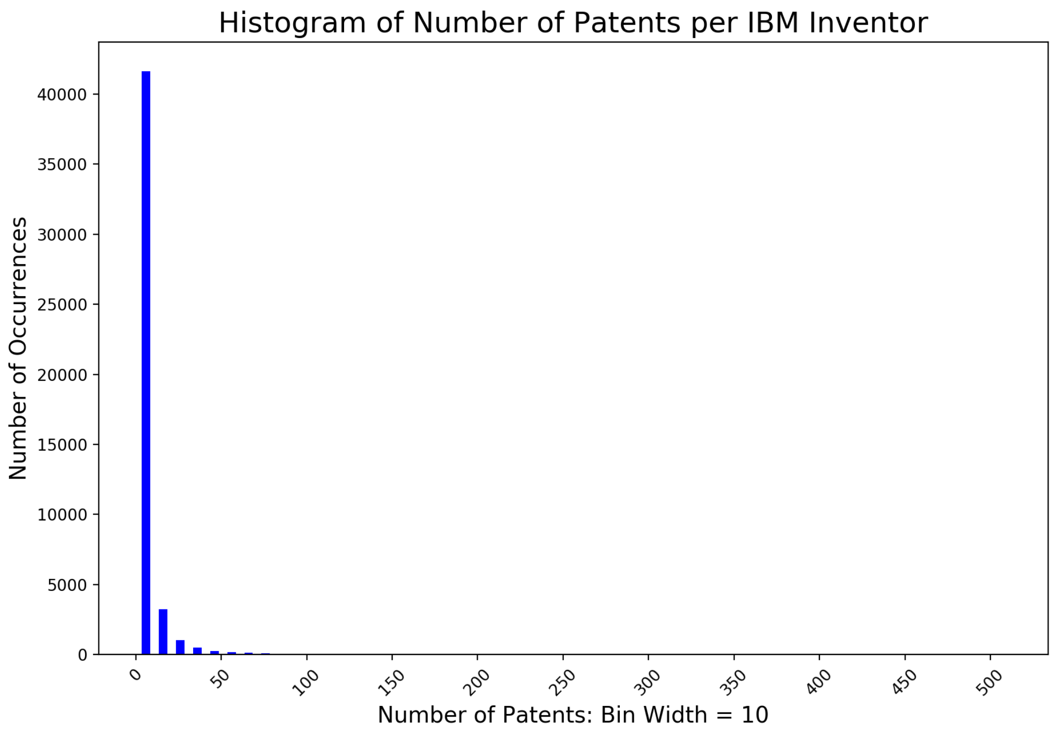


Figure 3.2: Distribution of Number of Patents Held per Inventor

It is worth looking at the whole set of numbers on a log scale, so that we can see what is going on at the tail of the histogram.

The log scale shown in Figure 3.4 gives a better picture of the numbers. There is one IBM inventor that has 510 patents alone. There is one that has 466, and two that have 465, and these are in the same bin towards the upper end of the x-axis. Conversely, at the lower end of the x-axis, there are 18,927 inventors with 1 patent, and 8,176 inventors with 2 patents, etc.

The International Patent Classification system, IPC, has eight sections. They are as follows:

- A: Human Necessities
- B: Performing Operations; Transporting

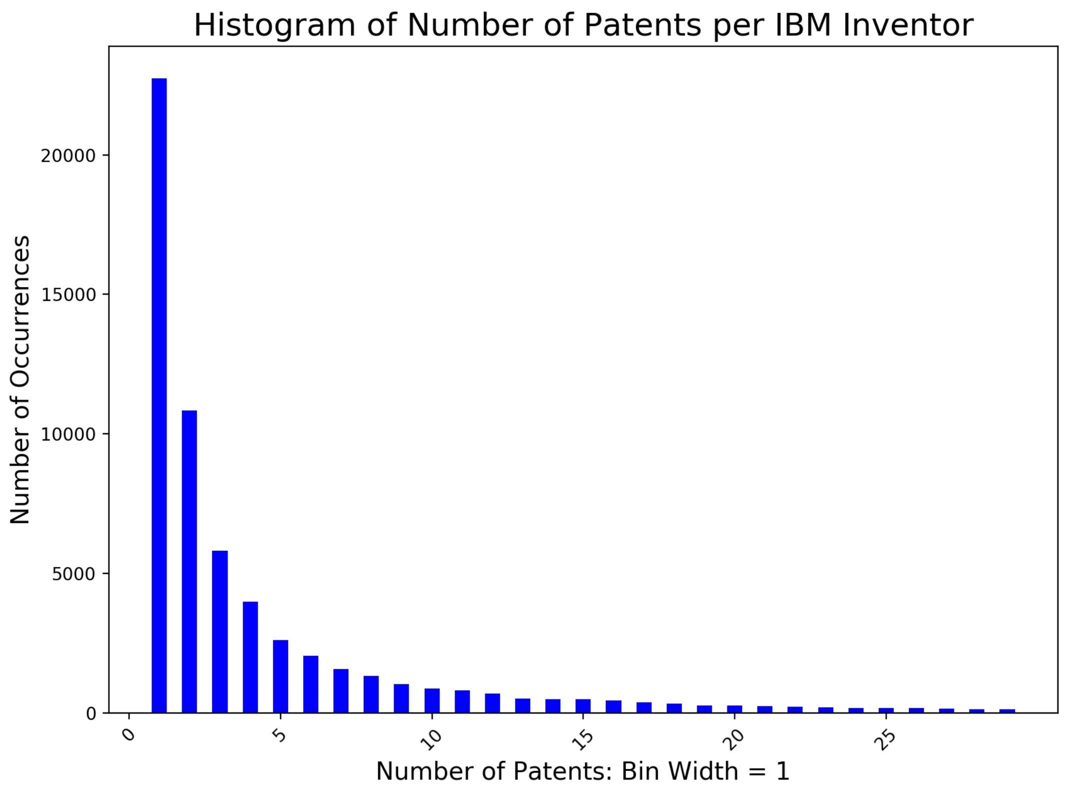


Figure 3.3: Distribution of Number of Patents, N, Held per Inventors where $N < 30$

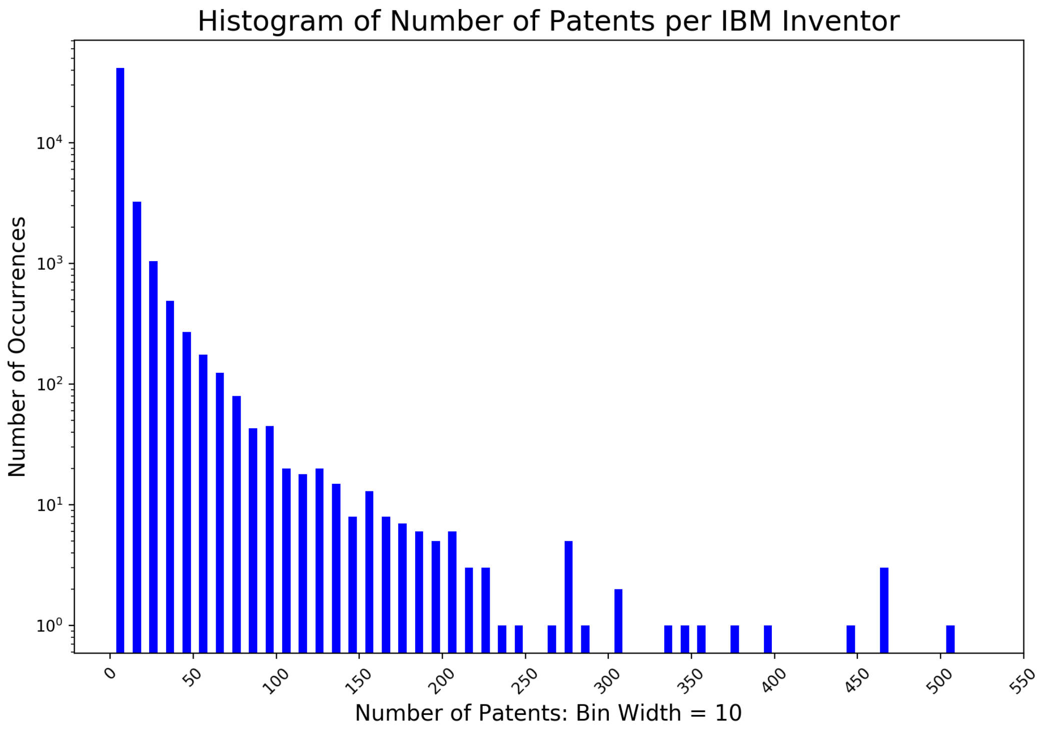


Figure 3.4: Distribution of Number of Patents Held per Inventor - Log Scale

- C: Chemistry; Metallurgy
- D: Textiles; Paper
- E: Fixed Constructions
- F: Mechanical Engineering; Lighting; Heating; Weapons; Blasting
- G: Physics
- H: Electricity

Each of these sections has various sub-classes and classes. We will look at some of them in further detail over the course of the research.

IBM's patents fall primarily in sections G and H, as shown in Figure 3.5, and the exact numbers are as follows:

- A: 992
- B: $5,390$
- C: $2,432$
- D: 283
- E: 162
- F: $1,068$
- G: $92,402$
- H: $45,723$

Summing these numbers gives $148,452$ patents, which is clearly higher than the $131,143$ patents we said IBM have earlier. Some patents can bridge more than one section, hence the discrepancy in the numbers.

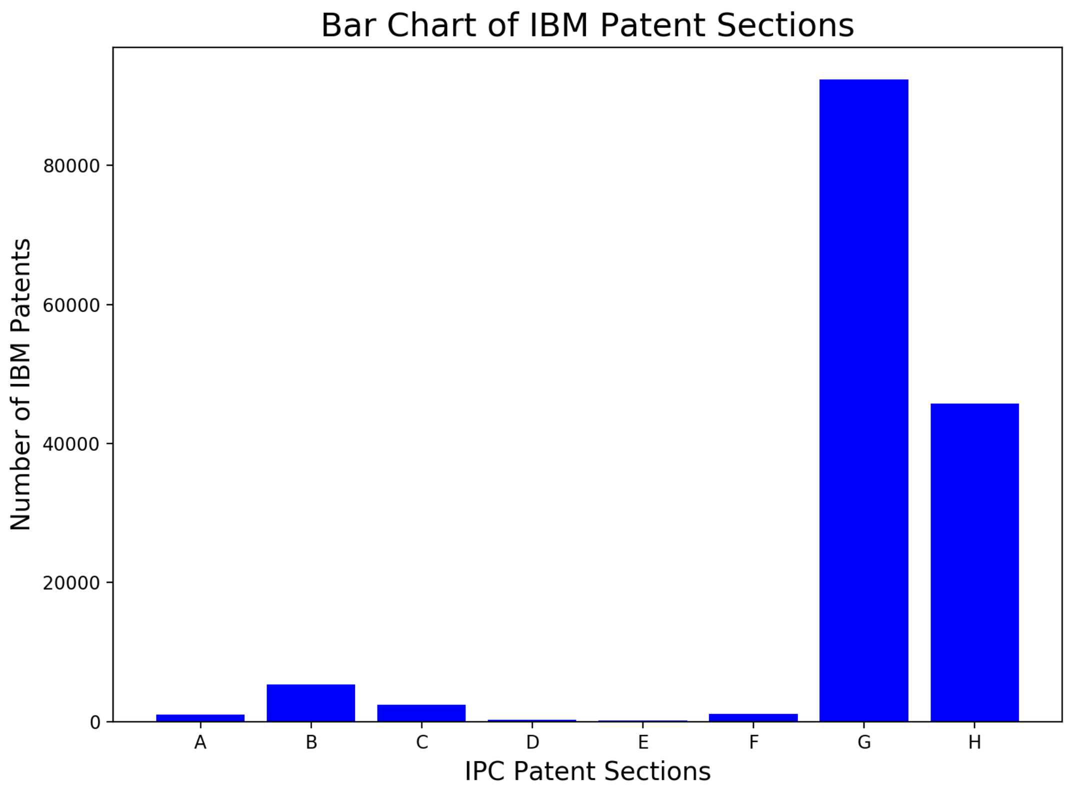


Figure 3.5: Bar Chart of IBM's Patents per IPC Section

The patents associated with IBM are distributed across numerous classes. "*Class*" is the term we will use to refer to the full classification id of a patent, for example *G06F*). In the classification id, G refers to the IPC section, in this case *"Physics"*. 06 refers to the IPC class, which is a subsection of section G, so in this case G06 refers to "*Computing; Calculating or Counting.*" Finally, F is the IPC sub-class, which gives us the full ID G06F, "*Electric Digital Data Processing.*"

Figure 3.6 shows the distribution of IBM's patents across some of these classes. This figure only shows the top 20 classes in which IBM holds patents. There are 638 different classes of IBM patents and many of these classes only have one patent in them. Clearly, as per Figure 3.6, G06F, which as we explained above is defined by the IPC as *"Electric Digital Data Processing,"*, is the class in which IBM holds the vast majority of its patents. This is followed by H01L, *"Semiconductor Devices; Electric Solid State Devices Not Otherwise Provided For."*

Looking at the level of IPC class, the top 20 contains nine classes, spread across two sections. G06, G11, G10 G09, G01, H01, H04, H05, H03. These classes, as defined by the IPC are as follows:

- G06: Computing; Calculating or Counting
- G11: Information Storage
- G10: Musical Instruments; Acoustics
- G09: Educating; Cryptography; Display; Advertising; Seals
- G01: Measuring; Testing
- H01: Basic Electric Elements
- H04: Electric Communication Technique

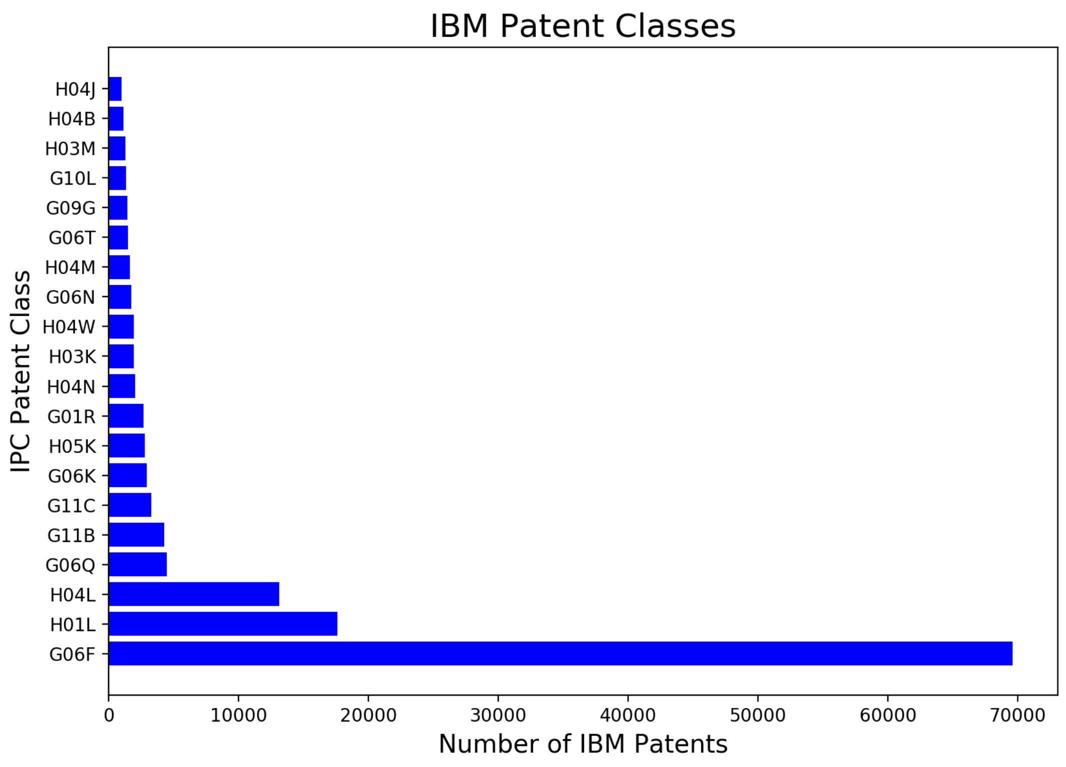


Figure 3.6: Number of IBM Patents in Different Classes

- H05: Electric Techniques Not Otherwise Provided For
- H03: Basic Electronic Circuitry

The definitions of all the IPC patent classes are available on the WIPO website. [‡]

3.3 Graph Creation

A graph, $G = (V, E)$, was created, where the vertex set, V, contained all the inventors of IBM patents, and the edge set, E, contained edges between them representing the existence of at least one patent on which both inventors are named.

[‡]`https://www.wipo.int/classifications/ipc/en/`

Algorithm 2: Creating Collaboration Network

```
Result: A Graph G(V,E) where V is the set of patent holders, and E is the set of
        edges representing collaboration between patent holders
Initialisation: Group the dataframe by unique patent ids, resulting in a Group
  with a title of patent id, and within the group are all of the inventor ids associated
  with that patent;
for Each Group do
    if Patent ID Group Has Only One Unique Inventor then
        Add node with Inventor ID;
    else
        for Each Pair of Inventors do
            if Pair is Not Connected then
                Create nodes for inventors and add edge of weight 1 between
                  Inventors;
            else
                Add 1 to current edge weight;
Invert all weight values;
```

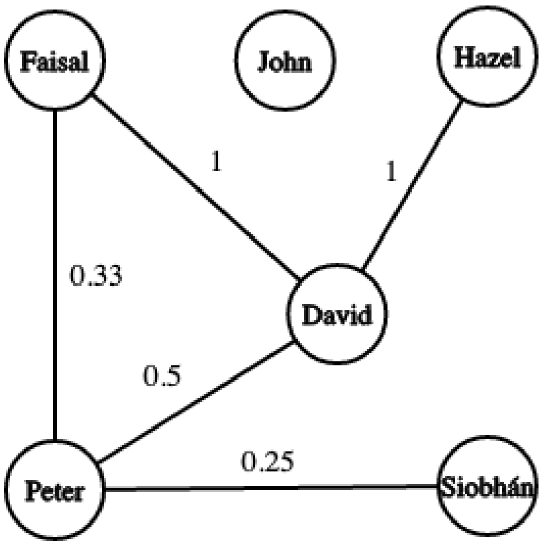


Figure 3.7: What a Graph Created by Algorithm 2 Would Look Like

The procedure for creating the graph was relatively straightforward and is outlined in algorithm 2. Pandas, as discussed in section 2.5.2, was used to group the data as outlined in the initialisation step of algorithm 2. The weighting of the edges and inversion of same means that the higher the weight of an edge, the fewer instances there are of the inventor pair having coauthored a patent in the past. A lower weight implies that the inventor pair have worked together on patents numerous times.

Figure 3.7 shows a very small scale example of what a graph created by algorithm 2 would look like. The number on the edge, as discussed earlier, is the weight of the edge, or the cost of traversing it. Since, in the last step of algorithm 2 all edge weights were inverted, this would imply that Peter and Siobhán have collaborated four times, as $\frac{1}{4} = 0.25$, Faisal and David are shown to have only collaborated once, as $\frac{1}{1} = 1$, and so on.

Python's NetworkX package, mentioned in section 2.5.1, was used for the adding of nodes and edges.

This resulted in a graph, $G = (V, E)$ with $61,707$ nodes (as expected due to the number of IBM inventors), and $284,411$ edges. The average degree of this graph is 9.2181, the median is 5, and the average clustering coefficient is 0.619. So on average, each inventor has collaborated with 9 other inventors. Average figures can be distorted by outliers, however.

Figure 3.8 shows the distribution of degrees among inventors width a bin width of five. It is clear that the majority of nodes have a small degree, somewhere between 0 and 5. By the time the value of the degree gets up past 50 the number of occurrences of such becomes minimal. Figure 3.9 shows the same data, but with a bin width of one, and with the x-axis only going to 50. It can be seen that the degree which occurs most is 3.

As before, with the distribution of number of patents shown in Figure 3.4, we should too examine the whole data set on a log scale. This is shown in Figure 3.10 with a bin width of five, the same bin width as in Figure 3.8.

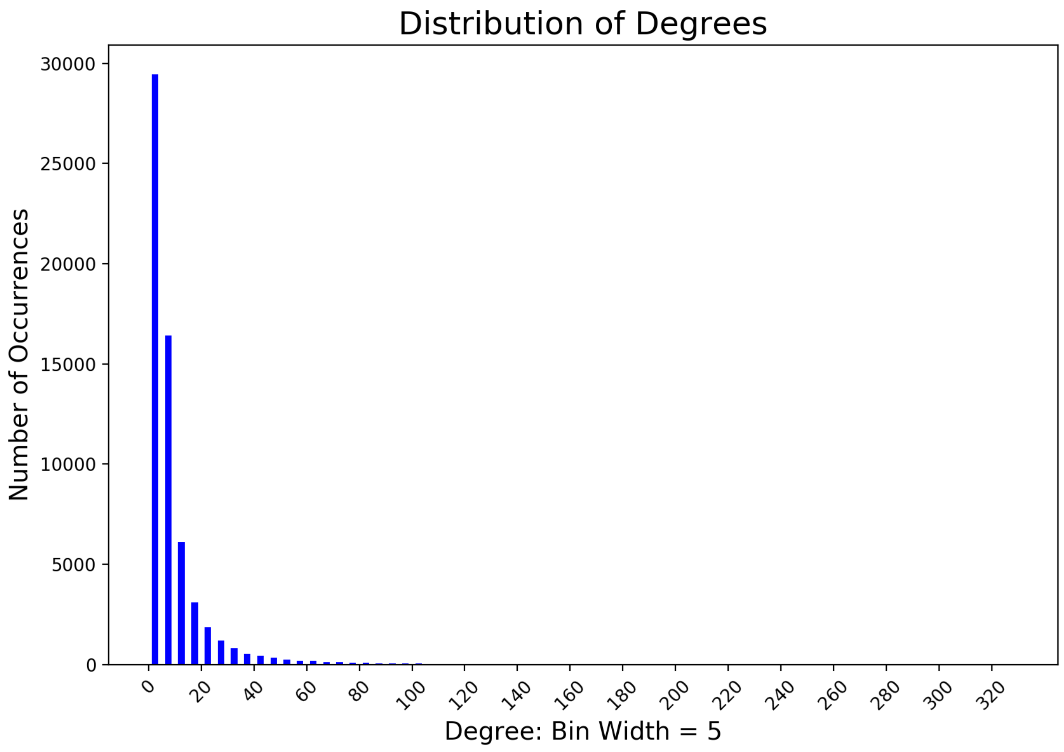


Figure 3.8: Histogram of Degrees of Inventor Network

Interestingly, the majority of occurrences are of degrees between zero and five, which is below the average degree of 9.2181. This suggests that there are, indeed, some inventors with large collaborative networks that are having an effect on the average figure and pulling it up. A cursory look at the degree data shows us that the minimum degree is 0, predictably, as there will be inventors who work alone, and the maximum is 332.

Taking a closer look at the network shows that there are $1,937$ isolated nodes, and $3,362$ connected components. Since every isolated node is considered a connected component, that means there are $1,425$ communities of two or more nodes in our graph. The largest connected component has $55,225$ nodes and $278,782$ edges. Perhaps some of the isolated IBM inventors could flourish if connected to the largest connected component, or even a larger component than their own isolated one.

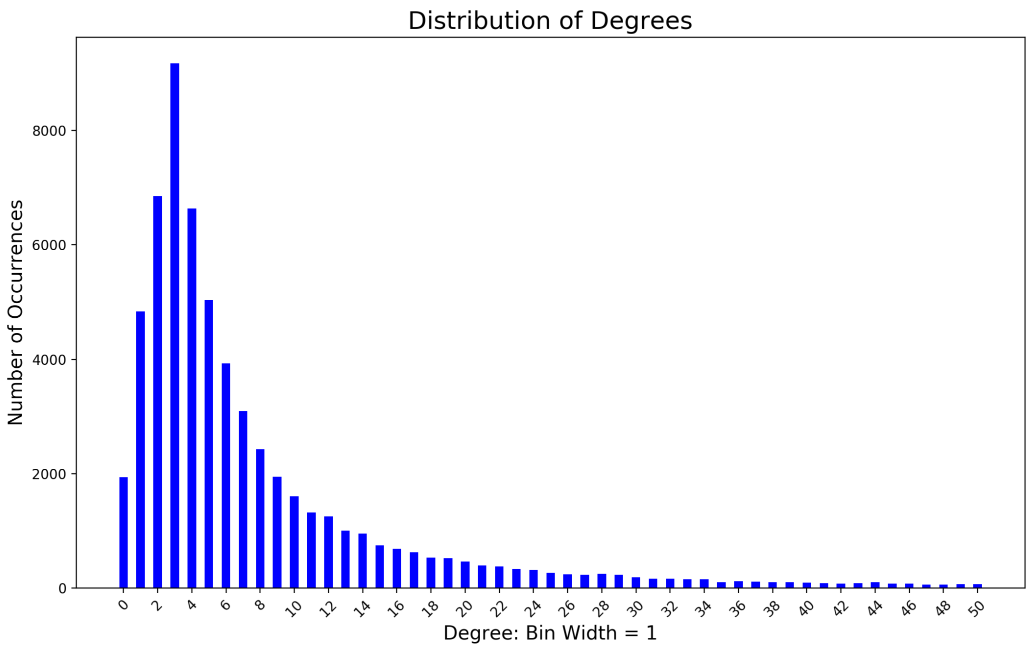


Figure 3.9: Histogram of Degrees of Values < 50

Four of these connected components are seen in Figure 3.11. They were selected by extracting random connected components that had specified numbers of nodes. They are randomly chosen connected components for visualisation purposes. The four connected components shown have five, ten, fifteen, and twenty-five nodes, and each node has an inventor id attached to it. The entire sample has 55 nodes. It can be seen quite clearly in the figure, that the graphs become difficult to visualise as they get larger. Even at 25 nodes, the number of potential edges is ${}^{25}C_2 = 300$, and it can be seen in Figure 3.11 that at higher numbers of edges, it can become quite difficult to draw any inferences from visual inspection of the graphs. As such, we won't be visualising many graphs over the course of this research.

Also worth noting is that Figure 3.11 is a weighted graph, as it is a subgraph of that which was created by algorithm 2. However, given the number of edges, and particularly overlapping edges, putting weights on them in the figure would surely make it a completely

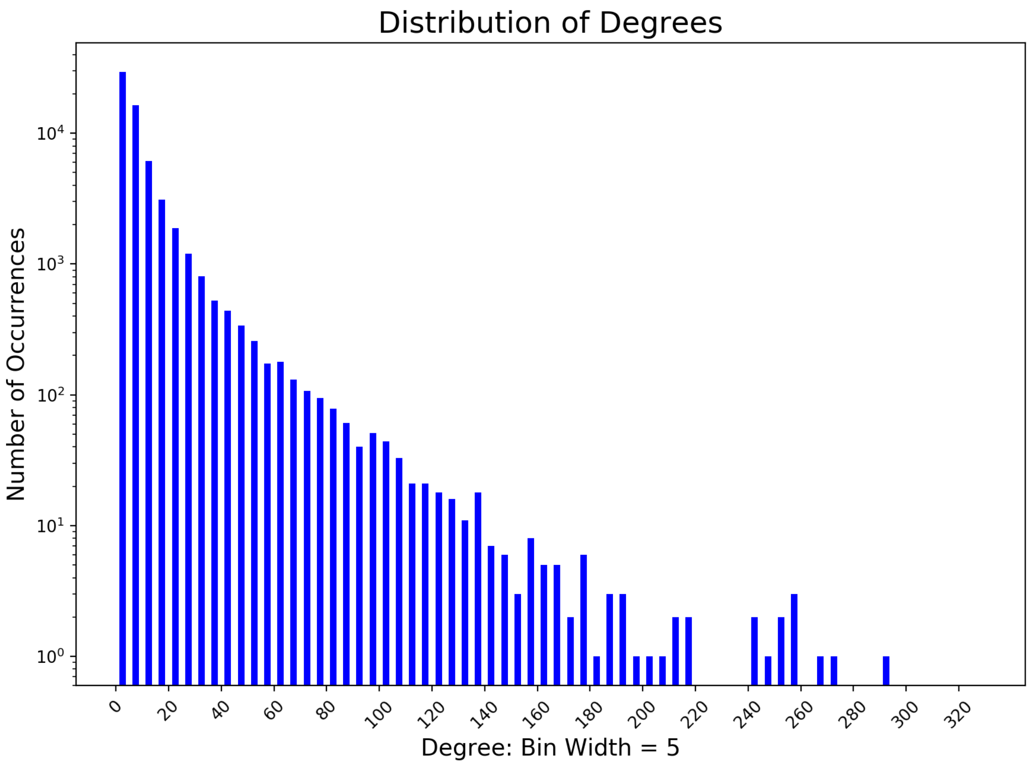


Figure 3.10: Histogram of Degrees of Inventor Network - Log Scale

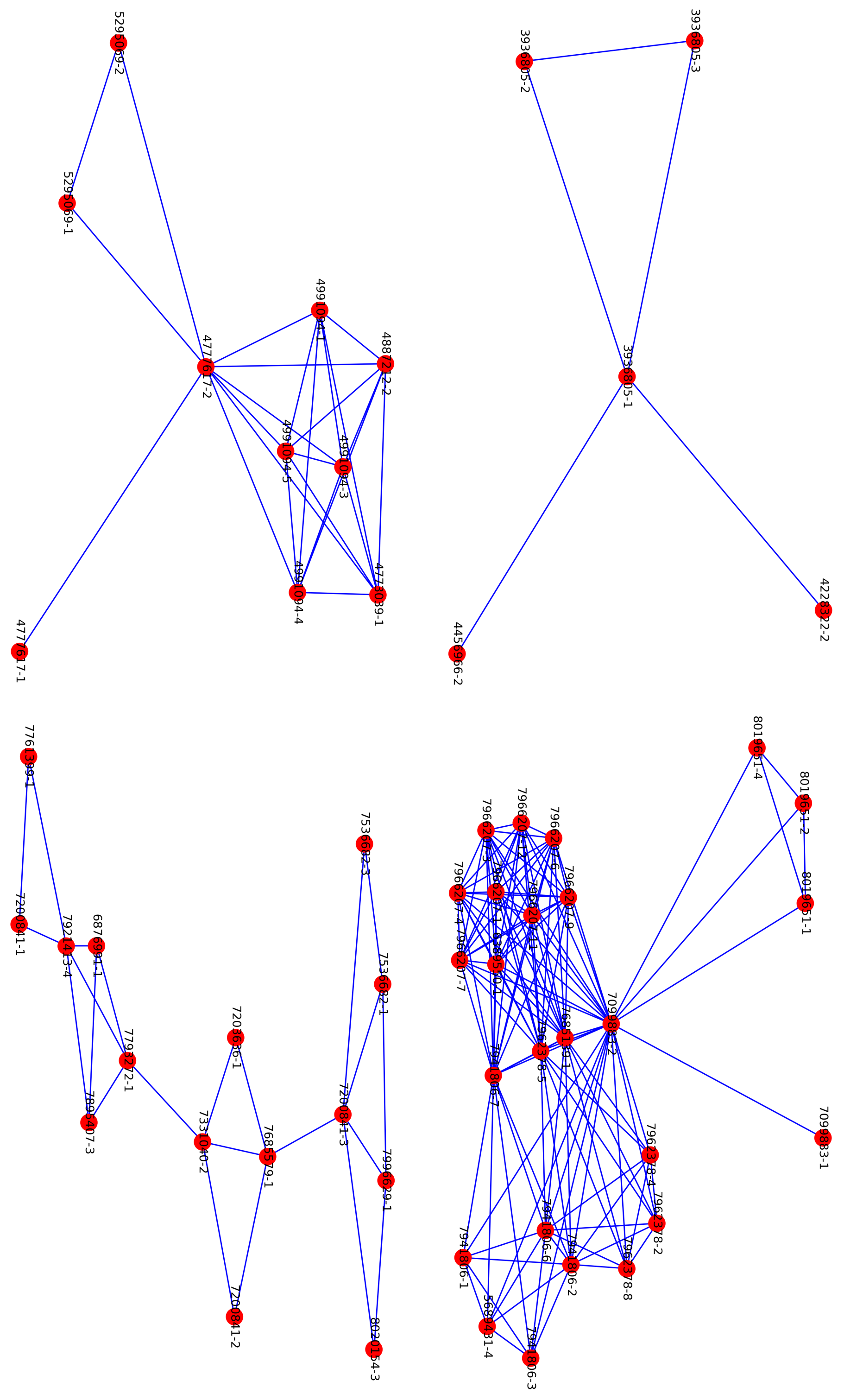


Figure 3.11: Some Connected Components From Inventor Network

illegible mess!

3.4 Summary

Reviewing what we have looked at in chapter 3 shows that we have explored the data source used, extracted some basic statistical analyses from same, and created our collaborator graph.

The reasons for choosing the USPTO website as the data source were explored and rationalised. Primarily that it is publicly available, and that any concerns about using IBM proprietary data were dismissed.

The raw data was briefly looked at, showing over seven million patents in the record held by nearly four million inventors, and the means by which the IBM specific patents were extracted was explained.

The IBM specific data was explored in greater detail. We discovered that there are $131,143$ IBM patents spread out across $61,707$ unique inventors. The time frame of the patents was also revealed to be spanning from 1976 to 2019. A plot was made of the number of patents IBM filed in each year, and a clear upward trend could be seen in that plot (Figure 3.1).

Staying with the IBM specific data, we then looked at how the patents are spread across the IBM inventors. We saw that the average number of patents held by IBM inventors is 7.07 and the median is 2. We also visualised some of the data. In Figures 3.4, 3.3 and 3.2 we looked at the distribution of numbers of patents for each inventor. It could be seen that the most commonly occurring number of patents is 1, but that there are inventors who hold more patents, with one inventor alone holding 510 patents.

Having looked at the distribution of IBM patents across inventors, we then explored the distribution of IBM patents across different topics. We discussed the structure of the

International Patent Classification system, comprising of sections, classes and sub-classes. We looked at what those sections were, and Figures 3.5 and 3.6 visualised the distribution of IBM patents across sections, and the top 20 full class identifications in which IBM holds most of its patents.

Finally, we introduced our collaboration graph. The creation of the graph was outlined in algorithm 3.7. Some visual representations of samples of the graph were displayed in Figure 3.11. The whole graph was analysed and it was found to have $284,411$ edges, and $61,707$ nodes, which was expected as each node represents an IBM inventor. The average degree of the graph was found to be 9.2181 and the median was 5. The distribution of the degrees of each node was displayed in Figures 3.8, 3.9, and 3.10.

Chapter 4

Tackling The Team Formation Problem

In this chapter we will begin by reviewing some literature around the team formation problem. We will look at how others have approached the problem, and we will look at *The Enhanced Steiner Tree*, as first proposed in *"Finding a Team of Experts in Social Networks"* [29] in a bit more detail, as it is the main process we will use to form our teams. Finally, we will move to the design, implementation, and results of our system for team formation.

4.1 Background

Team formation is the problem of identifying a set of individuals with skills that are required by a task for its completion. The concept of the individual inventor has also been researched extensively, with some interesting findings regarding elementary school math grades and the likelihood of someone holding a patent in later life [30].

One can imagine a setting where a stream of incoming tasks requires specific skills, and an automated system is finding such individuals who possess the required skills from an interconnected community, or a collaboration graph. Online social networks (e.g., Facebook, Linkedin, enterprise networking tools etc.) in our personal and professional lives provide us with opportunities to connect with each other and work together on common tasks. At the same time it is quite challenging to find a team of suitable experts. However, there have been approximate solutions to the problem and many of these rely on social or collaboration networks among individuals. The proposed solutions leverage techniques from graph theory, utilising topological features of social and business networks, combined with graph theory to examine the existing structures of teams, interactions, and collaborations within the enterprise [27, 25]. Other works have depended on machine learning, to see what individual characteristics are likely to contribute in forming a link between colleagues and suggest people with these features collaborate [31].

Much of the online team formation work [29, 26] focuses on reducing the communicative cost among potential team members in a network. The *communicative cost* in a network is considered as a measure of how effectively team members can collaborate with each other [29]. There is a correlation between team size and cost, both communicative and financial [32, 28]. One of the problems that will be addressed over the course of this research is producing small, ideally minimal teams, that have the required skills to complete a task. In producing a small team, the aim to keep costs down is satisfied.

In this chapter, a framework will be proposed which incorporates both an individual's attributes as well as topological features from an individual's network into a machine learning link prediction task to improve the Enhanced Steiner Tree algorithm for team formation [29]. The aim here is to use machine learning to produce an augmented graph, containing both real and predicted links, before feeding it into the Enhanced Steiner Tree Algorithm, in order to return a minimal team which covers the necessary skill set. In doing so, the aim is to combine previous work both on link prediction using machine learning

[31] and work carried out using minimal spanning tree solutions [29]. The hope is that by combining the two approaches possible shortcomings of the two will be overcome. The Enhanced Steiner Tree algorithm provides good solutions to the team formation problem, but one possible shortcoming pointed to is that it may sometimes neglect to consider isolated nodes or nodes that sit on unconnected components of the network. As discussed in chapter 3, our graph has many components. Making use of these components could hold potential.

Sometimes, it may be better to recommend collaboration between two inventors who aren't closely connected. There may be a valuable inventor who has all the required skills, and has parameters that are conducive to a good collaborative relationship with at least one member of the potential team. It may be more prudent to recommend this person to the team instead of bringing in many more inventors to cover the skills that this one inventor has. This will help in terms of finding small teams, and keep cost down, because, as mentioned, there is a positive correlation between team size and team cost [28, 32]. For example, if we look back at Figure 3.11, we can see four distinct connected components. An inventor who possesses the skills required to unlock a particular problem could sit on the component with 5 nodes, while the rest of the team could sit on the component with 25 nodes. It may be fruitful to connect an inventor from one component to an inventor from the other, provided they have parameters that suggest potential for a good collaborative relationship. These two inventors will essentially act as a bridge between the components.

To evaluate this scheme, it will be implemented on a collaboration graph consisting of IBM patent inventors. The data is drawn from the US Patent Office (USPTO) data set, whose structure was explained further in chapter 3. The scheme will be evaluated in terms of the team sizes produced.

4.2 Introduction to The Enhanced Steiner Tree

Our model will use the the Enhanced Steiner Tree Algorithm to find a small team of experts whose expertise covers that which is required for a given task. The Enhanced Steiner Tree Algorithm was introduced by Theodoros Lappas, Kun Liu, and Evimaria Terzi in their 2009 paper, *Finding a Team of Experts in Social Networks* [29]. Our research will draw upon this work, and as such, it seems prudent to discuss it with some detail.

They create a graph, $G = (V, E)$, where V is a set of individuals, and E is the set of weighted edges between them. The edges between them represent that they have collaborated in the past, and the weights are interpreted such that a low weight between two individuals implies that they can collaborate or communicate more easily, and a high edge weight implies the opposite.

The finer details of the enhanced Steiner Tree will be laid out in algorithms 5, and 6. The authors of the paper use a few different metrics to measure the success of their algorithm. These metrics include communicative cost and cardinality of (or number of people in) the team. The communicative cost measurements they use on the graph that the algorithm returns are the diameter, and the cost of traversing the minimum spanning tree of the graph.

The authors *"enhance"* their Steiner tree algorithm by creating Steiner nodes which refer to each required skill, and connect inventors to those nodes if and only if they possess that skill. The set of required skills will become known as T. We will propose enhancing that graph even further, augmenting the graph, by adding edges between individuals based on a machine learned algorithm predicting the existence of such.

The authors specifically state, *"In the definition of the Team Formation problem and its specializations, we focused on minimizing the communication cost among team members. Other notions of the "effectiveness" of a team can lead to different optimization functions.*

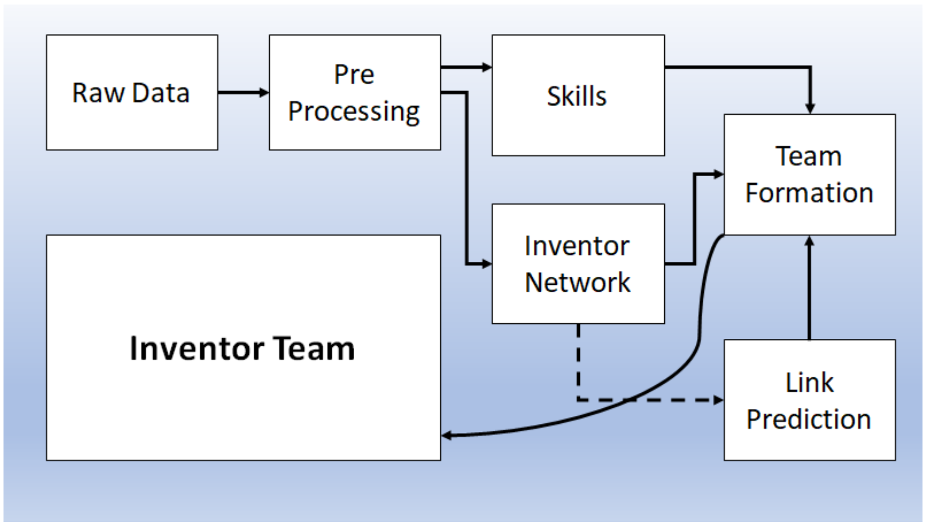


Figure 4.1: Block Diagram of Proposed Framework

For example, if the communication cost was not a concern, we could define as our goal to find $X' \subseteq X$, such that $C(X', T) = T$ and $\mid X' \mid$ is minimized" where $C(X', T)$ is the *cover set* of the individuals X' with respect to task T. The cover set means that at least one individual in X' has the skills to cover the required skills in task T. Given our reference to the correlation of cost of teams with size of teams [32, 28], we will focus primarily on keeping team size down, while covering the necessary skills.

4.3 Proposed Framework

In Figure 4.1 a high level diagram of the full proposed framework for team formation is shown. The block containing *Raw Data* was discussed in chapter 3 and consists of looking at the data gleaned from the USPTO. The *Pre Processing* block was also discussed in the same chapter. This pre processing stage was where we cross referenced the USPTO files to build our input dataframe.

The *Inventor Network* looks at the creation of the collaboration graphs on which the vast majority of analyses will be carried out. *Skills* will discuss the distribution of patent classes amongst different inventors and define expertise in particular classes.

This information will be fed into the *Team Formation* section, along with machine learned edges from the *Link Prediction* block. This element of the proposed framework demands specific attention, as it is the novel aspect of our contribution to the team formation problem. We will use a machine learning model, trained on data acquired from graph theoretical analyses, among other things, to find where there are edges between inventors on a collaboration graph. We will use the false positives, the instances where the model incorrectly predicted there was an edge, to augment the collaboration graph. The resulting output will be an *Inventor Team*.

4.4 Implementation

We will now discuss each of the stages in the implementation of our scheme in greater detail.

4.4.1 Skills & Classes

In order to implement the scheme as seen in Figure 4.1, (having dealt with the raw data and pre-processing in chapter 3) we must take a look at the distribution of patent classes among inventors so that we can see where particular inventors expertises lie. As seen in section 3.3, the average number of patents held by each IBM inventor is 7.07, and this is across 638 different patent classes. As seen in Figure 3.6, one class, G06F, dominates the landscape of IBM patents. As such, an inventor who holds three or four patents in this class may not be considered an expert, because there may be many inventors who have multiples of that in this class. However, an inventor who holds three or four patents in a class where IBM has fewer patents may be considered an expert to IBM, as they may be the most experienced inventor based available to IBM in that particular class.

In order to develop the *skills* section of the framework the distribution of the number of patents held in each class was calculated. In order to do this, a python dictionary with two keys was created. The two keys were *inventor_id* and *full_class_id*. The returned value was the number of patents an inventor held in a particular patent class. Using this data, a python list was created for each patent class. This list consisted of the number of patents held by each inventor in that class. The list was sorted and zeros were removed. After these zeroes were removed a cutoff figure for finding the top quartile of numbers of patents held was established.

We defined anyone holding more patents in a class than its cutoff figure as an expert in that class. For the purpose of the framework, we use the terms class and skill interchangeably.

4.4.2 Inventor Network

As discussed in section 3 the graph of IBM inventors, $G = (V, E)$, has $61,707$ nodes (as expected due to the number of IBM inventors), and $284,411$ edges.

A graph of this size would incur a substantial computational cost for many measurements. Research was carried out into the runtime of network analyses on "real world graphs" by Maher and Malone in *Analysing and Predicting the Runtime of Social Graphs* [33]. We determined that for the purpose of testing the efficiency of our proposed scheme trimming the graph is desirable.

This was achieved by creating a smaller graph broken up by patent classes. To do this, a python dictionary was created with the key being an inventor id, and the returned value being a set containing all classes in which that inventor holds a patent or patents. A number of patent classes are then selected, and using this dictionary, a list of inventors that hold patents in the specified classes is created. The split graph is then created by inducing the subgraph (see definition 2.4) on all nodes in this list. Pseudocode for this process is outlined in algorithm 3.

```
Algorithm 3: Creating Patent Class Split Graphs
Result: A Graph H(V,E) where V is the set of patent holders with patents in the
        specified classes, and E is the set of edges representing collaboration
        between those patent holders
Initialisation: Given the collaboration graph, G, described in algorithm 2, a set
  of patent classes, and the dictionary with inventors as key, and all classes in which
  they hold patents as value;
Create empty list, L;
for Each Patent Class in Set do
    for Each Inventor Key in Dictionary do
        if Patent Class is in Inventor's Set then
            Add node to list, L;
Induce subgraph of G on all nodes in L
```

4.4.3 Creating Machine Learning Training Dataset

For the purpose of training a machine learning model we select some patent classes in which inventors in our dataset hold patents. As an example, we chose H03K and G06K, *Pulse Technique* and *Recognition of Data; Presentation of Data; Record Carriers; Handling Record Carriers*, respectively. These classes don't hold any particular significance other than there exists a sizeable number of IBM patents within them; any classes can be selected. Algorithm 3 was applied to our graph G, with these patent classes to give us a trimmed graph, H.

From analysis of the trimmed collaboration graph H, another dataset was created for the purpose of training a machine learning model. The aim is to produce a set of features that might predict if two inventors might collaborate, even if that is not reflected in the existing patent data set.

Because we need to consider every single possible pair of nodes, the numbers involved become massive quite quickly. This is mitigated by the fact that we trimmed the graph G, as outlined in section 4.4.2. The number of potential pairs of nodes is given by ${}^nC_2$, where n is the number of nodes. In our case, the graph H has 5,698 nodes, and 22,342 edges. Taking every potential pair of inventors from this many nodes gives 16,230,753 possible combinations.

From here the machine learning dataset was created with the following headings: *vi, vq, bonding_vi, bridging_vi, bonding_vq, bridging_vq, vi_collab_ratio, vq_collab_ratio, vi_cluster, vq_cluster, vi_patents, vq_patents, common_neighbors, expert_jaccard, resource_allocation, connected.* We will explain each of these in turn. For each pair of inventors, vi and vq refer to each inventor of the pair. Some of the measurements relate to each inventor individually, and these are denoted by having vi or vq in their name.

The bonding and bridging capital [34] of each inventor was calculated by getting the communities of the graph, using the best partition function in NetworkX's community module, and seeing how many of the neighbors of each inventor are in the same community, and how many reside in different communities. Best partition is a standard function in NetworkX implementing the Louvain algorithm [35]. The bonding capital is given by:

$$\text{Bonding Capital} = \frac{|\text{Neighbors in same community}|}{|\text{Total number of neighbors}|}. \tag{4.1}$$

Conversely, the bridging capital is given by:

$$\text{Bridging Capital} = \frac{|\text{Neighbors } \textbf{not} \text{ in same community}|}{|\text{Total number of neighbors}|}. \tag{4.2}$$

In their paper *Individual-level Social Capital in Weighted and Attributed Social Networks* [34], Sharma, McAreavey, Hong, and Ghaffar discuss at length the importance of social capital. Much of this centres around the ability of a person to bond or to bridge within networks.

The collaborative ratio [36] of each inventor is given by:

$$\text{Collaborative Ratio} = \frac{|\text{Collaborative patents of inventor}|}{|\text{Patents of inventor}|}, \tag{4.3}$$

where a *collaborative patent* is any patent which has more than one inventor. This is intuitively a good metric to use, as a higher collaborative ratio implies the inventor is more open to collaborating with other inventors. For example, if an inventor has 100 patents, and 97 of them are held by the inventor alone, and nobody else, then it may be fair to say that that inventor isn't always open to the idea of working with others. Such an inventor would have a collaborative ratio of 0.03. If we flip the numbers, such that an inventor has 100 patents and only 3 of those patents are held by that inventor alone, and nobody else, then that inventor would have a collaborative ratio of 0.97, and it would be fair to say that such an inventor is open to the idea of working with others.

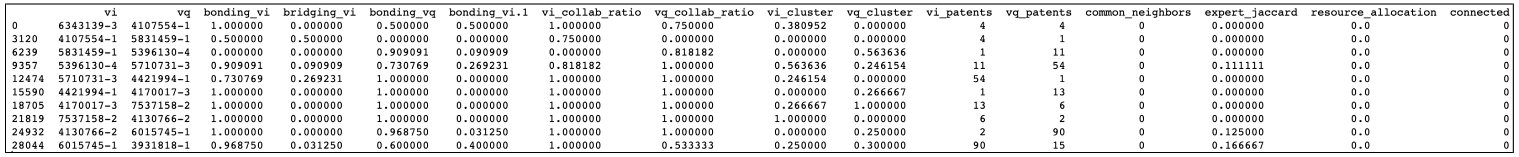
The values vi_cluster and vq_cluster are simply the clustering co-efficient of each inventor. Likewise, vi_patents and vq_patents are simply the total number of patents held by each inventor and common_neighbors is self explanatory.

The Jaccard Index [37], named after the botanist Paul Jaccard, is a well known similarity measure used extensively in research for decades [38, 39, 40]. The Jaccard Index of two sets is given by:

$$J(A, B) = \frac{| A \cap B |}{| A \cup B |}. \tag{4.4}$$

To find the expert_jaccard between two inventors, we take the Jaccard Index of the set of patent classes in which each inventor held expertise. We define expertise by an inventor being in the top quartile of inventors in a patent class, as measured by number of patents held in that class, as seen in section 4.4.1.

Resource allocation is the *resource allocation index* of the pair of nodes as given by Python's NetworkX module. Finally, the connected value is 1 if there is an edge between



```
               vi         vq  bonding_vi  bridging_vi  bonding_vq  bonding_vi.1  vi_collab_ratio  vq_collab_ratio  vi_cluster  vq_cluster  vi_patents  vq_patents  common_neighbors  expert_jaccard  resource_allocation  connected
0      6343139-3  4107554-1    1.000000     0.000000    0.500000      0.500000         1.000000         0.750000    0.380952    0.000000           4           4                 0        0.000000                  0.0          0
3120   4107554-1  5831459-1    0.500000     0.500000    0.000000      0.000000         0.750000         0.000000    0.000000    0.000000           4           1                 0        0.000000                  0.0          0
6239   5831459-1  5396130-4    0.000000     0.000000    0.909091      0.090909         0.000000         0.818182    0.000000    0.563636           1          11                 0        0.000000                  0.0          0
9357   5396130-4  5710731-3    0.909091     0.090909    0.730769      0.269231         0.818182         1.000000    0.563636    0.246154          11          54                 0        0.111111                  0.0          0
12474  5710731-3  4421994-1    0.730769     0.269231    1.000000      0.000000         1.000000         1.000000    0.246154    0.000000          54           1                 0        0.000000                  0.0          0
15590  4421994-1  4170017-3    1.000000     0.000000    1.000000      0.000000         1.000000         1.000000    0.000000    0.266667           1          13                 0        0.000000                  0.0          0
18705  4170017-3  7537158-2    1.000000     0.000000    1.000000      0.000000         1.000000         1.000000    0.266667    1.000000          13           6                 0        0.000000                  0.0          0
21819  7537158-2  4130766-2    1.000000     0.000000    1.000000      0.000000         1.000000         1.000000    1.000000    0.000000           6           2                 0        0.000000                  0.0          0
24932  4130766-2  6015745-1    1.000000     0.000000    0.968750      0.031250         1.000000         1.000000    0.000000    0.250000           2          90                 0        0.125000                  0.0          0
28044  6015745-1  3931818-1    0.968750     0.031250    0.600000      0.400000         1.000000         0.533333    0.250000    0.300000          90          15                 0        0.166667                  0.0          0
```

Figure 4.2: A Snippet of the Machine Learning Dataframe

vi and vq in the collaborator graph, and 0 if there is no edge.

We now have a dataset for a machine learning algorithm which will attempt to predict if two nodes are connected based on the features above. A snippet of this dataset can be seen in Figure 4.2.

4.4.4 Machine Learning Model Training

The Data

The training data consisted of 13 columns from the dataset created in section 4.4.3 and as shown in Figure 4.2. These 13 columns were *bonding_vi, bridging_vi, bonding_vq, bridging_vq, vi_collab_ratio, vq_collab_ratio, vi_cluster, vq_cluster, vi_patents, vq_patents, common_neighbors, expert_jaccard, resource_allocation.* The columns *vi* and *vq* are not included in the training data, as these columns are just identifiers for the inventors. As such, *vi* and *vq* don't provide any information relevant to predicting edges, and it is unnecessary to include them.

The final data needed for the training of the model is the *connected* column. This provides the classification for the model, i.e. the target variable we are trying to predict. In our case it is a simple binary classification. 0 if there is no direct edge between the inventors on our trimmed graph H, 1 if there is. However, machine learning can also be used for multi-class classifications, where there are more than two possible classes, and indeed, multi-label classification, where the final classification can have more than one label associated with it, although we will stick to binary classification alone in this instance.

A common problem with machine learning training arises when there is a skewed data set. As there is no link between most pairs of inventors, the connection column exhibits substantial skew that would impact predictions. As discussed in section 4.4.3, there were $16,230,753$ potential edges. Of these $16,230,753$ potential edges in our trimmed graph

H there were only $22,342$ edges. This means our classification column, *connected*, has $16,208,411$ zeros, and only $22,342$ ones. 99.9% of the classification column is zero.

This means that any potential machine learning algorithm could predict a zero for every instance, or that there are no edges at all in the graph H, and return an accuracy of 99.9%. Clearly, this is problematic, and must be addressed.

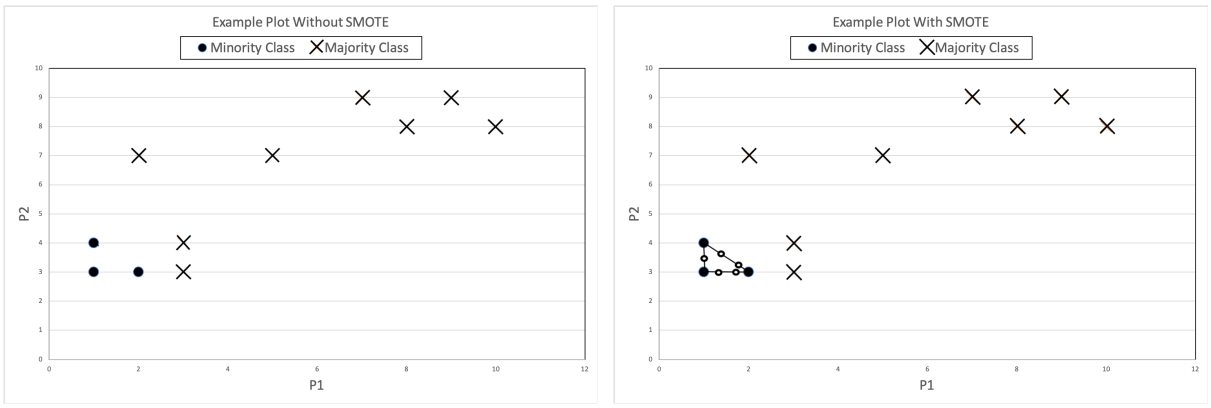
SMOTE

To counteract this, we employ the use of SMOTE [41], synthetic minority oversampling technique, as mentioned in section 2.5, to ensure a more accurate model is trained.

If we imagine a 2D plot, with all the points on this plot representing a value in our classification column. The x-axis, for simplification and explanatory purposes can be imagined as parameter one, and the y-axis can be imagined as parameter two, P1 and P2 respectively. The points fall somewhere on this plot based on their parameters, and are a binary classification, i.e. they can only fall into one of two classes. An example of this proposed plot is shown in Figure 4.3a. In our example plot, the difference between the shape of the points represent which class the point is in.

SMOTE creates lines between the minority class points, and creates synthetic points on on these lines, as shown in Figure 4.3b, with the solid filled circles in the minority class representing our real data, and the unfilled circles in the minority class representing the synthetic points. These points are then used as synthetic data, from which the machine learning model can glean information to improve the efficacy of its training.

In the case of our model, there are thirteen parameters. As such, plotting these parameters would be extremely difficult, as it would have to be done in thirteen dimensions, however, SMOTE handles data with such higher dimensions.



(a) A Skewed Dataset (b) Skewed Dataset Treated With Smote

Figure 4.3: How SMOTE Works

XGboost

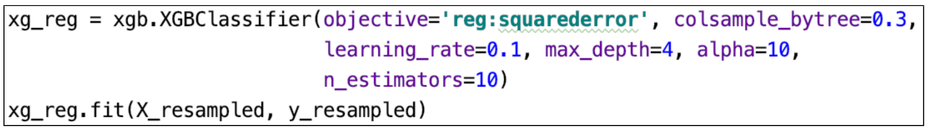
We mentioned XGboost briefly in section 2.5. XGboost is a library in Python that provides gradient boosting methods. Gradient boosting is a technique used in machine learning to produce an ensemble of prediction models, leading to a final classification output.

Prior to settling on XGboost, we experimented briefly with various machine learning techniques. These techniques included, but were not limited to, logistic regression, classification and regression trees, and support vector machine. We used XGboost with SMOTE oversampling as it returned the most accurate prediction model.

The data needs to be fed to the machine learning algorithm in a particular way to allow it to work. The data is first split into two separate pandas dataframes; one consisting of the 13 data columns we discussed, and one consisting of the binary classification. We will call them *data* and *label*, respectively.

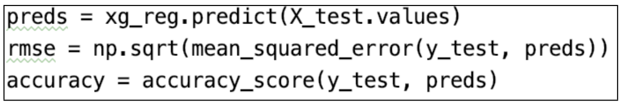
These dataframes are split into train and test sets. This allows us to do a small check on the accuracy of the model, prior to testing it on unseen data. In our case, 20% of the data was retained from training the model for the purpose of doing an accuracy check.

We now have four sets of data. We have *X_train, X_test, y_train*, and *y_test*. *X_train*, and *y_train* are 80% of the *data* and *label* dataframes, respectively, used for training the



```
xg_reg = xgb.XGBClassifier(objective='reg:squarederror', colsample_bytree=0.3,
                           learning_rate=0.1, max_depth=4, alpha=10,
                           n_estimators=10)
xg_reg.fit(X_resampled, y_resampled)
```

Figure 4.4: Snippet of Code Showing XGBoost Parameters



```
preds = xg_reg.predict(X_test.values)
rmse = np.sqrt(mean_squared_error(y_test, preds))
accuracy = accuracy_score(y_test, preds)
```

Figure 4.5: Snippet of Code Showing Call to ML Accuracy Measurements

model. *X_test* and *y_test* are the 20% of the *data* and *label* dataframes, respectively, used for performing a spot check on the accuracy of the model.

The training sets, *X_train*, and *y_train* are resampled using SMOTE, which leaves us with *X_resampled* and *y_resampled*.

We are now ready to train the machine learning model. A snippet of code, shown in Figure 4.4, shows the setting for all of the XGBoost parameters used in our case. The figure also shows that we are fitting the model to to our SMOTE resampled data, as previously explained.

The spot check was performed on this model's predictions against the remaining 20% of the data to get an idea of its accuracy. This was done by calling on the *mean_squared_error* and *accuracy_score* functions within sklearn's metrics module, as shown in Figure 4.5. The values returned for root mean squared error and accuracy were 0.173, and 97% respectively. This bodes well for our model.

4.4.5 Machine Learning Model Testing

Following the training of this model, as outlined in section 4.4.4, a second graph split between patent classes in which IBM holds patents was created, as before and as discussed in 4.4.3. This time the classes used were H04J and G10L; *Multiplex Communication* and *Speech Analysis or Synthesis; Speech Recognition; Speech or Voice Processing; Speech or*

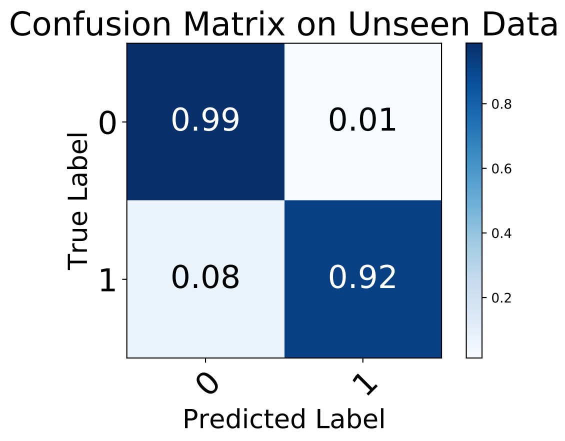


Figure 4.6: Confusion Matrix of Machine Learning Model on Unseen IBM Data

Audio Coding or Decoding, respectively. As before, these classes hold no particular significance other than there exists IBM patents in these classes.

Algorithm 3 was applied to the working graph, G, with these new patent classes, to provide a new unseen graph, J to analyse and test the model on.

A dataset of the same format as in section 4.4.3 was created. The model was tested on this new, unseen, dataset and the predicted *connected* column was compared against the actual one. From this, the accuracy of the model was measured.

Figure 4.6 shows that the model provides good accuracy across all classifications. The model is predicting ones, or there being an edge between inventors, with a 92% accuracy. Conversely, it is incorrectly predicting that there is a zero, or no edge between inventors, 8% of the time. The main area of interest for us is the upper right quadrant of Figure 4.6. Our model correctly predicts that there is no edge between inventors, 99% of the time, and predicting incorrectly that there is an edge between inventors just 1% of the time. This 1% of *false positives* are what we will use to create additional, synthetic links on our graph, J.

```
Algorithm 4: Creating Machine Learning Augmented Graph
  Result: An Augmented Graph with Machine Learning False Positive Edges added
  Initialisation: Gather all pairs of nodes between which there was an edge
    predicted by the machine learning model but no edge in the collaboration graph;
  for Each Pair of Nodes do
      Add edge between them;
```

4.4.6 Creating New Links

Having built a model to predict links, we are now ready to introduce the graph that is augmented with extra edges that are predicted by that model. This is to overcome a potential limitation of using the Enhanced Steiner Tree Algorithm [29] directly on the collaboration graph, in that it may not consider collaborations that sit in unconnected components.

As such, we propose that artificial links are created on the graph between pairs of inventors for which our machine learning model incorrectly predicted there was a link, or the 1% as seen in the top right quadrant of Figure 4.6. The logic here is that the machine learning model predicted, based on the parameters discussed in section 4.4.3, that these two inventors had attributes that were considered to be conducive to a collaborative relationship. This procedure is outlined in algorithm 4.

The graph was augmented by creating an edge between every pair of inventors for which the model predicted an edge, but there didn't already exist such an edge. We also need to provide a weight for the edge. The machine learning model returns a probability, P, of there being an edge between a pair of inventors. If $P >= 0.5$ then the model returns a 1, or an edge. If $P < 0.5$ then the model returns a 0, or no edge. The edge weights were set as $1 - P$ where P was the probability of an edge existing as predicted by the model. This weight is applied to all edges, including those included in the original collaboration graph.

This provides us with our augmented graph on which we will run the Enhanced Steiner Tree algorithm to identify a team.

4.4.7 The Enhanced Steiner Tree Algorithm

The Enhanced Steiner Tree Algorithm is outlined, almost exactly as its authors did so in their paper, in algorithm 5.

Algorithm 5: Steiner Tree Algorithm

```
**Result:** Team $X_0 \subseteq X' \subseteq X$ and subgraph $G[X']$
**Input:** Graph $G(V, E)$;
required nodes $X_0$ and Steiner nodes $X \setminus X_0$;
$X' \leftarrow v$ where $v$ is a random node from $X_0$;
**while** $X_o \setminus X' \neq \emptyset$ **do**
    $v* \leftarrow \arg\min_{u \in X_0 \setminus X'} d(u, X')$
    **if** $Path(v*, X') \neq \emptyset$ **then**
        $X' \leftarrow X' \cup \{Path(v*, X')\}$
    **else**
        Return Failure
```

Viewing the algorithm raises the obvious questions, what are our Steiner nodes and our required nodes? Recall, that in section 4.4.1, we defined an inventor as being an expert in a particular patent class if they were in the top quartile of inventors in that class based on the number of patents they hold in that class.

Our call on the algorithm will be as described in algorithm 6.

Algorithm 6: Enhanced Steiner Tree Algorithm

```
**Result:** Team $X' \subseteq X$ and subgraph $G[X']$
**Input:** Graph $G(V, E)$;
Individuals' expertise set $\{X_1, ..., X_n\}$ and task $T$;
$H \leftarrow EnhanceGraph(G, T)$
$X_H \leftarrow SteinerTree(H, \{Y_1, ..., Y_k\})$
$X' \leftarrow X_H \setminus \{Y_1, ..., Y_k\}$
```

The task T, mentioned in algorithm 6, is a set and will consist of various patent classes. These classes can be considered the skills needed for a particular invention. For example, if we have an idea that involves electronic digital data, it would make sense to include the class G06F which, as mentioned in 3, is *Electric Digital Data Processing*, and other similar patent classes in our task T. The line, *"EnhanceGraph(G,T)"* in algorithm 6 makes a pass over the graph G. An additional node Y_j is created for every skill in T and these nodes are edge connected to an inventor if and only if that inventor has expertise in that skill. The distance between inventors and these new nodes is set to be greater than the sum of pairwise distances of nodes in G.

This raises the issue of calculating the distance between every single pair of nodes in the graph. As discussed, in section 4.4.3, a graph with as few as $2,128$ has $2,263,128$ potential pairs. Given how computationally expensive it is to run Dijkstra's algorithm in NetworkX, and every function in NetworkX for finding paths calls on it at some stage, we need to find another way.

Thankfully, all edge weights were set to be $1-P$, where P was the probability of there being an edge between inventors as per the machine learning model. Since probabilities are always numbers between zero and one, we know that the sum of pairwise distances of our graph can never be greater than ${}^nC_2$, where n is the number of nodes in the graph.

As an aside, initially, when G was created, and before we used machine learning probabilities, each additional collaboration between inventors added 1 weight to their edge. So if two inventors had collaborated 20 times their edge weight would be 20. A lower weight would have been considered a weaker collaboration. This seems intuitively incorrect however, and the decision was made to invert all edge weights at the end of algorithm 2. This way, a lower edge weight implies more chance of collaboration.

4.4.8 Testing

Given the relation between team size and cost, the main criterion for testing was the team size necessary for the given skills.

This raised the obvious point that, having added in edges predicted by the machine learning model, the cardinality of the team would almost always be smaller, as the graph

Algorithm 7: Creating Randomly Augmented Graph

```
Result: A Graph with Random edges added
Initialisation: Gather all pairs of nodes between which there while $\#Edges\ Random\ Graph < \#Edges\ ML\ Augmented\ Graph$ do
    Choose random pair of unconnected nodes and add edge between them;
```

is now better connected. In fact, even if we added edges at random, we would expect teams to be smaller. We take advantage of this intuition, and use teams generated from a randomly augmented graph as a comparison. To generate this randomly augmented graph, we add edges at random between pairs of nodes until the number of edges of this randomly augmented graph equals the number of edges of the machine learning augmented graph. This is outlined in algorithm 7.

In the evaluation we will compare the average team size required for different sets of required skills. Since there are many skillsets, we compare the average team size as a function of the number of skills required for a particular task T.

Specifically, a task T is a set of patent classes that could be interpreted as the skillset required for the task at hand. Recall, in section 2.4 we mentioned that the authors of *Patent Partner Recommendation in Enterprise Social Networks* [36] state that refining recommendations by subtopics could be useful. Since our task, T is made up of patent classes, and we have defined experts in those classes, this is our way of refining the recommendations based on topics. For this task, the the Enhanced Steiner Tree algorithm [29] can be run on (1) the original graph, (2) the machine learning augmented graph, and (3) the randomly augmented graph.

4.5 Results

This evaluation was initially performed by selecting 10 patent classes at random. The algorithm was run 100 times, and the average team size recorded. This was repeated for

Task	G	G Random-Augmented	G ML-Augmented
T_1	7.84	7.82	**6.52**
T_2	8.79	6.41	**6.11**
T_3	6.99	5.5	**4.48**
T_4	13.33	11.23	**8.4**

Table 4.1: Average Cardinality of Teams for Task of size 10 - IBM

four different random tasks, T, of size 10.

Table 4.1 shows the results for these four initial tests performed on differing tasks T, all of size 10. Our augmented graph returns a smaller team than the original collaboration graph, as expected, but also consistently returns a smaller team than the graph with randomly created links, even though the random graph and augmented graph both have the same number of edges.

Following that, a random T was selected as before, but with a size of 8. The same tests were again run, in this case 50 times, and the average number of members in each team recorded. T was increased by 1 random element after each iteration until the size of T was 24 patent classes.

The results displayed in Figure 4.7 show the outcome of the tests for differing numbers of patent classes in task T. The plot shows the number of elements in a task T on the x-axis, and the average number of team members returned having run the Enhanced Steiner Tree algorithm 50 times on all three graphs.

The average cardinality of the team returned for the augmented graph is lower than that of the original collaboration graph, intuitively and as we discussed in Section 4.4.8. More interestingly, the average cardinality of the team returned when the Enhanced Steiner Tree algorithm is run on the machine learning augmented graph is consistently lower than the graph which was augmented with random links, even though the random graph has the same number of edges as the machine learning augmented graph.

This result is replicated for nine other random tasks T as shown in Figure 4.8. In this

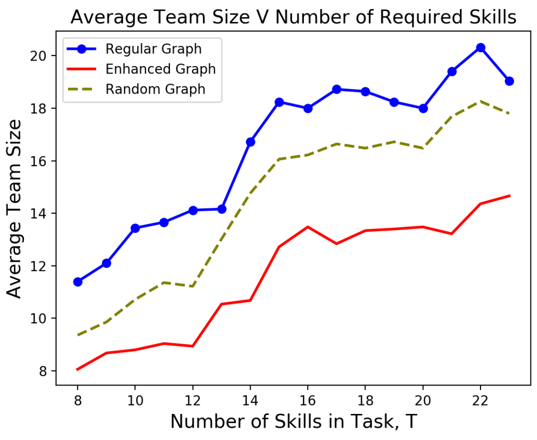


Figure 4.7: Average Cardinality of Team V Number of Skills in task, T

table, each graph follows the same legend and has the same axes as the graph shown in Figure 4.7, although some of the y-axis scales differ. This shows that the machine learning augmented graph generally returns a smaller team, than that returned from the randomly augmented graph and the regular graph, which covers the skillset across differing tasks of differing sizes.

We repeated the same experiment, with the same testing procedure on data from Samsung patents. The Samsung collaboration graph had $46,268$ nodes, and $252,548$ edges. It had an average degree of 10.9167, and an average clustering coefficient of 0.553. This compares quite closely to the IBM collaboration graph figures for average degree and average clustering coefficient; 9.2181 and 0.619 respectively. The confusion matrix for the Samsung graph is shown in Figure 4.9, and it can be seen that the machine learning model has returned good accuracy again.

The results for these tests are shown in table 4.2 and Figure 4.10. It can be seen in Figure 4.10 that our machine learning augmented graph consistently returned smaller teams, as it had also done for the IBM patents.

While the team size is an important factor, one might also want to consider the communicative cost. One crude way to represent this would be the sum of weights of the

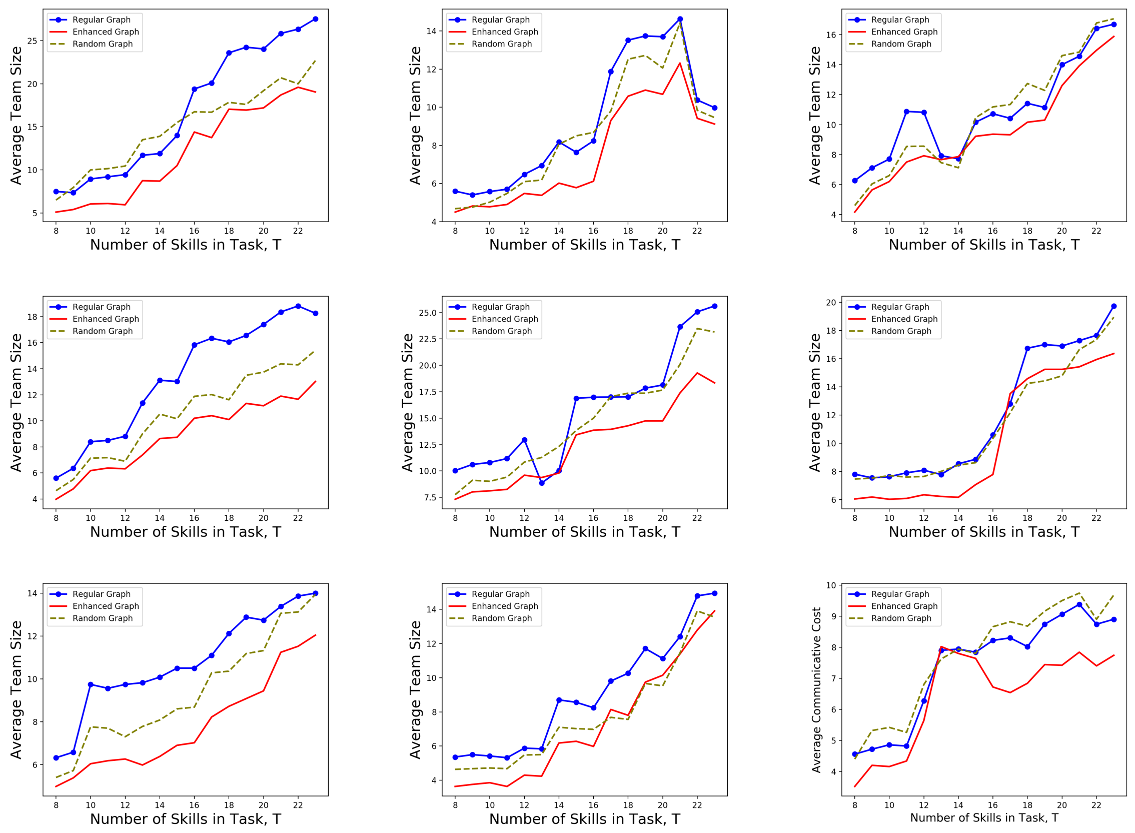


Figure 4.8: Average Cardinality of Teams for Differing Task Sizes - IBM

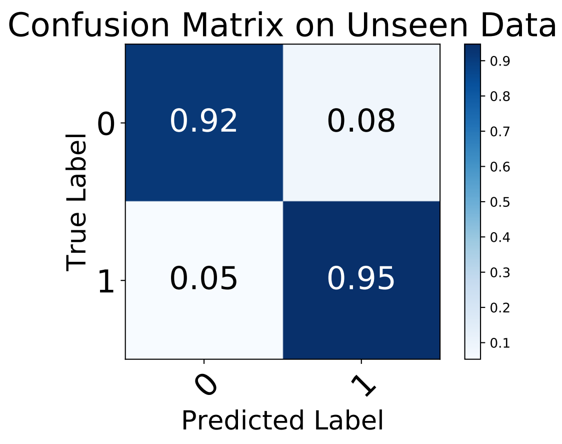


Figure 4.9: Confusion Matrix of Machine Learning Model on Unseen Samsung Data

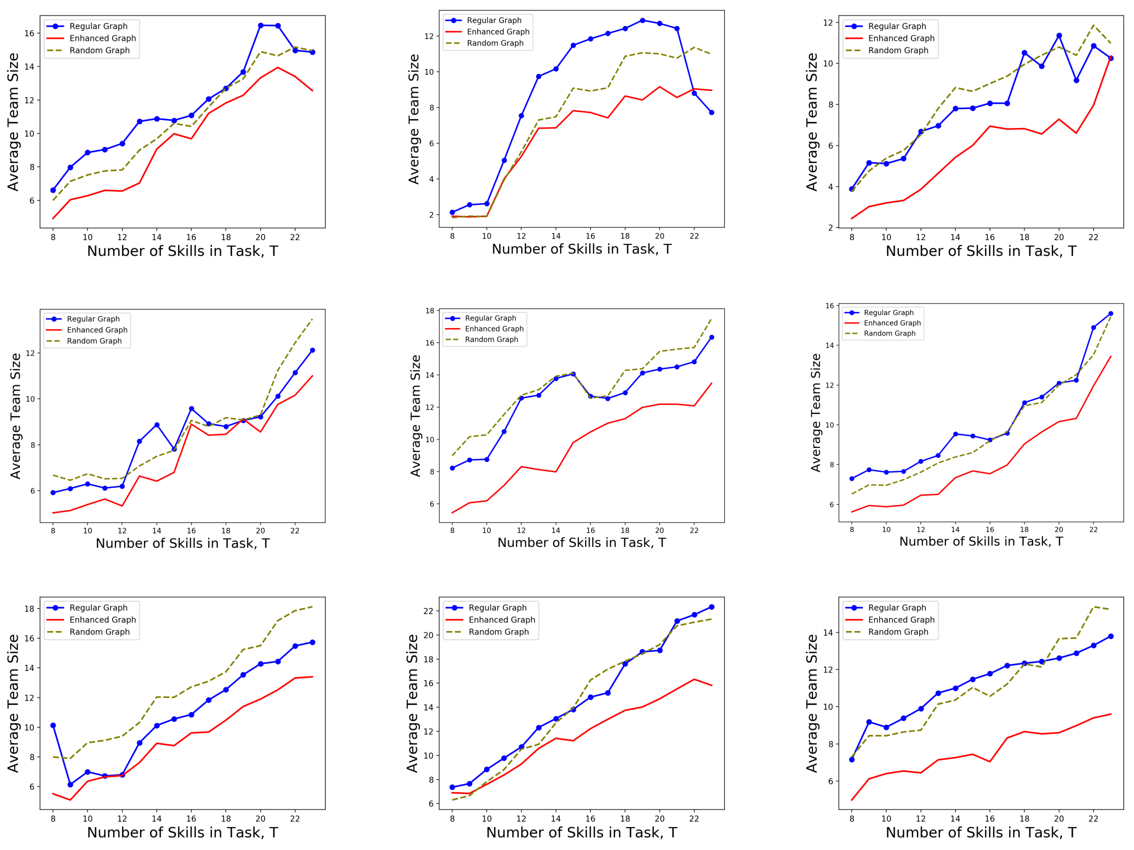


Figure 4.10: Average Cardinality of Teams for Differing Task Sizes - Samsung

Task	G	G Random-Augmented	G ML-Augmented
T_1	12.29	9.79	**8.93**
T_2	5.73	4.62	**3.82**
T_3	7.39	6.92	**5.79**
T_4	7.01	6.51	**5.65**

Table 4.2: Average Cardinality of Teams for Task of size 10 - Samsung

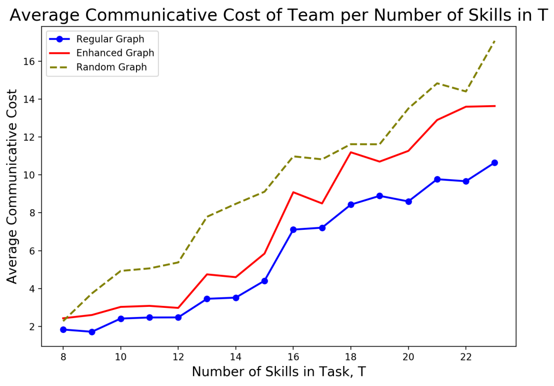


Figure 4.11: Communicative Cost based on Machine Learning Probabilities - IBM 1

links used by the team. Some of the results of these tests are shown in figures 4.11, 4.12, 4.13, and 4.14. Interestingly, although the average cardinality of the team suggested by the machine learning graph is lower, the sum of weights tends to be higher at larger numbers of skills. The physical or financial communicative cost is, of course, determined by the circumstances. Consequently, it may be interesting to explore other, more meaningful measurements of the communicative cost.

4.6 Summary

In this chapter we have laid out our system, implemented it, and tested its efficacy. We started with some review of literature and previous approaches to the team formation problem in section 4.1. Following on from looking at previous literature, we introduced the primary method we used to find teams in section 4.2, the Enhanced Steiner Tree Algorithm.

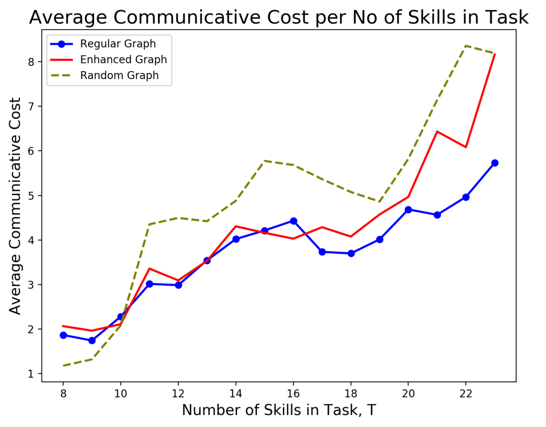


Figure 4.12: Communicative Cost based on Machine Learning Probabilities - IBM 2

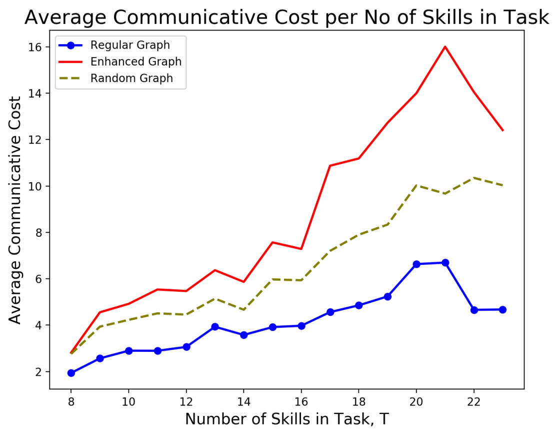


Figure 4.13: Communicative Cost based on Machine Learning Probabilities - Samsung 1

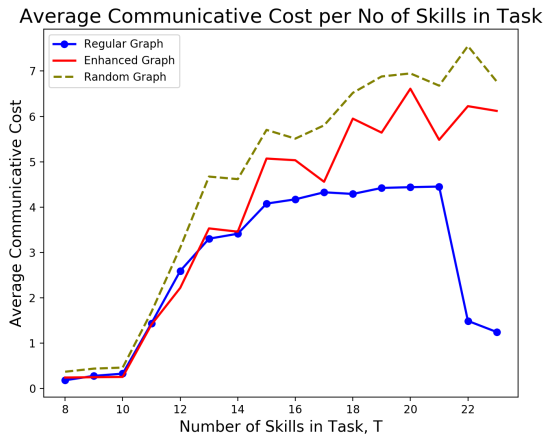


Figure 4.14: Communicative Cost based on Machine Learning Probabilities - Samsung 2

We also discussed the method in further detail in section 4.4.7.

We laid out our full system in a block diagram in section 4.3. We displayed it in Figure 4.1, and we explained the components of the block diagram.

In section 4.4 we implemented our system. We discussed how the expertise of inventors was determined in section 4.4.1. We discussed how we created our trimmed graphs from our existing collaboration graph in section 4.4.2. The machine learning aspect of the system was discussed extensively in sections 4.4.3, 4.4.4, and 4.4.5.

The machine learning model and its predictions were used to augment the collaboration graph in section 4.4.6. This is the principle part of our contribution. We suggest that a collaboration graph may exist where patents don't. For example, there may be inventors who work with other inventors on different tasks, tasks not involving patents. We suggest that a collaborative relationship could exist there despite the absence of a shared patent to date, and the machine learning model suggests the same, so we augment the collaboration graph with that edge.

Finally, in section 4.5 we test the scheme. Having previously mentioned the correlation between team size and costs, we are motivated by finding smaller teams with the necessary skills to complete a given task. We tested our augmented graph for differing sizes of tasks against the regular collaboration graph, and a graph with randomly added edges. We tested the algorithm 100 times for a task of static size. Then we tested the algorithm 50 times for tasks of successively increasing size. We also repeated this second test on Samsung patent data. When the algorithm was run, our augmented graph consistently returned smaller teams for most task sizes.

Chapter 5

Conclusion

We aimed to provide a method to find small teams that cover necessary skillsets using a combination of graph-based techniques and machine learning.

This began with a cursory look at Graph Theory and its history in chapter 1, and a more in depth look at the graph theoretical definitions that would be used over the course of the research in section 2.1.2.

Next, a suitable set of data was selected. This was the USPTO database and it was discussed in chapter 3. In section 3.2 the data was analysed statistically, which included looking at the breakdown of patents across different inventors and different patent classes.

We built upon the Enhanced Steiner Tree algorithm, introduced in 4.2, which provides a good solution to the team formation problem, but can sometimes neglect to include inventors which may not be well connected in the network. We use machine learning to predict links to connect these inventors based on potential relationships.

Our results show that when we add these predicted links we consistently get a smaller team which covers the skills required for a given task T, compared both to the original collaboration graph and a randomly augmented graph. This suggests our augmented graph is able to return more compact teams with good collaborative potential.

One particularly interesting observation we made is that when we augment the graph with machine learning we end up with more isolated nodes than if we randomly augment the graph. For example, the machine learning augmented graph for IBM has 69 isolated nodes, but the randomly augmented IBM graph has no isolated nodes, i.e. it is one large connected component. This was the same for the Samsung collaboration graph; the machine learning augmented graph has 102 isolates and the randomly augmented graph has no isolated nodes. However, as seen in the results, our machine learning augmented graph consistently returns smaller teams. So although the average degree of both the machine learning augmented graph and the randomly augmented graph is the same, the machine learning augmented graph is clearly having edges added in a manner which is conducive to finding smaller teams.

A substantial area for future research is the communicative cost of the teams. There may be a more direct method available to users for measuring potential communicative cost. This could be traded off against the cost of additional team members in an expanded model.

Another area to be considered in future is how to measure the efficacy of a team, be it pre-existing or newly formed by this method. Measuring the success of a collaboration could give a whole new dimension on the team formation problem. Perhaps future research could look at geographical distance as a measure of how effectively a team can collaborate. Given the proliferation of high speed internet in big companies, however, maybe geographic distance is lessening in importance. Perhaps access to high speed internet may be another way to look at communicative cost.

Another aspect for potential future research is how to define the success of a collaboration. One aspect of this that we did briefly consider over the course of this research is patent citations, e.g. a patent being cited more times than others being an indicator of how successful it was. This, however, proved to be a fruitless endeavour, and also would only measure the success of individual patents, rather than collaborations. This also opens up the opportunity to discuss how we can measure the success of an individual patent. In future research, another way to measure the success of a collaboration may be through user feedback.

How an inventor is classified as an expert could also be researched further in the future. In section 4.4.1 we used the number of patents each inventor held in a particular class to determine expertise. Perhaps another, more representative system for defining expertise could be developed.

Using the patent classes as identifiers is yet another area that may be worth further examination. Over the course of this research we did experiment with some latent Dirichlet allocation (LDA) textual analysis of each patent abstract. The hope was that we would be able to break all of the patents down into 15-20 categories identifiable by a single keyword. This wasn't very fruitful, as the words returned as being prominent in each grouping of patents were no more descriptive than the 8 group titles already provided by the International Patent Classification system.

All inventor links are temporal. Some inventors may retire, move companies, or die. This can be mitigated by trimming the collaborative graph that is fed into the algorithm to only include specific active years. However, maybe there is another area for research available that could make our system more time dynamic.

Another potential use for our framework could be in finding collaborations other than those which centre around patents. For example, one might consider applying our team formation problem to company directors. The collaboration graph could consist of vertices representing company directors, and the edges between them representing any time those two directors served on a company board together.

Bibliography

[1] J. A. Bondy, U. S. R. Murty, *et al.*, *Graph Theory With Applications*, vol. 290. Macmillan London, 1976.

[2] P. Keane, F. Ghaffar, and D. Malone, “Using Machine Learning to Predict Links and Improve Steiner Tree Solutions to Team Formation Problems,” in *International Conference on Complex Networks and Their Applications*, pp. 995–1006, Springer, 2019.

[3] J. L. Gross and J. Yellen, *Handbook of Graph Theory.* CRC press, 2004.

[4] L. Euler, “Solutio Problematis ad Geometriam Situs Pertinentis,” *Commentarii academiae scientiarum Petropolitanae*, pp. 128–140, 1741.

[5] C. Hierholzer and C. Wiener, “Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren,” *Mathematische Annalen*, vol. 6, no. 1, pp. 30–32, 1873.

[6] A. Kaur, A. Kanda, and S. Deshmukh, “A Graph Theoretic Approach for Supply Chain Coordination,” *International Journal of Logistics Systems and Management*, vol. 2, no. 4, pp. 321–341, 2006.

[7] S. M. Wagner and N. Neshat, "Assessing the Vulnerability of Supply Chains Using Graph Theory," *International Journal of Production Economics*, vol. 126, no. 1, pp. 121–129, 2010.

[8] P. Suárez-Serrato, M. E. Roberts, C. Davis, and F. Menczer, "On the Influence of Social Bots in Online Protests," in *International Conference on Social Informatics*, pp. 269–278, Springer, 2016.

[9] Y. Yamaguchi, T. Takahashi, T. Amagasa, and H. Kitagawa, "Turank: Twitter User Ranking Based on User-Tweet Graph Analysis," in *International Conference on Web Information Systems Engineering*, pp. 240–253, Springer, 2010.

[10] P. J. Carrington, J. Scott, and S. Wasserman, *Models and Methods in Social Network Analysis*, vol. 28. Cambridge university press, 2005.

[11] M. M. Skeels and J. Grudin, "When Social Networks Cross Boundaries: A Case Study of Workplace Use of Facebook and Linkedin," in *Proceedings of the ACM 2009 international conference on Supporting group work*, pp. 95–104, ACM, 2009.

[12] A. Badawy, E. Ferrara, and K. Lerman, "Analyzing the Digital Traces of Political Manipulation: The 2016 Russian Interference Twitter Campaign," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 258–265, IEEE, 2018.

[13] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on information theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[14] J. A. Fax and R. M. Murray, "Information Flow and Cooperative Control of Vehicle Formations," *IEEE transactions on automatic control*, vol. 49, no. 9, pp. 1465–1476, 2004.

[15] C. Stam, W. De Haan, A. Daffertshofer, B. Jones, I. Manshanden, A.-M. van Cappellen van Walsum, T. Montez, J. Verbunt, J. De Munck, B. Van Dijk, *et al.*, "Graph Theoretical Analysis of Magnetoencephalographic Functional Connectivity in Alzheimer's Disease," *Brain*, vol. 132, no. 1, pp. 213–224, 2008.

[16] O. Mason and M. Verwoerd, "Graph Theory and Networks in Biology," *IET systems biology*, vol. 1, no. 2, pp. 89–119, 2007.

[17] B. Bollobás, *Modern Graph Theory*, vol. 184. Springer Science & Business Media, 2013.

[18] E. W. Dijkstra *et al.*, "A Note on Two Problems in Connexion With Graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[19] B. Fortz and M. Thorup, "Internet Traffic Engineering by Optimizing OSPF Weights," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, vol. 2, pp. 519–528, IEEE, 2000.

[20] F. Hwang, D. Richards, and P. Winter, *The Steiner Tree Problem*, vol. 53. Elsevier, 1992.

[21] M. Brazil, R. L. Graham, D. A. Thomas, and M. Zachariasen, "On the History of the Euclidean Steiner Tree Problem," *Archive for history of exact sciences*, vol. 68, no. 3, pp. 327–354, 2014.

[22] V. Jarník and M. Kössler, "O minimálních grafech, obsahujících n danỳch bodů," *Časopis pro pěstování matematiky a fysiky*, vol. 63, no. 8, pp. 223–235, 1934.

[23] G. Choquet, "Étude de certains réseaux de routes," *Comptes Rendus Hebdomadaires des Séances de lAcadémie des Sciences*, vol. 206, pp. 310–313, 1938.

[24] M. S. Granovetter, "The Strength of Weak Ties," in *Social networks*, pp. 347–367, Elsevier, 1977.

[25] J. Tang, S. Wu, J. Sun, and H. Su, "Cross-Domain Collaboration Recommendation," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1285–1293, ACM, 2012.

[26] X. Wang, Z. Zhao, and W. Ng, "A Comparative Study of Team Formation in Social Networks," in *International conference on database systems for advanced applications*, pp. 389–404, Springer, 2015.

[27] J. Zhang, Y. Lv, and P. Yu, "Enterprise Social Link Recommendation," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pp. 841–850, ACM, 2015.

[28] P. C. Pendharkar and J. A. Rodger, "The Relationship Between Software Development Team Size and Software Development Cost," *Communications of the ACM*, vol. 52, no. 1, pp. 141–144, 2009.

[29] T. Lappas, K. Liu, and E. Terzi, "Finding a Team of Experts in Social Networks," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 467–476, ACM, 2009.

[30] A. Bell, R. Chetty, X. Jaravel, N. Petkova, and J. Van Reenen, "Who Becomes an Inventor in America? The Importance of Exposure to Innovation," *The Quarterly Journal of Economics*, vol. 134, no. 2, pp. 647–713, 2018.

[31] M. Al Hasan, V. Chaoji, S. Salem, and M. Zaki, "Link Prediction Using Supervised Learning," in *SDM06: workshop on link analysis, counter-terrorism and security*, 2006.

[32] N. Gorla and Y. W. Lam, "Who Should Work With Whom?: Building Effective Software Project Teams," *Communications of the ACM*, vol. 47, no. 6, pp. 79–82, 2004.

[33] R. Maher and D. Malone, "Analysing and Predicting the Runtime of Social Graphs," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pp. 370–376, IEEE, 2016.

[34] R. Sharma, K. McAreavey, J. Hong, and F. Ghaffar, "Individual-level Social Capital in Weighted and Attributed Social Networks," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 1032–1037, IEEE, 2018.

[35] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast Unfolding of Communities in Large Networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[36] S. Wu, J. Sun, and J. Tang, "Patent Partner Recommendation in Enterprise Social Networks," in *Proceedings of the sixth ACM international conference on Web search and data mining*, pp. 43–52, ACM, 2013.

[37] P. Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[38] M. Rahman, M. R. Hassan, and R. Buyya, "Jaccard Index Based Availability Prediction in Enterprise Grids," *Procedia Computer Science*, vol. 1, no. 1, pp. 2707–2716, 2010.

[39] L. Hamers *et al.*, "Similarity Measures in Scientometric Research: The Jaccard index Versus Salton's Cosine Formula.," *Information Processing and Management*, vol. 25, no. 3, pp. 315–18, 1989.

[40] C. Izsak and A. Price, "Measuring b-Diversity Using a Taxonomic Similarity Index, and its Relation to Spatial Scale," *Marine Ecology Progress Series*, vol. 215, pp. 69–77, 2001.

[41] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-Sampling Technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.