Rhodes University

Computer Science Department

Honours Project

Identifying direct relatives of a social network site user and obtaining their personal contact details

Submitted in partial fulfilment of the requirements for the degree of
BACHELOR OF SCIENCE (HONOURS)
of Rhodes University

Author:
Darryn Cull

Supervisor:
Mr. Yusuf Motara

Grahamstown, South Africa
November 17, 2015

Abstract

Usage of social network sites has steadily grown in the past few years and with it so has the amount of cyberbullying and online harassment. This paper explores a potential remedy to the problem of cyberbullying using an internet based practice called "doxxing" in order to determine the bullies direct family members through social network sites. To do so, a Google Chrome extension was developed to interact with Facebook to collect data and identify direct relatives of the targeted user. Based on the results found, it can be seen that the application was successful in determining the direct relatives of a Facebook user while achieving the research goal by ensuring any Facebook user can make use of the application in order to counter cyberbullying.

ACM Computing Classification System Classification

Thesis classification under the ACM Computing Classification System (2012 version, valid through 2015):

Human-centred computing — Social networks

Human-centred computing — Social networking sites

Human-centred computing — Social network analysis

Human-centred computing — Heuristic evaluations

Human-centred computing — Social network security and privacy

Human-centred computing — Social recommendation

Information systems — Web applications

Information systems — Web crawling

Mathematics of computing — Graph theory

Acknowledgements

I would firstly like to thank my supervisor, Mr. Yusuf Motara, for his mentorship and guidance throughout this research. His advice and counsel have been invaluable to me and his knowledge and expertise played a key role in this research.

I also thank my family for their emotional and financial support throughout the year. Without the support of them I would not be where I am today giving me the strength and will I needed to complete this research.

I would also like thank my colleagues in the Honours Laboratory, for their support in my endeavours and without whom this year would not have been as fulfilling as it has been.

I would like to acknowledge the financial and technical support of Telkom SA, Tellabs, Easttel, Bright Ideas 39, THRIP and NRF SA (UID 75107) through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter the research problem will be described followed by the goals of this research and the planned approach to obtain said goals. After this, the overall structure of this document will be explained.

1.1 Problem Statement

Usage of social network sites has steadily grown in the past few years (Statista, 2015) allowing friends and family to stay connected online. What this means is people are able to communicate more easily by sending direct private messages through these sites as well as communicate more publicly by posting on another person's or your own wall, allowing anyone with access to view and comment on the post. This opens people up to public opinion which can be very rewarding at times when the comments are useful or relevant to the post On the other hand, it also allows for negative behaviour. People are now able to voice inappropriate opinions or thoughts to whomever they wish and have access to, which is not in and of itself a bad thing unless it is at to the detriment of someone else.

A problem that has become apparent with online social networks is the ever-increasing amount of bullying or harassment taking place on them. Bullying has always been a problem within schools, although now that social network sites exist, the bullying does not end when school does and can now follow the child home. Cyberbullying and online harassment have become a major problem since there is not always someone policing, giving complete freedom to get away with saying anything to anyone. Along with this, the Internet trend of "trolling" has also taken off which is an excuse given online when

someone deliberately provokes a negative emotional response from the target by posting cynical or sarcastic remarks and in some cases socially unacceptable comments.

1.2 Research Goals

The goal of this research is to find a potential remedy to the problem mentioned above providing a way to counter cyberbullying and online harassment. Currently there are very few remedies to these problems for children; one such remedy is to tell your own guardians what has happened asking for their guidance or assistance on how to deal with the problem and possibly solve the problem. This is still a highly recommended approach, along with guardians not allowing their children to use social network sites unless the guardian is allowed to keep tabs on what is happening. In the cases where children are allowed to use social network sites freely and get bullied, there is a high likelihood that they will not tell their guardian in the same way that they do not tell their teacher when bullied at school.

The remedy proposed in this research will make use of an Internet based practice usually used by hackers called "doxxing." This involves collecting personal information about a target using social media and any other means possible, usually followed by openly revealing the information to the internet. Doxxing may be used as a remedy for online harassment by altering the search parameters to only determine the harasser's direct family members through social network sites. This will allow the victim to send a private message to the harasser's direct family members making them aware of what has been said or done, hopefully remedying the problem. In order to do this, a tool will need to be developed that is easily accessible to all users of the social network site. This generally includes all ages ranging from age 13 and up since most social networking sites require the user to be older than 13, although this is not always true as many children under the age of 13 sign up regardless.

1.3 Research Approach

To identify direct relatives of a social network site user, an application must be developed that has access to the site, is simple to acquire and easy to use. Thus the proposed approach is to develop a web browser add-on that can be used to target a user in order to determine their relatives. The reasons for this approach are the accessibility of add-ons through browser specific web stores, the straightforward installation process of simply

clicking a button and finally the inherent connection to the Internet. Considering the time constraints of this project, the scope of the application will be limited to a single browser and a single social network site; the browser being Google Chrome and the social network site being Facebook. The reason for using Google Chrome as the selected browser is due to its convenience of automatically synchronising browsers which synchronises workspaces plus its well designed developer mode and the reason for choosing Facebook is due to how widely used it is plus the rich data it provides. Using the Internet connection, the application will collect data on the targeted user by requesting the information from the social network site. The data collected will then be analysed using graph theory and heuristics to determine who the direct relatives of the targeted user are. Once done, the application should show the results with the option to contact the resulting direct family members.

1.4 Document Structure

- An in depth investigation into social network sites will be given in Chapter 2. This will include information pertaining to how they work and how they are used by the public followed by an investigation into the specifics of privacy settings and what they imply. Chapter 2 will also investigate the use of social network graphs and the potential methods that can be used by the application. Finally, an investigation into previous work relating to this research will be added.

- The design decisions made for the application will be analysed in Chapter 3, including the possible paths that were not taken. The chapter will begin by describing the general structure of a browser add-on followed by an investigation into the possible architectures that could be used by the application. This will be followed by an in-depth analysis of each operation of the application including user interaction (input and output), data collection and heuristics. The analysis of these will show the possible methods for each operation followed by a discussion of the methods and a final decision about which method will be used. Also included in Chapter 3 will be a discussion of the possible APIs or libraries that can be used by the application.

- The implementation process will be described in full in Chapter 4. This will be broken down into sections relating to each operation of the application. The operations to be included are the implementation of user input, data collection, heuristics and finally user output. The obstacles encountered during implementation will also be described in Chapter 4 along with the methods used to overcome the obstacles.

- Coverage, efficiency, accuracy and usability will be tested in Chapter 5. The chapter will begin with a description of the testing process and parameters used. This will be followed by an analysis of the application's social network graph coverage to determine the likelihood of successfully determining direct family members. The efficiency of the application will then be analysed based on how efficiently data is collected. The application's accuracy will analysed based on how correctly it determines the direct relatives of targeted users. The usability will be shown and analysed using images of the application's user interface.

- A conclusion will be drawn in Chapter 6 which will include a more in-depth summary of each chapter, a conclusion of the results produced, a final statement of the research and any future work to be done or considered. Discussions will be given in each section along with the conclusion produced.

Chapter 2

Literature Review

Social networks have been analysed in literature for many years from the perspective of mathematicians and social scientists. However, literature about social network sites is few and far between, especially from the view of a computer scientist. In this chapter, literature about social networks will be reviewed, although more emphasis will be on literature surrounding social network sites. The four overarching topics of this literature review include social network sites, privacy, social network graphs and work relating to the design and implementation of third party applications for social network sites. It is important to establish the background knowledge needed to develop an application which uses these topics to achieve a goal. In order to identify particular members of a social network site, one must first have a good understanding of what a social network site is and how they function. It is also important to be aware of the ethics involved, such as whether privacy is being invaded or preserved. Additionally, one must have a working knowledge of how to use social networks: how to traverse, analyse and visualise social network graphs.

2.1 Social Network Sites

Boyd and Ellison (2007) define a social network site as "a web-based service that allows individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system." The nature of the connections made can vary from site to site.

Early work on computer-mediated communication suggested that moving from face-to-face interactions rich with context to text-based media would create an "impoverished communication environment - fraught with misunderstandings, flaming and antisocial behaviour" (Haythornthwaite, 2005). In spite of this, as new social network sites emerged and became more familiar, the use of "common and group conventions" (Haythornthwaite, 2005) have made social network sites an integral method for maintaining interpersonal connections. Social network sites are unique since they enable users to express their social networks and make them visible to those they choose. However, this does not imply that social network sites are perfect: misunderstandings still happen and anti-social behaviour has become a regular occurrence online, from "flaming" to "trolling".

In most cases, generating a profile consists of answering a generic list of questions about oneself to create a unique set of pages where one can create a digital representation of oneself, though the visibility of a profile depends on the site hosting it and/or the users preference. MySpace allows users to choose whether they want their profile to be visible to the public or "friends only." On the other hand, sites like Friendster and Tribe are crawled by search engines, making profiles visible to anyone regardless of whether they have an account. By default, Facebook allows a user to see the profiles of other users in the same social network, unless the owner of a profile has denied permission to those in the same social network.

After creating a profile, one is usually prompted to identify other users with whom they have a connection. The label for said connection varies from site to site, although generally they are labelled "Friends". In some cases the connection is uni-directional: Twitter's "Follower" connection is good example of this. This means that a user can follow another user without their approval, seeing the content they produce. However, in most cases, the connection made between two users is bi-directional, meaning both users must accept the connection before they can view each other's content.

One of the main attractions to social network sites is the ability to communicate with other users of the site, whether that communication is public or private. Most social network sites provide some mechanism to allow these communications. Facebook, for example, allows users to post on their own profile or on a friend's profile, making the communication visible to all with access to see said profile. They also allow users to privately message another user.

2.2 Privacy

Social network sites have become an omnipresent technology, "tending to become invisible once widely adopted, ubiquitous and taken for granted" (Debatin, Lovejoy, Horn, & Hughes, 2009). Jones and Soltren (2005) identified flaws in Facebook that facilitated privacy breaches and data-mining. At the time, users' passwords were being sent without encryption, meaning a third-party man-in-the-middle attack could intercept and record the passwords. Jones and Soltren (2005) also found that Facebook gathered information about users from other sources unless the user specifically opted out. This opt-out option was later removed.

In the previous section, it was noted that a user could restrict who could see their profile page. This is one of the key features for privacy that Facebook gives users, but this feature didn't work as intended for the first three years of its existence. Information posted on restricted profiles showed up in searches unless the user opted out of allowing searches (Jones & Soltren, 2005). This issue was only fixed after a technology blogger made the loophole public and contacted Facebook.

In September 2006, Facebook introduced the "News Feed," which tracks and displays the online activities of a user's friends, such as uploading pictures, befriending new people, writing on someone's wall, etc. None of the aforementioned activities themselves were private actions, but the aggregated public display on the default page outraged Facebook users, who felt their privacy had been breached (Boyd, 2008). Subsequently, Facebook added privacy controls for what can be seen on the news feed and by whom.

Facebook can also be used by third parties for data mining, phishing and other malicious reasons. For example, Jagatic, Johnson, Jakobsson, and Menczer (2007) launched a phishing experiment at Indiana University on a selected group of students to gather information on students' friends using social network sites. They sent phishing emails to students spoofed to look like it came from a friend known through social network sites, where the control group received emails from unknown sources. The experiment had a 72 percent success rate within the social network whereas the control group only had a 12 percent success rate. The authors also added that other experiments in phishing on social network sites had similar results: "We must conclude that the social context of the attack leads people to overlook important clues, lowering their guard and making themselves significantly more vulnerable" (Jagatic et al., 2007).

When a user signs up to Facebook, they are assigned a unique identifier in the form of a number which can later be changed by the user to relate to them, such as their name or

nickname. Although, when changing the unique identifier, it can not be changed to an identifier used by another Facebook user. This identifier will always point to the same user, regardless of the changes made to their Facebook profile including first name or last name. Although, the identifier can be changed by a user at any time if necessary.

2.2.1 Privacy Implications

The population of Facebook users studied by Gross and Acquisti (2005) is, "by large, quite oblivious, unconcerned, or just pragmatic about their personal privacy." Users generously provide personal data while using minimal limiting privacy settings. This visibility, variety and richness of personal information provided on Facebook, when combined with the scope of the network and the public linkage to users' real identities, opens users to the risk of a variety of attacks on their online and physical persona. These risks range from identity theft to online or physical stalking, from embarrassment to blackmail. Not all of these risks are common among other social network sites, since not all social network sites use real identities or divulge as many personal details as Facebook.

Stalking

The information available on Facebook profiles can determine the likely physical location of a user. Gross and Acquisti (2005) found when researching Facebook users from an academic institution (860 profiles from Carnegie Mellon University) that students often made the physical location of their residences accessible on their profile pages as well as at least two of classes they attend. Since a students' life during university generally revolves around going to classes, it is possible for a potential stalker to know a particular student's whereabouts throughout the day. The authors also added that the research was done outside of the semester, speculating that the number of classes shown on profiles may be even higher during semester.

Re-identification

"Data re-identification typically deals with the linkage of datasets without explicit identifiers such as name and address to datasets with explicit identifiers through common attributes" (Samarati & Sweeney, 1998). An example is linking hospital discharge data to voter registration lists thus allowing sensitive medical information to be identified. A

more related possible use for re-identification is using data with explicit identifiers from one social network site to identify data from another social network site. As an example, one could use uploaded images in common to link an anonymous profile from one site to a Facebook account using a real identity, thus de-anonymising the user.

Online Harassment

Ybarra and Mitchell (2004) define online harassment as "an intentional and overt act of aggression toward another person online." Examples of this include making hurtful comments toward someone, or intentionally embarrassing another user. Finkelhor, Mitchell, and Wolak (2000) conducted a survey on internet use by youth aged 10 to 17 years of age in the United States finding that one in five were exposed to sexual solicitation, one in seventeen were harassed or threatened and 63% being stressed, embarrassed or upset while only a fraction reported the incident. Privacy settings being used incorrectly or not being used at all can facilitate online harassment. For example, a teenager could post a message publicly for strangers to see, giving them the option and means to abuse the poster in public, which can be both hurtful and embarrassing. Abuse on the internet can lead to psychosocial trauma, emotional distress and can cause mental health consequences for teenagers (Ybarra & Mitchell, 2004). Studies are unable to show a correlation between social network sites and the increase in online harassment so it is unclear whether social network site are solely to blame or that it is the result of the use of different types of online technology as well as teenage attitudes on internet use (Ybarra & Mitchell, 2008). A common occurrence in the literature in this area is blaming the lack of parental supervision online, however, there is also considerable speculation about whether teenagers accept parents adding them to their social network online.

2.2.2 Family and Social Networks

On Facebook, individuals voluntarily disclose information, although many may not actively consider the audience whom they are revealing information to and the variety of people included, for example the user's parents. In some cases, users may regret posting information since they did not realise before the fact that particular people (e.g. parents) are part of their social network or did not foresee people's reactions to the revealed information. Young adults could view this as an invasion of their privacy as they feel like they must monitor what gets posted by themselves and by their friends to prevent parents from

seeing. This can make young adults feel that their parent is invading conversations that they would otherwise not be privy to, particularly if the parent was to comment on the conversation. When a parent does see information about their child that they disapprove of, "boundary turbulence" (Kanter, Afifi, & Robbins, 2012) can occur between parent and child which can result in the child deleting or editing their personal information. In other cases, some young adults may not be bothered by their parent being their friend on a social network site as they feel they can use privacy control settings to restrict what they see, although they may have little control over what their parent posts about them.

Due to the aforementioned issues surrounding parents on social network sites and the lack of empirical data on the number of parents on social network sites, it is unlikely that all young adults "friend" their parents online. At present, "individuals aged 18-25 account for the majority of Facebook users, while the most rapidly increasing demographic is individuals aged 35 and older" (Kanter et al., 2012). This could mean that parents are following a similar trend which may increase the number of family networks within social networks online.

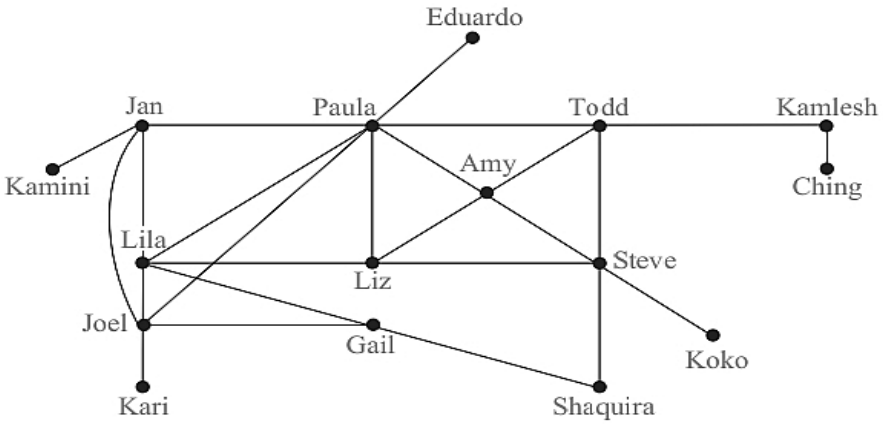
2.3 Social Network Graphs

"Graphs are discrete structures consisting of vertices and edges that connect these vertices" (Rosen, 2011). Graphs can be used to model many different problems, with many types of graphs available being used depending on the constraints of edges. These constraints vary; graphs may be directed or undirected, weighted or unweighted, cyclic or acyclic, and so forth. A directed edge is one with a direction, meaning you can only traverse that edge along the given direction, and thus an undirected edge is one without direction. A weighted edge is one with a given value. An example of this is a road map: think of plotting a route through a country picking the best roads to take where the vertices are towns, the edges are roads and the weights are distances. On the other hand an unweighted edges is one with no value or cost to it. A cyclic graph is one in which there are a number of vertices connected in a closed loop where an acyclic graph is one in which this can not happen. The number of nodes in a graph is known as the order of the graph, and the number of edges is known as the size of the graph.

Social network graphs are a specific type of graph used to represent social structures based on the kinds of relationships between people. "In these graph models, individuals are represented by vertices; relationships between individuals are represented by edges"

(Rosen, 2011). An example social network graph can be seen in Figure 2.1. The constraints, as mentioned above, on these graphs change depend on the social dynamics being viewed. Facebook would be seen as a undirected weighted cyclic graph. It is undirected due to Facebook friendship being a mutual agreement, weighted since some Facebook friends can be added as family members and cyclic since a friend of a friend can too be a friend. Twitter, on the other hand, would be viewed as a directed unweighted cyclic graph. There are many other uses of social network graphs apart from social network sites. For example, epidemiology uses social networks to study how patterns of human contact affect the spread of diseases.

Figure 2.1: An Acquaintanceship Graph (Rosen, 2011)



There are two main algorithms available to traverse a known graph: a depth-first search or a breadth-first search. Both algorithms start from a root node. A depth-first search visits a child node before visiting a sibling node, meaning it traverses down a graph until it can not go further, then backtracks until it can move sideways to uncharted territory. A breadth-first search is often used to find the shortest path from one node to another by visiting sibling nodes before visiting child nodes. Traversing a graph becomes more difficult when the full extent of the graph is unknown. This is seen as a variant on graph traversal called graph exploration, where only the root node along with all direct edges and the nodes at the end of these edges are known. When a new node is visited, more edges and nodes can be found, revealing more of the graph.

A problem that can occur when traversing a large graph is that classic methods become too slow or fail to traverse the whole graph: in other words, these methods do not scale well. A way around this is to use heuristics to trade accuracy or precision for speed. A heuristic can be considered a short cut, using a “rule of thumb” or “educated guess”

based on trial and error and data to get to a solution faster. The greedy algorithm is a good example of a heuristic algorithm; when looking for the best route, it provides a good but not optimal solution in a relatively short amount of time. It does this by picking the current best next pick regardless of whether it excludes possible future moves that are more optimal. "These algorithms effectively solve significantly larger problems than have previously been solvable using heuristic evaluation functions" (Korf, 1990). As an example of a simple heuristic: when searching through a list of names for a person's parents, there is a very high likelihood in many cultures that at least one of the parents will have the same last name as the person.

Another technique for identifying key actors in a social network graph is using graph overlaps. As the name suggests; this is done by looking at the graphs generated by two actors and overlapping them to see which actors they have in common. This is useful when looking for a social site user's family members if a family member is already known since all nodes they do not have in common (do not overlap) are likely not to be a family member producing a sub-graph. Set-density can be applied to the produced sub-graph to capture "how much intra-group similarity there is compared with the similarity between the group and the outside world" (Goldberg, Kelley, Magdon-Ismail, Mertsalov, & Wallace, 2010). This ties into local optimality which implies that a groups density can not be improved by removing or adding a single actor, so a family of actors is a sub-graph with local optimality. In other words, a parent on a social network site will possibly have very few actors in common to their child on the same social network.

2.4 Related Work

In this section, literature relating to the design and development of applications to analyse, visualise and extend social networks will be reviewed. What did they do? What are their primary results? Which challenges did they overcome? Which challenges remain? Which algorithms were developed? What analysis was done? What previous work did they rely upon? How scalable are their techniques? These are some of the questions to be answered.

2.4.1 Social Network Analysis and Mapping

Adamic and Adar (2003) designed an application which collects and analyses home pages of users to build a social network using the text on the page, the outgoing links, the

incoming links and the mailing lists they are subscribed to. The text on the page provided a "semantic insight into the content of a user's page" (Adamic & Adar, 2003) which helped identify the types of links produced by the incoming and outgoing links to other pages. The mailing lists a user was subscribed to provided a community structure the user was already a part of. The information gathered after collection and analysis was then used to visualise a user's social network. Adamic and Adar (2003) found that the application produced many false negatives, meaning a user gets matched with someone they know except there is no explicit link confirming the relationship. Another problem encountered by the authors was that not all students had personal home pages, which causes missing nodes in the mapped network, creating a lack of information. The authors noted that for the application to scale from a university setting to the entire globe would require changes to the analysis and assumptions made; this included adding more data sources such as demographic information, as well as accounting for the number of possible relationships where a user can be a fan of someone (one of thousands) or family member (one of a few). The conclusion Adamic and Adar (2003) came to was that "not only is it possible to find communities, but we can describe them in a non-obvious way."

Krebs (2002) used social network analysis to map networks of terrorist cells using data collected from news articles, search engine results and publicly released documents. The three major issues covered in their work were

- Incompleteness – the inevitability of missing nodes and links that the investigators will not uncover.
- Fuzzy boundaries – the difficulty in deciding who to include and who not to include.
- Dynamic – these networks are not static, they are always changing. Instead of looking at the presence or absence of a tie between two individuals, looking at the waxing and waning strength of a tie depending upon the time and the task at hand.

Krebs (2002) came to the conclusion that in order to successfully map a terrorist network, the agencies and/or countries involved in combating the terrorist cell need to share information and knowledge so "a more complete picture of possible danger can be drawn."

2.4.2 Recommender Systems

Given a snapshot of a social network at a given time, it is possible to predict likely interactions between users. The "link-prediction problem" as Liben-Nowell and Kleinberg

(2007) put it, is "based on measures for analysing the proximity of nodes in a network." The authors showed how this can be done in many different ways using already existing techniques found in graph theory and social network analysis producing varying results. They used methods based on node neighbourhoods such as common neighbours (Newman, 2001), which scores two users' similarity based on the number of connections they have in common, and Jaccard's coefficient (Salton & McGill, 1983), which measures the probability that two users have a given neighbour from a randomly selected neighbour from either user. Other methods based on the ensemble of all paths were used such as the Katz coefficient (Katz, 1953), which defines a measure that directly sums the collection of paths, exponentially damped by length to count short paths more heavily. Some higher-level approaches were also used, including low-rank approximation, unseen bigrams and clustering. Liben-Nowell and Kleinberg (2007) found that "there was no single clear winner" but "there is indeed useful information contained in the network topology alone." Liben-Nowell and Kleinberg (2007) conclude by saying "there is clearly room for improvement in performance" considering the highest performing method (Katz clustering) is "correct on only about 16% of its predictions."

A similar study was done by De Meo, Ferrara, and Fiumara (2011), focusing more on the similarity between two users based on the knowledge of social ties existing among users and the analysis of activities users are involved in. They use this data to draw a local measure of similarity between the two users, then consider the network as a whole to obtain a global measure of similarity based on the Katz coefficient. Finally they combine the scores of similarity into a unique value by applying linear regression. De Meo et al. (2011) found that applying a global measure of similarity can "partially correct errors produced by each single activity" implying that a collection of techniques working together outperforms any single technique. The authors mention future work will include applying their research to the link-prediction problem, using similarity as a measure of proximity in the network.

Kautz, Selman, and Shah (1997) designed an application, called ReferralWeb, similar to that of Adamic and Adar (2003). However, instead of focusing solely on analysing and visualising the network, a recommender system was built on top. ReferralWeb is "an interactive system for reconstructing, visualising, and searching social networks" (Kautz et al., 1997). The system is used specifically in academia for searching through papers, articles and the like to build a graph of references, allowing a user to search for academic writing by a particular person, referenced by a particular person, or referencing a particular person. Kautz et al. (1997) state that "by instantiating the larger community, the user can discover connections to people and information that would otherwise lay hidden."

2.4.3 Third Party Applications

Nazir, Raza, and Chuah (2008) developed and launched three applications using the Facebook Developer Platform in order to collect data on the usage of these applications. The applications gained a combined user base of more than eight million users, analysing the rich data procured based on geographical distribution of users, user interactions and modelling these interactions through interaction graphs. The three applications developed included a social gaming application called "Fighter's Club" in which users pick virtual fights with their Facebook friends that last from 15 to 48 hours and allow the aggressor and defender to request help from friends to become supporters. The other two applications developed were non-gaming applications; "Got Love" was designed to allow users to pick and display a distinct set of 'loved' friends on their profile page, "Hugged" was designed similar to that of Facebook Pokes where a user can send a virtual 'hug' to a friend (this can be done multiple times). One of the key findings is that "application dynamics can significantly affect the structure of interaction graphs, hence weakening the association between them and the underlying real-world (friendship) relationships between users" (Nazir et al., 2008). This finding is based on the fact that users of the social gaming application would often Friend strangers in order to increase the number of supporters they can gain distorting the natural community structure. On the other hand, they found that non-gaming applications tend to exhibit strong community structures.

Facebook Friend Mapper

Facebook Friend Mapper[1] is a Google Chrome Extension designed to generate the friends list of a user who is using privacy settings to hide their friends list. In order for it to work, a user must have at least one mutual friend with the target user. It works by using the mutual friend to see which friends they have in common with the target user, generating a partial list of the target's friends. It then uses the generated list to find users whose privacy settings allow the application to view their friends list, applying the previous method again to generate more of the target's friend list. The application continues doing this until a defined depth is reached, and once completed it outputs the list produced.

[1] `https://chrome.google.com/webstore/detail/facebook-friends-mapper/ikfdhlkcdllmkklmdbhfjkofjmehionn`

Chapter 3

Design

In this chapter, design choices on how the application will be implemented will be made along with a description for each possible choice and a justification of the choice made. The chapter will include decisions made on the application's overall architecture; how the application will communicate with the user; a description of how data collection will work; and how heuristics will be used to produce a result. Finally, possible libraries will be discussed, both internal and external to JavaScript.

3.1 Document Object Model (DOM)

The document object model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document. It is separated into three parts; the core DOM which is the standard model for all documents, XML DOM which is the standard model for XML documents and the HTML DOM which is the standard model for HTML documents. In this document, when the DOM is referred to, it is in fact the HTML DOM that is being referred to. The HTML DOM defines the elements of a HTML document as objects, the properties of all HTML elements, the methods to access all HTML elements and the events for all HTML elements. In other words, the HTML DOM is the standard for getting, changing, adding or deleting HTML elements within a HTML document. Each element of the document, including the document itself, is a node organised in a tree structure called the DOM tree. Each element node is made up of attributes forming sub-nodes called attribute nodes which themselves can have further sub-nodes containing

the text of the element known as text nodes. Some example HTML elements include `<a>` elements, `<div>` elements and `<p>` elements. These example elements could include attributes such as an ID or a Class which could be used to identify the element when using the DOM API in order to modify, add to or delete the element. Once an element is identified, the DOM tree can also be traversed using the API by moving up the tree to parent nodes, down the tree to child nodes or across the tree to sibling nodes.

3.2 Chrome Extension Framework

There are two main files needed to create a Chrome extension; a JavaScript object notation (JSON) file and a JavaScript file. There is also the option of a HTML file for various uses such as a extension pop-up to display information. In this section an example extension will be used to give more detail about the general stucture of each file mentioned. The extension was built to output all hyper-links on the current page.

The JSON file, named `manifest.json`, is "nothing more than a metadata file in JSON format that contains properties like your extension's name, description, version number and so on" (Google, 2015b). It is used by Chrome to determine what the extension is allowed to do and what permissions it will need to do so. An example `manifest.json` file can be seen in Appendix A.1. In this example the name and description of the extension have been defined as well as its version. The two more important lines are the permissions and the browser action lines, these are what tell Chrome that the extension needs permission to view all URLs and what HTML file to use for the pop-up.

The optional HTML file in this example is used to display the output from the JavaScript showing the user all links on the current page. As can be seen in the example code in Appendix A.2, the HTML includes a filter field and a check box allowing the user to search through the links using either plain text or regular expressions (regex). The code to implement this functionality would be added to the JavaScript files.

The JavaScript files contain the logic of the extension and use the event-based architecture provided by the `chrome.*` API to interact with the browser. An event-based architecture is one in which the system detects and reacts to events such as user interactions as well as producing its own events; where an event can be defined as an emitted change in state. This and the functionality of JavaScript allow asynchronous communication shown in Figure 3.1; meaning a request can be sent by the JavaScript application to Chrome, and while Chrome handles the request, the application can continue to execute until an event

occurs when the response is complete forcing the application to react and deal with said response. This becomes very useful when collecting data to be analysed since multiple requests can be sent to the `chrome.*` API in quick succession due to the JavaScript being able to continue running. There are other useful API libraries both internal and external which may be useful during implementation.

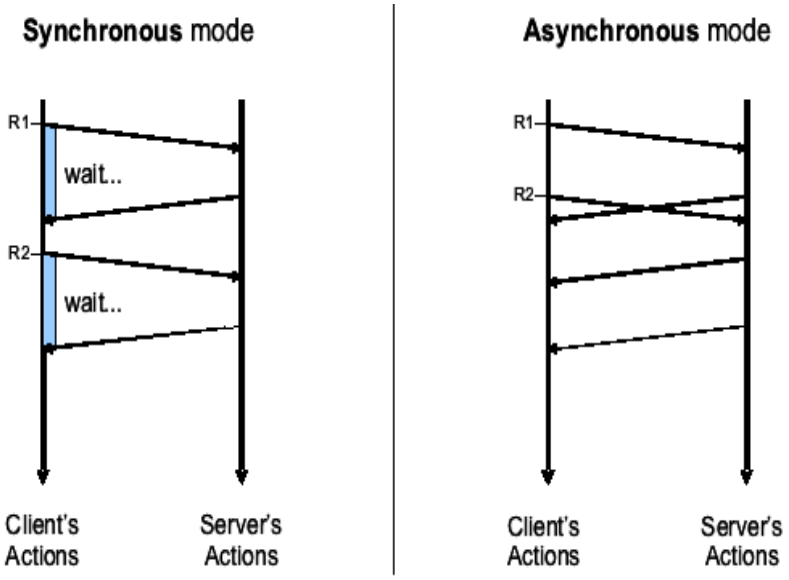


Figure 3.1: Synchronous call in comparison to asynchronous call[1].

In this, when the extension button is pressed, the HTML is parsed which, in turn, executes the `popup.js` script as can be seen in Appendix A.2. A function in said script is executed as a response to the event of the pop-up appearing. This response uses an API function call to determine which Chrome window and tab is in use. The script could run other code while waiting for responses but in this example there was no need. When the response is received, `popup.js` uses an API function to execute another script, named `send_links.js`, on all frames of the current page shown in Appendix A.3 and Appendix A.4 respectively. This script collects all hyperlinks on the current frame and can be run multiple times depending on the number of frames. It includes a callback to a function in `popup.js` in order to respond with data. Again, while this script is executed, `popup.js` could be running additional code. Finally, when `popup.js` receives the data from any `send_links.js`, it displays them in the HTML file.

There are some restrictions placed on the injected script (also known as content scripts). One such restriction is that they are executed in a special environment called an isolated

[1] `https://www.withfriendship.com/user/sathvi/push-technology.php`

world meaning "they have access to the DOM of the page they are injected into, but not to any JavaScript variables or functions created by the page" (Google, 2015a). Google Chrome treats all embedded content scripts as if there is no other JavaScript on the page, and vice versa, JavaScript running on the page cannot see the content scripts. This can be used as an advantage since each content script can make changes to its own JavaScript environment without conflicting with the page or other content scripts. If, however, a content script needed to communicate with another content script, this could be done by using the DOM as shared memory to pass data across.

3.3 Architecture

In this section two possible architectural paths will be examined followed by a discussion and a decision on the which architecture will be used during implementation. Each architecture will be analysed based on merits and downfalls, as well as possible implementation problems. The two architectures chosen are a monolithic architecture and a client-server architecture; these are likely not the only viable architectures although the two chosen are likely the best options based on time and complexity. In order, the monolithic architecture will be analysed first followed by the client-server architecture.

3.3.1 Monolithic Architecture

A monolithic architecture is one in which the application is self-contained meaning the software is single-tiered combining the user interface, business logic and data access into a single program on a single platform. In the case of a Chrome extension, this means all programming logic will be done using JavaScript in a single content script which once run will complete the entire task from end to end. This implies that the content script will be responsible for user interaction (both input and output) as well as data collection and finally the business logic needed to determine the results. A benefit to this approach is that it is based on a single platform meaning the developer can concentrate on understanding and improving the code base for this platform. This also happens to be a drawback since the single platform is JavaScript (of which the developer has limited knowledge), adding extra time to development since the developer will have to learn the platform first. A possible disadvantage to this architecture would be the difficulty of changing lower level sections of code or fixing a bug without disrupting the entire application.

3.3.2 Client-Server Architecture

A client-server architecture divides workload between client and server using a distributed application. In this architecture the client would would only handle user interaction and request the results needed for output from the server. The server would be responsible for data collection and the business logic to produce the required results. A merit to this approach would be that the server need not be implemented in JavaScript but instead can be implemented using any desired platform such as Python. This would reduce the developer's workload since a platform the developer knows well could be chosen, reducing the amount of learning needed to implement the business logic since JavaScript would only be needed for the client. However the disadvantage is needing to deal with authentication and authorization on the server which are needed for data collection. This problem is due to the server not being authenticated to Facebook as the user. This is needed to collect data since the application can only do so by acting as the user. This means the Facebook API would need to be integrated into both the client and server sides to allow access to the needed information and the user would need to accept giving the application permission to view their Facebook account. This adds too much complexity to an otherwise simple problem. Another problem with this approach is the need for a persistent server to handle requests; this reduces the scalability of the application since as more people start using the application, more resources would be needed to scale the server.

3.3.3 Monolithic vs. Client-Server

When considering the advantages and disadvantages of each architecture, it becomes apparent that a monolithic approach would be more appropriate for this application since it is more scalable and simpler to develop than a client-server architecture as well as being self-contained. This means that once development is completed the application can be easily shared and used by many. The disadvantages of a monolithic application are more easily overcome since they revolve around programming challenges as opposed to functionality problems which come with the client-server architecture such as networking, authentication and authorization obstacles. Therefore, a monolithic approach will be used to develop the application.

3.4 User Interaction

In this section, the possible design choices of user interaction will be analysed in order to decide on the methods used for user input and output to the user. Chrome allows extensions to interact with the user in many ways, thus the methods discussed will not be all-inclusive but rather narrowed down based on suitability to this application. User input will be discussed first, followed by user output.

3.4.1 Input

Firstly, consider what input is needed. The application targets a Facebook user from the origin point of an abusive post or comment; thus the input to the application would have to include the target user's unique identification (user ID) as well as possibly the origin post or comment. There are two possible ways of doing this:

- One such way is to use the Chrome extension pop-up button in the top right corner of the Chrome browser interface which once pressed prompts the contents of the extensions' included HTML file, which could contain a user input box for a link to the post/comment or for a link to the target user's profile page. Some possible flaws in this approach include the need for untrusted user input meaning input validation would be required as well as the possible miscommunication or ambiguity of the input field impacting usability.
- Another possible approach could be using the inherent ability of Chrome extensions to edit the document object model in order to add a button to every post and comment on every page using JavaScript. With this approach, the user would only need to press the added button on the abusive post or comment giving the application an origin point based on the DOM tree which will allow the application to gather the needed input being the target's user ID. This approach removes the need for input validation and improves the applications' usability since all the user is required to do is press a button.

Based on the pros and cons of each approach, the later of the two (using buttons) will be the design choice for this application since Chrome and JavaScipt together make implementing this simple and convenient for both the developer and the applications user.

3.4.2 Output

Again the first point to consider is the output that will be generated. The application will be used to determine the direct relatives of a target Facebook user, implying the output would contain at the very least the names of said relatives likely along with a link to each relative's Facebook profile. The possible methods for doing so are similar to that of user input; one method making use of the pop-up HTML file and the other method utilizing the document object model by editing the page to add an overlay containing the output. Both of these methods are similar in their implementation since they both rely on editing HTML using JavaScript in order to display the output. Although, editing the DOM of the current Facebook page would be the preferred approach in this case since the application will already be doing so and to allow for the application to more easily be ported to other browsers due to their possible differences in dealing with pop-ups. Also, using an overlaying dialog box will easily allow for a possible loading icon to be shown to the user as a means of proof the application is actually executing due to the likely variation in runtime the application will have based on data collection. The only reason for using the pop-up box being that it is a default option to any Chrome extension which is not reason enough to use it. Thus, an overlay will be generated by the application as a means of output to the user.

3.5 Data Collection

Data collection plays a key role in this application since a conclusion must be made based on available input data; in order to produce the most correct result, more information will need to be gathered. To do this, the application must request the needed information from Facebook. There are two ways in which this can happen; one way is to use the Facebook API, the other is to request Facebook pages through HTTP containing the required information and parsing the HTML received to retrieve the data. The Facebook API sounds like the reasonable choice in this situation, however, the API is designed more for use in applications to add Facebook features such as inviting friends to use the application or adding Facebook messenger to the application so users can communicate with one another while using the application. Another major use of the Facebook API is for analytical applications to view statistics on the applications user base. The other approach, using HTML parsing and HTTP requests will give the application access to a wider variety of information such as the information provided on a user's profile page which

includes but is not limited to the user's work, education, locations, contact information, basic information, friends, family, relationships. However, this information is not always freely available due to privacy settings possibly blocking access based on whether the target user is a friend of the user or whether their privacy settings are set to public or private. Based on the data provided by each approach, using HTTP requests and parsing the responses for informations will be the design choice for data collection. The reason the information provided by the second method would be more valuable to the application is so that heuristics can be run based on the information found allowing connections between users to be made and defined enabling the application to come to a more accurate conclusion on who is directly related to the target user.

3.6 Heuristics

A heuristic technique in general is any approach to problem solving which uses a method not guaranteed to be optimal or perfect, yet which is sufficient for the immediate goals. Some common terms used for heuristics include an educated guess, rule of thumb, intuitive judgement or common sense. In computer science; a heuristic is a technique designed to solve a problem faster when other methods are too slow or a technique used to find an approximate solution when there are no other solutions. In this application, heuristics will be used to determine the direct relatives of a target user using the target's friend list as potential candidates along with the data collected about the target and each of their friends. The heuristics used by the application will vary from common sense, "do they have the same last name," to simply using the data collected, "does their profile page show who their family are." Most, if not all, of the heuristics used by the application will not be scientifically proven to determine a person's family member, instead they will form a set of weak classifiers. Each heuristic on its own will not guarantee a correct solution, but, combining a few heuristics weighing each one based on its probability to produce a more correct solution and selecting the candidates that score the best might produce a satisfactory result. However, this approach will likely fail if not enough information was able to be gathered or produce an incorrect result if the target's direct relatives are not amongst the list of candidates produced using their friend list. If the application fails to produce a result, the output will reflect this, though there is no way for the application to know that it has produced an incorrect solution and so the final decision of whether the candidates produced by the application are directly related to the target user is up to the user of the application. For this reason, the application will not automatically send the

harmful post/comment targeted by the application to the resulting candidates produced but will instead provide a link to the possible relatives' Facebook profiles along with a copy of the post/comment so the user can make the final decision on whether to notify the family.

3.7 System APIs

System APIs are sets of functions provided by Chrome or JavaScript which when called, allow the application to gain information about and manipulate the browser along with most of its contents. In order to accomplish the goal of this application, some useful system APIs will be used. In this section, these APIs will be examined based on what they are capable of and what use they will be to the application.

3.7.1 Window

The `Window.*` API contains methods that allow the application access to the browser and are provided by JavaScript. This access could be used to get information about the current window, open a new window, close a window, resize a window or move a window. Also included is the document object model; thus using `Window.*`, the application will be able to get information from the DOM as well as manipulate the DOM. Based on the access provided, it will be a suitable API for user input since it includes methods to add objects to the DOM meaning buttons can be added everywhere needed as well as possibly being used to parse information received in HTML form since it comes with DOM traversal methods. Some example methods are shown below along with a description.

- `window.onload`
  This method gets called the moment a web page has loaded and its state becomes idle. To use this method, a callback function must be passed to `onload` so that the moment the page becomes idle, the callback function is executed allowing the application to react to the event. This method may be useful to add buttons to every Facebook page as they finish loading.

- `window.setInterval`
  This method is similar to `onload`, although instead of the event being a page loading it responds to a time event. To use this method, a callback function must be passed

to `setInterval` along with a time interval in milliseconds. The method will then be run after every time interval running the callback function allowing the application to create timed events and react to the event. This may be useful since most of Facebook's pages employ an infinite scroll mechanism which will likely not be caught by onload, meaning a timed event can continue to add buttons where needed even after the page has been loaded.

- `window.document.getElementsByClassName`
  This method, along with several similar methods, can be used to retrieve information from the DOM of the currently selected document within the currently selected window. This method is used to fetch all DOM elements with a given class name passed to the method; others like this method allow the use of identification or other identifiers. After a DOM element has been fetched, the application can read information from the element or add a DOM element to it changing the page for the user which will also be useful for adding buttons to each Facebook page.

3.7.2 Chrome

The Chrome API contains methods used to manage the browser's settings, preferences and data such as permissions, privacy and history respectively provided by Google specifically for Chrome. For this reason, the API will likely not be useful to the application since the capabilities of the API are out of the scope of the application and use of the API if necessary will likely make porting the application from Chrome to other browsers more difficult. There is no need for the application to change the user's browser or manipulate how it functions.

3.7.3 AJAX

Asynchronous JavaScript and XML (AJAX) is an API included with JavaScript that allows the application to make asynchronous requests to a server and receive the response. This is done using callback functions; the application sends an HTTP request passing a callback function with it so when the response is received, the callback function can handle the response. Due to its asynchrony, this means while the application waits for a response it can continue running other code instead of blocking computation until the reply is received. This API will likely be useful for data collection since the information needed by the application is contained in Facebook pages which can be requested by using

the library. Also, despite the name, XML is not required and the request need not be asynchronous, however, making use of the asynchrony would be advantageous for speeding up data collection since multiple requests can be sent sequentially with the responses dealt with in any order.

3.8 jQuery

jQuery is an extensive library built for speed and portability making it "fast, small and feature-rich" (jQuery, 2015). The library provides simple to use abstractions of the built in `Window` and AJAX libraries making use of these libraries to create an API which can traverse and manipulate the DOM as well as send request and receive responses. The library also includes functions to handle events such as a user pressing a button, which will likely be useful after adding buttons to a Facebook page since the API will allow the application to not only respond to the button press but also know where on the page the event occurred, giving the application a starting point for DOM traversal. The library makes use of jQuery selectors which are used to find (or select) HTML elements based on their id, classes, types, attributes, values of attributes, etc. and are based on the existing CSS Selectors, and in addition, it has some own custom selectors[2]. An example selector for finding a `<div>` element with the class name "element" would make use of a full stop in the selector string to make explicit that a class is being used, which would look like "div.element". Other special characters or key words would be use to specify different search parameters. A few of the many possibly useful methods are shown below along with a description.

- `$.on`
  Attaches an event handler function for one or more events to the selected elements. This may be used by the application to respond to button clicks.
- `$.get`
  Loads data from the server using a HTTP GET request. This will likely be used as a means of retrieving data from Facebook.
- `$.find`
  Gets the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element. This could be used to traverse HTML responses in order to find the requested data.

[2] `http://www.w3schools.com/jquery/jquery_selectors.asp`

3.9 Summary

Applying the design decisions made in this chapter; the application should be capable of handling user interaction using buttons as user input and a dialog box for output, able to collect data about the target user and all of their connections, and finally intelligent enough to use the information gathered to produce candidates that are likely to be directly related to the target user. The design decisions made include the use of a monolithic content script to run on every Facebook page, which can dynamically add buttons to every post and comment. When a button is pressed, it starts the process of data collection done using jQuery as a tool to request data and parse the response for information. Heuristics are applied to form a decision on which users are likely related to the target user, outputting the possible candidates to a dialog box on the page where the button was pressed.

Chapter 4

Implementation

In this chapter, the design choices made will be implemented and high level descriptions of how they were achieved will be given along with changes to the design made during implementation. Obstacles encountered during implementation will also be mentioned together with the solution found to overcome the obstacle. The chapter will follow the path of implementation taken by the developer; starting with input, moving to data collection, then heuristics and finally output.

4.1 Input

In order to get user input working, buttons needed to be added to every post and comment. To accomplish this within the content script, four functions were written; two of which respond to events calling the other two for adding the buttons. The two event listeners used both come from the `window.*` library; the first being an `onload` event which gets called every time a Facebook page is loaded and the second event being a timed event which occurs every 2000 milliseconds. Due to the use of the manifest file included with the extension, these functions only run on pages served by Facebook based on the permissions set. The contents of these two functions are similar, they both search the current page for "Like" buttons, one as part of a post and the other as part of a comment passing the found nodes to the other two functions mentioned. These functions then add a "Dislike" button providing a unique class name to posts and to comments respectively allowing the application to tell where a button press originates from. The reason for searching for "Like" buttons on the page is to simplify the problem of knowing where buttons are

needed or not needed based on whether the current user has enough privileges on the current page. With this done, buttons have been added everywhere needed, but there is no way to respond to a button press. So to do this, another two functions were written using the jQuery libraries' `.on()` method which given the body of the page as the origin of the event along with the type of event, a selector for each unique class created when generating the buttons and finally a callback function which will be executed when the event occurs. With all of the above implemented, the application can dynamically add buttons to any Facebook page and begin execution once a button is pressed. A benefit to this approach is that when the callback function runs in response to a button press, the function knows where on the page the button got pressed based on the unique class and so can capture the post and comments which includes the abusive post/comment being targeted along with the target's Facebook user name.

4.2 Data Collection

After getting input working, the application already had some relevant data which could be used to gather more. The information gathered through user input included the target users, as well as all users included in the post, user ID and name. To gather more information, the application needs to make requests to Facebook. To do so, the application uses jQuery's `.get()` method which, when supplied with a sub domain of the current domain and a callback function, asynchronously makes the request, allowing the application to continue running until the response is received and the application is idle, at which point the callback function gets run, parsing the response. For each page containing needed information that is unique to the others, a new function will have to be written since parsing the page will vary based on the given page. The following subsections detail the different functions written to gather data.

4.2.1 Friend List

To be able to determine a user's family members, a list of candidates would be needed. Firstly, the application will make the assumption that the target user is friends with their relatives on Facebook since finding a direct relative with no connection to the target would be a more difficult task. If the target's friend list is retrieved, the application can use it as the list of candidates. To do this, the application will use a feature on Facebook which allows a user to see the mutual friends between two users that can be found by navigating

to a URL, `https://www.facebook.com/[ID1]?and=[ID2]&sk=friends`, which includes the user IDs of the two users in question. Using this, the application can generate the majority of the target user's friend list by using the target's user ID along with a known friend of the target. If the target user made the original post, it can be assumed that the user of the application is a friend of the target and so can use "me" as the known friend. Otherwise, if the target user only commented on a post, it can be assumed that the target user is friends with the poster and so the poster can be used as the known friend. These assumptions were made based on the general format of a user's Facebook feed and what posts make it into said feed. Having the target user's ID and a known friend's ID, the application can request the mutual friends they have using the already described method. Once the response has been parsed and the IDs of all their mutual friends have been stored, the application can use these new IDs to request more mutual friends. To do this, once the function is completed, it will recursively call itself passing a new ID to be requested every time until all unique friends have been used as the known friend in the request. Once the entire process is complete, the application has gathered a list of the target's friends, although the list is not necessarily complete. The reason the list isn't complete is due to outliers in the target's friend list, outliers being friends or groups of friends that are connected to the target user by a single link. In other words, outliers are people who are friends with the target user, but not friends with any of the target user's other friends. However, this is not likely a disadvantage or obstacle, since it is unlikely that the target's direct family members fall into this category although it is possible.

4.2.2 Profile

With more information, better heuristics can be run and a more correct solution can be found. On Facebook, the best source of information about a particular person is their profile page. So to gather this information a function was written which, given a user ID, will request the profile page of the user and parse it for as much information as possible. A problem with collecting data from Facebook profile pages is that, depending on the relationship between the target of the function and the application user as well as the privacy settings of the target, much of the information available of a profile page could be unavailable to the application. The function written to do this uses the same method previously described at the start of this section. One section of a given profile page stands out from the rest, which includes details about a user's family and relationships. This page includes the names and IDs of every user the target user claims to be their family.

However, this does not guarantee these users are actually family members since there is a known trend on Facebook to add a good friends as a family members on this page.

4.2.3 Obstacles Encountered

After implementing the data collection, it was found that the application was synchronising the asynchronous requests sent. In other words, the application would send a request asynchronously, but instead of continuing computation the application waited for the response to arrive and be dealt with before moving to the next piece of code. This is a waste of time and resources, and so was fixed allowing the application to continue running other logic while waiting for responses. This allowed the application to start running heuristics based on what information had already been collected while waiting for more information to arrive.

When implementing the parsing of mutual friends and and the target user's family and relationships profile page, it was found that the data relating to other users arrived commented out within the HTML of the page. When requested by a browser, Facebook hides the sensitive information contained in the HTML within a comment during transport; after the browser receives the HTML, a script is run to uncomment the HTML just before the browser displays the HTML to the user. The reason behind this might be to prevent generic parsers from being able to read the information. However, using methods from jQuery, the application was able to read this information by searching the HTML received for the HTML tag denoting a comment and manually uncommenting it creating a sub-string producing the HTML within. This HTML was then passed to jQuery allowing the application to parse it normally.

Due to the possibility of privacy settings being enabled on the target's profile page and the chance that the user did not specify particular details, measures had to be taken to catch the possibility of the information not being available. If a page is requested and parsed for a particular element, and this element isn't there, depending on the specifics of the operation the application could throw an error or assign the data as undefined. To prevent these issues, the application was programmed to deal with all possibilities of information availability. As an example: on a user's family and relationships profile page, they can set the privacy settings to not show their relationship but still show family members or vice versa meaning if the application looks for both and only one is found it must still know which section the information came from while not causing errors immediately or further in computation.

4.3 Heuristics

During the data collection process, heuristics are run simultaneously due the asynchrony of JavaScript requests. This makes the application more efficient by allowing data received to be processed while more data is being collected, and if more data is needed, the application will continue to wait until the data is received moving to an idle state allowing the AJAX callback functions to be called. The heuristics included in the application are run on each user, and if the heuristic is successful, a score is added to said user based on the strength of the heuristic and the strength of relation between the users. These score will help the user of the application decide whether family members outputted are more or less likely to be actual family members. Intimate relations can be found by the application based on the information retrieved from the target's profile page which include marital relationships and non-marital intimate relationships, although if no information is found the application will not look any further. The heuristics used in the application are applied to both the generated friend list as well as the family members found on the target's profile page. If no family members are found on the user's profile page, the chances of finding a direct family member amongst the friend list is likely to be reduced.

The first heuristic is used during data collection, which is determining family members using the target's family and relationships profile page. This heuristic produces a list of possible family members along with their type of relation to the target user, although in some case the type is omitted. The next heuristic used involves checking whether the target user has the same last name as any of the friends in the generated friends list. If someone is found, they are likely to be a family member although this not guaranteed since friends may have the same last name with no relation, a user may have taken a use name or may not have a last name. Also, this excludes direct family with different last names, such as divorced or remarried parents taking different names. Another heuristic to help solve this problem is checking the profile page of a known family member for their family relations which, depending on the user's privacy settings, could produce more possible direct family members since the connection is known.

4.4 Output

In order to get output working, two functions were written. The first function creates an output dialog the moment the application is run styled to match Facebook; this was done

so that a loading icon can be shown while the application runs since the computation time can be quite long depending on the number of friends the target user has; thus, something was needed to show the application is actually running. Once the application is finished, the loading icon is removed from the dialog box and the output is added. The output shown includes the application's guesses about who the target user's direct family members are as well as the friend list generated. Each guess provided by the application will be accompanied with a scoring to give an idea of how likely they are to be related. If no information is known about the direct family of the user, the output will reflect this by showing the target's friend list along with zero or no scoring.

4.5 Summary

While some of the methods used were more challenging than others, the application can now determine a Facebook user's direct family members given enough data or access to data. The application has a working method for input from the user, for output back to the user, for data collection and finally for determining relatives using heuristics. Overall, there were no significant changes needed from design to implementation, however there were obstacles encountered that were not considered during design which needed to be overcome. As for further work, this will be covered in the last chapter.

Chapter 5

Tests and Results

In this chapter, the application will be tested on 16 Facebook users where the total number of friends for each user is known. The results produced will be analysed in three categories including coverage, efficiency and accuracy. In each of these sections, a description will be given on what the category incorporates along with data and an analysis of that data. Finally, a summary of the chapter will be included.

5.1 Testing

In order to test the application, data will be needed. To get the data, code was added to the application to produce the necessary data needed for the three categories mentioned. The data required for analysis includes the number of requests sent for family members, the number of requests sent to gather friends, the number of friends found, the number of family members found from the user's profile page, the total number of nodes found, the total number of friends the user has, the number of family members found, the number of false positives and finally the number of false negatives. This data will be used in each category of analysis based on its relevance to that category. The users chosen for testing include users with little to no privacy settings as well as users with high privacy settings and was carried out from the perspective of a user friends with all targeted users. The reason the application was not tested on users that are not friends with the application user is so that the values obtained through testing can be checked against actual values only known by being friends with target user.

5.2 Coverage

The application requires nodes in the social network graph of the target user in order to determine the target's direct relatives. If some nodes are left out then the likelihood of obtaining the target's direct family will decrease. So, to determine whether the application covers enough of the graph its coverage needs to be determined. This is done by comparing the number of friends generated by the application to the total number of friends the target user has producing a percentage of friends found for each user targeted during testing. This percentage only includes direct friends found using mutual friends and does not include the total number of nodes found during data collection.

User	Total Generated	Actual Total	Coverage (%)
1	288	336	85.71
2	671	712	94.24
3	195	214	91.12
4	266	365	72.88
5	450	481	93.56
6	71	80	88.75
7	350	363	96.42
8	812	905	89.72
9	423	574	73.69
10	391	412	94.90
11	129	134	96.27
12	202	223	90.58
13	395	440	89.77
14	247	264	93.56
15	188	207	90.82
16	444	490	90.61

Table 5.1: Table of coverage of friends found.

As can be seen in Table 5.1, or in Figure 5.1 below, the application scores in the top 20 percent coverage on all users bar two with a maximum reaching 96.42 percent. This implies the application is capable of covering a high proportion of the target user's social network graph, although the application did score a minimum of 72.88 percent which means the coverage can vary based on the current user being targeted. With a mean of 89.54 percent and a standard deviation of 6.96 percent, the application's coverage varies

between 82.58 percent and 96.50 percent which includes the maximum and excludes the two lowest users. This means that the application has on average a 10 percent chance to not find the target's direct family members since they may not be covered by the application's data collection. This does not include all data collection, only the traversal of the target user's direct friends and does not include the data collected from user's profile pages. The two lowest scoring users also happen to be the two users with high privacy settings which implies privacy may play a role in the application coverage since they are the only two to fall outside one standard deviation.

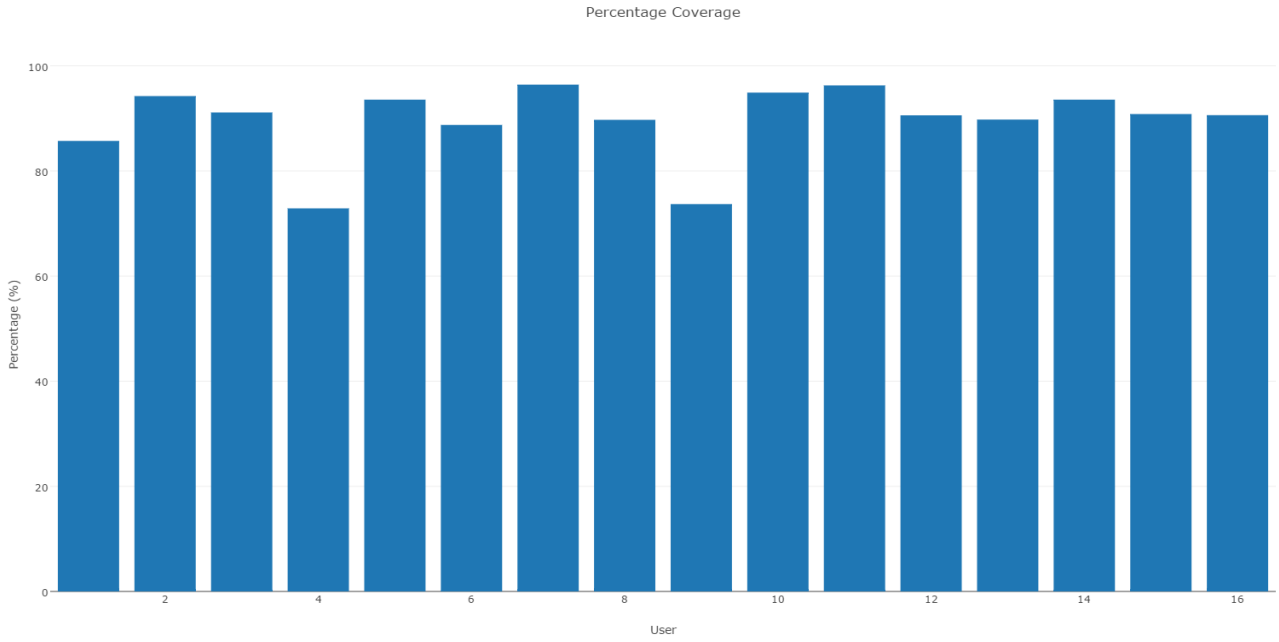


Figure 5.1: Graph of coverage of friends found.

5.3 Efficiency

For an application to be successful, it must also be efficient by not wasting time or resources. Given that the application collects data by sending requests to Facebook, time is not a good measure of the efficiency of the application due to the non-deterministic number of friends a user can have and instead its efficiency can be determined by comparing the amount of data collected to the number of requests sent. In this case, the amount of data collected is reflected by the number of nodes found in the complete social network graph including the target's friends and the family members found during data collection. The measure of efficiency will be done by dividing the total number of requests sent by

the total number of nodes found producing a percentage. This percentage shows number of requests needed per node, thus the lower the percentage is the better the result since less requests were needed to produce more nodes where 100 percent implies one node was found per request sent.

User	Friend Requests	Family Requests	Nodes	Requests/Node (%)
1	285	286	1226	46.57
2	668	669	1774	75.37
3	195	196	634	61.67
4	265	266	752	70.61
5	447	448	1334	67.09
6	67	68	251	53.78
7	339	340	1199	56.63
8	807	808	2612	61.83
9	423	424	1371	61.78
10	391	392	1397	56.05
11	130	131	437	59.73
12	201	202	737	54.68
13	391	392	1171	66.87
14	246	247	864	57.06
15	188	189	788	47.84
16	441	442	1328	66.49

Table 5.2: Table of efficiency of requests per nodes found.

As can be seen in Table 5.2, or in Figure 5.2 below, most users scored between 50 and 70 percent, though four fall outside this boundary. This implies that the application was able to find fewer than two nodes per request sent for these users which also include the two users who scored above 70 percent. On the other hand, the application also managed to score less than 50 percent on two users implying it was able to find more than two nodes per request sent. It should also be noted that the application sends a request for every unique friend found, thus the number of friend requests sent and the number of family requests sent will always be similar to the number of friends found. With a mean of 60.25 percent and a standard deviation of 7.86 percent, the requests per node varies between 52.39 percent and 68.11 percent. It seems efficiency varies significantly depending on the current target user with a quarter of the users falling outside one standard deviation from the mean.

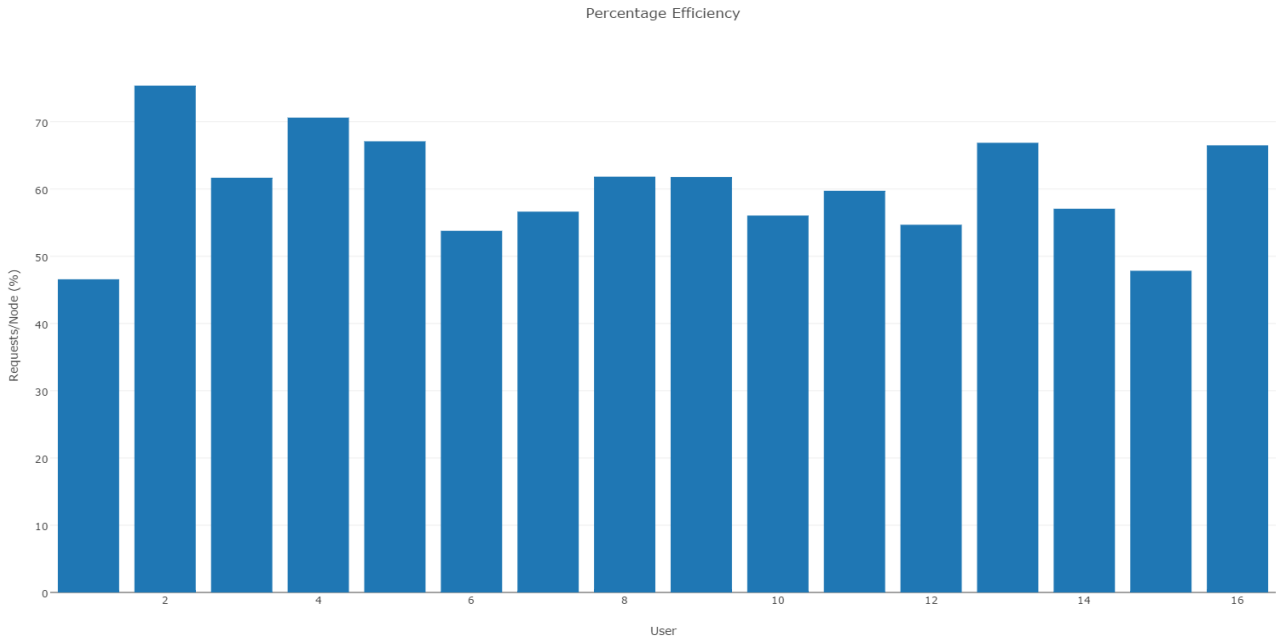


Figure 5.2: Graph of efficiency of requests per nodes found.

5.4 Accuracy

To determine the correctness of the results produced by the application, its accuracy must be calculated. Accuracy is the degree to which the result conforms to the correct value; in this case accuracy will be determined by summing the number of true positives and true negatives dividing this result by the sum of true positives, true negative, false positives and false negatives. True positives will be considered as correctly determined family members, true negatives as all friends determined not to be direct family, false positives as those incorrectly determined to be direct family and finally false negatives as direct family members incorrectly determined to be not related to the target user. This calculation will produce a ratio out of one where achieving a value of one implies complete accuracy of results.

User	Total Friends	Family Found	F/Positives	F/Negatives	Ratio	Alt Ratio	Parent (Y/N)
1	336	5	0	0	1.0000	1.00	Y
2	712	9	0	0	1.0000	1.00	Y
3	214	1	0	0	1.0000	1.00	N
4	365	7	0	0	1.0000	1.00	Y
5	481	4	1	0	0.9979	0.75	Y
6	80	5	0	0	1.0000	1.00	Y
7	363	15	0	0	1.0000	1.00	Y
8	905	7	0	0	1.0000	1.00	Y
9	574	3	1	1	0.9965	0.50	Y
10	412	4	1	0	0.9976	0.75	N
11	134	2	0	0	1.0000	1.00	N
12	223	4	0	0	1.0000	1.00	Y
13	440	10	5	0	0.9886	0.50	Y
14	264	4	0	0	1.0000	1.00	N
15	207	3	0	0	1.0000	1.00	Y
16	490	10	1	0	0.9980	0.90	Y

Table 5.3: Table of accuracy ratio.

As can be seen in Table 5.3, or in Figure 5.3 below, the accuracy of the application is exact for just over half the users since seven out of the twelve scored a perfect ratio of one while the remaining five scored within 0.02 of one. These ratios show that the application can accurately reduce a large friend list to a short list of likely family members. However, they do not highlight the incorrect results well enough to show how likely the results are to be family members due to the difference in size between the target's total number of friends and the total family members found. In order to more closely examine the results, an alternative ratio was calculated to try to highlight the incorrect results by removing true negatives from the calculation. The alternative ratio can be seen in Table 5.3 which shows the variation in accuracy for the less than exact results ranging from 0.50 to 0.90. This shows that when the application fails to produce completely accurate results, it will only fail on a maximum of half the results produced. Another point to note is that the application only produces one false negative between all 16 users showing that it is highly improbable to miss a direct family member and instead is more probable to produce false positives. Also, the false positives produced by the application are ranked low by the application indicating to the application's user that they are less likely to be related to the target user.

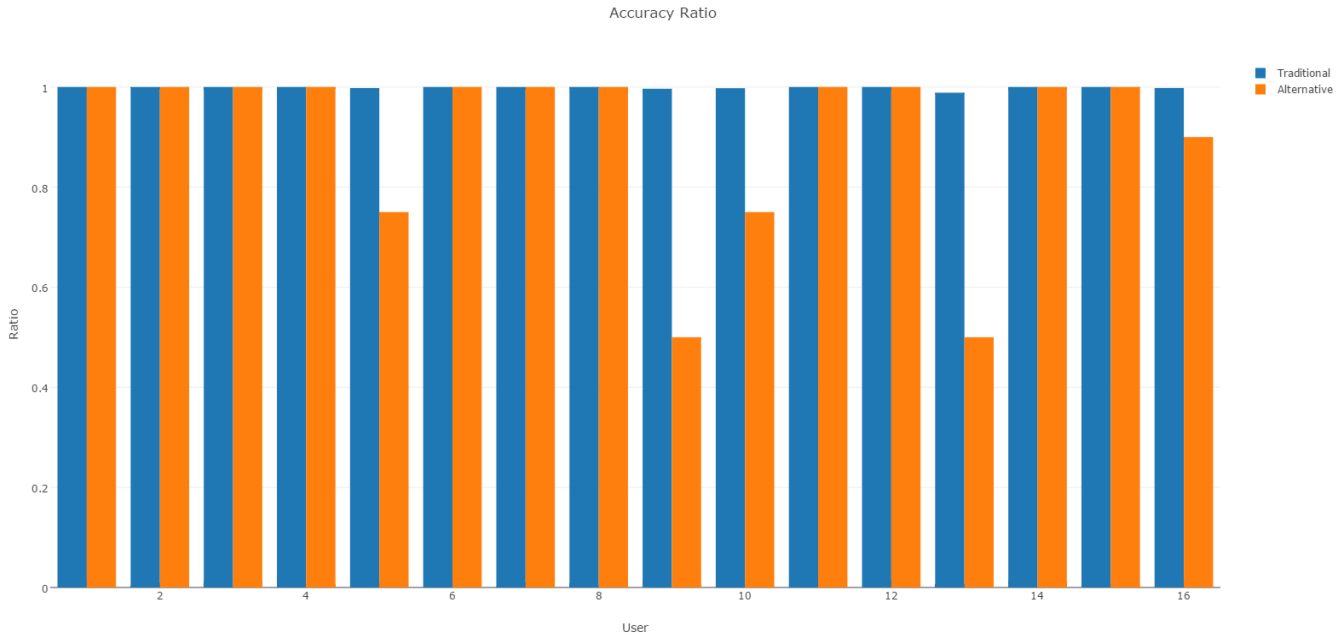


Figure 5.3: Graph of accuracy ratio.

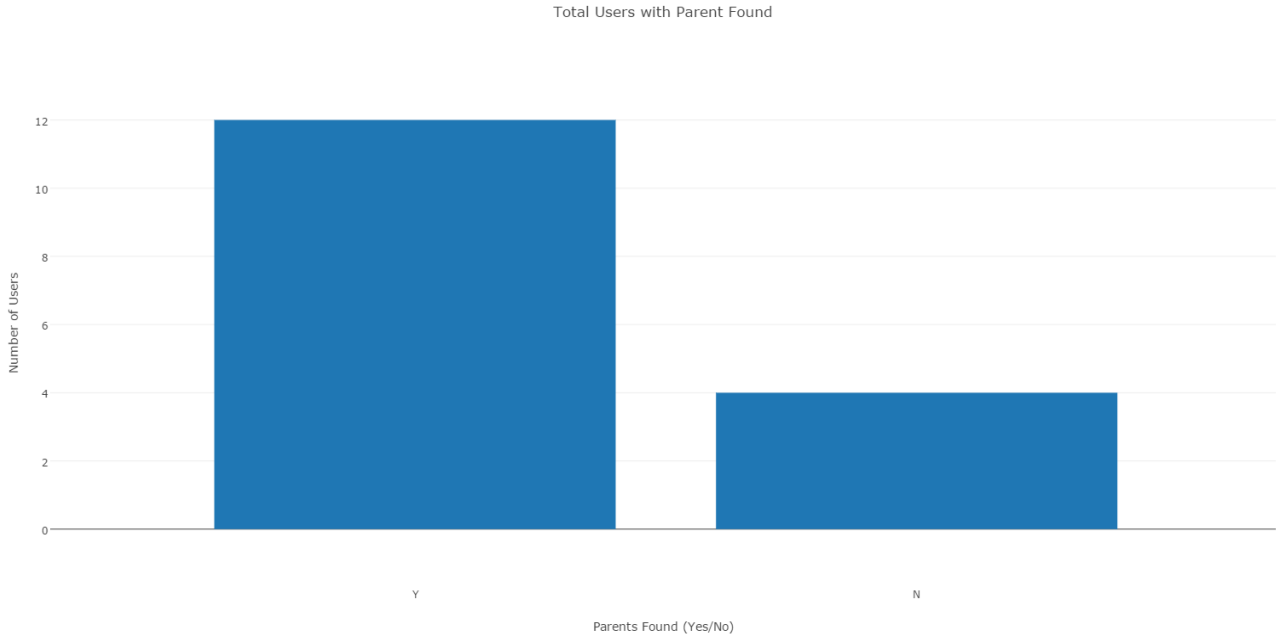


Figure 5.4: Graph of total users with at least one parent found and with no parent found.

As can be seen in Figure 5.4, or in Table 5.3 above, the application successfully managed to obtain the parents of the targeted users 75 percent of the time. The other 25 percent

is made up of four users who are not friends with their parents as can be seen in the table since the only user with a false negative is user nine for whom the application still managed to find a parent. This implies the application's accuracy in finding parents is 100 percent since all users that are friends with at least one parent were found by the application.

5.5 Usability

The ease of use of the application is an important aspect of usability. Since the application is aimed at users of all ages ranging from age 13 and up with the possibility of younger users, the application must be simple to use in order to accommodate all possible users. In order to determine whether the application is usable, the user interface of the application will be analysed to determine how learnable it is, how memorable it is and how easy it would be to make a mistake. These categories will be analysed in comparison to Facebook's user interface as a baseline for a usable interface since all users of Facebook will have a working knowledge of how to use Facebook's user interface.

Figure 5.5: Dislike button added to post.

The application was designed to simplify the process of using it by automatically adding a button to all posts and comments, as can be seen in Figure 5.5. This simplifies the

process by using an already known feature of Facebook, the “Like” button, which users are accustomed to already meaning it should be easy to learn, very memorable and difficult to make a mistake. It also make the application pleasant to use since the “Like” button on Facebook is regarded as a satisfactory feature to which users enjoy using.

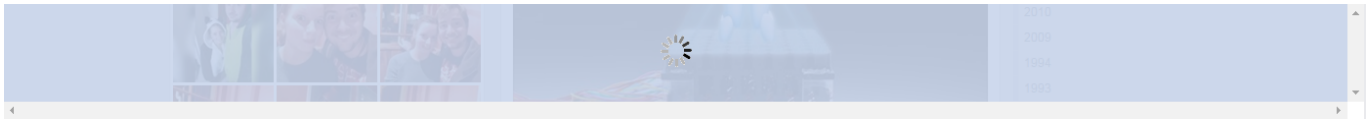


Figure 5.6: Output overlay with loading GIF shown.

After the button is pressed, the output overlay is shown with a moving loading icon known as a GIF as can be seen in Figure 5.6. This was done to prevent users from assuming the application is not running since execution time can vary based on the targeted user. This should also prevent users from making errors while using the application by not clicking the added button multiple times which could cause the application to break or produce incorrect results. Once the application finishes executing, the results are shown in the overlay as can be seen in Figure 5.7. The output produced shows the direct relatives found by the application as hyperlinks to the profile pages of the users with the score produced by the application followed by the rest of the target user’s friend list. Using a hyperlink shown as the user’s name is a common occurrence on Facebook so this should be memorable to users of the application making it simple to understand how to proceed. In order to contact the intended user; the application’s user need only click on the hyperlink, which will redirect to the intended user’s profile page where the application’s user should be familiar enough with general layout of a Facebook profile to find the “Message” button. This will allow the application’s user to contact the intended user in a way that is familiar to a Facebook user which should improve the usability of the application by not forcing the application’s user to learn anything new and using memorable details of the Facebook user interface.

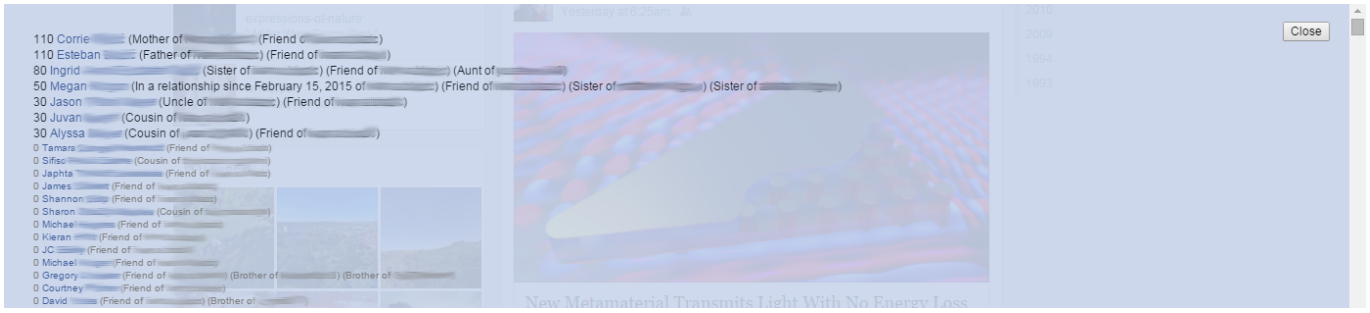


Figure 5.7: Output overlay showing output.

5.6 Summary

Overall, the application seems to have a high coverage of the target user's friend list using the algorithm described in Chapter 4 as can be seen in Table 5.1 with an average coverage of 89.54 percent. This implies the application is not retrieving (on average) 10 percent of the target user's friend list giving a 1/10 chance of not finding a direct relative. The application doesn't score very well in efficiency having an average of 60.25 percent efficiency implying that on average the application sends 0.6 requests per node found which means for every three requests, five nodes are found. This is not a bad result since efficiency is not the goal of the application and instead higher coverage and accuracy is more necessary for success. With respect to accuracy, the application scores exact accuracy over half the time and when this is not the case, using the conventional calculation of accuracy, scores near exact accuracy ranging between 1.00 and 0.9886 meaning the application is capable of correctly determining that a user's friend is not a direct family member. On the other hand, using the alternative calculation of accuracy it can be seen that the application sometimes falters on correctly determining direct relatives producing some false positives. The application also seems to be overall quite usable although further testing might be needed to prove this.

Chapter 6

Conclusion

In this chapter, the research conducted in this document will be concluded. This will be done by firstly summarising each chapter prior to this one giving an overview of what has been done, although this will exclude the results chapter since a conclusion to the results produced will be provided next followed by the conclusion this research. Discussions will be provided in both these sections on the success of the application and the success of the overall research. Finally future work to be done on the application will be mentioned as well as possible future research to be done.

6.1 Summary of Chapters

The introduction of this research was given in Chapter 1. This included the problem statement which outlined the purpose of the research being a means to help remedy cyberbullying and online harassment due to the continuous increase in use of social networking sites. Following this, the goals of this research were outlined proposing the use of doxxing to help remedy online harassment allowing victims to private message the relatives of the harasser making them aware of what has happened. The proposed approach was then given, detailing the use of a web browser add-on to automatically collect data and run heuristics as a means of producing the direct family members of a targeted social network site user.

Detailed background knowledge was provided in chapter two using resources from books, papers, articles, the internet and so on pertaining to the necessary fields of knowledge

needed to produce this research. This included a definition for and background information on social network sites such as how they work and how they are used by the general public. Following this, privacy on social network sites was investigated, detailing some of Facebook's privacy mishaps and privacy changes made throughout the years. Privacy implications were also given describing how freely social network site user divulge personal information and the risks involved with this including but not limited to online stalkers, re-identification and online harassment. The interactions between family members on social network sites was also investigated to help provide ideas for possible heuristics further down the line. Next, social network graphs were researched providing background knowledge of what they are, how they work and what algorithms could be used to help determine connections between people including a description of what heuristics are. Lastly, previous works of research or development relating to this research were provided and discussed to outline results already found and problems already encountered as a means to avoid repeating research.

The design choices made for the application were provided in Chapter 3 starting off with a detailed description of how a Google Chrome extension works and the general structure of the files needed to produce a working application. This was followed by a description of the possible architectural paths that could be taken including a monolithic architecture and a client-server architecture. A discussion was provided on the advantages and disadvantages of each coming to the decision of using a monolithic architecture since it will scale better since no cost is needed to upkeep or scale a server and instead the application will run completely client-side. Next, the methods for user interaction were discussed with a decision made to use buttons embedded into each Facebook page as user input and a pop-up dialog generated by the application for output. Following this, methods for data collection were proposed with descriptions of how they would work and how they could be implemented using HTML requests and parsing the responses. A description of heuristics and the possible heuristics to be used by the application were given detailing some of the information that may be needed to produce an educated guess as to which friends of the targeted user may be directly related. Lastly, both the possible internal and external libraries available for use were described providing examples of how they could be used by the application to accomplish the design decisions made.

The methods used during implementation were provided in chapter four and are broken down into four major areas of implementation; input, data collection, heuristics and output. The chapter is organised in order of implementation beginning with user input detailing how buttons were embedded into every Facebook page automatically by the application and how the application is able to respond to a button press knowing its

origin location on the given page. Next was data collection describing in more detail how HTML requests were sent by the application using jQuery's AJAX methods and how the responses were parsed and traversed to retrieve the necessary information. Also included were the obstacles encountered during the implementation of data collection since this was one of the more complicated areas to successfully get working. After this, the heuristics used were mentioned along with a description of how they were implemented and used to determine a user's direct family members. Finally, the applications method for output is given describing how the dialog box was added to the page and how the user can use this to send private messages to the provided users with proof of the abusive post or comment.

6.2 Concluding Results and Final Statement

Testing was carried out on 16 users including users with little to no privacy settings as well as users with high privacy settings and was executed from the perspective of a user who is friends with all targeted users. This was done so that values such as the total number of friends each user has can be determined by viewing the user's profile page. The results produced fall into three categories: coverage, efficiency and accuracy.

6.2.1 Coverage

Coverage determines the percentage of a user's friend list generated by the application and is used to show the likelihood of the application missing a direct relative by not covering them during data collection. The results for coverage show that on average the application generates 89.54 percent with a standard deviation of 6.96 percent meaning the variation in coverage is quite low although this also implies that the application is in general missing 10 percent of the targeted user's friend list producing a 1/10 chance to miss a direct relative. It must also be noted that the friends not being covered by the application are very likely to be nodes in the social network graph that have no other connections to the target's graph whether singular nodes or cliques. This implies that if a family member is found, the chances of other family members falling into this 10 percent are quite low since connections can be made using the known family member. Thus to conclude, the application seems to have a successful coverage of the target user's social network.

6.2.2 Efficiency

Efficiency is calculated by determining the requests per node found and is used to show the amount of work needed to produce a social network graph. The results of efficiency calculations show that on average the application sends 0.6025 requests per node found which implies the application is very efficient considering this means it takes three requests to produce five nodes. Although, with a standard deviation of 0.0786 requests per node, the efficiency of the application does not vary by much implying the application's efficiency is deterministic. This can also be shown by considering that the application sends a request for more friends and for family information every time a new unique friend is found meaning the number of friend requests and the number of family requests will always be close to that of the total friends found. The two lowest scoring users, the only two to fall outside one standard deviation from the mean, also happen to be the two users with the highest privacy settings. Although, due to the low number of test cases, this can only imply that privacy settings affect the applications efficiency and further research would need to be done to prove this. To conclude, the application's efficiency is not optimal, however, efficiency was not the goal of the application.

6.2.3 Accuracy

Accuracy is used to determine the correctness of the results produced by the application and is the degree to which the results conform to the correct value. To calculate accuracy, two formulas were used. This includes the conventional calculation of accuracy determined by summing the number of true positives and true negatives dividing this result by the sum of true positives, true negative, false positives and false negatives. Also included is an alternative calculation used to highlight false negatives by removing true negatives from the aforementioned formula. Using the conventional calculation of accuracy it has been shown that the application is highly capable of correctly reducing a large number of friends into a short list of likely family members. Using the alternative calculation of accuracy it has also been shown that the application produces many false negatives with some users scoring only 50 percent accuracy. However, the application gives each family member generated a scoring of their likelihood to be a direct family member, and in these cases, the false positives produced always score low in this respect. Thus, the applications accuracy is in general relatively high especially when considering that over half the users tested scored perfect accuracy using both calculations.

6.2.4 Final Statement

Based on the results described above, it can be seen that the application was successful in determining the direct relatives of a Facebook user. The application can cover enough of the target's social network graph to accurately determine their direct relatives albeit not efficiently. The application also seems to be sufficiently usable by the target audience by making use of known features within the Facebook user interface to simplify the application's user interface. This makes the research goal more achievable by ensuring any Facebook user can make use of the application in order to counter cyberbullying or online harassment.

6.3 Future Work

There are many possible additions that could me made to the application produced as well as much more research to be done in this field. Firstly, the application could be improved by collecting more data about each individual user which can then be used to add more heuristics. The amount of possible data that could be collected is near endless including information about work and education, places lived and personal information such as date of birth or religious views. This can all be found on a user's profile page depending on privacy settings and/or the relationship between the application and the target user. More heuristics could be added using this information such as checking age differences, similar religious views or whether they have lived in similar locations. On top of this, the user's wall could be scraped for information on who they communicate with and what they communicate about. Other possible extensions to the application include building a centralised database accessed by each application user to share collected data, although this will likely break Facebook's terms of use. A more in-depth addition that could be made to the application is to integrate other social network sites to gather even more information such as LinkedIn which contains plenty of freely available information. Further research that could be done it to possibly field-test the application to see how well it remedies online harassment or cyberbullying using a test group such as a high school or a university campus. Other future research to be done could be to further test the boundaries of privacy settings by testing the application on a group of willing strangers with little to no connection to the application user to see if the application maintains its accuracy.

References

Adamic, L. A., & Adar, E. (2003). Friends and neighbors on the web. *Social networks*, *25*(3), 211–230.

Boyd, D. (2008). Facebook's privacy trainwreck: Exposure, invasion, and social convergence. *Convergence: The International Journal of Research into Media Technologies*, *14*(1), 13–20.

Boyd, D., & Ellison, N. (2007). Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, *13*(1), 210–230.

Debatin, B., Lovejoy, J. P., Horn, A.-K., & Hughes, B. N. (2009). Facebook and online privacy: Attitudes, behaviors, and unintended consequences. *Journal of Computer-Mediated Communication*, *15*(1), 83–108.

De Meo, P., Ferrara, E., & Fiumara, G. (2011). Finding similar users in facebook. *Social Networking and Community Behavior Modeling Qualitative and Quantitative Measurement, IGI Global*, 304–323.

Finkelhor, D., Mitchell, K. J., & Wolak, J. (2000). *Online victimization: A report on the nation's youth.* (Tech. Rep.). National Center for Missing and Exploited Children, Crimes against Children Research Center.

Goldberg, M., Kelley, S., Magdon-Ismail, M., Mertsalov, K., & Wallace, A. (2010). Finding overlapping communities in social networks. In *Social computing (socialcom), 2010 ieee second international conference on* (pp. 104–113).

Google. (2015a, Mar 31). *Content scripts.* `https://developer.chrome.com/extensions/content_scripts`.

Google. (2015b, Mar 23). *Getting started: Building a chrome extension.* `https://developer.chrome.com/extensions/getstarted`.

Gross, R., & Acquisti, A. (2005). Information revelation and privacy in online social networks. In *Proceedings of the 2005 acm workshop on privacy in the electronic society* (pp. 71–80).

Haythornthwaite, C. (2005). Social networks and internet connectivity effects. *Information, Community & Society*, *8*(2), 125–147.

Jagatic, T. N., Johnson, N. A., Jakobsson, M., & Menczer, F. (2007). Social phishing. *Communications of the ACM*, *50*(10), 94–100.

Jones, H., & Soltren, J. H. (2005). Facebook: Threats to privacy. *Project MAC: MIT Project on Mathematics and Computing*, *1*.

jQuery. (2015, Sept 25). *What is jquery?* `https://jquery.com/`.

Kanter, M., Afifi, T., & Robbins, S. (2012). The impact of parents friending their young adult child on facebook on perceptions of parental privacy invasions and parent–child relationship quality. *Journal of Communication*, *62*(5), 900–917.

Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, *18*(1), 39–43.

Kautz, H., Selman, B., & Shah, M. (1997). Referral web: combining social networks and collaborative filtering. *Communications of the ACM*, *40*(3), 63–65.

Korf, R. E. (1990). Real-time heuristic search. *Artificial intelligence*, *42*(2), 189–211.

Krebs, V. E. (2002). Mapping networks of terrorist cells. *Connections*, *24*(3), 43–52.

Liben-Nowell, D., & Kleinberg, J. (2007). The link-prediction problem for social networks. *Journal of the American society for information science and technology*, *58*(7), 1019–1031.

Nazir, A., Raza, S., & Chuah, C.-N. (2008). Unveiling facebook: a measurement study of social network based applications. In *Proceedings of the 8th acm sigcomm conference on internet measurement* (pp. 43–56).

Newman, M. E. (2001). Clustering and preferential attachment in growing networks. *Physical Review E*, *64*(2), 25–102.

Rosen, K. (2011). *Discrete mathematics and its applications* (7th ed.). McGraw-Hill.

Salton, G., & McGill, M. J. (1983). Introduction to modern information retrieval.

Samarati, P., & Sweeney, L. (1998). *Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression* (Tech. Rep.). SRI International.

Statista. (2015, Feb 25). *Statistics and facts about social networks.* `http://www.statista.com/topics/1164/social-networks/`.

Ybarra, M. L., & Mitchell, K. J. (2004). Youth engaging in online harassment: Associations with caregiver–child relationships, internet use, and personal characteristics. *Journal of adolescence*, *27*(3), 319–336.

Ybarra, M. L., & Mitchell, K. J. (2008). How risky are social networking sites? a comparison of places online where youth sexual solicitation and harassment occurs. *Pediatrics*, *121*(2), 350–357.

Appendix A

Code Listings

A.1 manifest.json

```
{
   "name": "Show Links",
   "description": "Show all links on current page.",
   "version": "0.1",
   "minimum_chrome_version": "16.0.884",
   "permissions": ["<all_urls>"],
   "browser_action": {"default_popup": "popup.html"},
   "manifest_version": 2
}
```

A.2 popup.html

```
<!DOCTYPE html>
<head>
<script src='popup.js'></script>
</head>
<body>
<input type=text id=filter placeholder=Filter>
<input type=checkbox id=regex>
<label for=regex>Regex</label><br>
<table id=links>
```

```
    <tr>
      <th align=left>URL</th>
    </tr>
</table>
</body>
</html>
```

A.3 popup.js

```
var allLinks = [];
var visibleLinks = [];

function showLinks() {
  var linksTable = document.getElementById('links');
  while (linksTable.children.length > 1) {
    linksTable.removeChild(linksTable.children[linksTable.children.length
  }
  for (var i = 0; i < visibleLinks.length; ++i) {
    var row = document.createElement('tr');
    var col0 = document.createElement('td');
    var col1 = document.createElement('td');
    var checkbox = document.createElement('input');
    checkbox.checked = true;
    checkbox.type = 'checkbox';
    checkbox.id = 'check' + i;
    col0.appendChild(checkbox);
    col1.innerText = visibleLinks[i];
    col1.style.whiteSpace = 'nowrap';
    col1.onclick = function() {
      checkbox.checked = !checkbox.checked;
    }
    row.appendChild(col0);
    row.appendChild(col1);
    linksTable.appendChild(row);
  }
}
```

```
function toggleAll() {
  var checked = document.getElementById('toggle_all').checked;
  for (var i = 0; i < visibleLinks.length; ++i) {
    document.getElementById('check' + i).checked = checked;
  }
}

function downloadCheckedLinks() {
  for (var i = 0; i < visibleLinks.length; ++i) {
    if (document.getElementById('check' + i).checked) {
      chrome.downloads.download({url: visibleLinks[i]},
                                                function(id) {
      });
    }
  }
  window.close();
}

function filterLinks() {
  var filterValue = document.getElementById('filter').value;
  if (document.getElementById('regex').checked) {
    visibleLinks = allLinks.filter(function(link) {
      return link.match(filterValue);
    });
  } else {
    var terms = filterValue.split(' ');
    visibleLinks = allLinks.filter(function(link) {
      for (var termI = 0; termI < terms.length; ++termI) {
        var term = terms[termI];
        if (term.length != 0) {
          var expected = (term[0] != '-');
          if (!expected) {
            term = term.substr(1);
            if (term.length == 0) {
              continue;
            }
```

```
          }
          var found = (-1 !== link.indexOf(term));
          if (found != expected) {
            return false;
          }
        }
      }
      return true;
    });
  }
  showLinks();
}

chrome.extension.onRequest.addListener(function(links) {
  for (var index in links) {
    allLinks.push(links[index]);
  }
  allLinks.sort();
  visibleLinks = allLinks;
  showLinks();
});

window.onload = function() {
  document.getElementById('filter').onkeyup = filterLinks;
  document.getElementById('regex').onchange = filterLinks;
  document.getElementById('toggle_all').onchange = toggleAll;
  document.getElementById('download0').onclick = downloadCheckedLinks;
  document.getElementById('download1').onclick = downloadCheckedLinks;

  chrome.windows.getCurrent(function (currentWindow) {
    chrome.tabs.query({active: true, windowId: currentWindow.id},
                      function(activeTabs) {
      chrome.tabs.executeScript(
        activeTabs[0].id, {file: 'send_links.js', allFrames: true});
    });
  });
};
```

A.4 send_links.js

```
var links = [].slice.apply(document.getElementsByTagName('a'));
links = links.map(function(element) {
  var href = element.href;
  var hashIndex = href.indexOf('#');
  if (hashIndex >= 0) {
    href = href.substr(0, hashIndex);
  }
  return href;
});

links.sort();

// Remove duplicates and invalid URLs.
var kBadPrefix = 'javascript';
for (var i = 0; i < links.length;) {
  if (((i > 0) && (links[i] == links[i - 1])) ||
      (links[i] == '') ||
      (kBadPrefix == links[i].toLowerCase().substr(0, kBadPrefix.length))) {
    links.splice(i, 1);
  } else {
    ++i;
  }
}

chrome.extension.sendRequest(links);
```