Social Interactome Recommender Project

Matt Blumen, Sarah Devlin, Mary Kwiatkowski, Bridget Larcher,
Wilson Rhodes, and Timothy Tran
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

Final Report
CS 4624 - Multimedia, Hypertext, and Information Access
Instructor - Professor Edward A. Fox
March 17, 2017
Client - Prashant Chandrasekar

Table of Contents

Table of Figures

Table of Tables

1 Executive Summary

This project was a continuation of efforts conducted by the CS4624 team of Spring 2016. The previous team built a procedure to find the top k friends based on information submitted via surveys[1]. This year's project dealt with analyzing data received through these surveys in order to draw conclusions about different homophily-based measures that are prominent between recovery buddies on a live social network, Friendica[2]. This year's team of students worked with data that had been collected from the social network, which contain information about articles read, modules liked, and other information that may be used to find commonalities and form a friendship.

After parsing the information, we evaluated friendships and the homophily-based measures that the two people have in common. We analyzed them to find trends through the visualization of data (histograms) and a top-down approach. Our main focus was our top-down approach, in which we determined the similarity scores of two recovery buddies given their similarities in demographics. When we identified pertinent demographics, we calculated the probabilities of similarities so that we can statistically describe how friendships are driven by similarities on Friendica[2]. This was part of our final deliverable. We also focused on diffusion. We analyzed the tendency of a user to attend a meeting, watch a video module, or complete other tasks because another recovery buddy did so. This helped us identify how the network experiences diffusion. We used diffusion to identify users that experience high or low amounts of interaction with other users and can identify their similarities through homophily-based measures.

One of this team's focuses included different aspects of weighting different feature types. This mainly meant tuning parameters and observing the changes that those parameters produced. We needed to understand how to tune these parameters and how to improve the outcome. Another focus this semester was making predictions on friendships based on the answers submitted by participants through the surveys given. These findings gave the team insight to distinguish contact from homophily[3]. The team gained a visual understanding of this information through histograms. Socio-economic status, gender, and number of addictive substances are key parts of homophily in this project that were visually observed.

Our results were trends in our top-down and diffusion approaches. Top-down resulted in 55 close relationships with many of them being of the same gender and income level. Our diffusion results gave us the level of influence particular users had on each other. Our final deliverables consisted of documentation of these results and the code that went into finding them. Our top-down and diffusion results, along with our analysis on Homophily are our main deliverables.

2 User Manual

2.1 Installing Technologies

2.1.1 Python

1. Visit the Python[4] download page.
2. Scroll to the Download section.
3. For Windows users, click on the link labeled either "Windows x86 MSI Installer (2.7.0)" or "Windows X86-64 MSI Installer" depending upon which version is being used.
4. For Mac users, click on the link labeled either "Mac Installer disk image (2.7.0) for OS X 10.5 and later" or "32-bit Mac Installer disk image (2.7.0) for OS X 10.3 and later" depending upon which version is being used.
5. Verify the MD5 checksum before opening the file (appropriate checksums are located towards the bottom of the Download section.
6. Open the file and the Operating System will set up and install everything.
7. For Windows users, you may need to set up the path environment variable. In the command prompt, enter the command "path %path%;C:\Python".
8. For Mac users, you may need to set up the path environment variable . In the bash shell, enter the command "export ATH="$PATH:/usr/local/bin/python".
9. To confirm that Python has been installed correctly, enter the command "python -h" in either the command prompt for Windows users or the bash shell for Mac users. If a list of options appears, Python has been successfully installed.

2.1.2 Java (JRE / JDK)

1. Visit the Java[5] download page.
2. Click on the button labeled "Free Java Download". You will be taken to the download page for your specific Operating System.
3. Click on the button labeled "Agree and Start Free Download".
4. Open the file and the Operating System will set up and install the Java Runtime Environment (JRE), which is used for running Java code.
5. To confirm that the JRE has been installed correctly, enter the command "java -h" in either the command prompt for Windows users or the bash shell for Mac users. If a list of options appears, the JRE has been successfully installed.
6. Visit the JDK download page.

7. Scroll to the Java SE Development Kit 8u131 section and accept the license agreement.
8. For Windows users, click on the link labeled either “jdk-8u131-windows-i586.exe” or “jdk-8u131-windows-x64.exe” depending upon which version is being used.
9. For Mac users, click on the link labeled “jdk-8u131-macosx-x64.dmg”.
10. Verify the SHA256 checksum before opening the file (appropriate checksums are located at the checksum page).
11. Open the file and the Operating System will set up and install the Java Development Kit (JDK), which is used for compiling and testing code.
12. For Windows users, you may need to set up the path environment variable. In the command prompt, enter the command “path %path%;C:\Program Files\Java\jdk1.8.0_131”.
13. For Mac users, you may need to set up the path environment variable . In the bash shell, enter the command “export ATH=”$PATH:/usr/local/bin/Java/jdk1.8.0_131”.
14. To confirm that the JDK has been installed correctly, enter the command “javac -h” in either the command prompt for Windows users or the bash shell for Mac users. If a list of options appears, the JDK has been successfully installed.

2.1.3 Excel (GiGraph)

1. Make sure you have Microsoft Office installed on your Operating System.
2. Visit the GiGraph[6] download page.
3. Click on the button labeled “Download from the Office Store”.
4. Click on the green button labeled “Add”.
5. Click on the link labeled “Open in Excel”. You will be redirected to Microsoft Excel[7].
6. Click on the button labeled “Enable Editing”.
7. Click on the button labeled “Trust this add-in”.

2.1.4 Git

1. For Windows users, visit the Git[8] download page for Windows and click the button labeled “Download”. Open the file and a Git Bash will be set up and installed for you to use Git. Alternative options include using Ubuntu Bash, that is now included with Windows 10, and typing the command “sudo apt-get install git-all”.
2. For Mac users, visit the Git download page for Mac and Git will be downloaded automatically.

2.2 Top-Down

2.2.1 Running the code

Using the Terminal (for Mac or Linux) or Git Bash[8] (for Windows)

1. Enter the command “git clone https://github.com/tttran/SIFriendRecommender.git” This will create a local copy of the repository.
2. Enter the command “cd SIFriendRecommender”. The user has now entered the project folder.
3. Enter the command “cd topdown” . The user has now entered with folder with the runnable Python[4] code for the top-down approach.
4. Enter the command “python topDown.py”. The user has now run the program and will receive outputs on the data.

2.2.2 Understanding the outputs

Once the code has finished running, it will output the data collected into a text-file titled Timepoint3Statistics.txt. Table 2.1 shows the four categories of output in the text file.

Table 2.1: Timepoint3Statistics Formatted Output

Output Name	**Description**
Reported Relationship Statuses	This output shows the number of users that reported they are “not close”, “somewhat close”, and “very close” to their recover buddies. This data is outputted for the Lattice Network, the Small World Network, and the combination of both networks.
Lattice Network Statistics	This output shows how many demographic similarities there exists in the Lattice Network among relationships that were reported “somewhat close” or “very close”.
Small World Network Statistic	This output shows how many demographic similarities there exists in the Small World Network among relationships that were reported “somewhat close” or “very close”.
Combined Statistics	This output shows how many demographic similarities there exists in both the Lattice Network and the Small World Network among relationships that were reported “somewhat close” or “very close”.
One-Way/Two-Way Relationship	This output shows how many one-way and two-way relationships exist in Timepoint3.

2.2.3 Visualization

Figures 2.1 and 2.2 were created by analyzing the top-down output of the Small World and Lattice Network relationships. The figures were created in Excel, using the similarity statistics to visualize what recovery buddies had in common.

Figures 2.1 depicts the number of distinct relationships among somewhat close and very close users in each demographic category. The orange bar represents the number of Small World Network relationships for each demographic. The blue bar represents the number of Lattice Network relationships for each demographic.

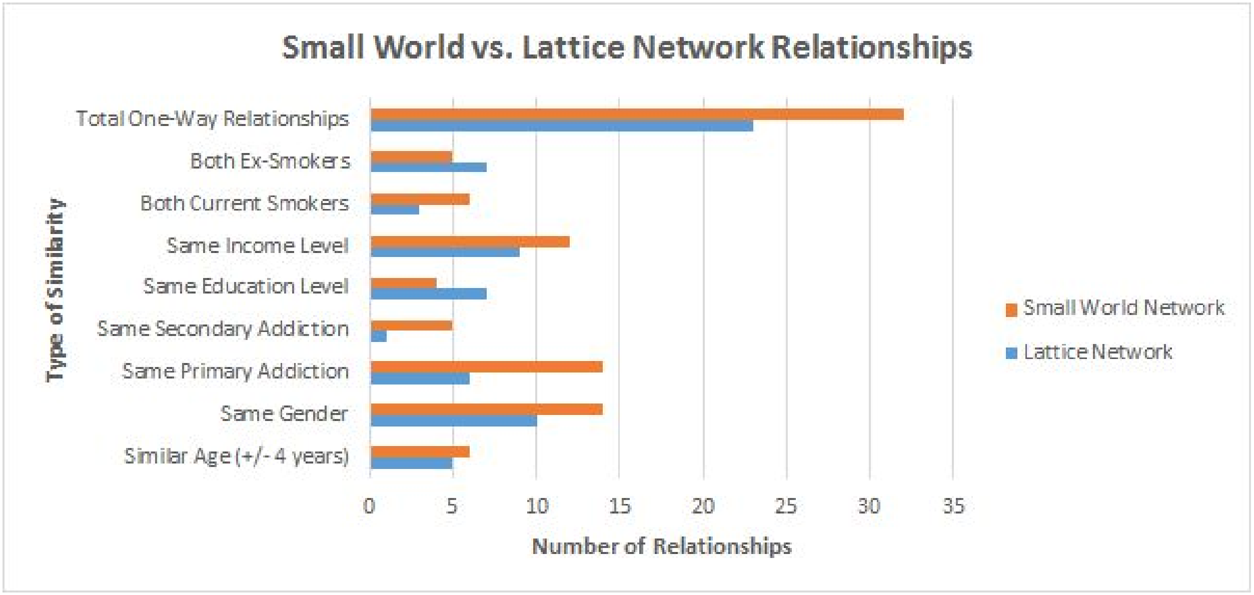


Figure 2.1: Small World and Lattice Network Demographic Relationships

Figure 2.1 depicts the comparison of the total number of “somewhat close” and “very close” users among the combined Lattice and Small World Networks in each demographic category. The blue section of the bars represent the number of users that shared the particular demographic in the relationship. The orange section of the bars represent the number of users that did not share the same demographics.

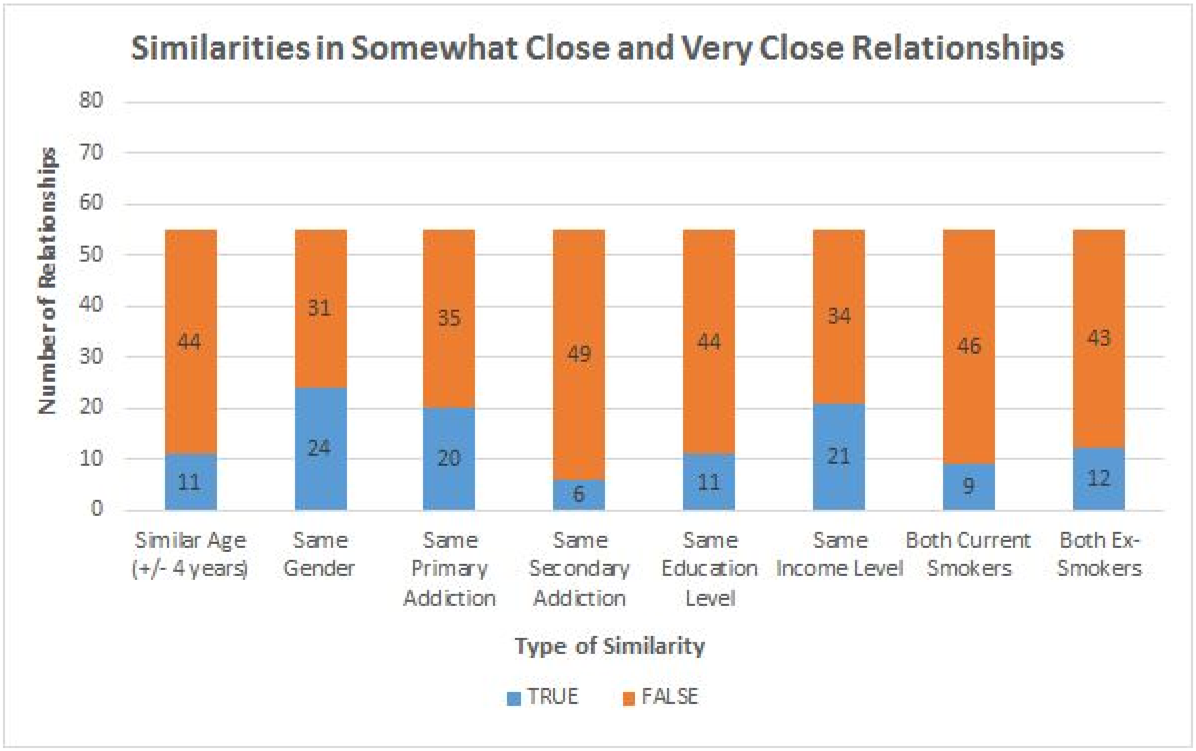


Figure 2.2: The Demographic Similarities Compared to the Total Number of Close Users in both Networks

2.3 Diffusion

2.3.1 Running the code

Using the Terminal (for Mac or Linux) or Git Bash[8] (for Windows)

1. Enter the command "git clone https://github.com/tttran/SIFriendRecommender.git" This will create a local copy of the repository.
2. Enter the command "cd SIFriendRecommender". The user has now entered the project folder.
3. Enter the command "cd diffusion" . The user has now entered with folder with the runnable Java[5] code for the diffusion code.
4. Enter the command "java -jar diffusion.jar" . The user has now run the diffusion program and will receive 4 output CSV files.

2.3.2 Understanding the outputs

The diffusion code will output the four CSV files, shown in Table 2.2.

Table 2.2: Diffusion Code Output Files with Descriptions.

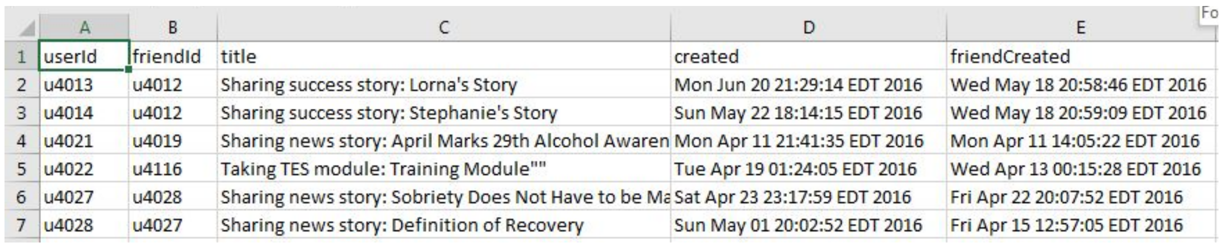
CSV name	Description

data.csv	This CSV contains all basic diffusion data in a format that should be easy to process further. Each row represents a post that a user made after their friend did.
network.csv	This CSV contains each relationship that exhibits diffusion along with the number of times that the user makes a particular post after their friend does.
lattice_network.csv	This CSV contains each relationship in the lattice network that exhibits diffusion in the same format as network.csv.
smallWorld_network.csv	This CSV contains each relationship in the small world network that exhibits diffusion in the same format as network.csv.

Each of the four files is detailed further in the Tables 2.3-2.6. This information can also be found in the data dictionary in our GitHub repository[9].

Table 2.3: The Format of data.csv Columns and a Visual Representation of Sample Data

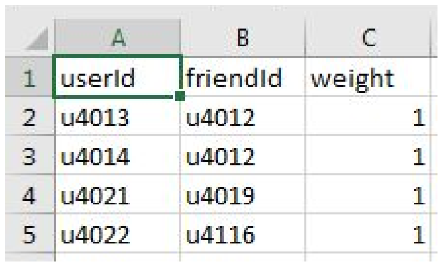
Column name	**What it contains**
userId	The ID of the user.
friendId	The ID of the friend.
title	The title of the post.
created	The date the user made the post.
friendCreated	The date the friend made the post.



	A	B	C	D	E
1	userId	friendId	title	created	friendCreated
2	u4013	u4012	Sharing success story: Lorna's Story	Mon Jun 20 21:29:14 EDT 2016	Wed May 18 20:58:46 EDT 2016
3	u4014	u4012	Sharing success story: Stephanie's Story	Sun May 22 18:14:15 EDT 2016	Wed May 18 20:59:09 EDT 2016
4	u4021	u4019	Sharing news story: April Marks 29th Alcohol Awaren	Mon Apr 11 21:41:35 EDT 2016	Mon Apr 11 14:05:22 EDT 2016
5	u4022	u4116	Taking TES module: Training Module""	Tue Apr 19 01:24:05 EDT 2016	Wed Apr 13 00:15:28 EDT 2016
6	u4027	u4028	Sharing news story: Sobriety Does Not Have to be Ma	Sat Apr 23 23:17:59 EDT 2016	Fri Apr 22 20:07:52 EDT 2016
7	u4028	u4027	Sharing news story: Definition of Recovery	Sun May 01 20:02:52 EDT 2016	Fri Apr 15 12:57:05 EDT 2016

Table 2.4: The Format of network.csv Columns and a Visual Representation of Sample Data

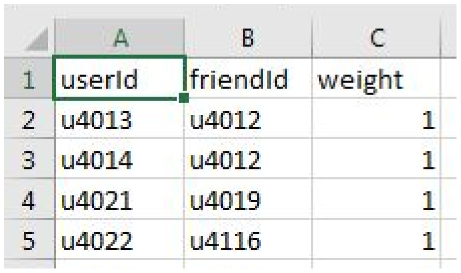
Column Name	What it contains
userId	The ID of the user.
friendId	The ID of the friend.
weight	The number of times the user made the same post after their friend.



	A	B	C
1	userId	friendId	weight
2	u4013	u4012	1
3	u4014	u4012	1
4	u4021	u4019	1
5	u4022	u4116	1

Table 2.5: The Format of lattice_network.csv Columns and a Visual Representation of Sample Data

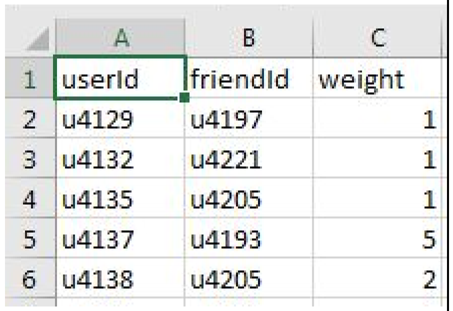
Column Name	What it contains
userId	The ID of the user in the Lattice network.
friendId	The ID of the friend in the Lattice network.
weight	The number of times the user made the same post after their friend.



	A	B	C
1	userId	friendId	weight
2	u4013	u4012	1
3	u4014	u4012	1
4	u4021	u4019	1
5	u4022	u4116	1

Table 2.6: The Format of smallWorld_network.csv Columns and a Visual Representation of Sample Data

Column Name	What it contains
userId	The ID of the user in the Small World network.
friendId	The ID of the friend in the Small World network.
weight	The number of times the user made the same post after their friend.

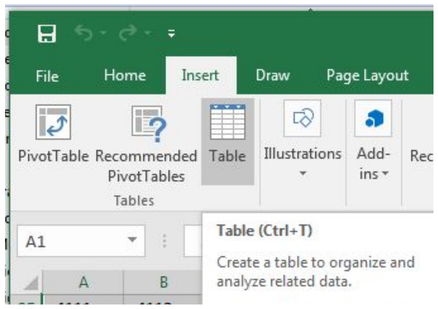
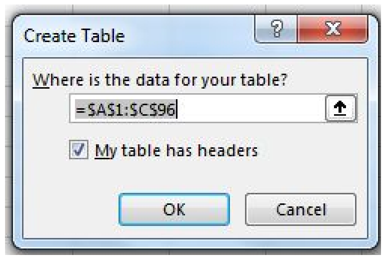
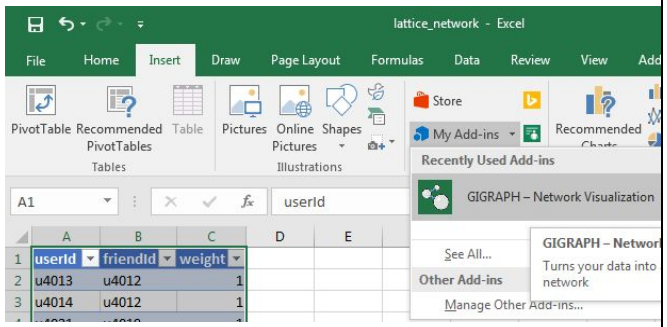


	A	B	C
1	userId	friendId	weight
2	u4129	u4197	1
3	u4132	u4221	1
4	u4135	u4205	1
5	u4137	u4193	5
6	u4138	u4205	2

2.3.3 Visualization

To visualize the information output by the diffusion code as a directed network graph, follow the instructions found in Table 2.7. It is important to note that data.csv is not in the correct format for visualization. network.csv, smallWorld_network.csv and lattice_network.csv have all been optimized for visualization and only those files should be used with the instructions.

Table 2.7: Instructions to Create Diffusion Network Visualizations using GiGraph[6] in Microsoft Excel[7]

	Instructions	
1.	Open the network.csv, lattice_network.csv, or the smallWorld_network.csv in Microsoft Excel[7].	
2.	Select all data (Ctrl+a) and click Insert and then click Table.	 File Home Insert Draw Page Layout PivotTable Recommended PivotTables Table Illustrations Add-ins Tables A1 Table (Ctrl+T) Create a table to organize and analyze related data.
3.	In the pop-up box, select the “My Table Has Headers” box and click Okay.	 Create Table Where is the data for your table? =$A$1:$C$96 ☑ My table has headers OK Cancel
4.	Next click Insert, then click My Add-Ins, and then click GiGraph[6] and click Start.	 lattice_network - Excel File Home Insert Draw Page Layout Formulas Data Review View PivotTable Recommended PivotTables Table Pictures Online Pictures Shapes Tables Illustrations Store My Add-ins Recommended Charts Recently Used Add-ins GIGRAPH – Network Visualization See All... Other Add-ins Manage Other Add-ins... Turns your data into network A1 userId userId friendId weight u4013 u4012 1 u4014 u4012 1
5.	Click Start, then click Continue twice, and then click Finish.	
6.	This should generate a network graph for you. You can click Settings or zoom in and out on the toolbar to manipulate the graph.	

The network graphs in Figure 2.3 and Figure 2.4 visually show the network of diffusion between users in the Lattice and Small World networks, respectively.

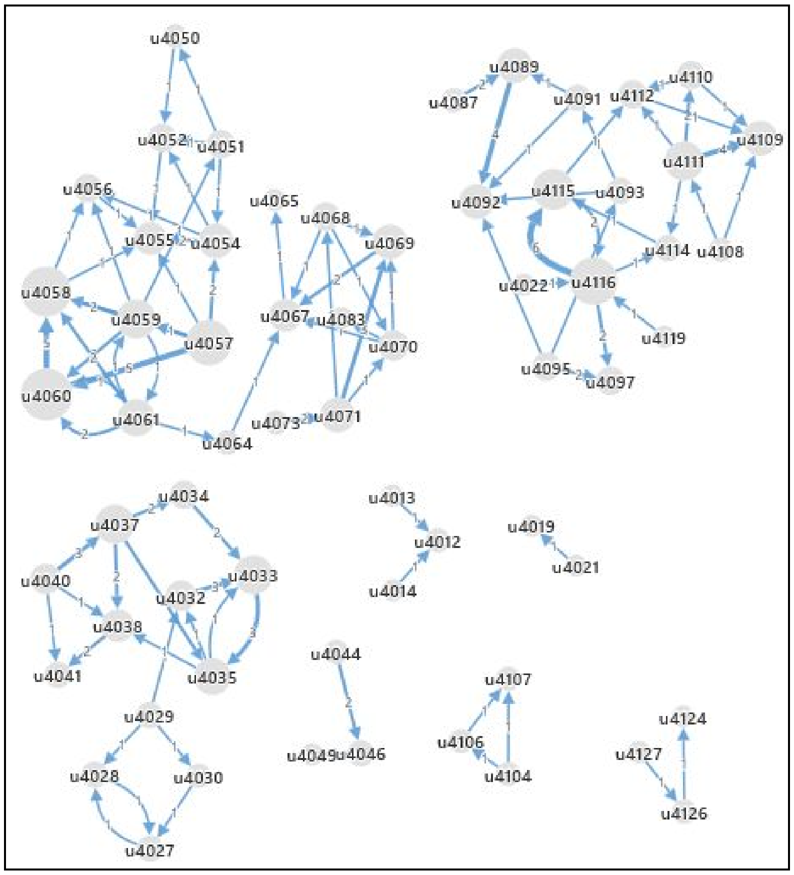


Figure 2.3: Lattice Network Diffusion

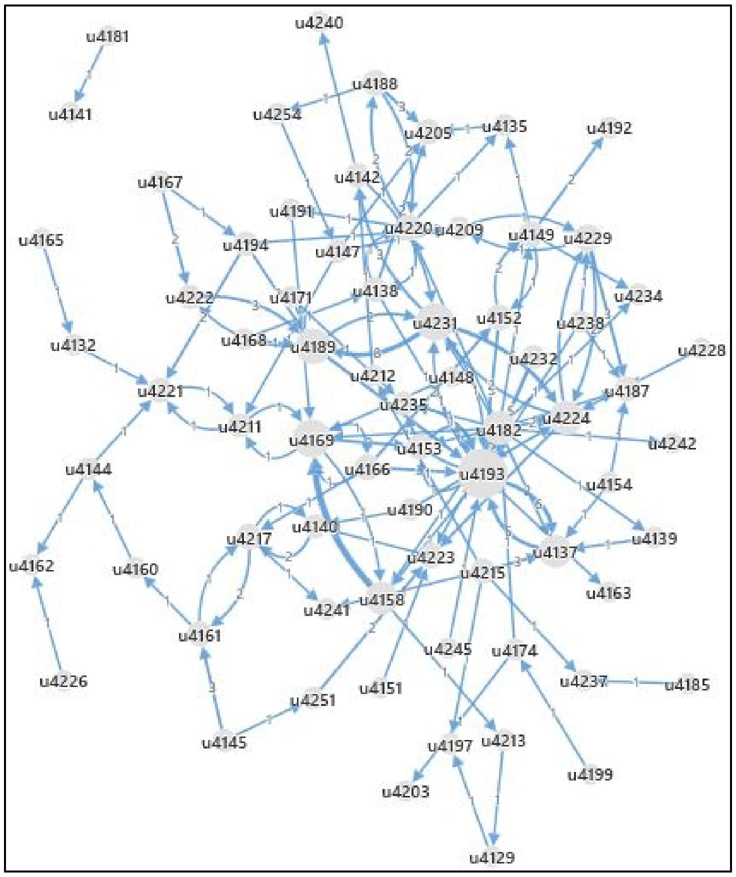


Figure 2.4: Small World Network Diffusion

3 Developer Manual

3.1 Development Environment

There are many existing tools for developing, compiling, and running code. However, we found that Atom[10], Eclipse[11], and MySQL Workbench[12] were very helpful in our data extraction process.

3.1.2 Installing Atom

Atom[10] is a text editor that will be used to develop the necessary Python[4] code. Please follow the instructions below to install the tool if it is not already installed:

1. Visit the Atom download page and click on the download link specified for the appropriate Operating System.
2. Open the file and the Operating System will set up and install Atom.
3. Open Atom to verify that it was successfully installed. There should be a Welcome Guide to help learn how to use the editing tool.

3.1.3 Installing Eclipse

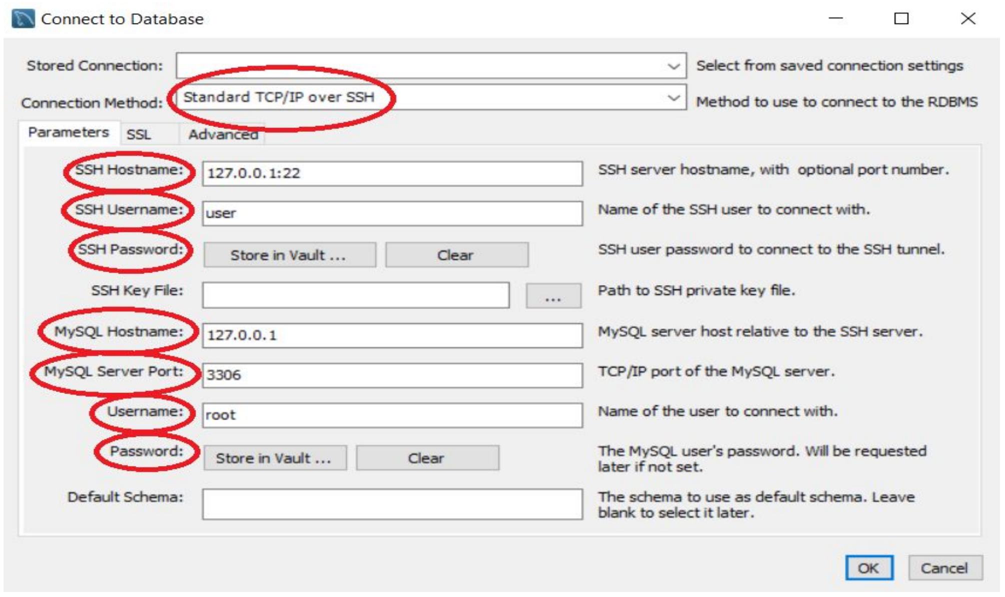
Eclipse[11] is an Integrated Development Environment (IDE) that will be used to develop the necessary Java[5] code. More specifically, Eclipse Neon, which is the latest version will be used. Please follow the instructions below to install the tool if it is not already installed:

1. Visit the Eclipse download page and click on the download link specified for the appropriate Operating System.
2. Click on the download link specified for the appropriate Operating System.
3. Verify the SHA512 checksum before opening the file (appropriate checksums can be found through a link next to the download link).
4. Open the file and the Operating System will set up and install Eclipse Neon.
5. Open Eclipse to verify that it was successfully installed. Once installed, you will have to ensure that JUnit is properly set up, which will be used for testing purposes. You will have to right click on an existing project and go to Properties > Java Build Path > Libraries, followed by clicking the button labeled “Add Library”. Select the option labeled “JUnit”.

3.1.4 Install & Connect to MySQL Workbench

MySQL Workbench[12] is a database connection tool that uses MySQL[13] and will be used to query the database for later analysis. Please follow the instructions below to install the tool if it is not already installed:

1. Visit the MySQL Workbench download page and click on the download link specified for the appropriate Operating System.
2. Click on the download link specified for the appropriate Operating System.
3. Open the file and the Operating System will set up and install MySQL Workbench.
4. Open Workbench to verify that it was successfully installed. Once installed, you will have to connect to the database. The Connection Method is Standard TCP/IP over SSH. You will also need the SSH Hostname, SSH Username, and SSH Password, as well as MySQL Hostname, MySQL Server Port, and MySQL Password in order to establish a successful connection. Figure 3.1 provides as a visual for the connection fields.



Stored Connection:		Select from saved connection settings
Connection Method:	Standard TCP/IP over SSH	Method to use to connect to the RDBMS
SSH Hostname:	127.0.0.1:22	SSH server hostname, with optional port number.
SSH Username:	user	Name of the SSH user to connect with.
SSH Password:	Store in Vault ... / Clear	SSH user password to connect to the SSH tunnel.
SSH Key File:		Path to SSH private key file.
MySQL Hostname:	127.0.0.1	MySQL server host relative to the SSH server.
MySQL Server Port:	3306	TCP/IP port of the MySQL server.
Username:	root	Name of the user to connect with.
Password:	Store in Vault ... / Clear	The MySQL user's password. Will be requested later if not set.
Default Schema:		The schema to use as default schema. Leave blank to select it later.

Figure 3.1: MySQL Workbench Connection Fields

3.2 Raw Data

3.2.1 Database

We had access to the Experiment 1 Replicate 2 (e1r2) database. The two tables we found most useful were items and contacts.

The contacts table contains friend relationships within Friendica[2]. It represents them by using a key (column 'uid') to represent each recovery buddy group, where the column ‘self’ is a boolean representing the owner of the recovery buddy group. The column ‘nick’ represents the user id, which is constant throughout Friendica.

The items table contains posts that users make. The significance of these posts is that they represent an activity done by a user in Friendica[2]. Every time a user clicks a link on the Friendica website, it is posted to their wall and a record is placed in the items table. The column ‘title’ represents what activity or link the user clicked on. There are 4 main types of activities: TES modules, assessments, video meetings, and news/success stories. The activity type can be found in the column ‘type’ or by parsing the content of the ‘title’. The column ‘created’ gives the date when the post was created, which corresponds to when the user started the activity in the post.

3.2.2 Surveys

We were provided 3 datafiles that were not stored in the database. These files are the responses to surveys taken by e1r2 participants and gives us their demographics and self-reported relationships with their buddies.

Usernames With Demographic Information.xls is an Excel[7] file that provides three different sheets containing different demographic data linked to the usernames of the e1r2 participants. It looks at homophily[3] within each network as well as over all users.

TimePoint2.csv and TimePoint3.csv contain two different survey responses from e1r2 participants at two different points in time. The survey asks users to report 6 buddys’ usernames, demographic information, and how close they feel to each buddy. TimePoint2.csv was compiled earlier in the experiment than TimePoint3.csv and they contain much of the same information (but with some column inconsistencies that should be addressed should you plan to compare the two files). We used only TimePoint3.csv in our experiment because the survey instructions were clearer for TimePoint3 which resulted in fewer instances of false data reporting.

3.3 Parsing Process

3.3.1 Queries

We wrote queries using MySQL[13] and ran them using MySQL Workbench[12] in order to use data from the e1r2 database in our code. The query results were exported as CSVs. Query 2 and Query 3 returned more than 1000 results, so the Limit command was added in order to return all results.

Query 1 is used to construct the items.csv used for diffusion (diffusion.jar) It returns all posts from the item table that are news/success stories or TES modules.

```
select `author-name` as id, title, file, created from friendica_e1r2.item where
(file = "<news>" or file = "<TES>") and wall=1 group by `author-name`, title, file
order by id;
```

Query 2 is used to construct the contacts.csv used for diffusion (diffusion.jar). It returns all contacts in the database who have a news or TES module type post in the items table.

```
select nick, self, uid from friendica_e1r2.contact where nick not like "admin"
and nick in (Select distinct `author-name` as id from friendica_e1r2.item where
wall = 1 and (file = "<news>" or file = "<TES>")) limit 0,2000;
```

Query 3 is used to construct the users.csv used for top-down cleaning (FriendCloseness.py). It simply returns uid, self, and nick for all contacts in the Small World and Lattice networks.

```
select uid, self, nick from friendica_e1r2.contact where nick not like "admin"
and nick between "u4001" and "u4256" and uid not like "2" LIMIT 0,2500;
```

Additional queries that were not used in our process, but may prove useful in future analysis, can be found in the appendix.

To generate a CSV from the query results, MySQL Workbench[12] has an "export as CSV" option. Queries 1-3 were used to generate the following CSVs, which can be accessed in the data inventory in our GitHub repository[9].

Table 3.1: The Format and Description of Columns in items.csv

Column Name	**What it contains**
id	The ID of the user in the database.
title	The title of the post.
file	The type of activity depicted by the post (TES, news, etc).
created	The date the post was created.

Table 3.2: The Format and Description of Columns in contacts.csv

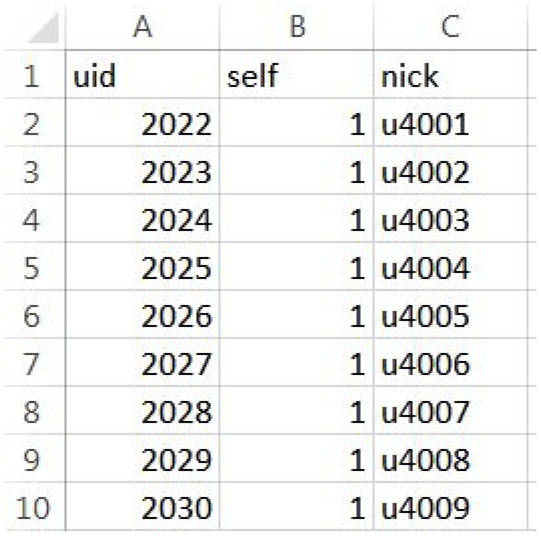
Column Name	**What it contains**
nick	The ID of the user in the database.
self	A boolean representation of the owner of the contact group.
uid	The key that maps a user to their group of contacts.

Table 3.3: The Format and Description of Columns in users.csv

Column Name	**What it contains**
uid	The key that maps a user to their group of contacts.
self	A boolean representation of the owner of the contact group.
nick	The ID of the user in the database.

3.3.2 Cleaning Code

In order to make sure that our top-down analysis was only looking at correct buddy pairings we needed to first clean the data given to us from the surveys and the database. A CSV file was created called users.csv that contained “uid”, “self”, and “nick” information from the database for every user. The second survey, TimePoint3.csv, was compared to this users.csv file to check if the buddies that each user entered in the survey were buddies they actually had in the database. In many cases users entered incorrect information on the survey, and this needed to be cleaned out.



	A	B	C
1	uid	self	nick
2	2022	1	u4001
3	2023	1	u4002
4	2024	1	u4003
5	2025	1	u4004
6	2026	1	u4005
7	2027	1	u4006
8	2028	1	u4007
9	2029	1	u4008
10	2030	1	u4009

Figure 3.2: Sample Data from users.csv

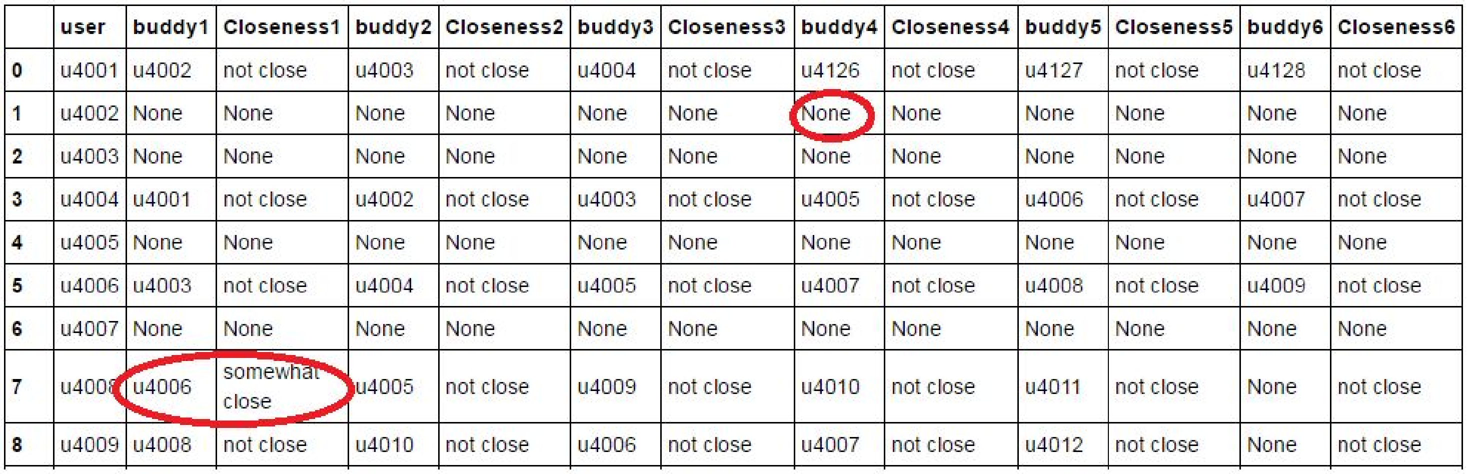
The two main cases of incorrect information were:
1. A user entered the name of a buddy into the survey that they were not actually assigned.
2. A user entered the name of a correct buddy into the survey, but entered it more than one time.

The second case needed to be looked at carefully, because we needed to make sure that a buddy pairing was not being recorded twice in our analysis.

The code was cleaned in a Python file titled: FriendCloseness.py. The files users.csv and TimePoint3.csv were imported and converted into pandas dataframes. All unneeded columns in the Time Point 3 dataframe, “tp3”, were removed. These were columns 0,1, 9-19, 21-31, 33-43, 45-55, 57-67, and 69-79. The dataframe was left with the columns: username, Buddy1, Buddy2, Buddy3, Buddy4, Buddy5, Buddy6, Close1, Close2, Close3, Close4, Close5, and Close6. These were the six buddies each user entered into the survey and the level of closeness to each.

The users.csv file was then sorted to order by “uid” and then descending by “self” value in order to pair each user by their many buddies. Due to the fact that a user could add more buddies for itself, the number of buddies paired to each user in the database could be greater than the six original buddies that were matched to them. An arraylist called “table” was used to store these new pairs of users and their buddies from the users.csv database information. This array list was produced to be in the same format as the Time Point 3 information, with one user on in the first column and each of their buddies in the subsequent columns. This table was then converted into a dataframe called “table2”.

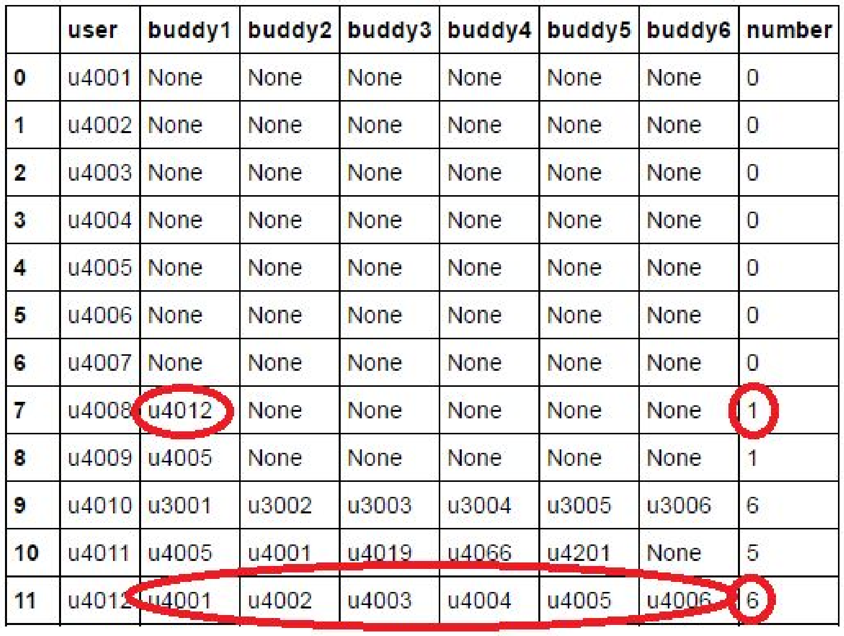
Two empty arrays were created to hold information about the closeness of the buddies, and the “sameness” of the buddies. Sameness here refers to the buddies that a user entered into Time Point 3 that were indeed buddies assigned to them in the database; the intersection of TimePoint3.csv and users.csv. Sameness took into account the correct buddies that were entered more than one time, deleting any duplicate correct buddies. The array of closeness values, taken from “tp3” were added to the sameness array. The array containing the sameness values, called “same” was then converted in a dataframe to be used as our output. The final table, consists of rows of individual users followed by each of the 0-6 correct buddies that they entered into Time Point 3, that were accurate based on the information in the database, followed by the level of closeness they felt toward each of those buddies. The dataframe was then converted into an output file called “sameClose.csv”.



	user	buddy1	Closeness1	buddy2	Closeness2	buddy3	Closeness3	buddy4	Closeness4	buddy5	Closeness5	buddy6	Closeness6
0	u4001	u4002	not close	u4003	not close	u4004	not close	u4126	not close	u4127	not close	u4128	not close
1	u4002	None	None	None	None	None	None	None	None	None	None	None	None
2	u4003	None	None	None	None	None	None	None	None	None	None	None	None
3	u4004	u4001	not close	u4002	not close	u4003	not close	u4005	not close	u4006	not close	u4007	not close
4	u4005	None	None	None	None	None	None	None	None	None	None	None	None
5	u4006	u4003	not close	u4004	not close	u4005	not close	u4007	not close	u4008	not close	u4009	not close
6	u4007	None	None	None	None	None	None	None	None	None	None	None	None
7	u4008	u4006	somewhat close	u4005	not close	u4009	not close	u4010	not close	u4011	not close	None	not close
8	u4009	u4008	not close	u4010	not close	u4006	not close	u4007	not close	u4012	not close	None	not close

Figure 3.3: Sample Data from “same” Dataframe

We then followed the same process to find the number of incorrect buddies entered into survey. These differences between TimePoint3.csv and users.cvs were vetted by the cleaning process in Figure 3.3, but with looking for the buddies not in common between the two files. This dataframe was created containing just a username, each of their 0-6 incorrect buddies, and then the number of incorrect buddies they had overall.



	user	buddy1	buddy2	buddy3	buddy4	buddy5	buddy6	number
0	u4001	None	None	None	None	None	None	0
1	u4002	None	None	None	None	None	None	0
2	u4003	None	None	None	None	None	None	0
3	u4004	None	None	None	None	None	None	0
4	u4005	None	None	None	None	None	None	0
5	u4006	None	None	None	None	None	None	0
6	u4007	None	None	None	None	None	None	0
7	u4008	u4012	None	None	None	None	None	1
8	u4009	u4005	None	None	None	None	None	1
9	u4010	u3001	u3002	u3003	u3004	u3005	u3006	6
10	u4011	u4005	u4001	u4019	u4066	u4201	None	5
11	u4012	u4001	u4002	u4003	u4004	u4005	u4006	6

Figure 3.4: Sample Code from “different” Dataframe

Cleaning the data in this way made it easier to discover which users displayed accurate information and which buddy pairings were true. Users that entered incorrect information could be ignored in determining what might make a good buddy selection process.

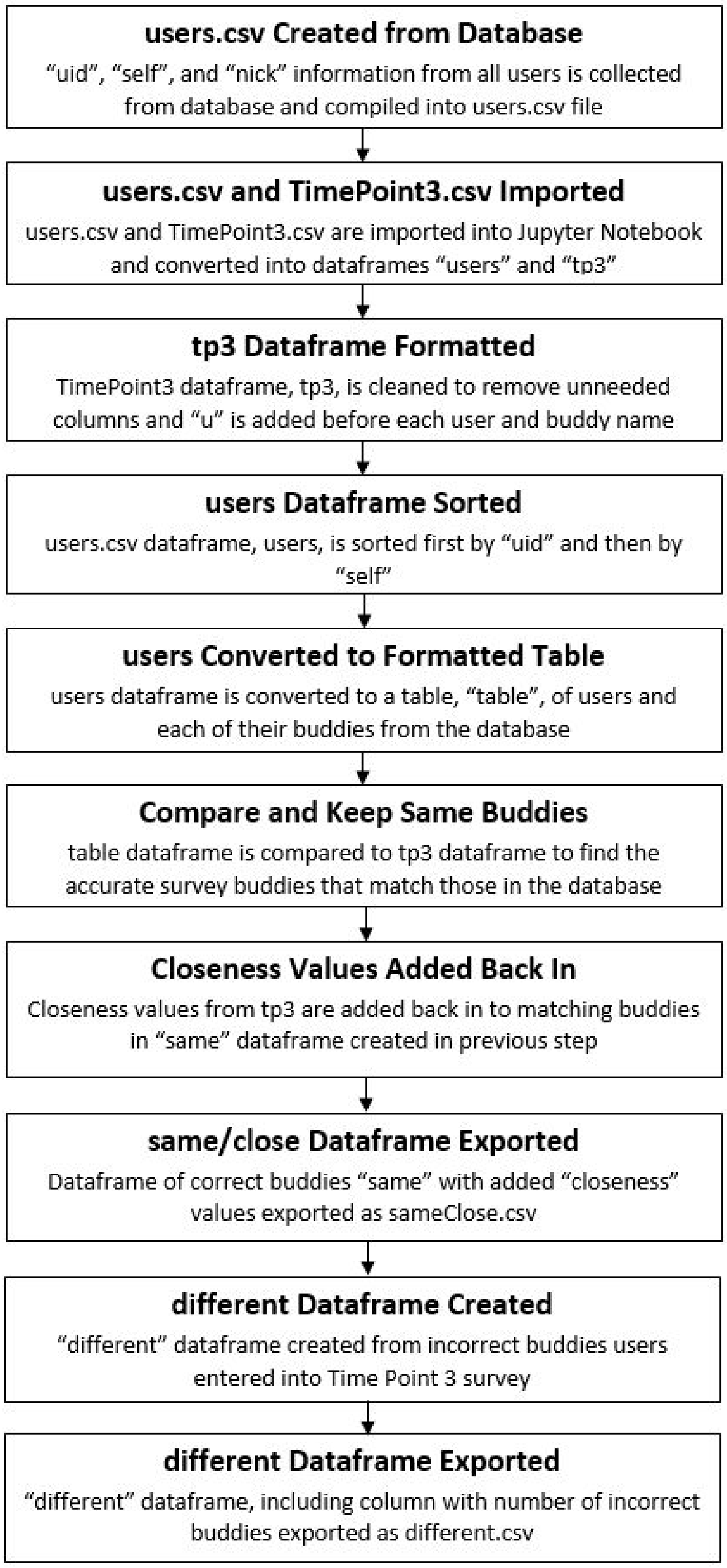

users.csv Created from Database
“uid”, “self”, and “nick” information from all users is collected from database and compiled into users.csv file
users.csv and TimePoint3.csv Imported
users.csv and TimePoint3.csv are imported into Jupyter Notebook and converted into dataframes “users” and “tp3”
tp3 Dataframe Formatted
TimePoint3 dataframe, tp3, is cleaned to remove unneeded columns and “u” is added before each user and buddy name
users Dataframe Sorted
users.csv dataframe, users, is sorted first by “uid” and then by “self”
users Converted to Formatted Table
users dataframe is converted to a table, “table”, of users and each of their buddies from the database
Compare and Keep Same Buddies
table dataframe is compared to tp3 dataframe to find the accurate survey buddies that match those in the database
Closeness Values Added Back In
Closeness values from tp3 are added back in to matching buddies in “same” dataframe created in previous step
same/close Dataframe Exported
Dataframe of correct buddies “same” with added “closeness” values exported as sameClose.csv
different Dataframe Created
“different” dataframe created from incorrect buddies users entered into Time Point 3 survey
different Dataframe Exported
“different” dataframe, including column with number of incorrect buddies exported as different.csv

Figure 3.5: Code Flow for FriendCloseness.py

3.3.3 Manual Cleaning

The file provided to us by our client, Usernames with Demographics.xls, required manual cleaning for use in Python[4]. This excel file contained 2 sheets that were of use to us, E1R2_Homophily_Lattice, and E1R2_Homophily_SmallW. Each of these sheets contained the demographics for each user in their respective network, in a format that was easy to parse within Python. Each of these sheets was converted to CSV file format and they were renamed to DemoLattice.csv and DemoSmall.csv. This was done because E1R2 was the only iteration of the study we were looking at for this project, so there was no need for an identifier. Upon closer inspection of these data sets, it was discovered that some columns in the CSV files contained the same name. This makes the files much more difficult to parse while using Python's built in CSV module, so some columns were renamed. Specifically, 8 columns had the same name because the referred to the same substance, but for either a primary or secondary addiction. The columns referring to a secondary addiction were prepended with an S, while the columns referring to a primary addiction were left the same. Table 3.4 is a full table of the renamed columns, note that the columns were renamed in the same way in both CSV files. Please note that all column names in Table 3.4 are written exactly as they were in the files we received. The misspelling of Opioids and Relievers is intentional.

Table 3.4: Renamed Columns in DemoLattice.csv and DemoSmall.csv

Original Column Name	Renamed Column Name
Alcohol	SAlcohol
Cannabis	SCannabis
Cocaine	SCocaine
Opiods	SOpiods
Presc. Pain Relivers	SPresc. Pain Relivers
Stimulants	SStimulants
Traq/Depres.	STraq/Depres.
Nicotine	SNicotine

3.4 Processing

3.4.1 Dataflow

Figure 3.6 depicts the entire dataflow for the project. All stages from the Social Interactome Data to our own code to the output files are noted.

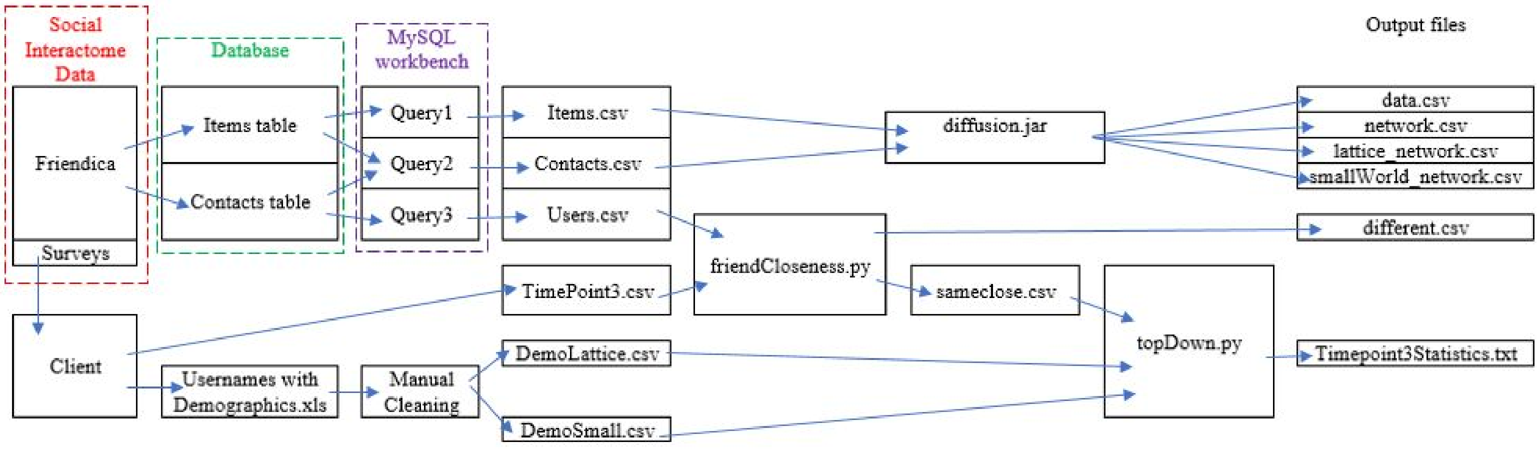


Figure 3.6: Dataflow for our Solution

3.4.2 Top-Down

Figure 3.7 represents the process in which our code flows for the top-down analysis. Two classes were created to facilitate this process: “friend”, and “user”. Every user in the study has an object for them, so there are 256 user objects. Each user object is initialized with up to 6 friends, depending on how much correct information they gave us on their survey from Timepoint 3. The number of correct friends is found in sameClose.csv. Even if a user has 0 correct friends listed, a user object is still created with 6 blank friends. This is done so that when other user objects compare to this user object for demographic similarities, there will be something to compare to. The friend class represents a user’s recovery buddy. Each friend contains the username of that friend, booleans for whether or not the friend shares demographic information with the user, and the level of closeness the user specified that they were to that friend.

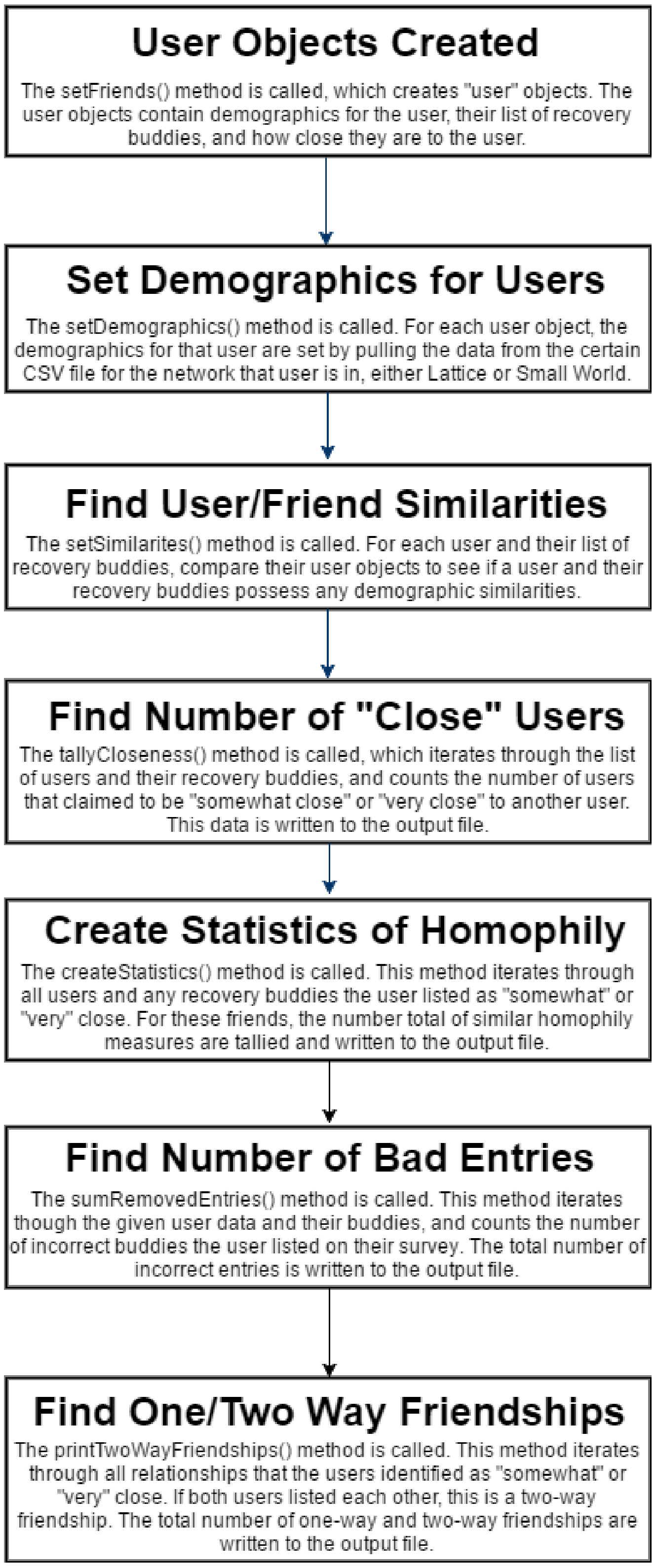

User Objects Created
The setFriends() method is called, which creates "user" objects. The user objects contain demographics for the user, their list of recovery buddies, and how close they are to the user.
Set Demographics for Users
The setDemographics() method is called. For each user object, the demographics for that user are set by pulling the data from the certain CSV file for the network that user is in, either Lattice or Small World.
Find User/Friend Similarities
The setSimilarites() method is called. For each user and their list of recovery buddies, compare their user objects to see if a user and their recovery buddies possess any demographic similarities.
Find Number of "Close" Users
The tallyCloseness() method is called, which iterates through the list of users and their recovery buddies, and counts the number of users that claimed to be "somewhat close" or "very close" to another user. This data is written to the output file.
Create Statistics of Homophily
The createStatistics() method is called. This method iterates through all users and any recovery buddies the user listed as "somewhat" or "very" close. For these friends, the number total of similar homophily measures are tallied and written to the output file.
Find Number of Bad Entries
The sumRemovedEntries() method is called. This method iterates though the given user data and their buddies, and counts the number of incorrect buddies the user listed on their survey. The total number of incorrect entries is written to the output file.
Find One/Two Way Friendships
The printTwoWayFriendships() method is called. This method iterates through all relationships that the users identified as "somewhat" or "very" close. If both users listed each other, this is a two-way friendship. The total number of one-way and two-way friendships are written to the output file.

Figure 3.7: Top-Down Code Process

The process begins with setFriends(). User objects are created by iterating through the data rows of sameClose.csv. This CSV file contains the each participant’s username in Timepoint3 and their responses to how close they were to their respective recovery buddies. After the user objects are created, setDemographics() parses through DemoLattice.csv and DemoSmall.csv to assign users with their demographic information. The CSV files contain the same data columns, which include:

- Age
- Gender
- Primary Substance Addictions
- Secondary Substance Addictions
- Education
- Income
- Whether they are a current or ex smoker

After the user demographics are set, similarities are calculated in both the Lattice Network and the Small World Network using setSimilarities(). This method leverages the helper method compareTwoUsers(), which checks each demographic similarity between a user and their recovery buddy. After the similarities are set, tallyCloseness() outputs the number of “not close”, “somewhat close”, and “very close” relationships to a file named Timepoint3Statistics.txt. Next, the method createStatistics() analyzes the list of relationships that were reported “somewhat close” and “very close”. It checks for similar homophily measures that were established in setSimilarities() and outputs the number of total similarities to Timepoint3Statistics.txt. Consequently, sumRemovedEntries() is called to count the total number of entries that contained flawed data. Specifically, it looks for the number of incorrect buddies listed in the surveys. The number of entries removed is then written into Timepoint3Statistics.txt. Lastly, the printTwoWayFriendships() method counts up the total relationships that were reported “somewhat close” and “very close”. If only one user reported that they were close to their recovery buddy, then it’s categorized as a one-way relationship. However, if both people reported the same response, then it would categorized as a two-way relationship.

3.4.3 Diffusion

The diffusion code is implemented in Java[5] and contains 5 classes which are described in Table 3.5.

Table 3.5: A Description of the Diffusion Class Structure

Class	Description	Variables
RunDiffusion	This is the main class that runs the diffusion algorithm.	ArrayList<User> users; ArrayList<Node> nodes; ArrayList<Node2> nodes2;
User	This class creates a User object that stores a Friendica user's id, the number to map their contact relationships to them in contacts.csv, an ArrayList of users who they are contacts with, and an ArrayList of post objects for each post they have made.	String ID; String friendNumber; ArrayList<User> friendList; ArrayList<Post> postList;
Post	This class creates a post object that stores post information.	String ID; String title; Date created;
Node	This class structures all diffusion information in an easily processed way. For each instance of diffusion, it stores the Id of the user and the id of the friend who exhibit diffusion, the title of the post they exhibit diffusion on, and the date the user and the friend each posted.	String ID; String friendId; String title; Date created; Date friendCreated;
Node2	This class creates a node2 object which is optimized for visualization. It has a source node (the userId, ID) a target node (the friendId) and the weight of the edge (weight).	String ID; String friendId; int weight;

The RunDiffusion class contains the methods displayed in Figure 3.8.

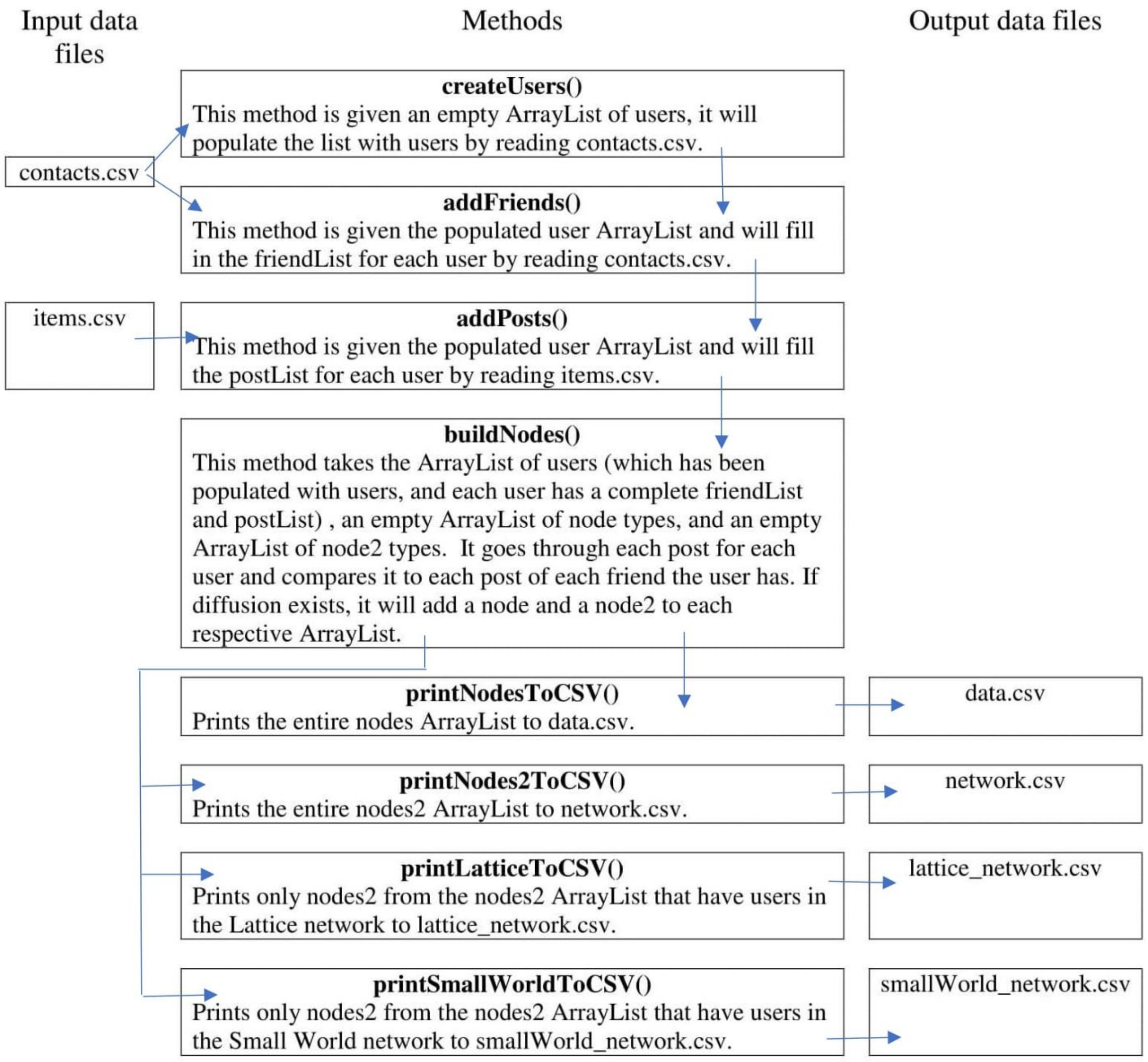

Input data files
Methods
Output data files
contacts.csv
items.csv
createUsers()
This method is given an empty ArrayList of users, it will populate the list with users by reading contacts.csv.
addFriends()
This method is given the populated user ArrayList and will fill in the friendList for each user by reading contacts.csv.
addPosts()
This method is given the populated user ArrayList and will fill the postList for each user by reading items.csv.
buildNodes()
This method takes the ArrayList of users (which has been populated with users, and each user has a complete friendList and postList) , an empty ArrayList of node types, and an empty ArrayList of node2 types. It goes through each post for each user and compares it to each post of each friend the user has. If diffusion exists, it will add a node and a node2 to each respective ArrayList.
printNodesToCSV()
Prints the entire nodes ArrayList to data.csv.
printNodes2ToCSV()
Prints the entire nodes2 ArrayList to network.csv.
printLatticeToCSV()
Prints only nodes2 from the nodes2 ArrayList that have users in the Lattice network to lattice_network.csv.
printSmallWorldToCSV()
Prints only nodes2 from the nodes2 ArrayList that have users in the Small World network to smallWorld_network.csv.
data.csv
network.csv
lattice_network.csv
smallWorld_network.csv

Figure 3.8: The Method Structure of Code with Regards to Input and Output Data Files

A fully commented copy of all Java files used for diffusion can be found on our GitHub[9] in the folder “SIFriendRecommender/diffusion/src/diffusion/”.

3.5 Output Data

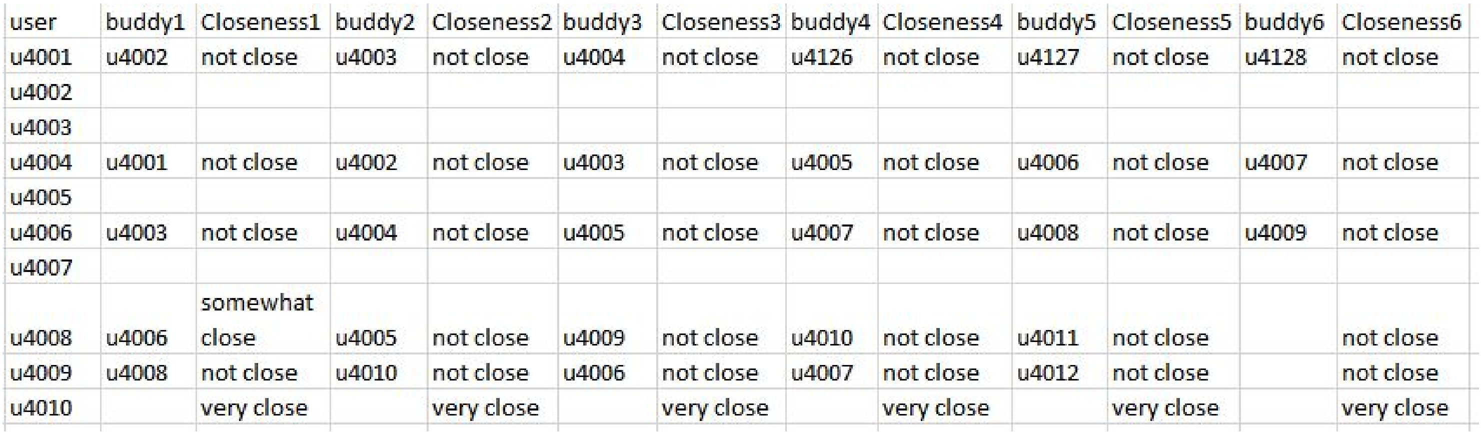
3.5.1 Accessing the Data Inventory

The Data Inventory consists of a detailed dictionary for every field value referenced in each CSV output file. To access the Data Inventory follow similar steps to “Running the Code” in sections 2.2.1 and 2.3.1: Using the Terminal (for Mac or Linux) or Git Bash[8] (for Windows)

1. Enter the command “git clone https://github.com/tttran/SIFriendRecommender.git” This will create a local copy of the repository.
2. Enter the command “cd SIFriendRecommender”. The user has now entered the project folder.
3. Enter the command “start DataDictionary.xlsx” . The user has now opened the Excel[7] file containing the Data Inventory.

3.5.2 Friend Closeness

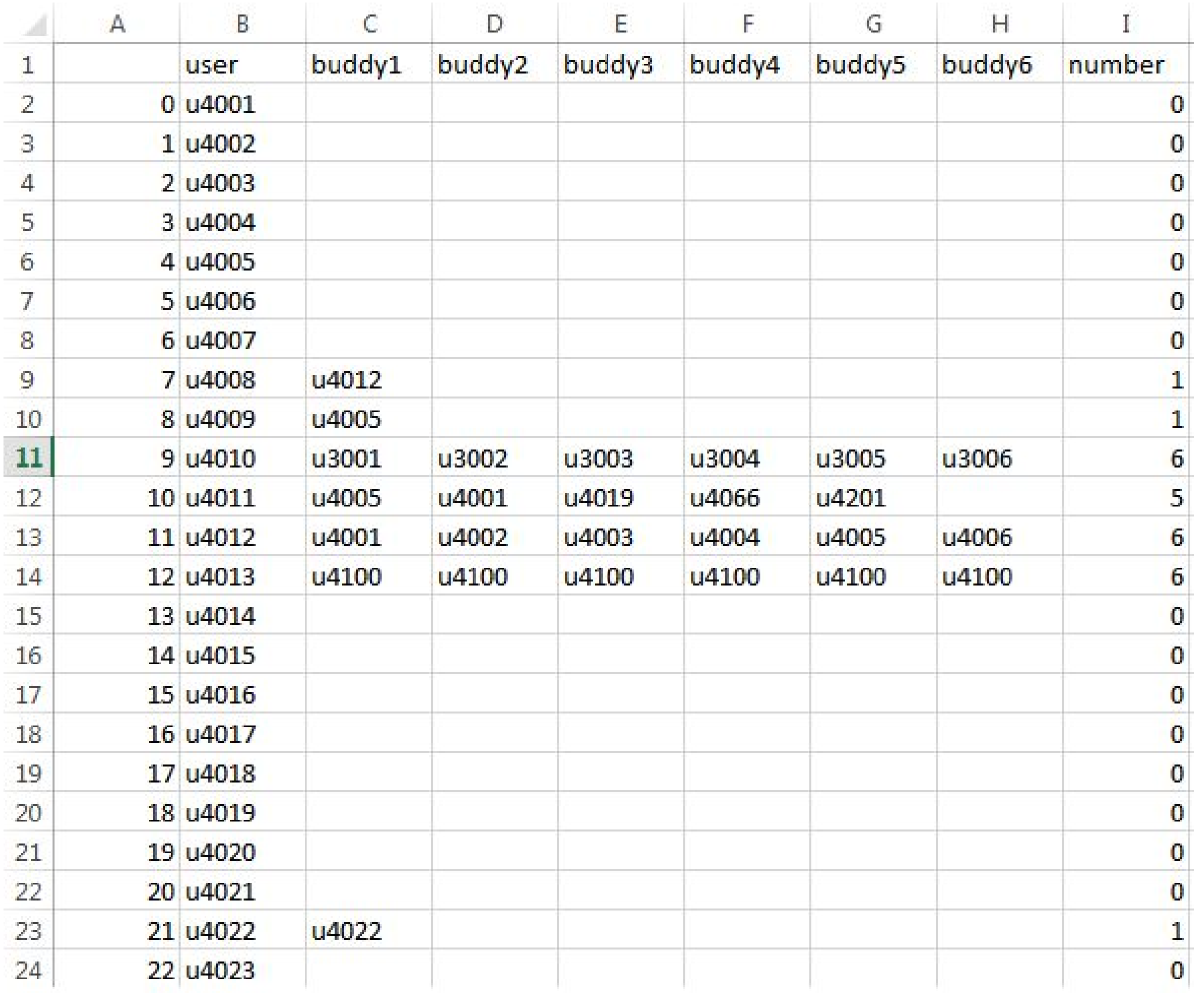
There are two outputs to the Friend Closeness code. The first is sameClose.csv. The sameClose data shows the recovery buddies a user entered into their Time Point 3 survey that were accurate buddies they were assigned in the database. It depicts the “closeness” value of each of those correct buddies. This output is used in the topDown.py code to assess the Time Point 3 Statistics.



user	buddy1	Closeness1	buddy2	Closeness2	buddy3	Closeness3	buddy4	Closeness4	buddy5	Closeness5	buddy6	Closeness6
u4001	u4002	not close	u4003	not close	u4004	not close	u4126	not close	u4127	not close	u4128	not close
u4002												
u4003												
u4004	u4001	not close	u4002	not close	u4003	not close	u4005	not close	u4006	not close	u4007	not close
u4005												
u4006	u4003	not close	u4004	not close	u4005	not close	u4007	not close	u4008	not close	u4009	not close
u4007												
u4008	u4006	somewhat close	u4005	not close	u4009	not close	u4010	not close	u4011	not close		not close
u4009	u4008	not close	u4010	not close	u4006	not close	u4007	not close	u4012	not close		not close
u4010		very close		very close		very close		very close		very close		very close

Figure 3.9: Sample Data From sameClose.csv

The second output data is different.csv. This data represents the incorrect recovery buddies that a user listed in their Time Point 3 survey. These buddies were inaccurate because they were either not actual buddies established in the database or were duplicates of correct buddies a user already listed. different.csv conveys the number of inaccurate buddies each user entered in order to easily determine the level of credibility a particular user had, ie: the larger the number, the less credible the user.

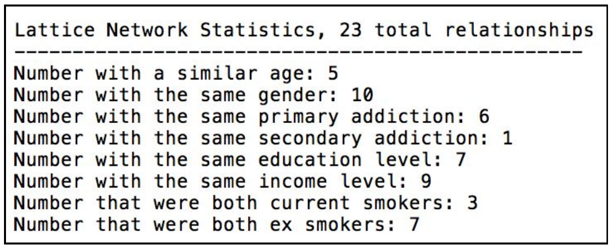


	A	B	C	D	E	F	G	H	I
1		user	buddy1	buddy2	buddy3	buddy4	buddy5	buddy6	number
2	0	u4001							0
3	1	u4002							0
4	2	u4003							0
5	3	u4004							0
6	4	u4005							0
7	5	u4006							0
8	6	u4007							0
9	7	u4008	u4012						1
10	8	u4009	u4005						1
11	9	u4010	u3001	u3002	u3003	u3004	u3005	u3006	6
12	10	u4011	u4005	u4001	u4019	u4066	u4201		5
13	11	u4012	u4001	u4002	u4003	u4004	u4005	u4006	6
14	12	u4013	u4100	u4100	u4100	u4100	u4100	u4100	6
15	13	u4014							0
16	14	u4015							0
17	15	u4016							0
18	16	u4017							0
19	17	u4018							0
20	18	u4019							0
21	19	u4020							0
22	20	u4021							0
23	21	u4022	u4022						1
24	22	u4023							0

Figure 3.10: Sample Data From different.csv

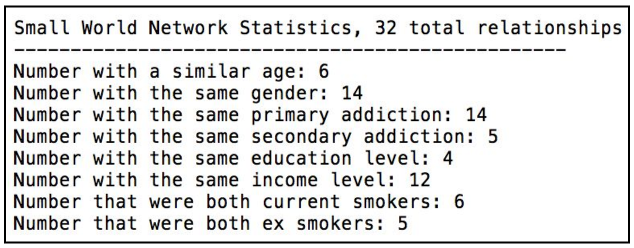
3.5.3 Top-Down

Figures 3.11 through 3.15 shows the results after running our Top-Down program.



```
Lattice Network Statistics, 23 total relationships
-------------------------------------------------
Number with a similar age: 5
Number with the same gender: 10
Number with the same primary addiction: 6
Number with the same secondary addiction: 1
Number with the same education level: 7
Number with the same income level: 9
Number that were both current smokers: 3
Number that were both ex smokers: 7
```

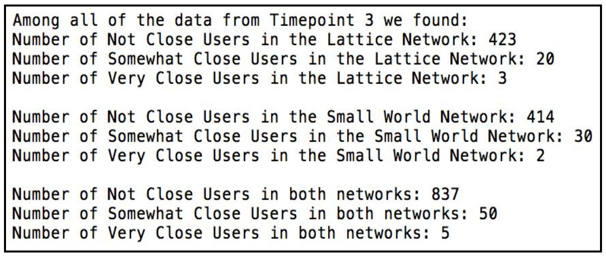
Figure 3.11: Lattice Network Top-Down Results



```
Small World Network Statistics, 32 total relationships
-----------------------------------------------
Number with a similar age: 6
Number with the same gender: 14
Number with the same primary addiction: 14
Number with the same secondary addiction: 5
Number with the same education level: 4
Number with the same income level: 12
Number that were both current smokers: 6
Number that were both ex smokers: 5
```

Figure 3.12: Small Word Network Top-Down Results

Figures 3.11 and 3.12 represent the number of similarities in demographics between each user and their recovery buddies in the Lattice Network and Small World Network.

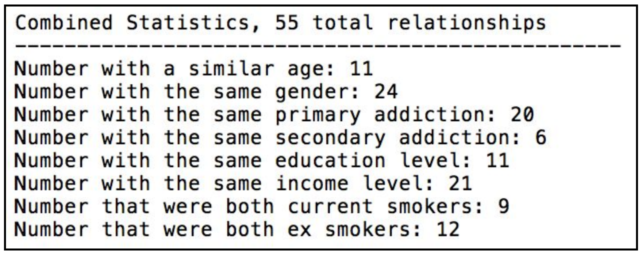


```
Among all of the data from Timepoint 3 we found:
Number of Not Close Users in the Lattice Network: 423
Number of Somewhat Close Users in the Lattice Network: 20
Number of Very Close Users in the Lattice Network: 3

Number of Not Close Users in the Small World Network: 414
Number of Somewhat Close Users in the Small World Network: 30
Number of Very Close Users in the Small World Network: 2

Number of Not Close Users in both networks: 837
Number of Somewhat Close Users in both networks: 50
Number of Very Close Users in both networks: 5
```

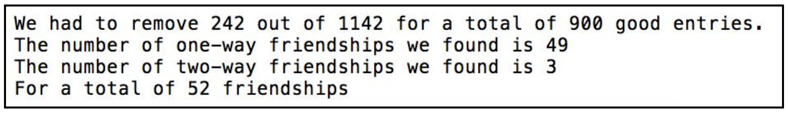
Figure 3.13: Closeness in Relationships for Each Network



```
Combined Statistics, 55 total relationships
--------------------------------------------------
Number with a similar age: 11
Number with the same gender: 24
Number with the same primary addiction: 20
Number with the same secondary addiction: 6
Number with the same education level: 11
Number with the same income level: 21
Number that were both current smokers: 9
Number that were both ex smokers: 12
```

Figure 3.14: Combined Top-Down Results

Figure 3.13 represents how many users reported they were “not close”, “somewhat close”, and “very close” to their recovery buddies. The data provides responses from both the Lattice Network and Small World Network. It is clear that many users did not feel they were close to their recovery buddies. Figure 3.14 shows the combined number of similarities in demographics for both the Lattice Network and Small World Network.



```
We had to remove 242 out of 1142 for a total of 900 good entries.
The number of one-way friendships we found is 49
The number of two-way friendships we found is 3
For a total of 52 friendships
```

Figure 3.15: One-Way/Two-Way Relationships Found

Figure 3.15 shows the of entries in the data that were cleaned. In addition, it shares the number of one-way, two-way, and total relationships found in Timepoint3.

3.5.4 Diffusion

For a detailed description of the output files and their formats, please see section 2.3.2 “Understanding the outputs” (pg 10).

<u>3.6 Testing</u>

3.6.1 Top-Down Testing

3.6.1.1 Top-Down unit testing

The unit tests for top-down will guarantee that the Python[4] code functions properly on an individual component basis.This includes testing that constructors can be created and values can be set within those constructors. The testing is split into three test classes: TestFriend.py, TestUser.py, and TestTopDown.py, which are all located on the GitHub repository[9].

Table 3.6: TestFriend.py

Test Method Name	**Description**
testConstructor()	This test ensures that Friend is instantiated.

Table 3.7: TestUser.py

Test Method Name	**Description**
testConstructor()	This test ensures that User is instantiated
testSetCloseness()	This test ensures that User can set new values for close for each Friend in the Friend list.
testPrintFriends()	This test ensures that User can print each Friend in the Friend list to console.
testPrintCloseness()	This test ensures that User can print all close values of each Friend in the Friend list to console.
testPrintDemographics()	This test ensures that User can print all demographics values to console.
testPrintSimilarities()	This test ensures that User can print all similarity values to console.

Table 3.8: TestTopDown.py

Test Method Name	**Description**
testSetFriends()	This test ensures that User objects are created with their friends set and added to the two User lists.
testSetDemographics()	This test ensures that each User in the two User lists add the appropriate demographics given an input CSV file.
testCompareTwoUsers()	This test ensures that two User objects can be successfully compared to determine shared demographics.
testOrderList()	This test ensures that a User list can be ordered correctly by username, alphabetically.

testClearFile()	This test ensures that a file can be closed successfully on the method call.

3.6.1.2 Top-Down Integration Testing

The integration testing for the top-down method will ensure that the Friend class, User class, and all of their various functions interact properly so that the program produces the correct output. These tests were written and can be found in TestTopDown.py, located in the GitHub repository[9].

Table 3.9: TestTopDown.py

Test Method Name	Description
testTallyCloseness()	This test ensures that closeness values can be tallied up and the results can be added to a CSV.
testCreateStatistics()	This test ensures that statistics regarding overall demographics can be gathered and added to a CSV.
testSumRemovedEntries()	This test ensures that entries can be removed from the two User lists and the result count can be added to a CSV.
testPrintTwoWayFriendships( )	This test ensures that the number of one-way and two-way friendships can be determined and added to a CSV.

3.6.2 Diffusion Testing

3.6.2.1 Diffusion Unit Testing

The unit tests for diffusion will guarantee that individual components of the code are functioning properly. This includes testing that the appropriate data structures are being created and filled with necessary information. The testing is split into four test classes: NodesTest.java, PostTest.java, UserTest.java, and RunDiffusion.java, which are all located in the GitHub repository[9].

Tale 3.10: NodesTest.java

Test Method Name	Description
testConstructor()	This test ensures that Node and Node2 are instantiated.
testFieldsNode()	This test ensures that the fields are the correct value for Node.
testFieldsNode2()	This test ensures that the fields are the correct value for Node2.

Table 3.11: PostTest.java

Test Method Name	Description
testConstructor()	This test ensures that Post is instantiated.
testFieldsPost()	This test ensures that the fields are the correct value for Post.

Table 3.12: UserTest.java

Test Method Name	Description
testConstructor()	This test ensures that User is instantiated.
testFieldsUser()	This test ensures that the fields are the correct value for User.
testAddPost()	This test ensures that User can add a post.
testAddFriend()	This test ensures that User can add a friend.

Table 3.13: RunDiffusionTest.java

Test Method Name	Description
testFindUser()	This test ensures that the list of User objects can find a User by id.
testFindUserFriendNum()	This test ensures that the list of User objects can find a User by friend number.
testCreateUsers()	This test ensures that the list of User objects can be filled given an input CSV.
testAddFriends()	This test ensures that the friend list of each User in the list of User objects can be filled given an input CSV.

testAddPosts()	This test ensures that the post list of each User in the list of User objects can be filled given an input CSV.
testBuildNodes()	This test ensures that a list of Node objects can be filled given a list of User objects.

3.6.2.2 Diffusion Integration Testing

The integration testing for diffusion will guarantee that the classes and functions interact properly and that the overall program produces correct output given certain input. These tests were written in RunDiffusionTest.java, located in the GitHub repository[9], where all RunDiffusion methods are combined to produce several new CSV files with information about possible diffusion in the input data.

Table 3.14: RunDiffusionTest.java

Test Method Name	**Description**
testNodesToCSV()	This test ensures that a CSV can be created and filled in with information about Node objects using the functions in Table 3.13.
testNodes2ToCSV()	This test ensures that a CSV can be created and filled in with information about Node2 objects using the functions in Table 3.13.
testPrintLatticeToCSV()	This test ensures that a CSV can be created and filled in with information about each User with id 4001-4128 using the functions in Table 3.13.
testPrintSmallWorldToCSV()	This test ensures that a CSV can be created and filled in with information about each User with id 4129-4256 using the functions in Table 3.13.

4 Lessons Learned

4.1 Timeline

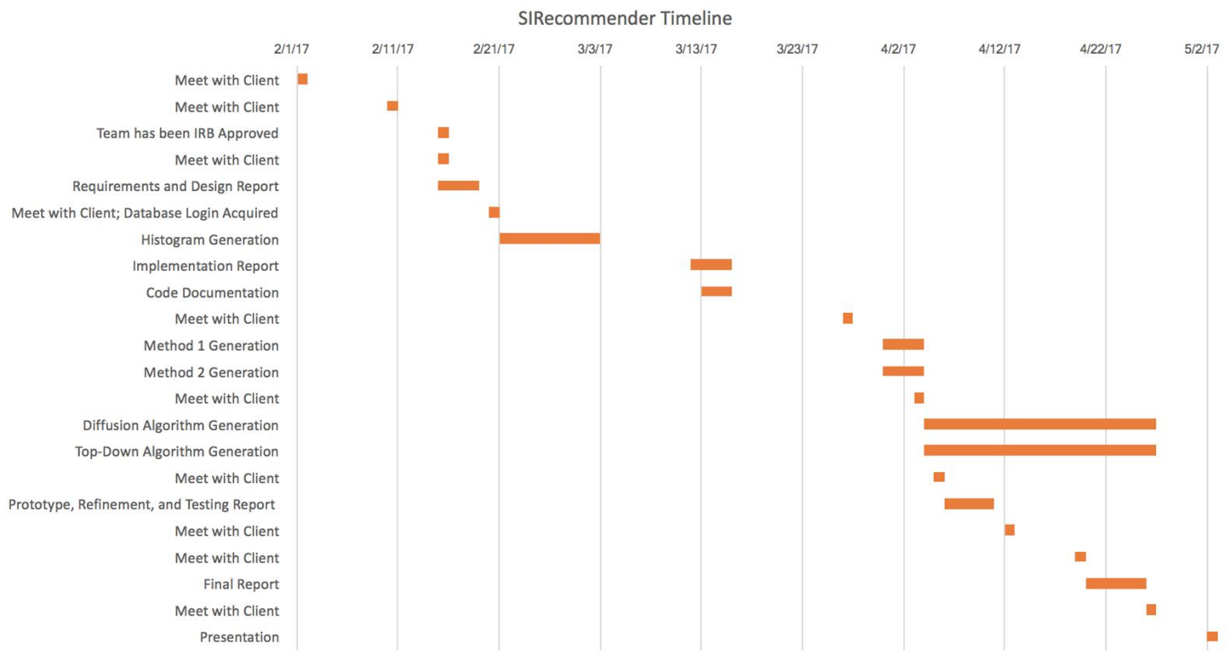


Figure 4.1: Gantt Chart for the SIRecommender Team

Our timeline was met with many changes in terms of requirements and delays at the beginning of our project. In Figure 3.16, it is apparent that our team was not able to access the database until February 20. This meant that we were unable to analyze the data necessary to start our project until a month into the semester. This complication is further explained in section 4.2.

As we gained access and became familiar with the database, we were advised by our client and Dr. Fox to create histograms based on user participation of TES modules, assessments, video meetings, and news/success stories. These visualizations were due by Spring break, which is why we were only able to focus on the Implementation Report during Spring break.

As our client was away for three weeks after our break ended, we turned to Dr. Fox for help regarding what to do next. From there we focused on our Method 1 and Method 2.

When our client returned, we met with him and were guided towards extracting and analyzing homophily-based measures based on our top-down approach and diffusion approach. These two methods are a main part of our deliverables.

4.2 Problems

Over the course of the SIRecommender project, our team faced many complications. These problems ranged from small to large, some even changing the course of our semester goal as a whole. As we worked through these issues, we got more of a sense for our requirements, rationalized our team's roles, and distributed tasks accordingly.

The major dilemma that changed our deliverables and semester goals was the change of requirements over the course of the semester. Our team was originally going to refine the previous team's SIRecommender algorithm[15] as one of our main deliverables. After receiving the login data to the database, we were unsure of what to do with that data; our end goal started becoming less clear. We were uncertain as to our goal being to revise the old team's code or to build a new friend recommender for the 12 week mark. The week before spring break, we were directed by our client to focus more on the visualization of data (i.e., histograms). Our client was away for three weeks after that meeting so we were confused as to what to do after the histograms and our Implementation report were completed. After some advising from Dr. Fox, we were redirected to analyzing methods of measuring homophily[3], which ended up being a major part of our deliverable. From there, we focused on our top-down and diffusion methods of evaluating homophily-based measures.

One of our first problems was the amount of time it took to gain access to the database. As the original five of us completed the tutorials necessary for IRB Approval[16], we added Wilson to our team. This meant that it took us an extra week to obtain approval to analyze the Social Interactome data. Furthermore, we had not received the login data for the MySQL[13] database until February 20th. This meant that we were not able to take a look at the data we needed for algorithm generation until a month into our semester project. Furthermore, we had difficulties with receiving the demographics survey until further into the semester.

Another issue with the database was that we did not receive permission to write functions into the MySQL Workbench[12] query line for easier access to data that we needed. Instead, the necessary data had to be exported to CSV. This was more of a roadblock or an extra step than a major problem, but some of the diffusion code would be written much more quickly if we had this permission.

As we began to work with the database to extract data for our histograms and analysis, we realized that some of the data measured was not helpful for us. For example, we were provided with how many video meetings a participant attended, but we were not able to identify the video meetings distinctly or for how long the participant attended. These data points would be extremely helpful because we would be able to identify if a participant attended multiple video meetings or simply reviewed the same one multiple times. For this reason, we could not use video meetings in our research as the data was incomplete.

Since the participants were incentivised, they may have been motivated to do only the bare minimum. This meant that we were provided with some data that we could not use because the participants entered false information. In time point 2 and time point 3, some data points from participants were either faked, wrong, or missing. This means that some of the names in columns were missing, participants clicked on different assessments or TES modules without spending sufficient time on them, or participants listed other recovery buddies as friends just to say they had other friends on the site. This caused for inconsistent and sometimes faked data. Wilson took the time to go through and manually clean some of the data, since some entries were missing or wrong. After we cleaned the data, we found that there were only three two-way friendships. There were only three pairs of people who said that they were close friends with one another via surveys. The lack of true, clean data was a problem that hindered our capability of truly analyzing the homophily-based measures between recovery buddies.

The size of our team was another drawback when it came to communication, scheduling meetings, and delegation of work. It was difficult for us to all meet at once, and when we did, it was difficult to keep everyone on the same page.

Though there were many of us, there were few of us with deep knowledge of SQL[13]. When our task was to create histograms, we realized that only Sarah was able to provide us with precise queries.

The complexity of the data set was another large drawback. This is mainly because it took so much time to understand and tackle what friendship means in a research sense. Our understanding of measuring friendship through demographics, assessments, and type of addiction was sharpened over time, but it took us much time to wrap our heads around that concept.

4.3 Solutions

Though our team experienced many complications throughout the semester, we worked hard to provide the Social Interactome team with visualizations of data from our diffusion and top-down methods.

Despite having a large team, we learned to delegate work based on our team members' capabilities. Wilson took charge of the top-down method because of his experience with Python, and Sarah, as lead developer, was in control of the diffusion method. The other members of the team worked under Wilson and Sarah and contributed more to revising the previous report and writing the new report. This made for a more organized structure of roles, and the clarity of these roles made communication much clearer.

Since our data on video meetings was unusable, our team thought it best to focus on other forms of data. Not being able to use video meetings as a research point for our histograms, graphs, and analysis was a dilemma we overcame by focusing on the other data we did have. This includes TES modules, assessments, demographics, and type of addiction.

False data was one of the most time-consuming complications that our team faced, mainly because it took communication, meetings, and collaboration to clean up the false data involved. Regardless, though, our team worked together to make sure the data was removed. This meant that we had less data to work with, but also that our analysis worked with clean data and provided true outputs.

As the Figure 4.1 shows, we met with our client, Prashant Chandrasekar, very often towards the end of the semester. During the days that we held meetings, we sometimes met for multiple hours. We saw Prashant as a reference for us to check information and ask questions. He was very helpful in guiding us when we were not sure of how to move forward with our project. Prashant helped us form more of a collaborative and organized environment when our team was disorganized and lacked communication.

Our main solution was our collaborative effort to provide deliverables that the Social Interactome team can use in the future to measure homophily[3]. When the team was unsure of where our project was headed, we didn't hesitate to email our client and Dr. Fox for more guidance. Teamwork and diligent effort was the main solution to the complications that our team faced throughout the semester.

4.4 Future work

This work is related to the Social Interactome (SI) project being conducted by the Addiction Recovery Research Center at Virginia Tech Carilion Research Institute (VTCRI) and the Computer Science department at Virginia Tech[17]. We expect our deliverables to be used within the Social Interactome team to draw conclusions about homophily. Specifically, we see our visualizations of data to be referenced when talking about demographics and addictions that friends have in common. We expect our top-down and diffusion code to be used, updated, and revised for future studies.

We expect our visualizations to be improved based on our analysis. Since we were short on time when we created these visualizations, we think that the Social Interactome team or a future undergraduate team can extend them to encapsulate more data.

Some of our code relies on a bubble sort, so we think that a future team can optimize our code to be more lightweight and easier to run. Another way to make our code easier to use is to either put the code into Jupyter Notebook[14] or to write a script to make the code run more seamlessly altogether.

In order to extract more data from our code, a future team can determine weightings based on our top-down analysis. These weightings will include the number of relationships between two friends on Friendica[2] based on the type of similarity they share.

5 Acknowledgements

We would like to acknowledge our client Prashant Chandrasekar (peecee@vt.edu) for his patience and support with our work. We would also like to acknowledge Dr. Edward Fox[17] for helping us develop our knowledge of the project, for giving helpful feedback on our project proposal and for taking time out of his day to answer our innumerable questions.

6 References

1. Andrew Baehre, Colin Gray, and Trevor Kinaman. 2016. SIRecommender Final Report and Presentation https://vtechworks.lib.vt.edu/handle/10919/70956. Accessed: 19 February 2017.

2. Friendica. Stable Release, 2017. Mike Macgirvin. Distributed social network. http://friendi.ca/ Accessed: 12 March 2017.

3. Miller McPherson, Lynn Smith-Lovin, and James M Cook. BIRDS OF A FEATHER: Homophily in Social Networks. http://aris.ss.uci.edu/~lin/52.pdf. Accessed: 09 April 2017.

4. Python. 2001-2017. Python Software Foundation. https://www.python.org/. Accessed: 11 April 2017.

5. Java. 2017. Oracle Corporation. https://www.java.com/en/. Accessed: 11 April 2017.

6. Buergin, Patrick. GiGraph. 2017. https://gigraph.io/. Accessed: 28 April 2017.

7. Microsoft Excel. 2017. Microsoft. https://office.microsoft.com/excel/. Accessed: 28 April 2017.

8. Git. 2017. Github, Inc. https://git-scm.com/. Accessed: 28 April 2017.

9. Timothy Tran, Wilson Rhodes, Sarah Devlin, Bridget Larcher, Mary Kwiatkowski, and Matthew Blumen. 2017. SIFriendRecommender Code https://github.com/tttran/SIFriendRecommender. Accessed: 11 April 2017.

10. Atom. 2017. GitHub, Inc. https://atom.io/. Accessed: 28 April, 2017.

11. Eclipse. 2017. The Eclipse Foundation. https://eclipse.org/. Accessed: 11 April 2017.

12. MySQL Workbench. 2017. Oracle Corporation. https://www.mysql.com/products/workbench/. Accessed: 28 April, 2017.

13. MySQL. 2017. Oracle Corporation. https://www.mysql.com/. Accessed: 11 April 2017.

14. Jupyter Notebook. 2017. Project Jupyter - Last updated Sunday, March 05, 2017. Open Source Programming Application. http://jupyter.org/. Accessed: 17 March 2017.

15. Andrew Baehre, Colin Gray, and Trevor Kinaman. 2016. SIRecommender Code https://github.com/baehre/SIRecommender. Accessed: 19 February 2017.

16. "VT Human Subject Protections Tutorial." *VT Human Subject Protections Tutorial | Institutional Review Board | Virginia Tech*. N.p., n.d. Web. 09 Apr. 2017.

17. Edward A. Fox. 2017. The Social Interactome of Recovery: Social Media as Therapy Development. https://www.cs.vt.edu/node/7439. Accessed: 19 February 2017.

7 Appendix

Requirements, Design, Implementation, Prototyping, Refinement and Testing Report

1 Executive Summary

1.1 Background

This project is a continuation of efforts conducted by the CS4624 team of Spring 2016. The previous team built a procedure to find the top k friends based on information submitted via surveys [1]. It will extend that implementation. This year's team of students will work with data coming in from a live social network, such as interactions with web resources, like TES modules, stories, video meetings, etc. to build an algorithm that incorporates this information as well.

One of this team's focuses considers different aspects of weighting different feature types. This mainly includes tuning parameters and observing the changes that those parameters produce. We need to understand how to tune these parameters and how to improve the outcome. Another focus this year is making predictions on friendships based on the answers submitted by participants through the surveys given. These findings will give the team insight to distinguish contact from homophily[2]. The team can gain a visual understanding of this information through histograms. Socio-economic status, gender, and number of addictive substances are key parts of homophily in this project that can be observed visually.

1.2 Expected Impact

This work is related to the Social Interactome (SI) project being conducted by the Addiction Recovery Research Center at Virginia Tech Carilion Research Institute (VTCRI) and the Computer Science department at Virginia Tech [3]. With the help of the code[14], which is a deliverable for the project, we will be able to find participants who are similar to one another. We will use this information for a live experiment that will be beginning in the near future.

1.3 Work Completed

The whole team has successfully achieved IRB Approval[4] and has examined the project data. This is a significant milestone because prior to approval, we were unable to access the database and work on our design. We have fully analyzed the previous team's contribution to the project[1][5]. Prashant has given us a rundown on what key aspects should be looked for in the data while we study the experiments that have been conducted.

We have developed a number of queries to gather the relevant data collected from user activity on Friendica[6]. We have created histograms to visualize certain queries in the MySQL database[7]. The histograms were made with Pandas[8] in Python[9]. Furthermore, the code is flexible with regard to reading through multiple queries. We have also begun to parse data from surveys collected from Friendica users.

Since the Implementation report, we have parsed the survey data for reliable entries. This has helped the team determine closeness between Friendica[6] users and identify trends between demographics we have collected and user friendship. The team has also installed Jupyter Notebook[10] to allow for further collaboration. In order to analyze our data, we have defined two different methods of defining friendship: a top-down method and the diffusion method. Our top-down method identifies similarity scores between recovery buddies so that we can identify pertinent homophily-based measures in common. The diffusion method analyzes how recovery buddies influence their peers within the Friendica[6] network. We have written the code, drawn conclusions based on these two methods, and computed probabilities of different homophily-based measures between recovery buddies.

1.4 Work in Progress

As we have defined important homophily-based measures, analyzed our data through our two methods, and extracted probabilities of these measures, we are working towards refinement and testing of our code so that we can draw optimal conclusions for our final deliverable. We will also compute weights on the prominent homophily-based measures so that the SIRecommender team can have data on what measures have shown importance over the course of the study.

2 User Manual

3 Developer Manual

3.1 Requirements

3.1.1 Users

We have identified two core groups of users for our algorithm, data clustering, and visualization.

The first group is the Social Interactome team that will use the cluster data and the friend recommender to aid in their research on homophily[2]. The Social Interactome[3] is an interdisciplinary research project that studies how social media can be used to help people in recovery from substance addiction. Because this is a research project focused on the interaction between people, our deliverables should provide data that the Social Interactome team will find useful to aid in further research. Additionally, our deliverables should be easy to visualize, and our code should be easy to adapt and understand. This element is key because the project is interdisciplinary and members of the team without a computing background should be able to interpret the results of the data received through the code.

The second group includes the research participants, with a history of substance abuse, who will directly use our friend recommending algorithm on Friendica[6]. These research participants can be further divided into two groups: those whose recovery buddies are organized using a lattice organization, and those whose recovery buddies are organized using a small-world organization. This distinction matters when querying data because the two groups were separated from interacting with each other at the time of the experiment. This distinction will also be important when finding a list of mutual friends because they were organized differently. Apart from those two cases, there should be little impact on our design or approach based on the initial organization of recovery buddies. It is also important to note that these participants are working through Amazon Mechanical Turk[11] or using Friendica[6] on their own computers/devices. Any algorithm we make should be able to function regardless of what machine it is run on.

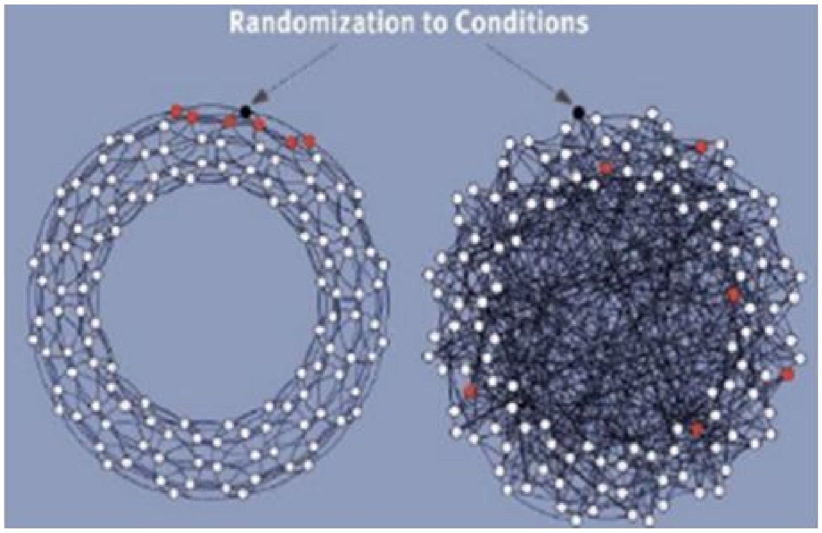


Figure 3.1 - Example of Lattice vs. Small world friend organization
(Pending citation information from our client Prashant)

3.1.2 Scenarios

There are a number of scenarios to consider when designing the functionality of our deliverables. In one scenario, the algorithm will be used in the middle of the 12 week period to suggest new recovery buddies for research participants. As the experiment progresses the number of research participants may change and the data collected may also change. We must reflect this by making our algorithm and our code as transparent and flexible as possible. Because of the complexity of a friend finder, we have come up with a table of special scenarios to consider based on participant data.

Table 3.1 - Special participant scenarios that require special consideration when designing the system

Scenario	**Why this scenario requires special consideration**
Participant with low site engagement	This person may have little information to aid us in recommending friends to them.
Participant with little in common with others	This person will still need friends recommended to them despite little compatibility.
Participant with a lot in common with many people	This person will need to have friends recommended to them in the best possible order.
Participants who are already recovery buddies with their best match	The recommender will need to update and remove current friends from consideration.
Participants who elect to remove themselves from the study	The recommender will need to update and not recommend people who are no longer in the system.
Participants who may have issued false information	We will need to recognize and remove instances of false data in order to keep the algorithm accurate.

Our algorithm will run concurrently with new data coming in. When users interact and participate in activities, the algorithm should be able to update friend recommendations. It will also work for both lattice and small-world organizations.

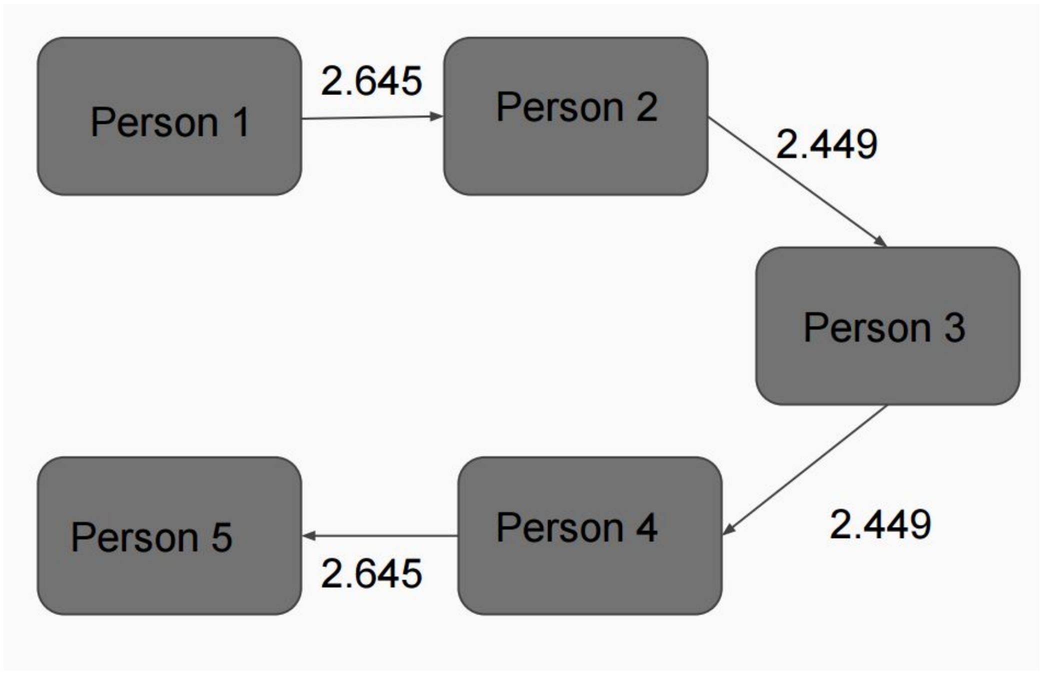


Figure 3.2 - Example of previous team's interpretation of the lattice network in tandem with similarity scores[1]

Figure 3.2 provides an example of last years team's lattice network with similarity scores as weights on the edges. The figure portrays the relationships between six recovery buddies and their similarities. The numbers represent their similarity scores and the links demonstrate the concept that the users are recovery buddies with each other in a chain-like fashion.

3.1.3 Data Involved

We have a variety of data to interact with for this project. There have been two iterations of the experiment; this gives us ample data to gather, parse through, and extract results from. We have demographic data such as age, income, gender, etc. This data has been supplied by participants through surveys associated with Friendica[6]. Later studies conducted on Amazon Mechanical Turk[11] also have led to data from surveys.

We have data on the participants' interaction with the Friendica[6] site. This data includes posts they have made to the social network, the video meetings they have attended, the news or success stories they have read, the TES (educational) modules they have completed, the assessments they have taken, and the messages they have sent and received from their recovery buddies.

We also have data on each user's recovery buddy network. There is an adjacency table representing each user and their relations to other users. This information is also supplemented by information on each user's 'connectedness' to their recovery buddies. This information is collected through a survey that asks each participant to share how close they feel to each of their recovery buddies and is compiled to create a 'social connectedness scale' ranking each participant's connection with their recovery buddies.

Because we have so much Personal Identifying Information (PII) there are stringent protocols in place to ensure the information is secure. For example, we will never use any participant's name in any part of the code, but will instead only reference them by their randomly assigned user ID. We have each also completed training and have been certified by the IRB (Institutional Review Board)[4] to ensure that we are contributing to the experiment in a safe and ethical manner.

3.1.4 Summary

To summarize, our system's most important requirement is to analyze, parse, and draw conclusions on the data given to us through surveys and CSV files of the previous team's work. To do this, we will create visualizations of data and build algorithms to help us extract what similarities are important between recovery buddies. This code will take into account each user's interaction with Friendica[6] throughout the initial 12 weeks of the experiment, including the educational modules they view, the posts they make, the video meetings they attend, the assessments they take, etc. Our system will also address all scenarios listed above, and do so in a manner that would satisfy both the SI team and users of the Friendica[6] system. The code will be implemented in a way that is easy to understand, be adaptable, be well documented for future use, and be secure in order to adhere to IRB requirements[4]. We will also test our code thoroughly and obtained comparison results that are as accurate as possible when compared to the social-connectedness scale.

3.2 Design

3.2.1 Approach

Our main goal for the SIRecommender project is to help the Social Interactome team measure homophily by identifying trends between recovery buddies on Friendica[6]. Our main questions are: 1) "Given the self-reporting of recovery buddies in Friendica, in what measures do they actually have things in common? 2) How often are people influenced by their peers to complete tasks within Friendica?"

Our first step is to parse the data and adapt the 2016 Social Interactome Recommender Team's code[1] to incorporate the data collected through the first 12 weeks of the experiment. The bulk of this information has been parsed from social media wall posts which contain information about articles read, modules liked, and other information that may be used to find commonalities and form a friendship.

After parsing the information, we will evaluate friendships and the homophile-based measures that the two people have in common. We will analyze them to find trends through the visualization of data (histograms) and a top-down approach. Our main focus is on our top-down approach, in which we determine the similarity scores of two recovery buddies given their demographics in common. When we identify pertinent demographics, we will calculate the probabilities of similarities so that we can visually describe how friendships are driven by similarities on Friendica[6]. This is part of our final deliverable at the end of the semester. We will also focus on diffusion. We analyze the tendency of a user to attend a meeting, watch a video module, or complete other tasks because another recovery buddy did so. This helps us identify how the network experiences diffusion. We use diffusion to also identify users that experience high or low amounts of interaction with other users and can identify their similarities through homophily-based measures. Our end deliverable will also consist of our histograms, data from the diffusion algorithm, and the probabilities from the top-down algorithm.

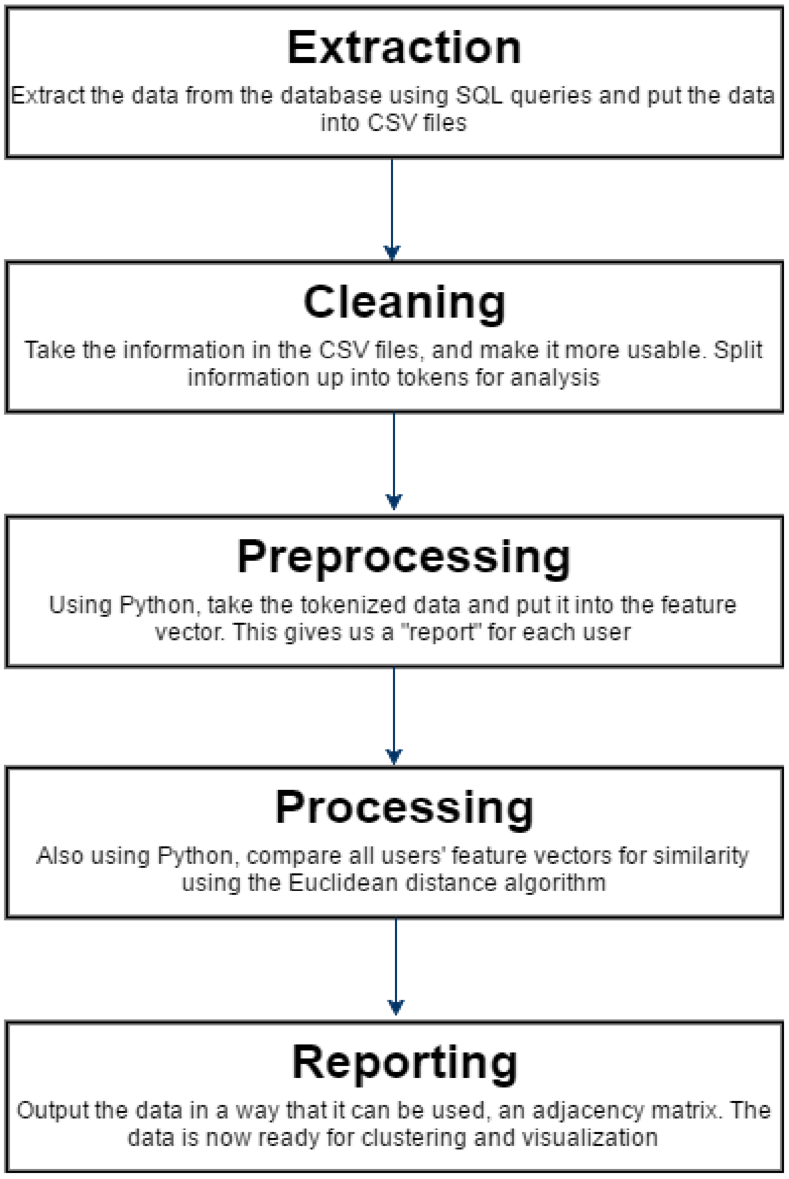
3.2.2 Interpreting Data

We have been granted database access in order to analyze the data and determine what information we want to use to compare mutual recovery buddies. After looking at the database, we can find some fields that will be useful in creating our algorithm. The demographics of the users such as gender, age, and type of addiction is a good place to start and should give us a baseline to recommend friends. We can then comb through the posts made by users and scrape the data, including the date/time of the post, the title, and the body of each post. Users with similar post content should have a higher chance of being recommended to each other.

We also have access to the video meetings each user has attended, the news and success stories they have read, the recovery modules they have completed, and the assessments they have taken. With this large amount of data to work with, we hope we will be able to create a fairly accurate algorithm to recommend friends. Users will be recommended based on overlap in these fields. For example, users that have read many of the same news or success stories have a higher chance of being recommended to each other. Also, users with similar responses on assessments would be more likely to be recommended to each other.

Another important data field to analyze is the amount of recovery buddies that a user already has access to. This is important to note because we want to avoid recommending current recovery buddies to users. In addition, we can compare friends lists of the current user and a recovery buddy. This is so that we can analyze if the two users have many recovery buddies in common-- in that case, they have a similar taste in recovery buddies. In this case, we can recommend the two users' recovery buddies to each other. In this way we will be basing some of our connections on the concept of 'mutual friends'.

3.2.3 Parsing Data



Extraction
Extract the data from the database using SQL queries and put the data into CSV files
Cleaning
Take the information in the CSV files, and make it more usable. Split information up into tokens for analysis
Preprocessing
Using Python, take the tokenized data and put it into the feature vector. This gives us a "report" for each user
Processing
Also using Python, compare all users' feature vectors for similarity using the Euclidean distance algorithm
Reporting
Output the data in a way that it can be used, an adjacency matrix. The data is now ready for clustering and visualization

Figure 3.3 - Flowchart showing the data parsing process

The raw data in the database is not useful in its current form, so we will be utilizing the parsing process detailed in Figure 3.3. The adjacency matrix we end up with is a representation of similarity scores for the users.

3.2.4 Testing Code

When testing our algorithm, there are many scenarios that we need to cover in order to form a complete assessment of our work. Instead of simply testing if our algorithm works, we need to consider some introspective questions as well. For example, one of the main questions of the study we are working on is: How can we best predict a connection between people? If our algorithm works, it should suggest people who are more similar to each other. This is a metric that may be difficult to measure, but we hope to see trends between our algorithm's similarity rankings and how users rank their friends that they like best. We can do this using the social connectedness scale, which is created by the Social Interactome team[3] based on survey responses from participants. Our hope is that our algorithm will output similarity values that correspond with the social connectedness values for each participant.

Aside from asking ourselves questions about our algorithm, our physical testing of our algorithm will be similar to how the previous group conducted their testing. The previous group used two different testing methods: testing through Steam, an online gaming community, and by checking similarity scores and looking at the values used to calculate those scores. We hope to leverage the first method to ensure that our code is robust and will work on multiple platforms, and use the second method to ensure that our algorithm is correct in recommending friends.

3.2.5 System Architecture

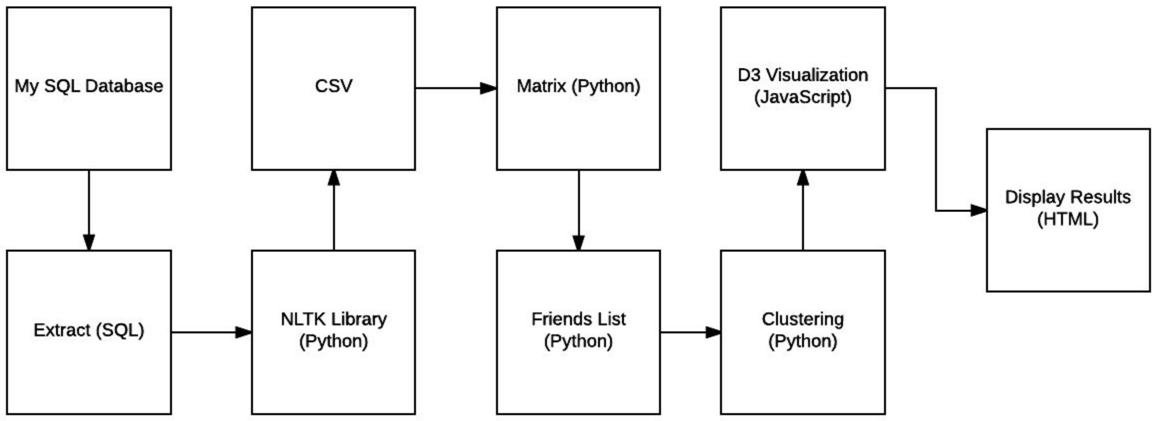


Figure 3.4 - Chart of proposed system architecture

3.2.6 Data Structures

3.2.6.1 Similarity Matrix

The main data structure the previous SIRecommender Team[5] selected to use was NumPy[12], which is an n-dimensional array object. NumPy[12] allowed for the previous team to hold values in their similarity matrices, making calculations easy for their algorithm. Based on our decisions in the near future regarding the algorithms we develop and paths we decide to pursue, our data structures are subject to change.

3.2.6.2 Data Cluster Vectors

In the event that we have time before the end of the project, we plan on also incorporating data clustering into our final deliverable. As of yet, we have not had substantial time to research the technology and incorporate it into our design. We would likely want to implement hierarchical clustering, since the data lends itself to connectivity-based clusters. These clusters will likely require their own matrix vectors, so as we continue to research clustering, we will be able to better define the data structure used for clustering. One such possibility is NumPy[12] because of its flexibility.

3.3 Implementation

3.3.1 Approach

Our approach to fulfilling all requirements has been adjusted over time to reflect the changing goals of our project as a whole. In this section we discuss the specific roles of each member on this project, the phases we have identified to break up the work into a cohesive timeline, and the technologies we need to use to accomplish our goals.

3.3.2 Roles

Of the six people on our team, we have broken down different roles for the different needs of the project. These roles are flexible in that if a teammate needs help with a part of the project that falls under their role, another teammate comes to their aid. In this way, we can emphasize the idea of teamwork while all working on our designated parts of the project. Note that every team member has a part in working on the algorithms involved and the reports; these are just specific roles for smaller, but necessary, parts of the project.

Bridget Larcher is the project lead. She is the main communication between Dr. Fox, Prashant, and the team itself. It is her job to schedule meetings and make sure everyone is on the same page. Her focus is on rewriting and revising parts of the report that are outdated or need updating. She also aids in data organization and visualization, e.g., histograms.

Sarah Devlin is the lead developer of the project. She mainly works on constructing the algorithm and helps the team with any developmental questions the team may have. She also focuses on developing the algorithm for the diffusion method that identifies the influence of recovery buddies to attend meetings, watch a video module, or complete other tasks. With background in Python[9] and MySQL[7], Sarah is an important member of the team.

Mary Kwiatkowski is the lead documentation contributor. She is the secondary lead on every report milestone. Mary aids in editing each report and making it flow with ‘one voice’ rather than having six different voices contributing to the report. She also has a background in working with Juypter Notebook[8] and has aided the team in its efforts to incorporate the application.

Matthew Bluman is the lead database manager and helps the team create CSV files when necessary. When creating histograms, Matthew is an integral part of the team. This is because he aids in searching the database with MySQL[7] queries and optimizing them in tandem with Sarah.

Timothy Tran is lead in quality control and testing. Without a quality control lead, our project won't be able to receive the emphasis on testing that it needs to be used live. Timothy also aids in histogram generation with Bridget and Matthew and develops the top-down method algorithm with Wilson. Incremental tests and quality checks will avoid possible problems in the future.

Wilson Rhodes is lead researcher and helps with connecting this year's project and last year's project. He studies the report from last year's team and helps us eliminate work that has already been done, raise problems that we need to address, and ask questions to Prashant or Dr. Fox that need to be approached. He and Timothy work on the generation of the top-down algorithm that identifies similarity scores between recovery buddies.

3.3.3 Phases

We have identified five main phases for implementation, given in Table 3.2.

Table 3.2 - Implementation Phases

Phase 1	Finding and testing data trends
Phase 2	Weighting friendship factors and updating the algorithm
Phase 3	Implementing work in Jupyter Notebook[10]
Phase 4	Testing the algorithm
Phase 5	Generating data clusters and visualization

3.3.3.1 Finding and testing data trends

This phase is the longest and most important part of our project. It involves parsing and interpreting the data so that we may find and analyze trends in the data, and their relation to friendship. The goal of this step is to find trends that indicate friendship between users.

This has been the most challenging phase thus far because of the ambiguity of how to define friendship given the Friendica[6] system and the data it has collected on participants. We are attempting to establish trends in friendship in two different ways.

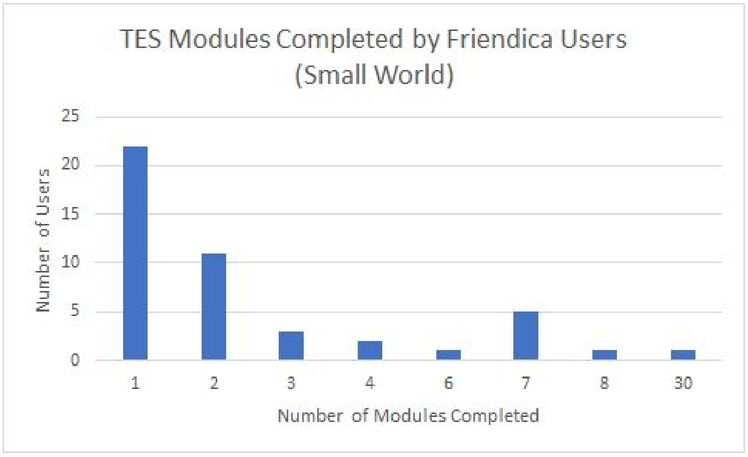
Method One:

We visualize demographic and participation data using histograms. We identify users who display similar amounts of participation and we explore other similarities among them (gender, socioeconomic background, etc.). Then we compare the similarities found to see if there is a trend that applies to more users, and we represent this data visually.

Method Two:

We identify friends based on participant surveys and then identify similarities between them. We then compare these similarities to the user group as a whole and represent this data visually.

For Method One, we have currently assessed user participation and are in the process of analyzing users with similar participation for other similarities. We have constructed a number of histograms for the data which can be seen in Figures 3.5 - 3.12 below.



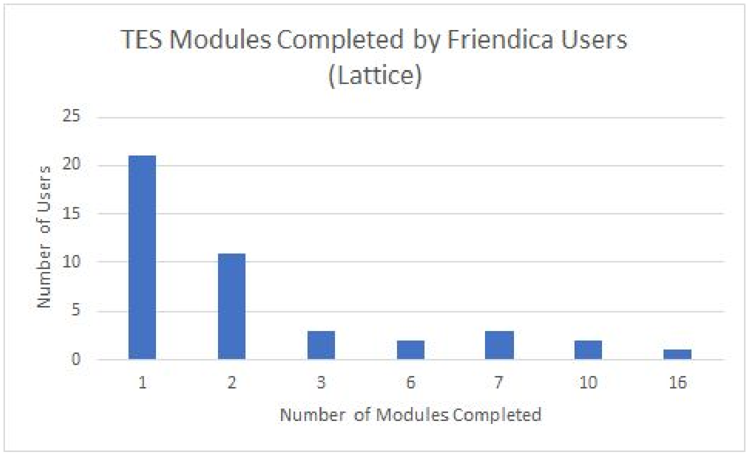
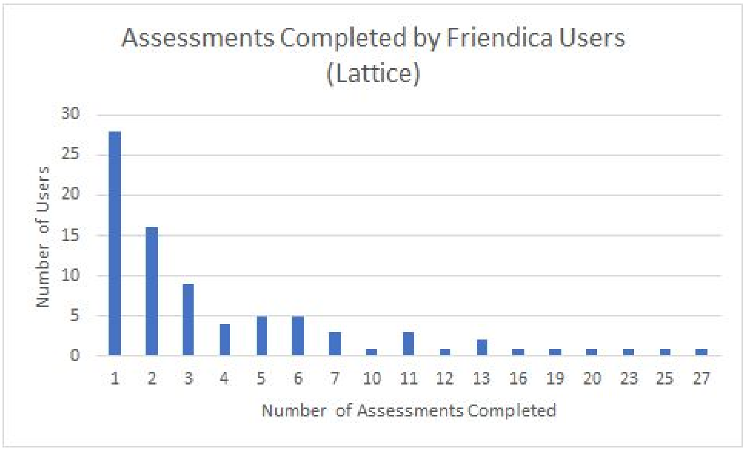


Figure 3.5, Figure 3.6 - User participation in educational modules for the Lattice and Small World organizations, respectively



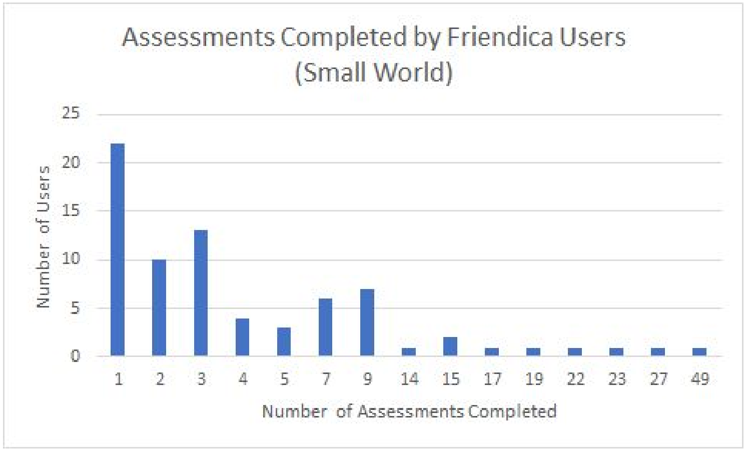
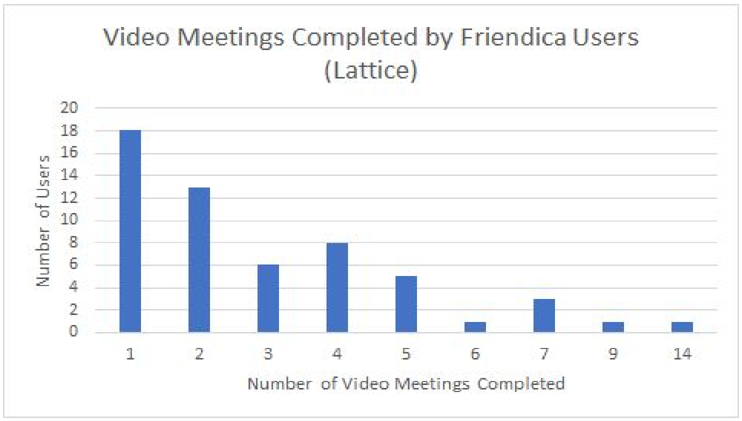


Figure 3.7, Figure 3.8 - User participation in assessments for the Lattice and Small World organizations, respectively



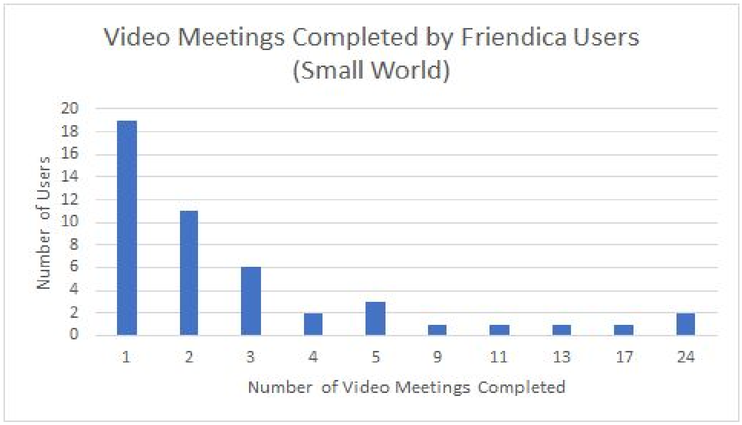
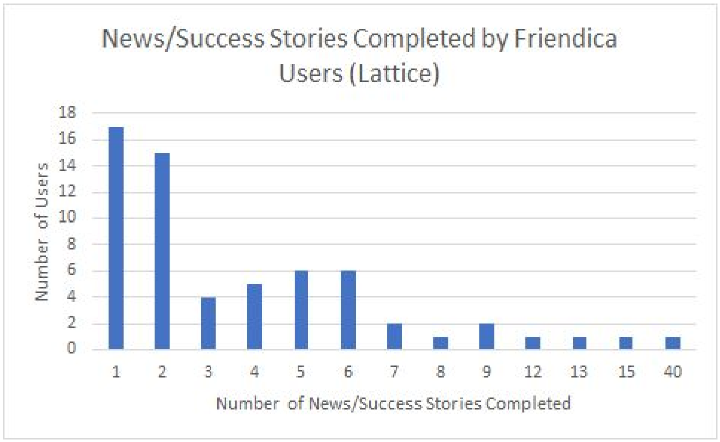


Figure 3.9, Figure 3.10 - User participation in video meetings for the Lattice and Small World organizations, respectively



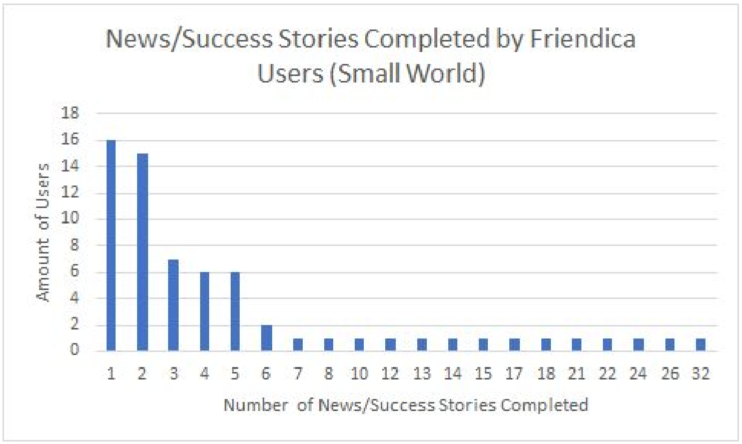


Figure 3.11, Figure 3.12 - User participation in news/success stories read for the Lattice and Small World organizations, respectively

We have completed parsing data for method 2. It was an arduous process because many participants entered false data, and those entries must be discovered and disregarded. We conducted this process by looking through answers manually. First we eliminated survey participants who disregarded or misunderstood the instructions. These were easy to identify because their answers did not fit the required form for a particular question. Next, we eliminated participants who listed the usernames of their friends inconsistently, because those should be constant throughout both surveys and are integral to the validity of our analysis. Finally, we disregarded entries where participants said they did not feel close to any of their recovery buddies, or if they said they felt close to their friend but their survey answers contradicted their statements.

3.3.3.2 Weighting friendship factors and updating the algorithm

Once trends have been identified in phase 1, we need to assess how much this information actually determines friendship. We can do this in 4 different ways. We can assign all trends equal weighting (which has been the method until this point), we can find an average of the weights that the SI team thinks are appropriate based on the data provided, we can infer the weights solely from the data (which is likely inaccurate due to the lack of available data), or we can bypass the weighting system altogether and explore other indications of friendship such as bi-directional communication between participants.

After establishing the weights for each category, we will need to apply it to the previous group's algorithm (which involves a Euclidean matrix) and make minor modifications to the code because they wrote it for the initial friend recommendations and we are adapting it for midway through the experiment.

3.3.3.3 Implementing work in Jupyter Notebook[10]

In order to easily collaborate and transfer code, data, and visualizations within the project team we plan to transfer our work over to Jupyter Notebook[10] to continue our evaluation. Jupyter's open source access and capability of implementing other languages along with Python[9] will greatly benefit our team in aiding each other in the coding process. Since Jupyter emphasizes a human friendly, text immersive environment, our team can cohesively read through each other's code and comments while being able to add thoughts and lines of our own when needed.

3.3.3.4 Testing the algorithm

Once our code has been moved to a Jupyter Notebook[10] so that the entire SI team can run the algorithm, we will continue to test and improve the code. The code will be tested to see that it meets all requirements, and it will also be tested for accuracy. This will be done by retroactively running the code on a finished experiment to see that the recommendations made represent the friend relationships somewhat accurately.

3.3.3.5 Generating data clusters and visualization

Our last phase is generating data clusters and creating visualizations of these data clusters. Once we know our system works somewhat well because of rigorous testing, we can use it to show the participants connections to one another through our algorithm, and we can compare these clusters to the relationships the participants actually have through Friendica[6]. The code to generate the data clusters and visualizations will also be uploaded to the Jupyter Notebook[10] in a format that is easily reused.

3.3.4 Timeline

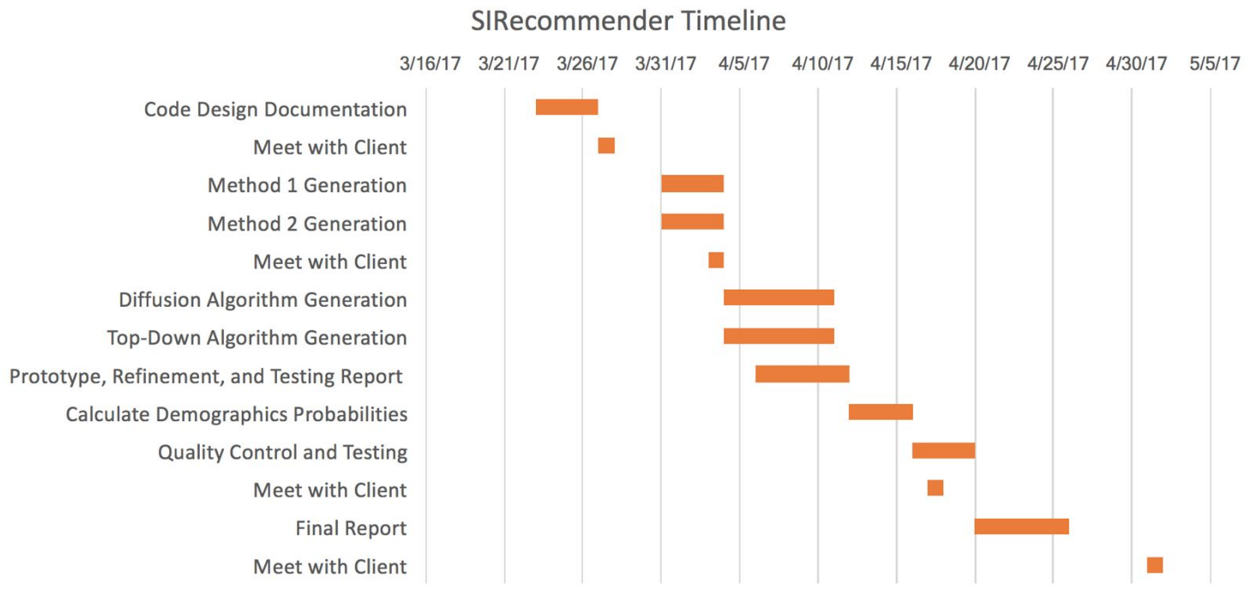


Figure 3.13 - Gantt Chart of our project timeline

Figure 3.13 shows a timeline of the milestones we will complete for this project. Frequent client meetings are important for the successful completion of this project, to insure we are working in the right direction.

3.3.5 Tools and Languages

3.3.5.1 Python

As stated above, Python[9] will be the central programming language that will be used throughout the duration of the project. So far, our group has used Python, specifically the library Pandas[8], to parse through the data and generate histograms of certain queries in the MySQL database[7]. Pandas[8] is a data analysis library for Python that allows us to easily display the data in graph form, i.e., in the form of histograms. With these histograms, we attempted to make connections/establish patterns with certain TES modules that were taken with regard to the users' recovery.

For the future, Python[9] will be used to improve the friend matching algorithm that last year's team developed. Based on the patterns we discover in the histograms, we will assign weights to the criteria that help best match friends. For example, if we find out that socio-economic status is the most important criterion that determines if two people are good friends, we will update our algorithm with a high weight for that criterion. Similarly, if we discover that gender plays little to no role in determining two people will be good friends, we can assign a very small weight to that criterion or remove it from our algorithm altogether.

3.3.5.2 MySQL Database

The data is stored in a MySQL database[7]. This will be the main resource we use for SQL data extraction using a number of queries. With these queries, we will be able to tokenize and cluster all of the data which should help us provide a more meaningful analysis.

3.3.5.3 Jupyter Notebook

A tool that Prashant recommended we use is Jupyter Notebook[10]. Jupyter Notebook is an open-source web application that will allow us to create and share documents that contain “live code”, data, and visuals. Our group will potentially use Jupyter Notebook to clean and transform data, model the data, and use other tools that the application has depending on the task at hand.

3.3.5.4 GitHub

To maintain and keep record of our developments for this project, a GitHub repository[14] was made and the team members have been added as collaborators. GitHub[13] is a web based version control system that extends from Git. GitHub allows us to develop and collaborate as a team. It also maintains past versions of the code documentation to allow backtracking and debugging. Currently, we have developed code to parse through the data for certain queries. This code can be pulled from all of the team members.

3.4 Prototyping

3.4.1 Top Down Approach

Prototyping began with our “top down” approach. The goal of the prototype was to first analyze the relationship quality that users reported, which could have either been “very close”, “somewhat close”, or “not close”. If recovery buddies stated that they were “somewhat close” or “very close”, we would look at the similarities in their demographics. We decided not to take into account the “not close” submissions because this would not provide us with accurate information to create a friend recommender. Similarities between “somewhat close” or “very close ” relationships would be used to calculate probabilities on how compatible two recovery buddies could be, given similarities in their demographics.

Demographics that would be compared from each recovery buddy includes:

- Age
- Gender
- Education
- Income
- Whether they’re a current smoker or ex-smoker
- Primary substance addiction
- Secondary substance addiction

Primary and secondary substance addictions may include:

- Alcohol
- Cannabis
- Cocaine
- Opioids
- Prescription pain relievers
- Stimulants
- Tranquilizers/depressants
- Dissociatives
- Nicotine
- Other (non-substances)

If these demographics are similar between recovery buddies that are close, then it will be taken into consideration as a factor in creating close relationships. Similarities might be based on categories, e.g., grouping together pain relievers, stimulants, and tranquilizers.

3.4.2 Diffusion Approach

We decided to investigate diffusion as another measurement for friendship or engagement. We have defined diffusion within our system to be the spreading of engagement through recovery buddies. We can measure diffusion because all user interaction with the Friendica site is posted and available for a user's recovery buddies to see and interact with. When Person A takes the same training module after their recovery buddy Person B takes the module, we can record this instance because Person A may have been motivated to take the module by seeing that Person B took it. Thus, our goal is to identify such instances, e.g., where a Friendica user completed a TES module and/or read a news/success story after one of their recovery buddies has.

Our analysis is based on the hard assumption that the time that the carrying out of an activity by a person after that the activity was carried out by their buddy is an instance of diffusion. However, just because one friend read an article after another friend read an article, we have no proof that these occurrences were connected by diffusion. They may not have seen or noticed that their recovery buddy completed the same activity.

The code was written with a number of constraints and requirements in mind. One such constraint was that we could not create procedures within the database to iterate through the code. Our response to this constraint was to write queries on multiple tables to extract the information into CSVs that we could manipulate outside of the database. The inputs are a list of users and their friends, and a list of posts that reflect user participation within Friendica. These can be found in two separate database tables and must be queried and converted to CSVs before passing them into the program. The outputs should give a description of how information has diffused within the system in an easily understood and easily modified way. Thus we decided to create two separate CSVs in order to organize our outputs in an efficient way.

Table 3.3 - Description of output files for the diffusion implementation

CSV	Description	Purpose
Diffusion_Raw.csv	Returns all relationships that show diffusion along with all important related data.	This CSV allows the Social Interactome team to manipulate the data in whatever way they choose.
Diffusion_Processed.csv	Returns a list of users along with summations of diffusion relationships per user.	This shows overall diffusion trends within the system and allows the Social Interactome team to see the data in a more easily understood format.

It is important to note that this method has a number of limitations based on limitations of the raw data from the experiment. There are no values distinguishing video meetings from each other in the database so we have no way of determining whether video meeting participation is for the same or different meetings. Additionally, assessments were incentivized in the experiment, so it is more likely that similar participation in assessments is due to the individuals actively seeking the incentives provided, not from viewing friends participating. As a result, we excluded such from our investigation of results.

Another limitation is our inability to tell whether or not a user actually looked at the news/success story or TES module, as opposed to simply clicking on it and then closing the tab.

3.4.3 Development and Application

3.4.3.1 Python

Python[9] was the primary language used to prototype the “top down” method. The language was instrumental in facilitating the tasks needed to fulfill the goals for the prototype. Python’s built-in commands, in addition to its relatively simple syntax, makes the code compact and readable.

3.4.3.2 Top Down Algorithm

Our code can found in our GitHub repository[14]. The runnable algorithm can be viewed in the “parser” branch of our repository. The file with the algorithm is named “topDown.py”[14]. We began with creating a “user” class that represented a user of Friendica. The constructor parameters of the user contain their respective username and their six recovery buddies. Within the constructor, we initialize variables to represent their demographics. We also initialize variables to represent demographics for each of the user’s recovery buddies. Each user executes the algorithm that includes the methods in Table 3.4.

Table 3.4 - Description of methods used for the top down implementation

Methods	**Description**
setFriend(timepoint)	This methods initially gives the user their recovery buddies that they were assigned.
setDemographics()	This method method assigns the user their demographics.
setSimilarities()	This method parses through the data tables and sets similarities for the user and their recovery buddies.
compareTwoUsers(user1, user2, friendNumber)	This method checks to see if two recovery buddies share the same similarities.
tallyCloseness()	This method is used to keep track of the users that reported close mutual relationships.

With all this data, we will calculate probabilities to determine how demographics contribute to a close relationship.These methods are operating on the data from the files, shown in Table 3.5, that we have queried from the MySQL database[7].

Table 3.5 - Description of CSV files used for top down implementation

CSV files	**Description**
DemoSmall.csv	This file contains the data from the Small World network, which includes demographics
DemoLattice.csv	This file contains the data from the Lattice network, which includes demographics
TimePoint2AllData.csv	This file contains data from TimePoint 2, which includes surveys about users' recovery buddies and their relationships
TimePoint3AllData.csv	This file contains data from TimePoint 3, which includes surveys about users' recovery buddies and their relationships

Additionally, our Python[9] code was designed with the notion of modularity, enabling our code to be easily edited to run on multiple replicate data sheets. This allows for future research/data samples to be easily parsed and analyzed.

3.4.3.3 Java

Java[15] is being used as a temporary language for the implementation of the diffusion algorithm. The developer working on diffusion is more comfortable working in Java so she chose to write the preliminary code in Java using Eclipse[16] in the interest of time. She is currently converting her solution to Python[9] and will conduct testing in Python. The initial Java code will also be used to check the results of the completed Python code.

3.4.3.4 Diffusion Implementation

Diffusion was implemented using multiple steps, the first of which was querying the database for the necessary data.

Table 3.6 - Diffusion query descriptions and importance

Query	**Description**	**Purpose**
User Friends (Appendix Query 10)	Returns all usernames along with an id corresponding to contact relationships	This information can be processed to create a network of contacts for each user with networks having a contact-network-id in common, and self-id identifying the owner of the network.
User posts (Appendix Query 9)	Returns the title of the post, the earliest date of creation, and the creator's username.	This information can be processed and compared with other posts to identify possible cases of diffusion when used in conjunction with the other query.

The queries listed in Table 3.6 were optimized to only include users that had TES or news/success stories. With this distinction there were still over 1000 records returned, so it is important to adjust the limit variable to ensure all relevant records are returned, not just the first 1000.

The code is currently written in Java and needs to be converted to Python for ease of use for the clients. Although the code was initially constructed hastily and poorly structured, Table 3.7 gives proposed classes to increase efficiency of the code.

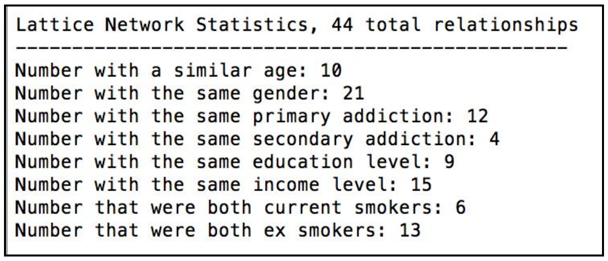
Table 3.7 - Proposed class structure for Diffusion solution

Class	**Variables**	**Purpose**
User	String Id ArrayList<User> friendList ArrayList<Post> postList	This class stores all information for each user, making it easy to access each instance.
Post	String Id String Title Date created	This class holds all relevant post information together.
Node	String Id String friendId String Title Date created Date friendCreated	This class is used to store instances of diffusion. It can be used to build chains and create networks by connecting Ids and friendIds.

3.4.4 Analysis

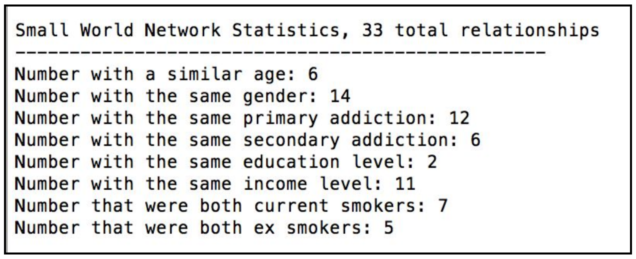
3.4.4.1 Top Down Results

After running our Top-Down program, the results are shown in Figures 3.14 through 3.16.



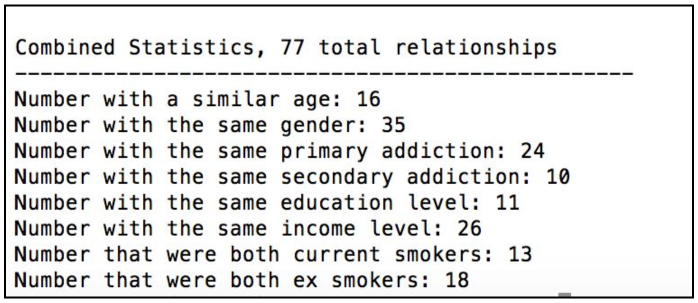
```
Lattice Network Statistics, 44 total relationships
--------------------------------------------------
Number with a similar age: 10
Number with the same gender: 21
Number with the same primary addiction: 12
Number with the same secondary addiction: 4
Number with the same education level: 9
Number with the same income level: 15
Number that were both current smokers: 6
Number that were both ex smokers: 13
```

Figure 3.14 - Lattice Network Top-Down Results



```
Small World Network Statistics, 33 total relationships
------------------------------------------------
Number with a similar age: 6
Number with the same gender: 14
Number with the same primary addiction: 12
Number with the same secondary addiction: 6
Number with the same education level: 2
Number with the same income level: 11
Number that were both current smokers: 7
Number that were both ex smokers: 5
```

Figure 3.15 - Small Word Network Top-Down Results



```
Combined Statistics, 77 total relationships
------------------------------------------------
Number with a similar age: 16
Number with the same gender: 35
Number with the same primary addiction: 24
Number with the same secondary addiction: 10
Number with the same education level: 11
Number with the same income level: 26
Number that were both current smokers: 13
Number that were both ex smokers: 18
```

Figure 3.16 - Combined Top-Down Results

This data represents the number of similarities in demographics between the users and their recovery buddies. Probabilities can be calculated for each demographic. It should be noted that in both the Lattice and the

Small World Networks, only 3 users identified another user as having a "very close" relationship.

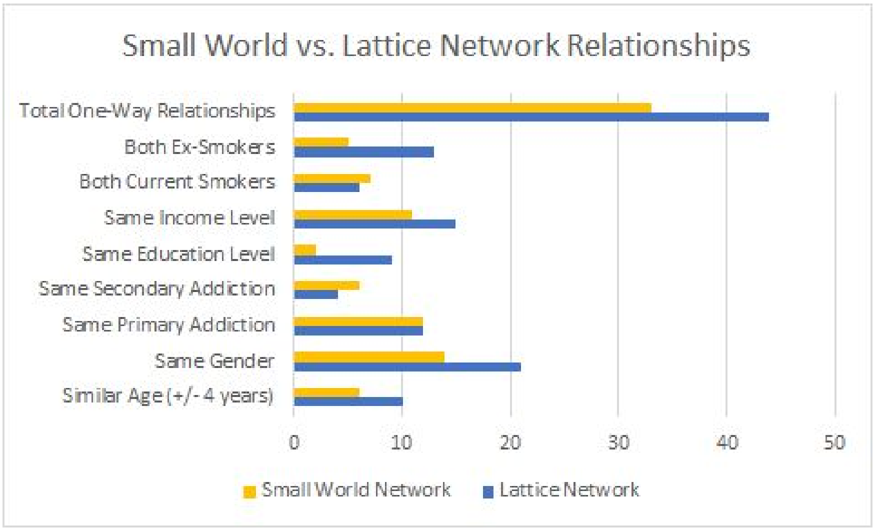


Figure 3.17 - Small World and Lattice Network Demographic Relationships

Figure 3.17 depicts the number of distinct relationships among somewhat close and very close users in each demographic category. It can be determined from this graph that the Lattice Network contained higher results in nearly all of the demographics, however the numbers were fairly similar in general.

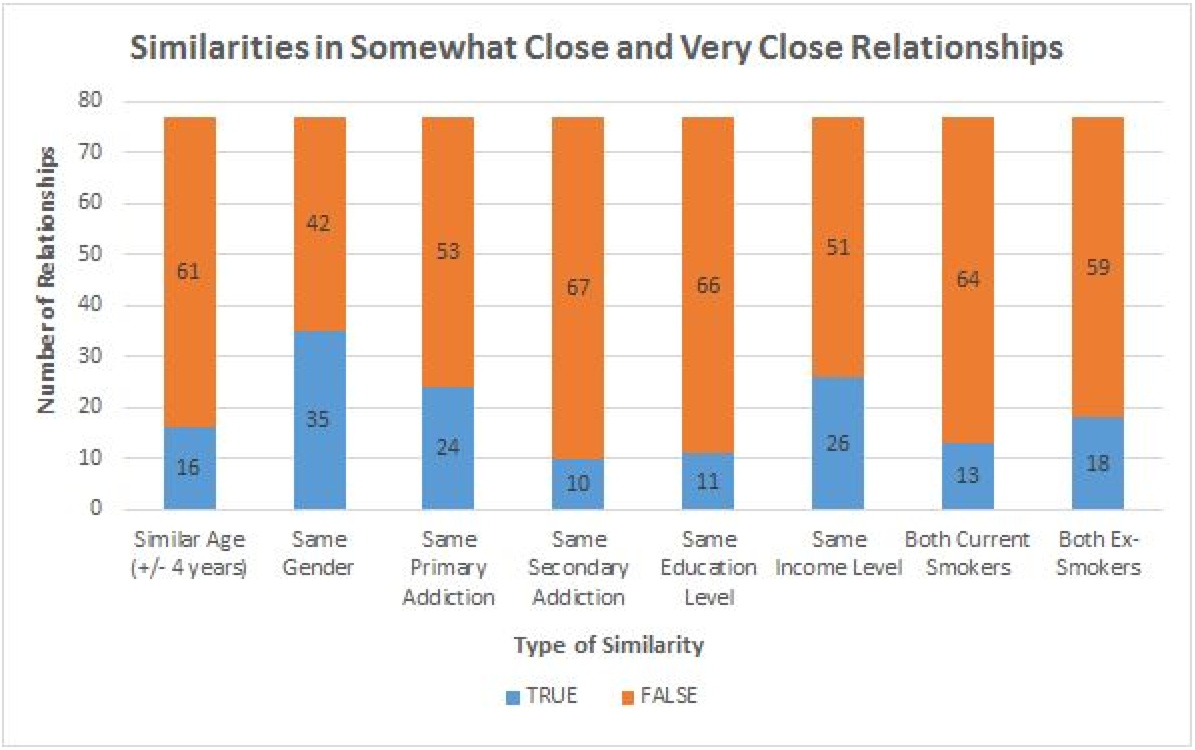


Figure 3.18 - The Demographic Similarities compared to the Total Number of Close Users in both Networks

Figure 3.18 compares the total number of somewhat close or very close users among the combined Lattice and Small World Networks in each demographic category. The total number of close users was 77. The blue sections of the bars display the number of users out of 77 that displayed that particular demographic in their relationship. The orange bars depict the number of relationships out of that 77 that did not display that demographic. From this we can determine the percentages of each demographic; see Table 3.8.

Table 3.8 - The Percentage of the Total in each Demographic

Demographic	**Number of Relationships**	**Percentage**
Similar Age (+/- 4 years)	16	20.78
Same Gender	35	45.45
Same Primary Addiction	24	31.17
Same Secondary Addiction	10	12.99
Same Education Level	11	14.29
Same Income Level	26	33.77
Both Current Smokers	13	16.88
Both Ex-Smokers	18	23.38

Table 3.8 calculates the percentages of each demographic when compared to the total number of close relationships. From this we can determine the likelihood, or probability that close friends would have this similarity in the future. For example, based on the percentages, there is a 45.45% chance that close friends share the same gender.

3.4.4.2 Diffusion Results

There were a number of interesting results from our analysis of diffusion. While looking at individual diffusion cases it was easy to identify how a particular story might have moved around a network of recovery buddies. For example, there was a particular story that was reviewed by 6+ recovery buddies.

Diffusion as a whole across the network was more difficult to make inferences about. One way we chose to analyze the network was by looking at the amount of diffusion as a percentage of posts per user, as shown in Figure 3.19.

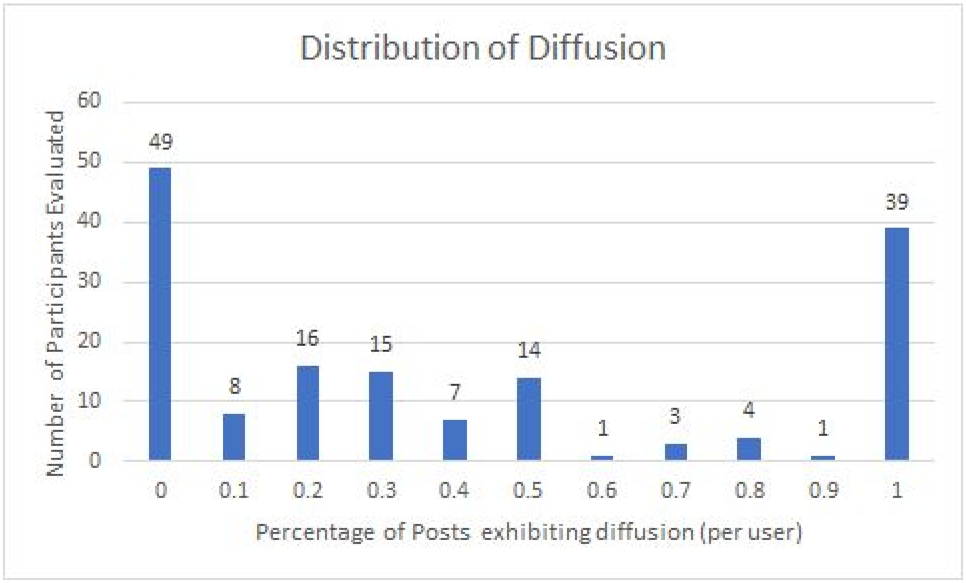


Figure 3.19 - Amount of diffusion exhibited by a number of users on Friendica

Another way of measuring diffusion was to look at the percentage of diffusion posts per person compared to the number of friends with whom they exhibited diffusion. This gives us an idea of how much diffusion an individual experiences and from how many people in their network they will see the same post, before they participate in the same activity.

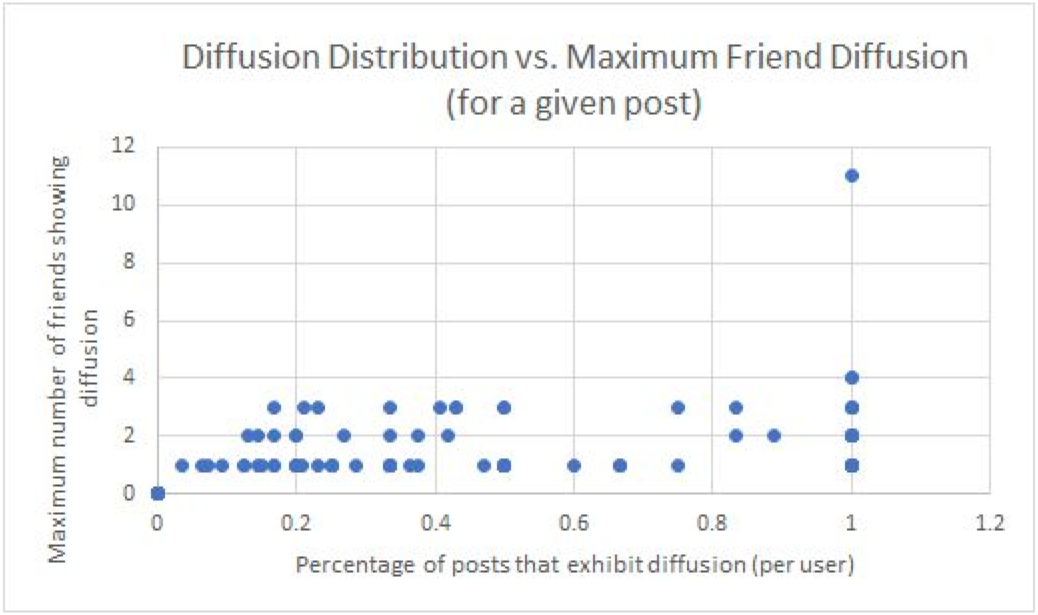


Figure 3.20 - Diffusion distribution based on number of friends showing diffusion

3.5 Refinement

3.5.1 Top Down Refinement

3.5.1.1 Adding classes

Our program could be refactored to eliminate repetitive code. Currently, it is designed to have each user contain their six recovery buddies all inside the constructor. Along with the user's data on their demographics, the constructor also contains the six recovery buddies' demographic data, which is not an optimal design. Creating a "friend" class would aid in refining the program by eliminating the repetition in our "compareTwoUsers"[14] method. The addition of this class will improve readability by adding a "friend" class to replace repetitive "if" statements that check for similar demographics.

In addition, adding a "skeleton" user class would allow us to have a user class that only stores demographics, not friends. This would allow us to save space for users that do not have any friends listed. A user might not have any friends listed because they filled out the Timepoint surveys incorrectly, and this class would allow us to have a special class just for these users that don't appear in the Timepoint surveys.

3.5.1.2 Weighted similarities

Currently, our code checks for direct similarities between demographics for education and socio-economic background. The algorithm should be adjusted to reflect a more interrelated data set within these two sets of demographic information. For example, a user with a bachelor's degree would have more similarities in their education with a recovery buddy that earned a master's degree versus a recovery buddy that earned a doctorate degree. Likewise, a user with a socio-economic background that earns $50,000 - $69,999 would have more similarities with someone who earns $70,000 - $89,999 as opposed to someone who earns $150,000 and above. To implement this, we could set the "compareTwoUsers" method to set users to have similar income if their income brackets were within 1 step of each other. For example a user in the $50,000 - $69,999 bracket could match with users in the $30,000-$49,999 bracket or the $70,000 - $89,999 bracket.

Furthermore, refinement could be continued within our "compareTwoUsers" method where the program compares the age between recovery buddies. As of now, the code determines that two recovery buddies are similar in age if they are within four years of each other. The range of four years was selected based on an educated guess.

With further reasoning, research, and data to back our decisions, the age range could be modified to a number more appropriate/accurate. In addition, weights could be added to age similarities. For example, a user that is 41 year old will have a higher similarity score to another user that is 41 years old versus someone that is 43 years old. Likewise, weights could be determined for income, so a higher weight is given in the same bracket, but a lesser weight if one bracket away.

Moreover, our program does not differentiate the difference between relationships that are “very close” and “somewhat close”. Adding weights to these categories could help establish more accurate results when determining which demographics contribute to close relationships. The “very close” relationships will have a stronger weight when determining probabilities than the “somewhat close” relationships. The “somewhat close” relationships could be refined to having a moderate impact on calculating the probabilities of certain demographics contributing to relationships.

3.5.2 Diffusion Refinement

3.5.2.1 Conversion to Python

The rest of our code[14] as well as the previous team’s code[5] are all written in Python[9]. For consistency and ease of use it makes sense to convert code now in Java[15] to Python. This will also allow the code to be included in the Jupyter Notebook[10] that will be delivered to the Social Interactome team.

There will be a number of large changes to the code so instead of transcribing the Java[15] code in Python and making changes from there, we will design the code more thoroughly and implement a more efficient and optimized version in Python the first time.

3.5.2.2 Inclusion of chains

We think the inclusion of code that will easily generate diffusion chains could be a great addition to the functionality of the code. A diffusion chain in the context of our program would be a user who experiences diffusion with a friend, and the friend experiences diffusion with another friend, and so on for a particular post. The creation of a node class (Table 3.7) would easily allow each instance to be linked, but we may need to reconsider the best way to implement the node class going forward.

Possible functionality to include with chains would be finding the length of a chain given the user that is the starting point or the user that is the ending point. We could also provide methods to calculate all chains and display these in a network graph. Another method we could provide is finding chains based off of the title of the post the chain is created from. The Social Interactome could use these chains to visually represent diffusion in a number of ways.

3.5.2.3 Restructuring classes and methods

The current code was built hastily to provide long awaited functionality to the system and does not follow the best coding conventions. An important part of rewriting this code will be breaking the code into segments that allow for better extensibility and understanding. The proposed classes listed above will provide some much needed structure to the code and make the solution easier to modify in the future. Utility methods will also improve developer understanding in the future and reduce the number of lines in the program making it also easier to read.

3.5.2.4 Improving efficiency

The current program makes use of several nested for loops that cycle through all the records to retrieve one. This is incredibly inefficient and can be adjusted, to improve the code. Improving efficiency will help the Social Interactome team long-term, especially if they begin to conduct experiments involving larger data sets where a drop in efficiency is very noticeable.

3.5.2.5 Generating a network graph

We would like to create another tool for the Social Interactome team to use to visualize diffusion as a network. We will need to process the information into source and destination nodes with a weight along the edge. The source node will correspond to a user who looked at a post after one of their friends, the destination node will be the friend who the user has viewed a post after, and the edge represent the number of posts where these two users exhibit diffusion in that direction.

3.6 Testing

3.6.1 Top Down Testing

3.6.1.1 Top Down unit testing

To ensure that our top down algorithm computes the desired results, we will provide unit tests to ensure that our program is functioning correctly. We are developing individual tests for each method in our Python code[14] to ensure each test outputs the correct data. Additionally, unit tests will be developed on a case by case basis, ensuring that tests will encompass all the possible behaviors from the program. See Table 3.9.

Table 3.9 - The Description of the each Top Down Unit Test

Test Method Name	**Description**
testSetFriends()	This test ensures that each user in the user list has correct closeness values. Due to the size of the list, we may have to test select users.
testSetDemographics()	This test ensures that each user in the user list has correct demographic values. Due to the size of the list, we may have to test select users.
testSetSimilarities()	This test ensures that each friend of a user in the user list has the correct similarity values. Due to the size of the list, we may have to test select users.
testCompareTwoUsers()	This test ensures that two selected users have correct similarity values after they are passed in the function.
testTallyCloseness()	This test ensures that the correct tally output is printed in the function.
testCombineLists()	This test ensures that given two lists, the return list is correctly modified to be the combination of the two lists.
testPrintAllFriends()	This test ensures that the correct friend output is printed during the function call.
testOrderList()	This test ensures that a given list is sorted correctly during the function call.

testCreateStatistics()	This test ensures that statistic values are generated correctly during the function call.

3.6.1.2 Top Down integration/validation testing

Following unit testing, we know that all functions are working independently. Integration testing will involve ensuring that all functions interact with each other correctly. Validation testing involves ensuring that all output is correct given certain input. Assuming all unit tests pass and each function is working properly, we can combine these two tests and perform a test that calls all functions with an input CSV file and examine the output created to verify that it is equivalent to the expected result. All of these individual methods (see Table 3.10) are supposed to be called sequentially and work together so in order to calculate the statistics so we must check that the statistics output is reasonable and the methods are run together without any major error.

Table 3.10 - The Description of the each Top Down Integration/Validation Step

Step	**Description**
Prepare CSV	This step involves the creation and preparation of CSV files to be used as input. This step also includes data cleaning to ensure successful results.
Define User Lists	This step requires creating lists of users and their respective friends at different timepoints and then combining those lists to create two complete Lattice and Small user lists.
Call Setup Functions	This step includes the primary work of the approach in that demographics must be set and similarities must be examined and tallied. setDemographics(), setSimilarites(), and tallyCloseness() should be called here.
Sort All Lists	This step is where all finalized lists are sorted to be better organized for later evaluation and analysis.
Call Output Functions	This step involves putting the results of the top down approach to output. printAllFriends() and createStatistics() are called here.
Evaluate Output	This is the final test that ensures that the output is exactly as expected when integrating all functions with given input.

3.6.2 Diffusion Testing

3.6.2.1 Diffusion unit testing

Unit tests (see Table 3.11) will be created to guarantee that the code behaves as it is expected to. A few key functionalities that will need to be tested include parsing and reading in the data, linking each user to their friends, linking each post to its creator, converting the date fields from strings into date formats, finding diffusion, and linking diffusion instances into chains.

Table 3.11 - The Description of the each Diffusion Unit Test

Test Method Name	Description
testParsing()	This test ensures that all data can be parsed and read correctly.
testLinking()	This test ensures that each user can be correctly linked to their friends and each post can be linked to its creator.
testFormat()	This test ensures that conversions can be performed successfully for tasks such as converting date fields from strings to date formats.
testDiffusion()	This test ensures that diffusion can be found and that each instance can also be linked into chains.

3.6.2.2 Diffusion integration/validation testing

Integration and validation tests can be performed together in this category since they will examine correct output that is created using a combination of all of the steps of the diffusion implementation given certain input CSV files. Because the input data is taken from queries from the database, we will need to ensure that we can handle all situations of data found within the database. For example, the code separates columns from the CSV by breaking each line using commas. There is certain information (such as post titles) that include multiple commas within one field. We will need to test that all data fields can handle having commas within them. This code is supposed to create two output CSVs, one describing all relationships that show diffusion and the other involving a list of users and diffusion relationships associated with each user. Both of these output files will need to be examined for correct output.

4.0 Lessons Learned

5.0 Acknowledgements

We would like to acknowledge our client Prashant Chandrasekar for his patience and support with our work thus far. We would also like to acknowledge Dr. Edward Fox [3] for helping us develop our knowledge of the project, for giving helpful feedback on our project proposal and for taking time out of his day to answer our innumerable questions. Finally we would like to acknowledge Andrew Baehre, Colin Gray, and Trevor Kinaman, the previous group who worked on the Social Interactome friend finder. Their algorithm[5] has been the basis of our work thus far, and their documentation[1] has helped us understand the system better.

6.0 References

1. Andrew Baehre, Colin Gray, and Trevor Kinaman. 2016.
   SIRecommender Final Report and Presentation
   https://vtechworks.lib.vt.edu/handle/10919/70956. Accessed: 19 February 2017.

2. Miller McPherson, Lynn Smith-Lovin, and James M Cook. BIRDS OF A FEATHER: Homophily in Social Networks.
   http://aris.ss.uci.edu/~lin/52.pdf. Accessed: 09 April 2017.

3. Edward A. Fox. 2017.
   The Social Interactome of Recovery: Social Media as Therapy Development.
   https://www.cs.vt.edu/node/7439. Accessed: 19 February 2017.

4. "VT Human Subject Protections Tutorial." *VT Human Subject Protections Tutorial | Institutional Review Board | Virginia Tech*. Web. 09 Apr. 2017.

5. Andrew Baehre, Colin Gray, and Trevor Kinaman. 2016.
   SIRecommender Code
   https://github.com/baehre/SIRecommender. Accessed: 19 February 2017.

6. Friendica. Stable Release, 2017. Mike Macgirvin. Distributed social network.
   http://friendi.ca/ Accessed: 12 March 2017.

7. MySQL. 2017. Oracle Corporation. https://www.mysql.com/. Accessed: 11 April 2017.

8. Pandas. 2016. Python Data Analysis Library. Open Source Programming Application. http://pandas.pydata.org/. Accessed: 09 April 2017.

9. Python. 2001-2017. Python Software Foundation. https://www.python.org/. Accessed: 11 April 2017.

10. Jupyter Notebook. 2017. Project Jupyter - Last updated Sunday, March 05, 2017. Open Source Programming Application. http://jupyter.org/. Accessed: 17 March 2017.

11. Amazon Mechanical Turk. 2005-2017. Amazon.com. https://www.mturk.com/mturk/welcome. Accessed: 09 April 2017.

12. NumPy. 2017. NumPy. http://www.numpy.org/. Accessed: 09 April 2017.

13. GitHub. 2017. https://github.com/. Accessed: 11 April 2017.

14. Timothy Tran, Wilson Rhodes, Sarah Devlin, Bridget Larcher, Mary Kwiatkowski, and Matthew Blumen. 2017.
SIFriendRecommender Code
https://github.com/tttran/SIFriendRecommender. Accessed: 11 April 2017.

15. Java. 2017. Oracle Corporation. https://www.java.com/en/. Accessed: 11 April 2017.

16. Eclipse. 2017. The Eclipse Foundation. https://eclipse.org/. Accessed: 11 April 2017.

7.0 Old Appendix

Lattice Participation Histograms

	Query Data	Query syntax
1	TES Modules	select Lattice_TES_modules_taken, count(Lattice_TES_modules_taken) as number_of_people from (select `author-name`, count(`author-name`) as Lattice_TES_modules_taken from (select distinct `author-name`, title from friendica_e1r2.item where file = "<TES>" and wall = 1 and `author-name` between "u4001" and "u4128") as stuff group by `author-name`) as Lattice_TES_modules group by Lattice_TES_modules_taken order by count(`author-name`) desc;
2	Assessments	select Lattice_assessments_taken, count(Lattice_assessments_taken) as number_of_people from (select `author-name`, count(`author-name`) as Lattice_assessments_taken from (select distinct `author-name`, title from friendica_e1r2.item where `author-name` <> 'Administrator' and body like 'I am taking the [bookmark=%' AND body NOT LIKE '%$%' AND wall = 1 and `author-name` between "u4001" and "u4128") as stuff group by `author-name`) as Lattice_assessments group by Lattice_assessments_taken;
3	Video Meetings	select Lattice_videos_attended, count(Lattice_videos_attended) as number_of_people from (select `author-name`, count(`author-name`) as Lattice_videos_attended from friendica_e1r2.item where file = "<Video Meeting>" and wall = 1 and `author-name` between "u4001" and "u4128" group by `author-name`) as Lattice_videos group by Lattice_videos_attended;
4	News/success stories	select Lattice_News, count(Lattice_News) as number_of_people from (select `author-name`, count(`author-name`) as Lattice_News from (select distinct `author-name`, title from friendica_e1r2.item where file = "<news>" AND wall = 1 and `author-name` between "u4001" and "u4128") as stuff group by `author-name`) as Lattice_News_1 group by Lattice_News order by count(`author-name`) desc;

Small World Participation Histograms

	Query Data	Query syntax
5	TES Modules	select Small_TES_modules_taken, count(Small_TES_modules_taken) as number_of_people from (select `author-name`, count(`author-name`) as Small_TES_modules_taken from (select distinct `author-name`, title from friendica_e1r2.item where file = "<TES>" and wall = 1 and `author-name` between "u4129" and "u4256") as stuff group by `author-name`) as Small_TES_modules group by Small_TES_modules_taken;
6	Assessments	select Small_assessments_taken, count(Small_assessments_taken) as number_of_people from (select `author-name`, count(`author-name`) as Small_assessments_taken from (select distinct `author-name`, title from friendica_e1r2.item where `author-name` <> 'Administrator' and body like 'I am taking the [bookmark=%' AND body NOT LIKE '%$%' AND wall = 1 and `author-name` between "u4129" and "u4256") as stuff group by `author-name`) as Small_assessments group by Small_assessments_taken;
7	Video Meetings	select Small_videos_attended, count(Small_videos_attended) as number_of_people from (select `author-name`, count(`author-name`) as Small_videos_attended from friendica_e1r2.item where file = "<Video Meeting>" and wall = 1 and `author-name` between "u4129" and "u4256" group by `author-name`) as Small_videos group by Small_videos_attended;
8	News/success stories	select Small_News, count(Small_News) as number_of_people from (select `author-name`, count(`author-name`) as Small_News from (select distinct `author-name`, title from friendica_e1r2.item where file = "<news>" AND wall = 1 and `author-name` between "u4129" and "u4256") as stuff group by `author-name`) as Small_News1 group by Small_News order by count(`author-name`) desc;

Diffusion Query for Posts

9	TES and news/success posts	select `author-name` as id, title, created from friendica_e1r2.item where (file = "<news>" or file = "<TES>") and wall=1 group by `author-name`, title, file order by id;

Diffusion Query for Friends

10	User Friends (just those with news or TES entries)	select nick, self, uid from friendica_e1r2.contact where nick not like "admin" and nick in (Select distinct `author-name` as id from friendica_e1r2.item where wall = 1 and (file = "<news>" or file = "<TES>")) limit 0,2000;