Group 2 Technical Report

Assignment 3

Jonathan Healy, Kevin Chen, Soroush Yousefi

1. **Design decisions (project architecture)**
    a. **preprocessing**
    b. **backend**
    c. **frontend**
    d. **jupyter notebook**

2. **Modeling choices (Data Modelling)**
    a. **graph database vs relational database**

3. **Scalability handling / cost measures (Algorithmic considerations)**
    a. **cost measures**
    b. **clustering and downsampling**

4. **Visualization graphs+Insights (Transforming Raw Data to Insights)**
    a. **Node-Link Diagram**
    b. **Edge-Bundled Graph**
    c. **Chord Diagram**
    d. **Matrix**

Step by Step process of what we did

1. **Importing data to Neo4j**

2. **Running clustering algorithms**

3. **Setting up backend**

4. **Connecting frontend to endpoints and visualizing data**

1. Design decisions

1.a Preprocessing

For this assignment, we decided to use a dedicated graph database, neo4J, to help us process and store the data. By picking up the basic Cypher query language, we managed to use it load raw data into the database. We have two scripts to store our data:

1. **General data preprocessing:**

   This query gives us a general structure of the graph data, First, it has one node object: Subreddit representing the aggregation of the source and target subreddit in the raw data. Second, it returns the huge set of connections: LINK which are set in between two Subreddit. Having this query, we now have a general multiple directional graph stored in the Neo4J graph database.

```
# load data and create links between source and target
query = '''
USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM "file:////soc-redditHyperlinks-title.tsv" as
row FIELDTERMINATOR "\t"
MERGE (s:Subreddit{id:row.SOURCE_SUBREDDIT})
MERGE (t:Subreddit{id:row.TARGET_SUBREDDIT})
CREATE (s)-[:LINK{post_id:row.POST_ID,
   link_sentiment:toInteger(row.LINK_SENTIMENT),
   date:localDateTime(replace(row['TIMESTAMP'],' ','T'))}]->(t)
'''
```

2. **Weighted data preprocessing:**

   To narrow the scope of the data we were working with, we created a CSV file in Jupyter with Python which stores only posts with negative link sentiments. Because there can be numerous links between the same source and target Subreddits we added the number of negative connections together and then imported the CSV into neo4j to create a weighted graph. An example query on this graph can be seen below in Figure 1.

```
# load data and create links between source and target
query = '''
USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM "file:////matrix1_neg.csv" as row
MERGE (s:Subreddit{id:row.source})
MERGE (t:Subreddit{id:row.target})
CREATE (s)-[:LINK{post_id:row.POST_ID,
   weight:toInteger(row.weight)}]->(t)
'''
```

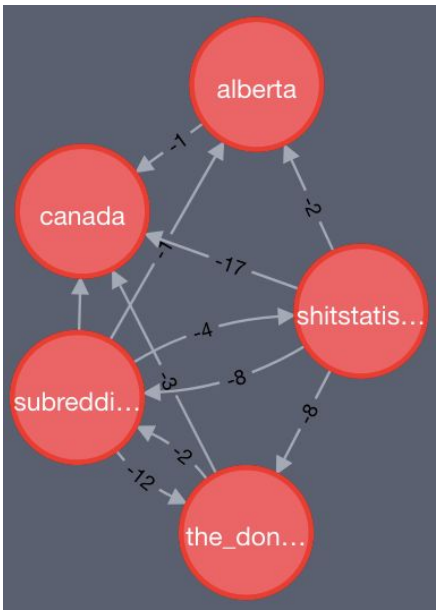

Figure 1: All shortest paths between ‘alberta’ and ‘the_donald’. Limit 4. Queried in the neo4j browser:

```
MATCH (a:Subreddit {id: "the_donald"}), (b:Subreddit {id: "alberta"})
MATCH p=allShortestPaths((a)-[:LINK*]-(b))
RETURN p LIMIT 4
```

1.b Backend

For this assignment, we mostly kept the frameworks that we used for the previous assignment. Flask is a very lightweight compact server framework which is easy to set up, access, and it comes with rich features that satisfies our standard well enough for the assignment. In order to work better with neo4j, our graph database, we imported NetworkX, a python module to better study the structure, dynamics, and functions of complex graphs. This has helped us to manipulate and modify the data format that backend framework desired.

Additionally, we implemented Docker, more specifically docker-compose, to run neo4j and set up the multiple parameters needed to make working on this project easier for everyone involved. With docker-compose we can set up passwords, ports, and install the packages we need etc. all with the command: docker-compose up neo4j.

1.c Frontend

For the frontend we used Reactjs and D3.js library. Since Reactjs is a component based library to write code for web applications, it is pretty useful when you want to reuse components. Starting from the first assignment we used Reactjs so whenever we moved to the next assignment, we had reusable components from previous assignments, which made our life easier!

D3.js is a strong visualization library which many projects use it to visualize data. We have successfully integrated D3.js with Reactjs to create reusable visualization components to use it in our visualizations. D3.js have powerful tools to take advantage of. For the third assignment, we were able to visualize graph data in four different ways which is very hard to visualize without using this library. This library is not only about static visualization. It has powerful tools to make your visualizations interactive. As data grows it is harder for users to explore them. So interactivity is a key solution to make it easier for users to get meaningful insights [1]. We used these two libraries to make reusable, interactive, and insightful visualizations.

Additionally, we used Neovis.js for Figure 6 and Figure 7. Neovis.js is based on vis.js and is maintained by the neo4j community. We were running short on time and we could not agree on whether we should add the html code running Neovis to our React frontend.

1.d Jupyter notebook

For this assignment we used a Jupyter Notebook to handle some of the more complicated tasks associated with processing data for neo4j. These tasks can be divided into two categories. The first was to work with the original TSV file to create both positive and negative, weighted CSV files based on link sentiment. Doing this made working with and visualizing data much easier. A source and target subreddit with 17 connections simply became 1 connection with a weight of 17 stored in the link between the two.

The second reason for using the Notebook was to work with IGraph so that we could create graphs based on PageRank, clustering, and betweenness centrality. IGraph is a little difficult to install and only ran in Jupyter for whatever reason. Ultimately we used the algorithms in IGraph to add information to our negative link sentiment weighted graph. Once this processing was done, Neovis.js was used to create the graph that can be seen in Figure 6. We also ran JGraph in Jupyter which can be used to create really cool 3d graph models that you can rotate around. We have not figured out how to add labels to this visualization yet.

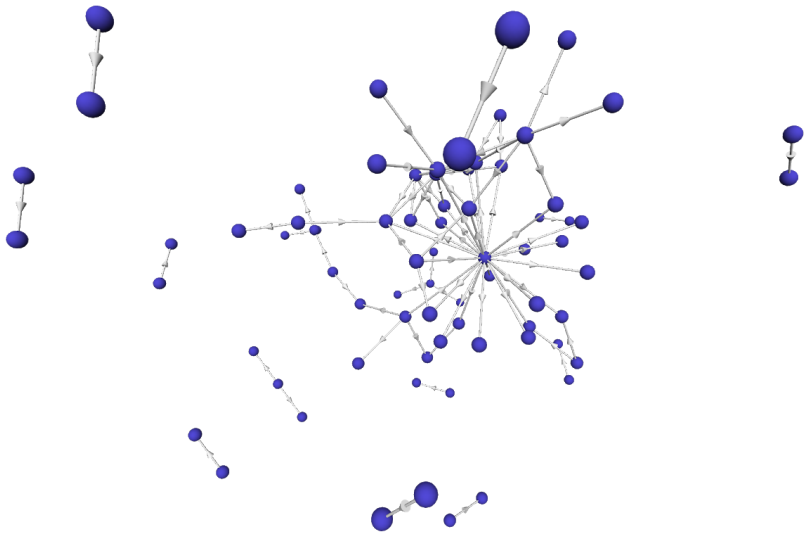

Figure 2: JGraph using the negative link sentiment weighted graph where link weights are less than -40.

2. Data modelling

2.a Graph databases vs relational databases

Neo4J is an enriched database that is sophistically designed to help storing different types of graph data. There are many reasons why we should switch from relational database to Neo4J. First of all, the Neo4j graph database contains various famous graph algorithms

such as centrality algorithms and community detection algorithms which can be easily accessed by us to use on our reddit data. Before we use each algorithm, we only need to study the idea and logic behind them instead of worrying about the correctness of the algorithms and the time to implement them ourselves. Second reason why we switched is the built-in data visualization it has for every cypher queries we wrote, this gave us huge advantage when we design a query, we can preview the correctness and patterns behind it before we move forward and commit in building better visualization in our frontend framework. Finally, unlike the relational database, Neo4j has the advantage to better show the connection between nodes(subreddits), the idea to individually store the data for edges in Neo4J has given us great advantage in making the connection and writing queries. Because of all of the reasons listed above, Neo4J has helped us resolve the connection issue which could easily be the potential problem with traditional relational database.

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2,500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Figure 3: From the book 'Neo4j in Action' [5]. A social network with ~ 1,000,000 people each with ~50 friends was used to find friends-of-friends connections up to 5 degrees.

Figure 4: From the neo4j website. Complex joins are much easier to visualize and carry out in Cypher vs. SQL.

To model our data in neo4j we used this basic format: (s:Subreddit)-[l:LINK]->(r:Subreddit) representing a source to target connection. In each node we store the id of the node which is the name of the Subreddit. With neo4j we can also store information in the links themselves. For our first database, LINKs contain the link_sentiment (+1 or -1), and the Timestamp. With our negative weighted database we collapsed the connections and with less nodes we stored the negative weights of

connections in a LINK as weight. Later on, via IPython we added the PageRank score and a Community Id to each LINK as well.

3. Scalability handling, Cost measure, and Algorithms

3.a Cost measure

Since we are using Neo4J which is a heavily designed graph database for any network. We would understand that one of the trade-off between Neo4J and other relational database to be the larger disk space that Neo4J requests. Since it needs more space to compute and store the relationship between each nodes while maintaining a higher performance than the relational database, it is understandable that it required more space and memory.

The performance of the Neo4J database has been exceptional. From the articles we read online showed that the performance of it to be a lot faster, and it has been very responsive to our queries given the medium graph size of our data.

3.b Scalability handling

Given that our dataset contains over 50,000 nodes and 0.8 million of edges, it is nearly impossible we could draw insights from the visualization of full size of the graph. Hence, we have been using several graph algorithms to help us reduce the size of the graph while not losing too much of the structure and dynamics of the data. We will talk about the algorithms in the section 3.b.1, 3.b.2, and 3.b.3.

3.b.1 Eigenvector centrality algorithm

Centrality algorithms have been widely adopted for sorting nodes in the graph data model. Eigenvector centrality calculates the eigenvector centrality score for each node to determine their ranks. The score is calculated as relative scores assigned to every node in the graph based on the concept that the connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. This measures the influence-factor each node in a graph. This algorithm has been used for visualising adjacency matrix, node-link diagrams, and edge-bundling diagrams, and by observing and comparing visualizations with various levels of resolution, we can justify and conclude that this algorithm is very efficient to show different levels of granularity of the graph without letting us lose the ability of drawing insights while maintaining the high connectivity of the data.

--- Eigenvector Centrality ---

	subreddit	**score**
0	askreddit	126.614243
1	pics	88.071382
2	worldnews	84.883255
3	videos	84.249410
4	funny	80.170607

3.b.2 Other centrality algorithms

These results were run on the negative weighted graph. The total number of nodes is 9136 which is far less than the54075 in the original TSV file we that were given. The average number of links per node is 4.705807 with a stddev of 30.043381. With all of these algorithms, ‘subredditdrama’ places first, ‘bestof’ places second and ‘the_donald’ only sneaks into the top 5 for third place in betweenness centrality.

Betweenness centrality refers to the number of shortest paths between two other members in the network that a given node appears in. Degree centrality is simply the number of connections that a node has in a graph. Finally, weighted degree centrality is like degree centrality but it takes into account the actual weights of the nodes. The weighted degrees are negative because these algorithms were run on the negative weighted graph in neo4j.

--- Total # of Nodes ---

	count
0	9136

--- Avg. Interactions ---

	min	max	avg_interactions	stdev
0	1	1557	4.705807	30.043381

--- Degree Centrality Top 5 ---

	node	degree
0	subredditdrama	1791
1	bestof	1178
2	askreddit	814
3	drama	760
4	funny	443

--- Betweenness Centrality Top 5 ---

	user	centrality
0	subredditdrama	7.582001e+06
1	bestof	3.943216e+06
2	the_donald	2.636204e+06
3	conspiracy	1.678288e+06
4	news	1.590740e+06

--- Weighted Degree Centrality Top 5 ---

	node	weightedDegree
0	subredditdrama	-8178
1	bestof	-3637
2	askreddit	-2474
3	drama	-2367
4	circlebroke2	-1684

3.b.3 Using Timestamps with Eigenvalue Centrality

--- Top 5 Positive and Negative By Year ---

	year	sentiment	top_5
0	2014	-1	[askreddit, adviceanimals, iama, todayilearned...

1	2014	1	[iama, askreddit, pics, videos, funny]
2	2015	-1	[askreddit, worldnews, videos, news, pics]
3	2015	1	[iama, askreddit, videos, pics, funny]
4	2016	-1	[askreddit, the_donald, worldnews, politics, n...
5	2016	1	[iama, askreddit, videos, pics, funny]
6	2017	-1	[askreddit, the_donald, politics, worldnews, n...
7	2017	1	[askreddit, iama, the_donald, videos, pics]

3.b.4 Weighted PageRank

Weighted PageRank is an extension to the regular page rank algorithm introduced by Google. Weighted page rank accounts for the importance of both inlinks and outlinks and distributes rank scores based on popularity. It has shown to be more effective than regular page rank (4). Regular page rank scores nodes evenly over their links while weighted page rank assigns larger rank values to more important nodes. It can be used alongside Eigenvalue Centrality and the nodes occupying the top 10 in both are very similar. We install the optional neo4j graph algorithms automatically in Docker to make it easier for other group members to work with the codebase.

```
CALL algo.pageRank.stream(
        // Node statement
        'MATCH (s:Subreddit) RETURN id(s) as id',
        // Relationship statement
        'MATCH (s:Subreddit)-[:LINK]->(t:Subreddit)
        RETURN id(s) as source, id(t) as target, count(*) as weight',
        {graph:'cypher',weightProperty:'weight'})
        YIELD nodeId,score
        RETURN algo.getNodeById(nodeId).id as subreddit, score
        ORDER BY score DESC LIMIT 10
```

--- Weighted PageRank Top 5 ---

	subreddit	**score**
0	askreddit	75.945676
1	pics	38.968152
2	funny	38.803899
3	videos	29.082333
4	worldnews	28.250103

3.b.5 Label propagation algorithm

Label propagation is a semi-supervised machine learning algorithm that clusters nodes in a graph to form different communities of nodes that show similar behaviours in the graph to some degrees. We include this algorithm in our edge-bundling diagrams and adjacency matrix for two main reasons. The first one is that it is one of the three clustering algorithms Neo4J provides, and by examining all three algorithms, it has the best result that included communities of subreddit that close to human recognition. For example, this algorithm grouped xboxone, redditgaming, leagueoflegends etc together, and we agreed that

it is the best algorithm among the three. The second reason is that during the time we use it to visualize the graph, we found out because it is semi-supervised machine-learning, it requires less to no parameters in the beginning. It helped us save big amount of time so that we can work on searching for more interesting aspects of the data. However, this algorithm seems to be efficient for large dataset, while we size down our dataset to 50 or 100 nodes, this algorithm seems to fail in clustering, as shown below, it creates many islands instead of communities. And because of the lack of experience with clustering algorithms and time limit, we suggest that in the future better clustering algorithm or even manually grouping can be used for small dataset.

--- Label Propagation Top 5 Communities ---

	label	size	subreddits
0	2108	3243	[18bfriendzonest, 1984isreality, 2016_election...
1	1802	1062	[1509, 16eralert, 1990boys, 2007scape, 2007sca...
2	6228	320	[195, 19thworldproblems, 40klore, 90210worldpr...
3	154	69	[asoiaf, atheistlegioncoc, cakeday, canadianbu...
4	6235	65	[357, adamcarolla, adamcarollacirclejerk, adel...

4. Visualization graphs and Insights

4.a Node-Link Diagram

One of the most used methods to visualize graph data is Node-link diagram. This method is widely used when you don't have many edges in your screen. The reason behind this is that having too many edges in the screen makes the visualization unreadable for the user (in contrast with Gestalt laws).

D3.js uses force directed algorithm [2] to depict the graph. Figure 5 and Figure 6 show samples of our visualizations with node-link diagram. Figure 5 uses D3.js library which uses force directed but Figure 6 uses neovis.js.

With these visualizations we understand the topological aspects of the data and the connectivity in each group that we want to focus on. The weak point of this visualization is when you increase the number of edges and nodes. It turns into cluttered image.

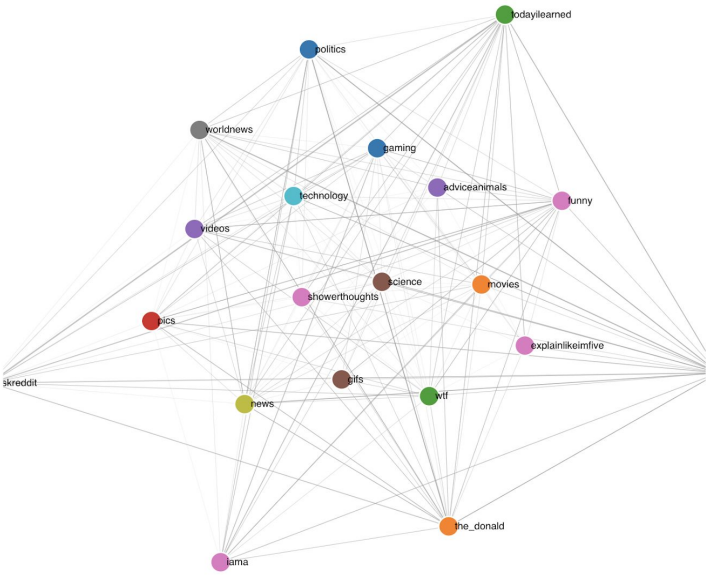

Figure 5: Correlation between top 20 subreddits

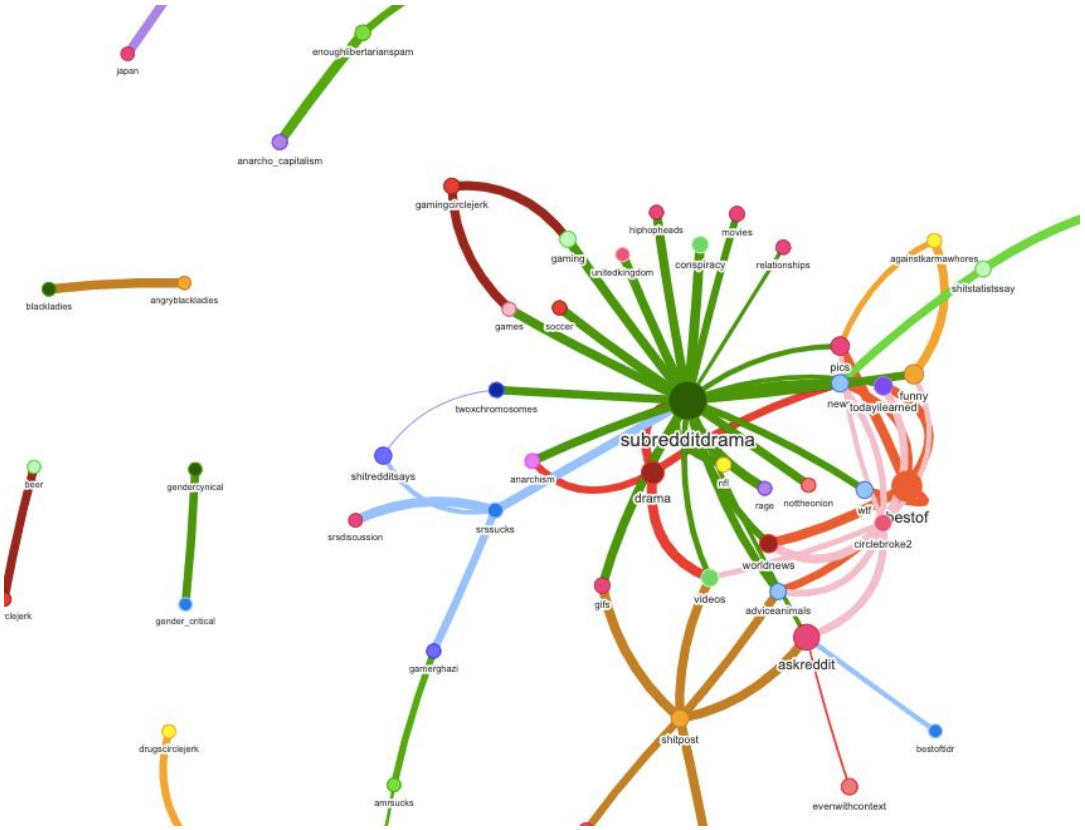

Figure 6: Negative link sentiment weighted graph made with IGraph, neo4j, and Neovis.js. Node size is based on PageRank score. Link size is weighted negative link sentiment and color is created using the IGraph community_walktrap detection algorithm. This query returned all links in the network with weights of less than -20: MATCH (s:Subreddit)-[r:LINK]->(m) WHERE r.weight < -20 RETURN *

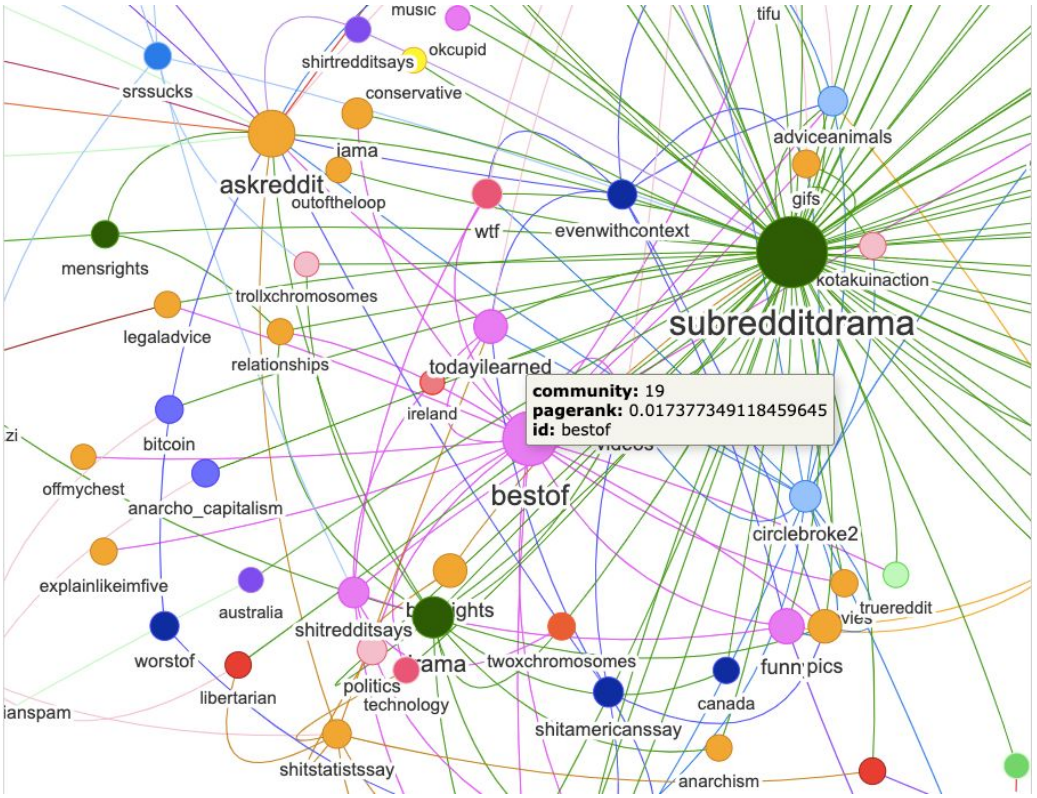

Figure 7: This is the same graph as the one in figure 6 except that weights are < -10 and the node link strength that is displayed is reduced for better visibility between source and target.

4.b Edge-Bundled Graph

Edge bundled graphs are widely used when it comes to graph data. The most important difference when the data accumulates with a certain ratio, edge-bundled visualization depicts more understandable images compared to node link(this article demonstrates why).

Demonstrated multiple edge-bundled graphs in our visualization to compare it with node-link. As you see the depictions you can find interesting patterns in this visualization more easily than node-link. So overall we can say that using edge-bundled graph will improve readability of the graph [3]. Figure 8 and Figure 9 will demonstrate how represented our data using this method. Figure 8 has 50 nodes and Figure 9 has 100 nodes. These visualizations would be more meaningful when interacting with them.

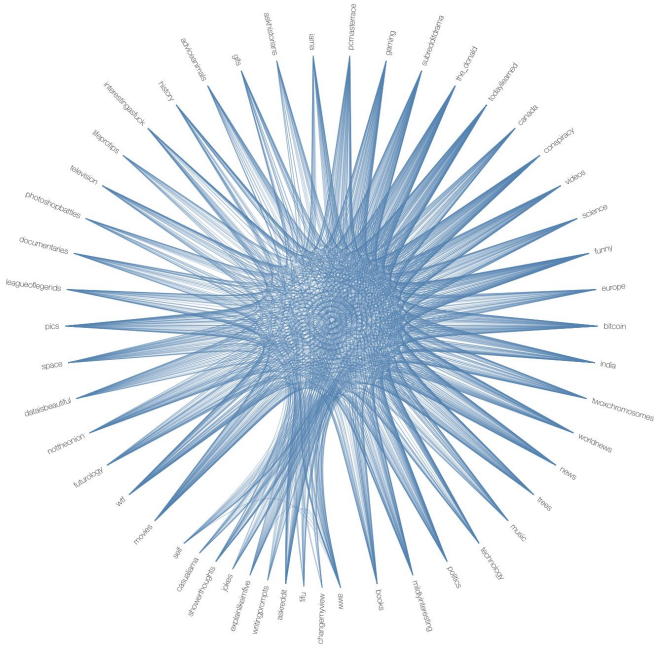

Figure 8: Edge-bundled graph with 50 nodes. This is an interactive visualization which you can see outgoing and ingoing edges by hovering on nodes.

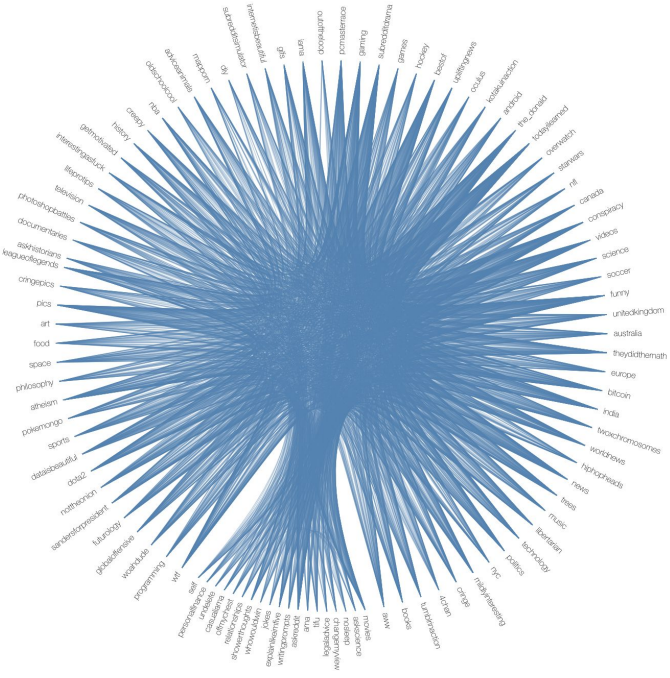

Figure 9: Edge-bundled graph with 100 nodes. This is an interactive visualization which you can see outgoing and ingoing edges by hovering on nodes.

4.c Chord Diagram - 2016 US election

We used a chord diagram to show relationships between Subreddits that had the most activity surrounding topics relating to the 2016 US election. We removed ambiguous Subreddits like ‘askreddit’ from the results. We started with ‘the_donald’ representing Donald Trump and looked for the Subreddits showing the most total activity (negative and positive interactions). From these results we created an adjacency matrix. The results are interesting as the top 3 nodes are ‘the_donald’ (pink), ‘conspiracy’ (red) and ‘hillaryforprison’ (black). This diagram is shown in Figure 10.

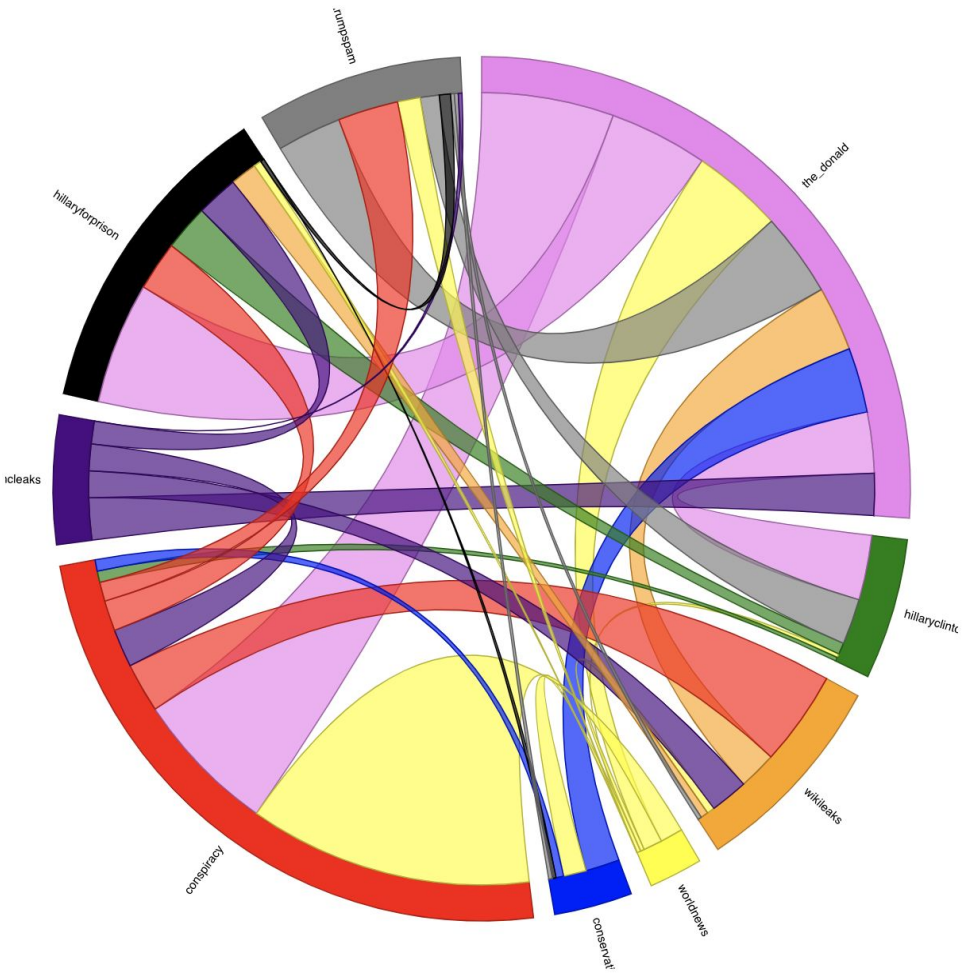

Figure 10: Total activity based on topics important to the 2016 US election. ‘the_donald’ (pink), ‘conspiracy’ (red) and ‘hillaryforprison’ (black), ‘hillaryclinton’ (green), ‘wikileaks’ (orange), ‘worldnews’ (yellow), ‘conservative’ (blue), ‘dncleaks’ (purple) and ‘trumpspam’ (grey).

4.d Adjacency Matrix

Utilizing an adjacency matrix is a well known method to visualize graph data. We used the methods mentioned Landesberger’s paper [3]. We used different visual variables to represent data (color for grouping, opacity for edge weight). We managed to implement three different representations for matrix visualization(by cluster, frequency, and the name of the nodes). Figure 11 shows a sample of our matrix, showing edges near each other with the same group.

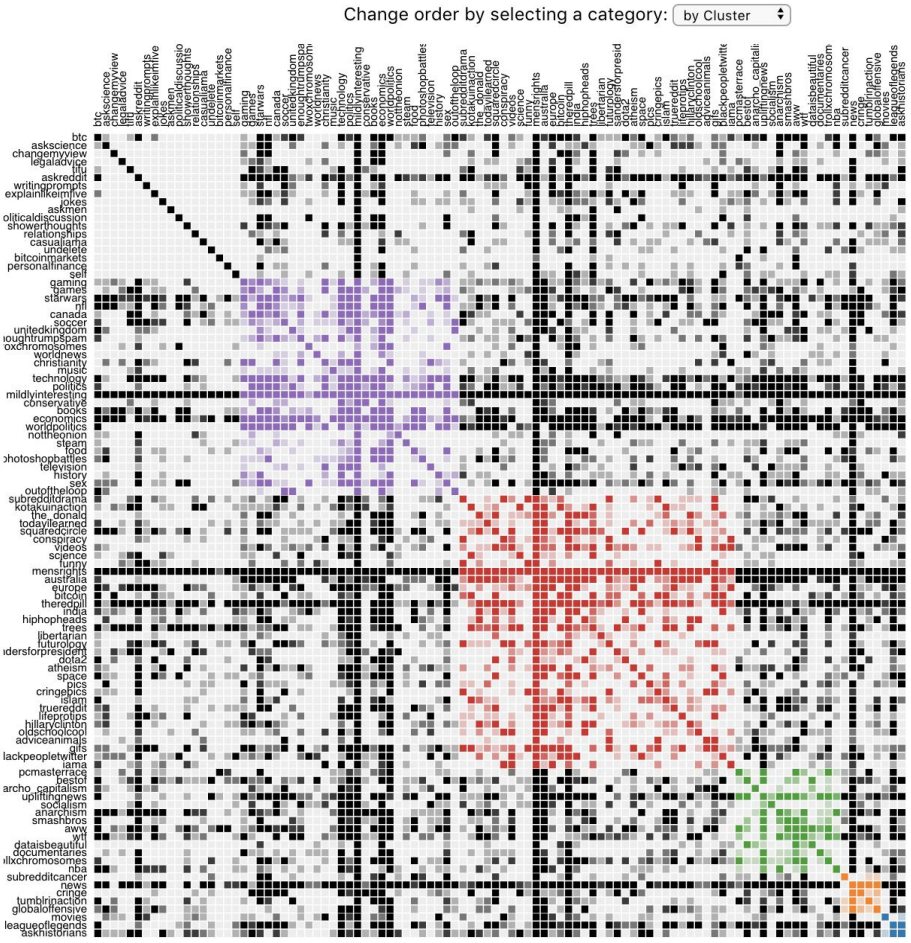

Figure 11: Adjacency matrix for 100 subreddits. Different colors show different groups and opacity shows different edge weights. This is an interactive visualization which you can change the order of nodes by name, frequency(edge weights), and clusters.

References

[1] Keim DA. Information visualization and visual data mining. IEEE transactions on Visualization and Computer Graphics. 2002 Aug 7;8(1):1-8.

[2] Fruchterman TM, Reingold EM. Graph drawing by force-directed placement. Software: Practice and experience. 1991 Nov;21(11):1129-64.

[3] Von Landesberger T, Kuijper A, Schreck T, Kohlhammer J, van Wijk JJ, Fekete JD, Fellner DW. Visual analysis of large graphs: state-of-the-art and future research challenges. InComputer graphics forum 2011 Sep (Vol. 30, No. 6, pp. 1719-1749). Oxford, UK: Blackwell Publishing Ltd.

[4] Xing, Wenpu, and Ali Ghorbani. "Weighted pagerank algorithm." *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.*. IEEE, 2004.

[5] Vukotic, Aleksa, et al. *Neo4j in action*. Manning Publications Co., 2014.