A Flexible Social Network Visualization

George Crowson
Earlham College
801 National Road West
Richmond, Indiana
ghcrows13@earlham.edu

ABSTRACT
Theres a massive amount of data being generated by individuals interacting with social networks. Mining this data and generating a useful visualization of that data is a difficult task that academic areas such as sociology, anthropology, and psychology often face. Additionally, social networks play an important role in many peoples lives. A flesible social network visualization can help users answer questions about their social networks or generalized social networks.

Categories and Subject Descriptors
E.1 [**Data**]: Data Structures; F.2.1 [**Analysis**]: Numerical Algorithms and Problems; H.2 [**Information Systems**]: Database Management; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; I.3.8 [**Computer Graphics**]: Applications; I.5.3 [**Pattern Recognition**]: Clustering

Keywords
social network visualization, Twitter, Tumblr, Python, TwitterAPI, PyTumblr, data aggregation, PSQL, mesh networks, web applications, HTML, CSS, Javascript, D3

1. INTRODUCTION
There are a variety of tasks that must be tackled in order to create a social network visualization. I collect data - users and posts - from Twitter and Tumblr using Python libraries to interface with their respective APIs. Data is coming from multiple sources and so it must be aggregated into a standardized schema. I store the data in PSQL tables that are normalized and indexed. I visualize the data as a mesh network implemented using HTML/CSS/JS and the D3 visualization library. The color/size of nodes can be customized by category such as gender, age, and location. The visualization features dynamic abstraction in that users are abstracted into groups to facilitate the exploration of the dataset. This abstraction is controlled by zooming in/out of the visualization. This project completes all of theses tasks in an effort to provide a reusable tool.

2. DATA COLLECTION
Many of the social network visualizations I've found have focused on egocentric visualizations of a users email. I believe this is because email is standardized and easy to collect. However, these visualizations are egocentric in nature because they focus on the data of a single user. These datasets are focused on answering questions about a users personal social network rather than answering questions about the users of social networks in general. In contrast Ive attempted to create a visualization that can handle large datasets so that more generalized questions can be asked about the users of social networks.

2.1 Scraping
Not all social network data is easily accessible through APIs. In these circumstances data must be manually scraped. For example, forums and discussion boards must be scraped as they rarely have native APIs. A downside of scraping is that it can lead to fragile programs that have the risk of breaking when the social network is updated. A concern of scraping social networks is that they pay to serve you data and therefore must be respected. Data scraping on a large scale can can result in behavior that is very similar to DoS and/or DDoS attacks. Obviously organizations dont like to have their servers flooded with requests. Consequently, data scraping should make steady requests, rather than a flood of requests, so as to respect the capacity of the organization's servers. In general, though, its better to use APIs as the organization is explicitly defining the interface that they deem reasonable.

2.2 APIs
I collected data from Twitter and Tumblr using an egocentric algorithm. My algorithm begins with a list of users, gets the friends of those users, and repeats the process until a given number of users or a given depth is reached. This is similar to exploring a tree using a depth-first algorithm. The number of users/leaves at a given depth increases exponentially. Users are only explored once, and so if a user is rediscovered theyre not explored. The posts of users are collected throughout this process so that they can be surfaced in the visualization. Similarly, rather than exploring users by looking at their friends its possible to explore users through looking at whom interacts with their posts. In the

future I would like to use post interactions to make an inference about the strength of connections between users.

2.3 Twitter

Twitters API was easy to work with through the TwitterAPI library for Python. The TwitterAPI library relies heavily on a single function, request(), that gets data of a particular type that meets the specifications of the query. This data is represented as a series of JSON dictionaries. Twitters API limits how many results are returned at once which requires the use of pagination to collect all users and/or tweets. The most inconvenient part about Twitters API is that it throttles requests. This makes it more difficult to collect large amounts of data from Twitter. In order to bypass this issue I run my query, check if the query failed due to throttling, wait until the throttle resets, and repeat the query. Twitters API and the TwitterAPI python library are effective tools for collecting data from Twitter.

2.4 LinkedIn

I originally wanted to include LinkedIn but was unable to implement their API in a reasonable amount of time. This led to LinkedIn being cut from the project. In retrospect, LinkedIn was the oddball of the three services. It offers social network features, such as friends and posts, but is substantially different than Twitter and Tumblr in terms of audience. I chose to replace LinkedIn with Tumblr.

2.5 Tumblr

Tumblrs API was easy to work with through the PyTumblr library for python. Tumblrs API is very similar to Twitters API in terms of implementation. However, PyTumblr has a different philosophy than TwitterAPI because it uses different functions to query for different types of data. For example, info() returns the authenticated user while blog_info() can get any user. Tumblr is somewhat restricted in that a users following/followers arent public information. I worked around this restriction by keeping track of whom interacts with a given users posts and I treat those interactions as a sign of friendship. This isnt optimal as post interactions are a less significant indication of relationship than following/follower. Additionally, Tumblrs API restricts the number of post interactions returned to 50 and doesnt support pagination for post interactions. Tumblrs API and the PyTumblr library were easy to work with but were more restricted than Twitters API in my experience.

3. DATA AGGREGATION

No single archive or tool captures all our social relations with others. [3] Focusing on a single social network limits the kinds of questions that can be asked about users. All social networks are built to meet different needs. Similarly, users treat social networks differently and express themselves differently on different social networks. Including multiple social networks in a visualization provides the ability to compare and contrast different social networks.

3.1 Social Network Similarities

Ive found that most features of social networks are analogous and can be translated to a standardized schema. For example, users and posts have similar functionality across different social networks. Users have a name, friends, and posts. Posts have an author, content, replies, likes, and shares. Due to the similarities between different social networks Ive used a standardized schema to make data from different social networks directly comparable.

3.2 Exceptions

However, all social networks are different, and when these differences dont fit into the standardized schema they become exceptions. Dealing with exceptions requires deciding on the algorithm to be used for combining data from various sources. [3] For example, Facebook pages/groups arent features that exist on Twitter or Tumblr. I could resolve this exception by treating Facebook pages as users and treating Facebook groups as just another form of friendship. However, forcing these exceptions to fit with the standardized schema makes the data more general and subsequently harder to understand in visualizations. A better solution for Facebook groups might be to let users be in groups and only have that feature apply to Facebook users. Exceptions are difficult to deal with appropriately given that they can be ignored, translated to fit into the standardized schema, or added to the standardized schema.

3.3 IBM's SONAR API

A good example of aggregating data from multiple social networks is IBMs SONAR API. The SONAR API uses IBMs internal social networks to generate friends lists based on how often users interact. The SONAR API enables users to determine the weight of different social networks so that their friends list can more accurately reflect their expectations. Similarly the aggregation algorithm can learn by asking the people in the network about its correctness. [3] These are a good ways to give users control over an otherwise abstract algorithm. However, giving the user too much control can lead to the user feeling overwhelmed by choice. I believe that users should have a minimal set of options that are maximally helpful in meeting their needs. I believe that IBMs SONAR API is lightweight while still enabling users the power to customize data aggregation.

4. PERFORMANCE

4.1 Parallelization

Collecting and processing data from social networks is a highly parallelizable task. One solution, described in "Parallel crawling for online social networks", is to use a centralized queue that delegates tasks to subthreads. [2] When a subthread is finished processing a user it queries the centralized queue for the next user. Its possible to increase performance and reliability by distributing the crawling of web pages across several agent machines. This solution is for scraping the pages of forums. However, its important to note that APIs that implement throttles, such as Twitters API, dont benefit from performance gains. The only way to gain performance when an API has throttles is by authenticating subthreads as different users which is against most APIs terms of service. Tumblrs API, however, has no throttles and benefits greatly from threading. Threading provides large performance gains in situations where the API isnt throttled.

4.2 API Queries

I view API queries as a finite resource and so I tried to make the most out of them. This involved storing and reusing as much data as possible. All of my queries request the maximum amount of records that the API supports. This is important because APIs keep track of how many queries youve ran: not the size or complexity of those queries. Similarly, it makes sense to reuse data from previous queries as it avoids network latency and redundant queries. For example, when I need data about a given user I check to see if Ive already collected data about that user. If so, I reuse the data. If not, I run a query to get the data. Queries are an important resource because waiting for API throttles to reset is an incredible waste of time.

4.3 Databases

When dealing with large amounts of data the storage and retrieval of that data becomes crucial. The structure of social networks lends itself to normalization and indexing. I normalized my database schema (figure 1) which helps PSQL access only the data relevant to a given query. I found indexing ID fields to lead to large performance gains because a subset of fields receive the majority of access. For example, the average query would be sped up considerably if users were ordered by their number of friends and how often their content is viewed or interacted with.

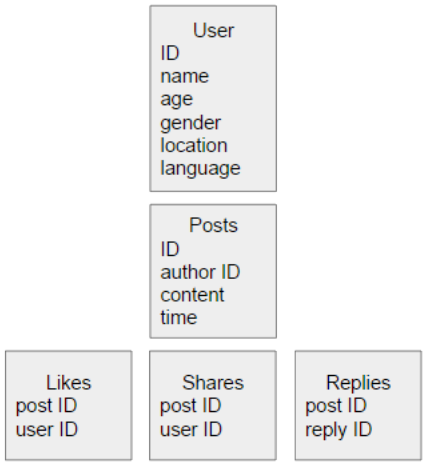

Figure 1: My database schema.

5. VISUALIZATION

5.1 Mesh Networks

Ive used a mesh network for my social network visualization as its the clearest way to show connections between users and groups. Indeed, most social network visualizations use mesh networks. I represent users as nodes and connections between users as edges between nodes. Nodes are rendered with the users name or the users profile picture. (figure 2) Nodes are colored based on the selected category. The color of the users node is determined by their value for that category. More concretely, if gender is selected a nodes color will be blue for males, purple for trans, and pink for females. (figure 3) In the future I would like to implement filters that enable nodes to be hidden dynamically based on the values of categories. Mesh networks have afforded me a lot of freedom as node appearance and positioning are flexible ways to visualize data.

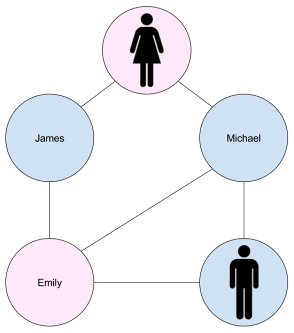

Figure 2: An example of a mesh network representing a social network.

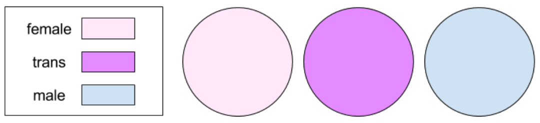

Figure 3: An example of nodes being colored by gender.

5.2 Pie Charts

A particularly flexible thing about mesh networks is that nodes can be represented as a pie chart. I use this to visualize the composition of users in a group node. (figure 4) The pie chart's data can be tied to a selected category. The composition of groups is likely to convey some form of information about the group. For example, perhaps one group is predominantly male, another group is predominantly female, and another group is a 50-50 mix. This says a lot about the dynamics of the groups. An alternative could be a weighted bar chart beneath the node, but this doubles the number of objects on screen, and I believe that this is very harmful to the legibility of the visualization. Consequently, it seems that pie charts are an effective way to convey percentage-based information about the composition of groups.

5.3 Animation

Social network visualizations can be animated, and one example of this is PostHistory which animates user interactions throughout time. [5] In the future I would like to implement some animations. A users node could temporarily expand in size when they post a status. The connection between users could be highlighted when a reply occurs. Groups could be animated to represent the fluidity of the users that participate in those groups. Similarly, the width of edges between users could be animated to reflect the strength of the connection between users over time. Viégas mentions how animation across time helped people interpret the significance of the underlying data. [5]

5.4 Relationships and Interactions

Its possible for social network visualizations to focus purely on relationships (Vizster [4] and SONAR [3]) or to include interactions from those relationships (Fizz [1]). In my visualization hovering over a user will cause all of that users interactions will be displayed next to those users and are

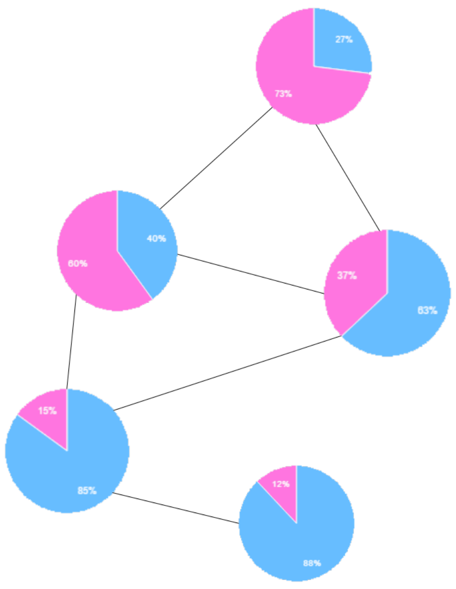

Figure 4: Group nodes can be replaced by pie charts that represent their compositions.

formatted as a timeline. These interactions are useful for providing context about specific users and their interactions. I believe that without interactions users would interpret the visualization as being much more abstract.

6. REDUCING VISUAL COMPLEXITY

6.1 Group Abstraction

At first my visualization was overwhelming due to the size of the dataset and the resulting density of nodes/edges. 5 Visualizing such large datasets was only manageable with an equally large amount of abstraction. This abstraction involved replacing a lot of individual nodes with a single group node, replacing a lot of group nodes with even larger group nodes, and repeating the process until there were a few dozen nodes. 6 For me it seemed natural to make these abstractions based on social groups: clusters of people that are densely connected. The users are evaluated for the strengths of their connections and are assigned groups for six different layers of abstraction. This metadata is generated by the server because its too intensive to consider running on the client. Dynamic abstraction has been the only way for me to visualize large datasets.

6.2 Group Fragmentation

A caveat of using group abstraction is that a given node might belong to two or more groups. In these circumstances a new group can be created between the existing groups to represent the nodes whom fall into this fragmented categorization. Edges are drawn between this composite node and the originating groups to indicate the connection.

6.3 Zoom Control

Viégas brings up adaptive zooming which describes the adjustment of a map, its contents and the symbolization to target scale in consequence of a zooming operation. [5] This

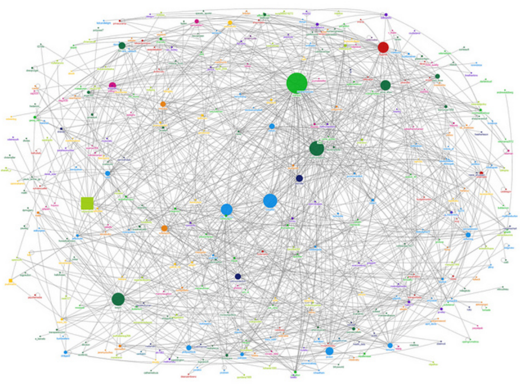

Figure 5: Visualizing too many nodes or edges at once makes visualizations difficult to interpret.

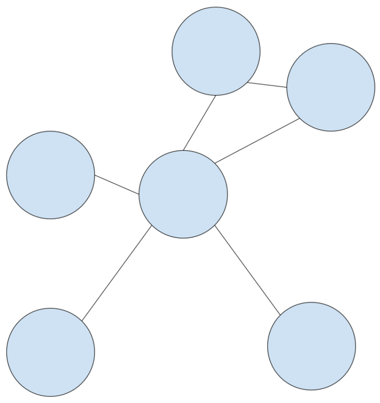

Figure 6: Group nodes can be used to represent the busy mesh network in figure 5 more legibly.

means that the visualization dynamically responds to the act of zooming by changing the degree of abstraction for nodes thatre currently in view. This helps prevent situations where clusters are so dense and so tight that nodes end up almost completely on top of each other and names of people ... become illegible. [5] Similarly, when zoomed out enough edges between nodes become incomprehensible and should be hidden. I use zooming as an intuitive way for users to control what level of group abstraction is used for the visualization.

7. CONCLUSIONS

Upon reflection I believe this project to be the beginnings of a very powerful tool. It enables programmers to easily collect and process data from a host of social networks. Its relatively easy to mine data for different types of metadata and to then add that metadata to the data model. Its robust to change in that adding social networks and/or fields wont change much about the implementation of the visualization. Its possible to extend the visualization to have feature parity with social networks so that users can check their social media, live, from within the visualization. Along similar lines, the visualization can be brought to life by animating posts and user interactions across time. With enough work and advertising I believe that this visualization is capable of having a large number of users.

8. ACKNOWLEDGMENTS

I would like to thank Veronica Dow for her significant help and support. Additionally, my conversations with Mic Jackson helped me develop this project and my related thoughts. Finally, I'm grateful for Charlie Peck as he was a guiding hand throughout the duration of this project.

9. REFERENCES

[1] Bloom. Fizz. 2011.

[2] e. a. Chau, Duen Horng. Parallel crawling for online social networks. *Proceedings of the 16th international conference on World Wide Web.*, 2007.

[3] e. a. Guy, Ido. Harvesting with sonar: the value of aggregating social network information. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.*, 2008.

[4] J. Heer and D. Boyd. Vizster: Visualizing online social networks. *Information Visualization*, 2005.

[5] F. B. Viégas and J. Donath. Social network visualization: Can we go beyond the graph. 4, 2004.