Social Coding: Evaluating Github's Network using Weighted Community Detection

CS224W Final Project Report

Joseph Marrama
jmarrama@stanford.edu

Tiffany Low
tlow@stanford.edu

***Abstract*— Relationships amongst users of collaboration networks can be characterized by the extent and quality of their connections. We improve upon the one-dimensional analysis of unweighted community detection by modifying Girvan Newman's algorithm for weighted networks and introduce weighted variants for measures of betweenness, conductance and modularity. We demonstrate the improvement of weighted community detection on Github's social graph.**

I. INTRODUCTION

Social networks are the zeitgeist of the 21st century. We possess the technology to stay connected continuously with our circle, and the resources to reach out to people from all around the world in an instant. The web has allowed communities to emerge and collaborate on challenges like building the largest compendium of all human knowledge, Wikipedia, providing the infrastructure for communities to gather around a common goal.

Github is a platform that promotes social computing in a software development context that has grown exponentially since its inception. GitHub's official motto is 'Social Coding', and we are interested to know to what extent this goal has been achieved. In this paper, we evaluate the breadth and depth of teamwork across GitHub's community using community detection methods.

Community detection gives a measure of the relative connectivity between nodes in a given network. When applied to social networks these methods yield invaluable information about the quality and extent of relationships between entities. Many standard community detection algorithms discard edge weight information. In doing so, they ignore the contribution of the strength of ties between nodes in determining community boundaries. We propose an algorithm for community detection using weighted networks and demonstrate that the communities found have network characteristics that correlate better with notions of collaboration and community.

II. RELATED WORK

In an age of social media, source control management is evolving to take on a more collaborative flair. Communities like GitHub, SourceForge, BitBucket and Redmine are all examples of online communities centered around collaborative coding. These systems typically include expected features such as version control, branching, and merging, and also incorporate social media features such as commenting, followers and groups.

Community detection approaches to social media networks have been conducted for services such as Facebook [1], Twitter [2] and Wikipedia [3]. In comparison, less attention has been paid to the similar problem of collaboration in open source development communities.

We first examined Newman and Girvan[4]'s approach to discovering communities in a social network. Although we found their methodology sound, their model lacked sophistication and left out important considerations about the quality of interaction between users [4]. The work by Brandes et al. [3] presented a powerful model of collaborative structures within Wikipedia and is the motivation for our research on the GitHub community [3]. The emphasize the use of a collaboration score between users to quantify the extent of user relationships, weighting positive and negative interactions. Finally, the paper by Jin et al. [5] attempted to combine the work of a *weighted* user interaction model with a community detection algorithm. Although their algorithm requires careful parameter tuning, their modeling approach used weighted edges in determining both modularity and betweenness scores.

III. DATASET

A. Data Collection

We considered the space of public commit events on GitHub between the months of July 2012 and October 2012. Commit events are generated when users push changes to a repository's files. Because users must receive approval from the repository's owner or collaborators if they do not possess push/pull access to the repository, commit events can be construed as signs of approval (either implicit or explicit) from the repository's community. This makes such events more significant as an indicator of collaboration between users. GitHub offers these datasets free for download at www.githubarchive.org and also via Google's BigQuery.

Our final dataset contains a total of 5428071 commit events across 262609 users and 623294 repositories. On average, each repository has 8.7 commits and each user contributes 20.67 commits. From the raw commit data we first constructed a heterogeneous commit graph consisting of both users and repositories as nodes. Each commit from a user u to repository r contributes an edge between the nodes u and r with weight 1. Using this, we then construct an undirected weighted collaboration graph, where users are nodes and edges between users are weighted with the collab-

oration score of each user. We run our weighted community detection and all other algorithms on the collaboration graph.

B. Pre-processing

To build the commit graph, we first had to convert the raw downloaded json from disjoint json events to a valid json list. We used a few simple `sed` commands to accomplish this. Next, we had to parse the json, filter out all non-PushEvents, and build the edge list of the commit graph. We wrote a short program in `Go` to do this. Our `Go` program outputs the full edge list of the commit graph, along with mappings from unique numerical user node ids to GitHub usernames and unique numerical repository ids to GitHub repository urls. The user and repository mappings are invaluable in that they allow us to look at users and repositories of interest on GitHub to verify our findings.

C. The Collaboration Graph

Our next step was to build the collaboration graph, as described above. Building this graph is trivial using SNAP and our commit graph, and it provides us with a simple yet powerful graph for analyzing community structure.

To build the collaboration graph, we first had to devise a measure of collaboration between two users. We chose to formulate the collaboration score between two users as *the sum over all repositories both users have committed to of the minimum number of commits each user has pushed to that repository*. In formal terms: letting A_r denote the number of commits user A has pushed to repository r, $CollabScore(A, B) = \sum_r \min(A_r, B_r)$. We chose this measure because it is simple and it satisfies our basic requirements of a collaboration score: it is high when a pair of users both contribute a significant amount to the same set of repositories, and it is low if a pair of users both contribute to the same repository but one user contributes next to nothing.

D. Preliminary Analysis

After constructing the collaboration graph, our first intuition was to look at the distribution of sizes of strongly connected components (SCCs). Figure 1 shows a histogram of SCC sizes in the collaboration graph, which nearly mirrors a power law distribution. Fitting a power law to the histogram, we obtained an alpha value of 3.08. This is fairly large, reflecting the propensity of GitHub users to collaborate with a small subset of users, if any. Nearly 77 percent of all users did *not* collaborate with anyone else. Most users publish exclusively to personal repositories and don't grant collaboration access to other users.

IV. Methods

In the context of network theory, a community is defined as a grouping in a graph where in-group network connections are dense and inter-group connections are sparse [4]. In a GitHub centric definition, a community is a grouping of users that are more likely to collaborate together. We make the assumption that based on past collaborative behavior between users, it is more probable for them to collaborate again in the future. This model abstracts away relationship dynamics between users to construct a big picture of collaboration in the network.

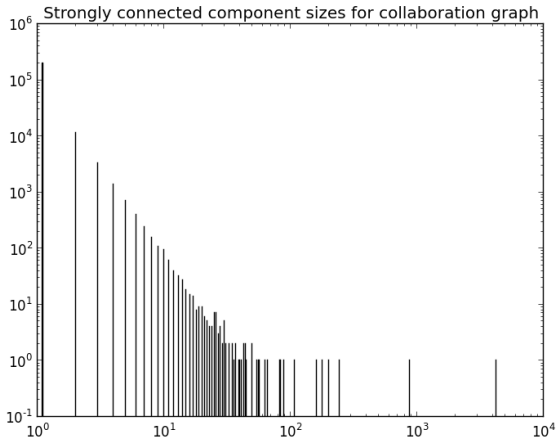

Fig. 1. Histogram showing size distribution of strongly connected components and communities

A. Divisive Clustering

Our main contribution is to propose a new weighted community detection algorithm that utilizes weighted betweenness and modularity. The algorithm is a typical divisive clustering algorithm, except that in each step we remove the edge with the maximum *weighted* betweenness, and we judge the optimal assignment of communities to be the one with the highest *weighted* modularity.

We apply Newman and Girman's divisive clustering algorithm [4] as a baseline for partitioning the collaboration graph into communities. At each step of the algorithm, we compute the betweenness of all edges and eliminate the edge with the highest betweenness value. As in all divisive clustering algorithms, we need to choose the optimal division of the graph to use as our community partition once we have removed all of the edges. Newman and Girvan[4] does this by choosing the split with the maximum *modularity*, a measure they devise to quantify how good a given community assignment is. Given each community assignment found while removing edges, they compute the modularity score of each assignment and choose the one that maximizes modularity as the ultimate community assignment.

In our proposed divisive clustering algorithm, we modify the betweenness and modularity measures to include the weight of edges. We propose two new measures, *weighted betweenness* and *weighted modularity* to use with the divisive clustering algorithm.

B. Weighted Betweenness

Our *weighted betweenness* measure is very similar to the formulation of shortest-path betweenness used in the related work on weighted networks[4] [6]. The general intuition is that shortest-path betweenness is proportional to the amount

of shortest paths that flow through a given edge. We can compute shortest path betweenness by doing the following for each node $s \in V$:

- Do a breadth-first search starting from node s, and keep track of the number of shortest paths from s to v, denoted $sigma_{sv}$, for each node $v \in V$ explored by BFS
- Working up from the leaves of the BFS tree constructed, compute $\delta_s(v,w) = \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \sum_x \delta_s(w,x))$, where edges $\{(w,x)\}$ are all of the edges connected to node w below w in the BFS tree (note: $\sum_x \delta_s(w,x) = 0$ when w is a leaf node).

The betweenness score for edge (v,w) is simply $\sum_{s \in V} \delta_s(v,w)$.

In order to incorporate edge-weights, we make one small adjustment to the above formulation of betweenness: we set $\delta_s(v,w) = \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \sum_x \delta_s(w,x))\frac{1}{\sqrt{W(v,w)}}$, where $W(v,w)$ is the edge weight of edge (v,w). This has the approximate effect of scaling back betweenness scores of edges by the inverse square root of their weight.

According to this scoring, edges adjacent to an edge with high weight will have a lower betweenness score as compared to those adjacent to an edge with low weight. In other words, comparing two similar subgraph structures in a network, we prioritize pruning relationships between users with weaker collaboration scores over those with stronger collaboration scores.

Studying the distribution of mutual commits between users, we found a high standard deviation. In addition, the quality of collaboration for a pair of users with 50 commits on shared repositories compared to a pair of users with 200 commits on shared repositories was questionable. Thus we chose to scale δ values by $1/\sqrt{W(v,x)}$ instead of other values because we wanted $W(v,x)$ to have a diminishing impact on δ as $W(v,x)$ scales upwards [1].

We chose to include this weight scalar in the recursive formulation of $\delta_s(v,w)$ instead of simply scaling normal shortest-path betweenness by it (that is, setting betweenness $B(v,w) = \frac{1}{\sqrt{W(v,w)}}\sum_{s \in V} \delta_s(v,w)$) because including the scalar in $\delta_s(v,w)$ helps capture our intuition that communities tend to form around pairs that have a high degree of collaboration.

C. Weighted Modularity

We can compute the *weighted modularity* of a given set of communities by constructing a kxk symmetric matrix E with the following property: $E_{ij} = (\sum_{(i,j)} W(i,j))/(\sum_{v,x} W(v,x))$, where edges $\{i,j\}$ are all edges that connect across cluster i and j, and edges $\{v,x\}$ are all edges in the graph. E_{ij} has a simple intuition: it is the ratio of edge weights that connect clusters i and j to total edge weight in the graph. Using this matrix, we define weighted modularity as $\sum_{i=1}^{k}(E_{ii} - A_i)$, where $A_j = \sum_{i=1}^{k} E_{ij}$. Intuitively, our formulation of weighted modularity should favor in-community high-weight edges and discourage between-community high-weight edges.

[1]We also tried other scalars, such as $1/(1 + \log W(v,x))$, which produced very similar results

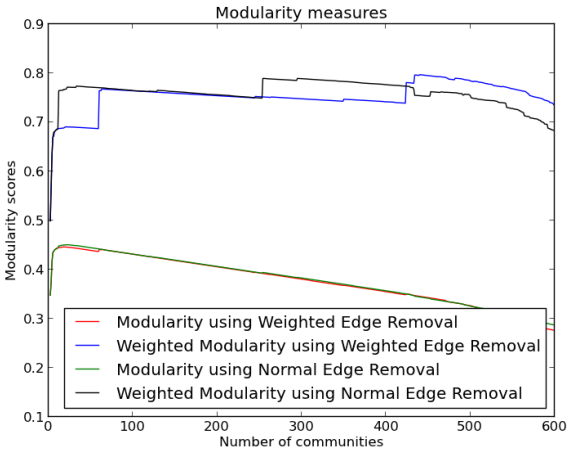

Fig. 2. Graph showing modularity and weighted modularity over communities produced by normal and weighted community detection on the next largest SCC

D. Weighted Conductance

Lastly, we define one new measure *weighted conductance*, to further evaluate the quality of found communities. Conductance, as defined by Leskovec et al. [7], is simply $\frac{c_S}{2m_S + c_S}$, where c_S is the number of edges connecting community S to other communities, and m_S is the number of edges inside community S. We define weighted conductance in a nearly identical manner as $\frac{W(c_S)}{2W(m_S)+W(c_S)}$, where $W(c_S)$ is the combined weight of all edges connecting community S to other communities, and $W(m_s)$ is the combined weight of all edges inside community S. A low weighted conductance score is indicative of a stronger community.

V. Results

In order to evaluate our algorithm in a tractable setting, we first ran our weighted community detection algorithm alongside the standard community detection algorithm on a small subset of the GitHub collaboration graph. For our initial graph, we used the second largest strongly connected component of the GitHub collaboration graph, which totals in at 879 user nodes and 3888 edges. This graph is both small enough to run our weighted community detection relatively fast and large enough to offer insight into the structure of collaboration on GitHub.

A. Measurements

In Figure 2, we can see that there is a large difference between weighted and unweighted modularity in both weighted and normal community detection. This corroborates with the greater average edge contribution for weighted modularity as compared to unweighted modularity. Since communities are more likely to contain high-weight edges than not, we would

expect to see a higher average weighted modularity. Interestingly enough, the weighted modularity obtained through normal community detection nearly matches that obtained through weighted community detection. This would indicate that edges with high scores can be grouped together by the structure of the graph alone.

In general we find that weighted modularity tends to have peaks at a higher number of communities as compared to unweighted modularity. There is a similar early peak in modularity score like what is observed for unweighted modularity, but the highest weighted modularity score tends to come at a larger number of communities. Weighted modularity shows a sawtooth graph, indicating that there are several tiers of communities that can be drawn given a set of vertices, with different cut-off points where the score increases sharply.

It can be observed that the unweighted modularity scores are similar over the number of communities for both weighted and unweighted edge removal. However, the peak weighted modularity values differ by a significant amount, as can be seen in Figure 3.

Our modularity measurements presented confusing results, largely because of the sawtooth structure of our weighted modularity plots and the high number of communities (440) our weighted community detection algorithm recommends. In order to figure out what was going on, we decided to take a look at the graph itself. Visualizing the 440 community graph (Figure 5, 6) , we quickly spotted an anomaly: there were hundreds of users who were assigned to their own community, and all of them were connected to node 21329.

B. Anomalies on GitHub

Node 21329 (The dark blue node in both Figure 6 and Figure 5) is connected to 509 other nodes, 468 of which are only connected to node 21329. It turns out this user is named Try-Git on GitHub, and it something of an automated bot designed to help people learn Git. This bot's recent history reveals that it automatically pushes to other users' Try-Git repositories. As is the case in GitHub, most of these other users don't collaborate with anyone else.

This result is quite promising, because it justifies the need for a large number of communities: no one is *really* collaborating with the Try-Git bot, so we *want* the users who have only collaborated with Try-Git to be in their own communities. For the most part, our weighted algorithm accomplishes this. The normal community detection algorithm instead chooses only 22 separate communities, which doesn't adequately differentiate between Try-Git users and members of real communities.

In Figure 6, we have emphasized user pairs with strong collaboration scores with thicker line widths. We find that the communities identified by weighted community detection tend to have a subset of users connected by strong edge weights, with members at its periphery connected with weak edge weights. This distinction tends to be lost by unweighted community detection, largely because it partitions the graph into too few communities.

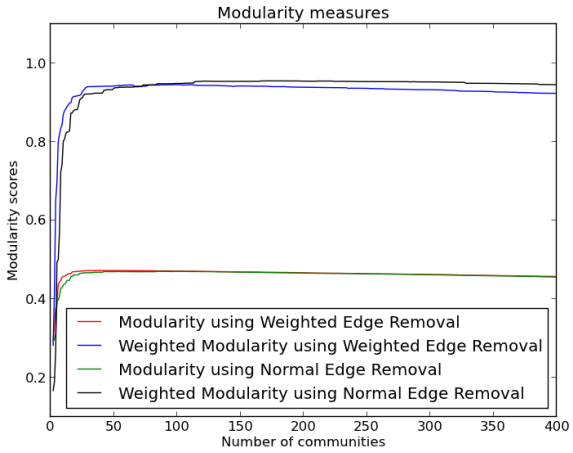

Fig. 9. Graph showing modularity and weighted modularity over communities produced by normal and weighted community detection on the largest SCC

C. Conductance

The weighted community detection algorithm also tended to more effectively minimize weighted conductance, as shown in Figure 7. This shows the conductance values of communities produced by weighted community detection, and Figure 7 shows those produced by normal community detection.

The difference between the two is especially apparent in the conductance of larger communities, as shown in Figure 7. While this is encouraging, conductance might not be the right measure for this particular graph, because we *want* all of the satellite nodes connected to Try-Git to form their own personal communities, which would give them a conductance value of 1. To analyze a less anomalous graph, we turn our sights towards the largest connected component of the collaboration graph. The average conductance (0.839) and weighted conductance values (0.829) are also very closely similar.

D. Scaling the Collaboration Graph

We next applied our algorithm to the largest strongest component in the Github social graph, consisting of 4209 nodes and 48782 edges. The conductance values for weighted and unweighted edge removal are shown in Figure 8.

Comparing modularity scores (Figure 4), we found the unexpected result that although both unweighted modularity scores were similar, the number of communities returned by the unweighted algorithm was significantly higher than that of the weighted algorithm. This produced higher weighted modularity scores for the baseline algorithm. Even so, the differences in the modularity scores for both measures are closely similar.

Where as the modularity values appear to be very closely matched for both algorithms, the weighted community detection yields communities with better conductance values, as can be observed in Figure 8. The average conductance values for unweighted community detection are 0.237 and

Edge Removal Algorithm	Weighted Modularity	Number of Communities	Unweighted Modularity	Number of Communities
Weighted	0.797	440	0.446	18
Unweighted	0.789	254	0.451	22

Fig. 3. Table of Max Weighted/Unweighted Modularity Score and Number of Communities for second largest SCC

Edge Removal Algorithm	Weighted Modularity	Number of Communities	Unweighted Modularity	Number of Communities
Weighted	0.945	97	0.473	42
Unweighted	0.955	171	0.470	98

Fig. 4. Table of Max Weighted/Unweighted Modularity Score and Number of Communities for largest SCC

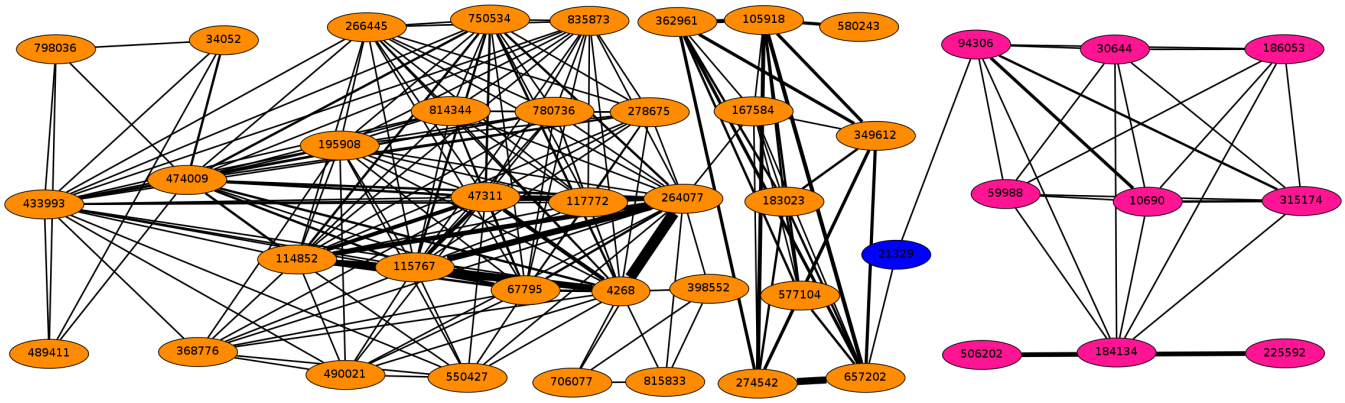

Fig. 5. Community structure found by normal community detection. The Try-Git user is in blue.

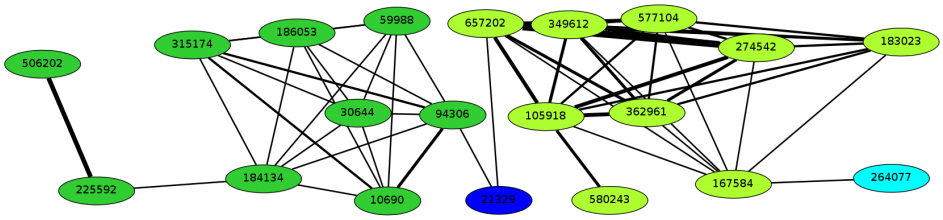

Fig. 6. Communities found using weighted community detection. The Try-Git user is also in blue.

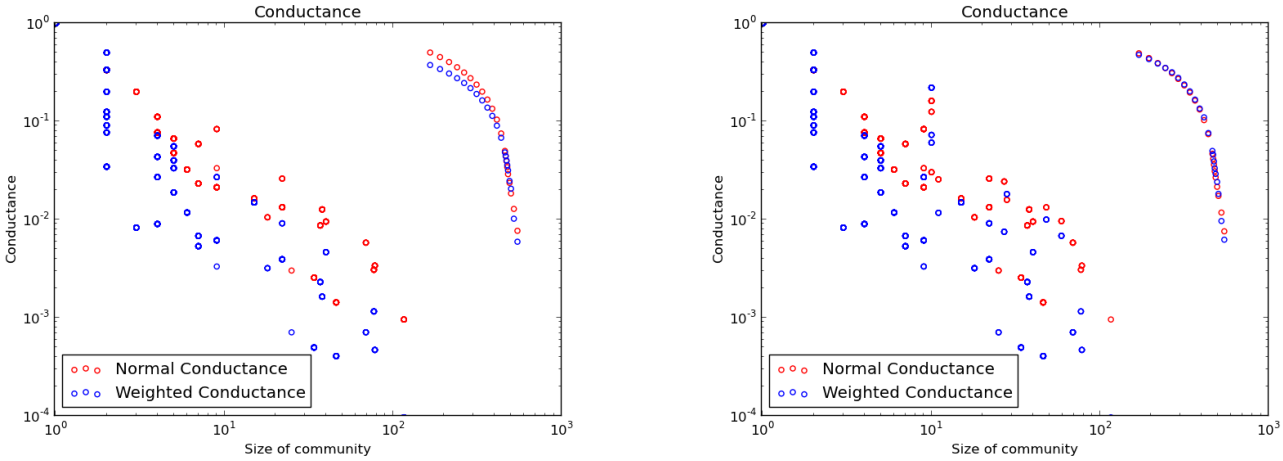

Fig. 7. Normal and weighted conductance values produced by weighted community detection (left) versus normal community detection(right) on the second largest strongly connected component.

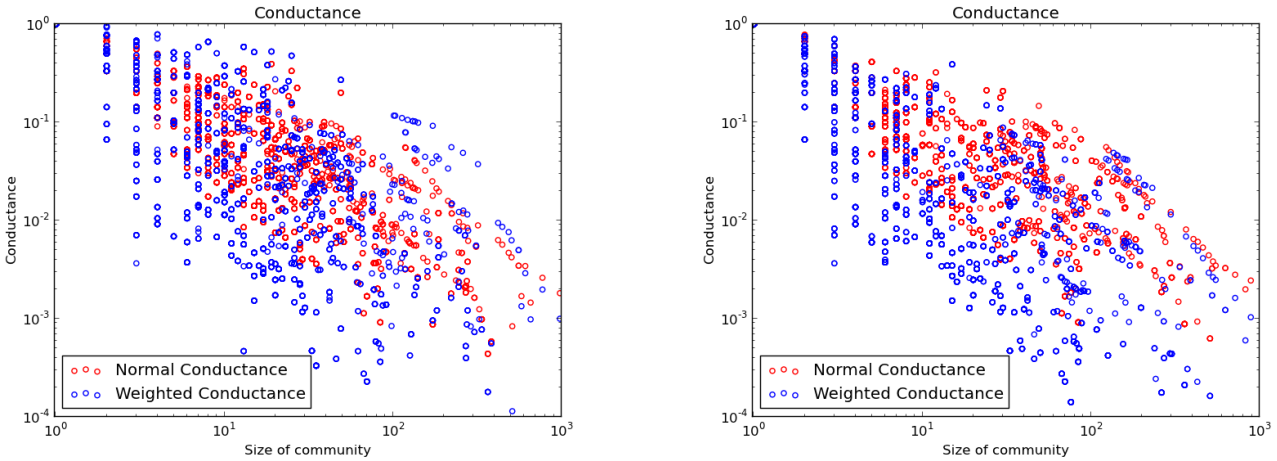

Fig. 8. Normal and weighted conductance values produced by weighted community detection (left) versus normal community detection(right) on the largest strongly connected component.

0.213 respectively for average unweighted and weighted conductance, compared to 0.170 and 0.168 for average unweighted and weighted conductance for weighted community detection. These results show that the weighted community algorithm performs at least as well as the unweighted algorithm.

VI. Future Work

We can also observe that the existing measures used are heuristics for a property that is not yet clearly defined. Future work is needed to be able to better access how successful a community partitioning is at identifying social groups within a community.

VII. Conclusion

Following network structure alone without assessing the weight of interactions between vertices in the graph would lead to inferior solutions found, such as the 'community' surrounding the Try-Git user. Modifying community detection to incorporate weights produces results that are at least as good as the baseline algorithms. In particular, incorporating weights allow for better assessment of which edges in the network should be emphasized over others.

References

[1] T. L. C. Marlow, L. Byron and I. Rosenn., "Maintained relationships on facebook." 2009. [Online]. Available: http://overstated.net/2009/03/09/maintained-relationships-on-facebook

[2] D. R. B.A. Huberman and F. Wu., "Social networks that matter: Twitter under the microscope." 2009.

[3] U. Brandes, P. Kenis, J. Lerner, and D. van Raaij, "Network analysis of collaboration structure in wikipedia," in *Proceedings of the 18th international conference on World wide web*, ser. WWW '09. New York, NY, USA: ACM, 2009, pp. 731–740. [Online]. Available: http://doi.acm.org/10.1145/1526709.1526808

[4] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review*, vol. E 69, no. 026113, 2004.

[5] J. Jin, L. Pan, C. Wang, and J. Xie, "A center-based community detection method in weighted networks," in *Proceedings of the 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, ser. ICTAI '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 513–518. [Online]. Available: http://dx.doi.org/10.1109/ICTAI.2011.83

[6] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 35, pp. 75 – 174, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0370157309002841

[7] J. Leskovec, K. J. Lang, and M. Mahoney, "Empirical comparison of algorithms for network community detection," in *Proceedings of the 19th international conference on World wide web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 631–640. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772755