Network Analysis and Community Detection on GitHub

https://github.com/rameshvarun/github-network-analysis

Jarrod Cingel
Stanford University
jcingel@stanford.edu

Varun Ramesh
Stanford University
vramesh2@stanford.edu

Abstract

Technology companies increasingly target open-source communities for hiring. In the same way that consumer brands analyze Facebook and Twitter to optimize advertising, recruiters should analyze GitHub to identify communities and important voices. We perform network analysis on two graphs generated from GitHub user interactions. The first ***User-Follow*** *graph captures follow relationships, while the second* ***User-PR*** *graph captures pull requests between users. For each of these graphs, we analyze degree distributions, PageRank, reachability, roles (using RolX), and communities (using the Leiden algorithm). With find that the GitHub graph is extremely unequal, with a distinction between "regular" users and "leaders." We also find that the graph has one large community that roughly corresponds to modern application development.*

1. Introduction

GitHub is the most prominent and widely-used online version control platform for software development projects. Users sign up for GitHub and create a profile, which includes self-reported "location" and "company" fields. Users can then create repositories (repos), which are units of code that can be published to from any Git client. Other users can open pull requests on a repo - pull requests represent patches that the creator wishes to be incorporated into that repo. The owner of a repo can either accept or deny the pull request.

On top of user profiles, GitHub also features "organizations," which are a special type of user that represents a group of regular users. Organizations can own repositories. Users can be part of any number of organizations.

In addition to code productivity tools, GitHub also incorporates features common across social networks [11]. Users have an information feed that is displayed on the homepage when they log in. Users can follow other users, which adds the followee's activity to the follower's information feed. Users can also watch repos, which puts activity related to that repo on the user's feed. Users can't follow organizations - only regular users.

Because of these characteristics, GitHub acts as a unique blend between a social network and engineering tool. The goal of this project is to perform a variety of analyses on this network. We begin by analyzing basic characteristics like degree distribution and reachability, then move on to perform more advanced analyses like role extraction and community detection. We also examine multiple ways to generate a graph from the GitHub data, in order to understand the benefits and trade-offs of each one.

2. Previous Work

2.1. Network Algorithms

Clauset et al. introduced the most common approach to community detection, often referred to as the Clauset-Newman-Moore (CNM) algorithm [3]. They define a community as a densely connected sub-region of nodes in a graph, and seek to optimize an objective called modularity. Later on, Blondel et al. introduce the Louvain algorithm [1] which is simpler, faster, and more effective at optimizing modularity. Traag et al. build on this to create the Leiden algorithm, which is even faster and produces better communities [12]. More details on the Leiden algorithm are available in section 4.5.1.

The RolX algorithm [7] was first introduced by Henderson et al. to establish structural similarities between nodes. The authors found that, unlike communities, roles provide information about node-level behavior. The authors applied RolX to a book co-purchasing network generated from Amazon data for political books. They used roles to distinguish between "central" books and "periphery" books.

Reachability analysis [2] was first introduced by Broder et al. The authors generated reachability plots from link networks on the world-wide-web. They discovered that the web had a "bowtie" structure - a single giant strongly-connected component with an inlink-set and an outlink-set.

PageRank [10] was first introduced by Page et al. The authors used the algorithm to find important nodes on an early version of the Web graph. They found that this was better for extracting relevant nodes from a query than title search alone.

2.2. Social Network Analysis

Several other authors have applied network algorithms to large real-world social networks. For example, Ugander et al. applied community detection to Facebook, which can be understood as an undirected graph with users as nodes and friendships as edges [13]. The authors find that communities tend to form among geographically-similar users, but without regard to traditional demographics like race and gender, especially when controlling for country and region.

Similarly, Huberman et al. examine Twitter as a social network [8]. They find that Twitter users only interact with a small subset of the people they follow. They conjecture that Twitter has a dense follower network, but a much sparser yet more influential interaction network.

Both of these studies focus on undirected networks. Facebook's friend relationship naturally lends itself to an undirected model, while for the Twitter network, a concept of friendship was artificially created to make the graph undirected. We build on this prior work by producing different analyses across directed graphs. We still use undirected graphs for community detection.

3. Data set

We obtained our data from the GHTorrent project [6]. We downloaded GHTorrent tables that were publicly available as part of Google's BigQuery service. The data is dated to April 1st, 2018. We generated two graphs from our data tables, the first of which attempts to capture influence, and the second of which attempts to capture actual code contributions.

3.1. User-Follow Graph

The **User-Follow** graph is a directed graph consisting of users. When a user follows another user, an edge is created pointing from the follower to the followee. This graph consisted of 24 million users, 25 million edges, and an average clustering coefficient of 0.025. However, we found that over 19 million nodes had no in or out follow edges. We decided to prune those nodes. This gave us a new graph with 4 million nodes (the number of edges stays the same). The pruned clustering coefficient was 0.133.

3.2. User-PR Graph

The **User-PR** graph is also a directed graph over users. When a user opens a pull request to a repository, we create an edge from that user to the owner of the repository. This graph consisted of 44 thousand nodes and 42 thousand edges. This graph has a clustering coefficient of 0.0123.

Unlike the **User-Follow** graph, the **User-PR** graph more accurately captures code contributions. Furthermore, it provides more equal footing to organizations, as organizations on GitHub cannot be followed, but can be contributed to.

4. Methodology

In order to perform our analysis, we used two network libraries for Python - SNAP [9] and igraph [4].

4.1. Degree Distribution

We began by replacing all of the directed edges in our graph with undirected ones, and by plotting the proportion of nodes with a given degree. We then included the directed edges and plotted the in and out degrees on the same graph.

4.2. PageRank

We analyzed our graphs using SNAP's implementation of PageRank, selecting the top 100 users by PageRank score. As a baseline, we compared these to the list of 100 most followed users.

4.3. Reachability Analysis

We performed a reachability analysis on each of the two networks. These reachability figures were estimated by drawing a sample of random nodes from each graph and computing their inward and outward breadth-first searches. We then sorted nodes by their reach and plotted the resulting data.

4.4. Role Extraction

We used the RolX algorithm [7] to extract roles of nodes in the graph. Our basic feature vector included the degree of each node, the number of edges coming into the egonet of each node, and the number of outgoing edges from the egonet of each node. We used recursive feature estimation with 1 level of recursion. Once this was completed, we used k-means clustering with $k = 8$ to identify centroids that corresponded to 8 roles.

We also used principal component analysis (PCA) on the resulting feature vectors - we plotted the first two principal components.

4.5. Community Detection

After initial testing, we found that SNAP's implementation of the CNM algorithm did not scale to the size of our data set. Instead, we chose to go with the Leiden algorithm, which is designed for much larger networks. We used the original Leiden algorithm implementation along with igraph in order to perform community detection on our data [12].

4.5.1 Leiden Algorithm

Community detection algorithms like CNM [3] and Louvain [1] attempt to optimize modularity. Modularity is defined below, where m is the number of edges in the network, A is the adjacency matrix, k_i is the degree of node i, and c_i is the community of node i.

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] 1[c_i = c_j]$$

The Louvain algorithm optimizes modularity by local moves - at each iteration nodes are greedily moved to the community that produces the greatest increase in modularity. After each series of local moves, the nodes in each community are merged into super-nodes in an aggregation step.

The Louvain algorithm is simple, but has many issues. One issue is that, when a bridge node is moved from one community to another, this can result in an internally disconnected community. Furthermore, the Louvain algorithm is slow because it visits all the nodes in the network on each local-move phase.

The Leiden algorithm fixes these issues with the Louvain algorithm. It does so by adding an extra step between local moves and aggregation called "refinement." After we generate a partition P from local moves, the Leiden algorithm refines each community in P into one or more sub-communities. The refined communities are generated by randomly merging nodes within a community. Once complete, we use the refined partition for the aggregation phase, but assign the super-nodes to the communities from the original partition. The Leiden algorithm also improves speed by only visiting a node when its neighborhood has changed.

However, the Leiden algorithm does not prevent the tendency of Louvain to incorrectly merge small communities into larger communities. This is a fundamental limit of modularity called the "resolution limit" [5]. This can be resolved by using other objective functions like the Constant Potts Model (CPM). However, we chose to stick with modularity for it's simplicity.

4.5.2 Evaluating Communities

To evaluate our communities, we borrowed a metric from the Louvain paper called homogeneity. Blondel et al. evaluate cell-phone network communities by determining what percentage of users in a community speak the dominant language of that community. We do the same analysis, but instead of language, we look at three features. The first is organizations, which is GitHub's notion of groups of users that collectively own and contribute to repositories. The second feature is the company a user works at - this is self-reported on the user's profile. Finally, we look at the user's location, also self-reported on their profile.

5. Analysis Results

5.1. Degree Distribution

The degree distributions for the User-Follow and User-PR networks are shown in Figure 1 and Figure 2, respectively. In the User-Follow network, we can see that the vast majority of nodes have only one neighbor, but this frequency diminishes sharply as degree (both in and out) increases. This is strongly indicative of a preferential attachment model - a few users get many followers. Once one has followers, it becomes more likely for them to obtain additional followers. This also applies to out-degree, meaning that users who follow many other users are likely to follow even more.

In the User-PR network, we observe a similar shape and trend, but see differences when we distinguish between in and out degree. Users with a small degree tend to make more contributions to other repos, but receive few contributions to their own repos. The out-degree in the User-PR network falls much more sharply than the in-degree. This means that users with a high degree receive more contributions than the contributions that they make.

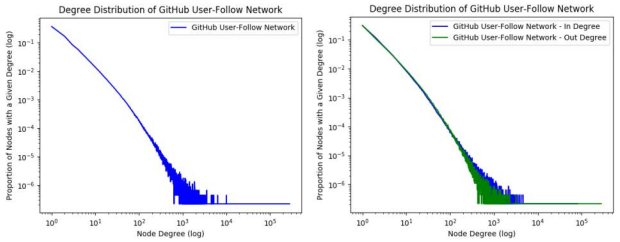


Figure 1. Degree distributions (Combined, In, and Out) for User-Follow network.

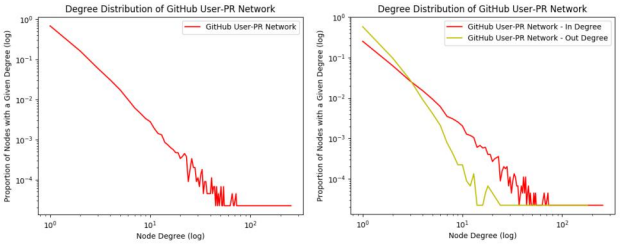


Figure 2. Degree distributions (Combined, In, and Out) for User-PR network.

Table 1. The top 10 users according to PageRank scores from the User-Follow graph.

Rank	Username	PageRank $\times 10^4$
1	torvalds	8.82
2	JakeWharton	6.93
3	michaelliao	5.25
4	githubpy	4.44
5	Tj	3.84
6	mojombo	3.30
7	paulirish	3.24
8	defunkt	2.93
9	ruanyf	2.90
10	gaearon	2.76

Table 2. The top 10 most followed users on GitHub.

Rank	Username	Followers
1	torvalds	80184
2	JakeWharton	48120
3	ruanyf	39102
4	Tj	37402
5	addyosmani	32666
6	paulirish	29690
7	yyx990803	29200
8	gaearon	27415
9	sindresorhus	25701
10	mojombo	25112

5.2. PageRank

5.2.1 User-Follow Graph

The top 10 users by PageRank is shown in Table 1 - this can be compared to the top 10 most followed users, as shown in Table 2.

It turns out these two methods of finding "important" users produce similar results. Many users appear in both lists, though in different orders. The Jaccard similarity between the top 100 most followed users and the top 100 users by PageRank is 0.626, demonstrating how similar the lists are.

Many of the top users on both lists make sense. `torvalds` is the username of Linus Torvalds, the creator of the Linux kernel. `JakeWharton` is the username of Jake Wharton, the maintainer of many popular Android development repos. Going down the list, we also see Tom Preston-Warner, the co-founder of GitHub, and Dan Abramov, the creator of the Redux framework for React.

However, some users on the PageRank list seem highly unusual. Take user `githubpy` for example, who has only three followers yet somehow ranks highly on the PageRank list. To illuminate this situation, we can look at the egonet for `githubpy`, shown in Figure 3. From this, we see that `githubpy` is most likely receiving it's PageRank score from `michaelliao`, a user who appears on the 100 most followed list and the top 10 PageRank list.

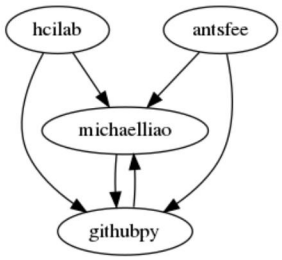


Figure 3. githubpy Egonet

5.2.2 User-PR Graph

The top 10 users by PageRank on the User-PR graph are shown in Table 3. Intuitively, this shows us the users which receive many pull requests on their repos. Unsurprisingly, these "users" are most often organizations like Mozilla or Ruby on Rails that have extremely popular, highly contributed to projects. Two individuals are in the top 10 - Hadley Wickham (`hadley`) and Kenneth Reitz (`kennethreitz`). These individuals are well known in the R and Python communities respectively, and each has several highly popular and heavily contributed to repos.

Table 4 shows the results of PageRank run on the reversed User-PR graph - it gives us

Table 3. The top 10 users according to PageRank on the User-PR graph.

Rank	Username	PageRank $\times 10^4$
1	mozilla	25.72
2	hadley	15.92
3	jenkinsci	14.59
4	twitter	11.96
5	rails	11.43
6	puppetlabs	10.71
7	kennethreitz	9.72
8	visionmedia	9.57
9	heroku	9.56
10	doctrine	9.28

Table 4. The top 10 users according to PageRank on the reversed User-PR graph.

Rank	Username	PageRank $\times 10^4$
1	GunioRobot	22.24
2	giovanniramos	5.04
3	kenmazaika	4.16
4	ertemplin	3.41
5	rrix	3.35
6	phil5	3.31
7	JonnyJD	3.29
8	srs81	3.27
9	calvinchengx	3.25
10	lamblin	3.16

users who open pull requests on a wide variety of repositories. As expected, the top slot is taken by `GunioRobot`, a bot. Although it no longer exhibits this behavior, from 2011-2012, `GunioRobot` opened hundreds of pull requests on repos with the only changes being the removal of trailing white-space in code. Some of the other users on the list, such as `kenmazaika` seem to legitimately be users who open lots of pull-requests to many repos - perhaps an underappreciated behavior.

5.3. Reachability Analysis

The results of our reachability analysis on the two networks are shown in figures 4 and 5. Figure 4 shows the reciprocity between inward and outward traversals in the User-Follow network. This would suggest that there is indeed a single large strongly connected component in the structure of the GitHub network, just like the bow tie structure of the World Wide Web [2].

The User-PR graph, on the other hand, does not seem to have a single large strongly connected component. The sampled nodes with the largest reachability have far lower proportional reachability than in the User-Follow counterpart. This can be explained based on the more stringent creation method of the User-PR graph. The acts of both owning a repository and creating a pull request on a repository both imply much higher level of responsibility, and therefore stronger association, than is implied in the construction of the User-Follow graph. Additionally, the results in the User-PR graph show that repository owners generally don't contribute to other repositories, resulting in fragmentation.

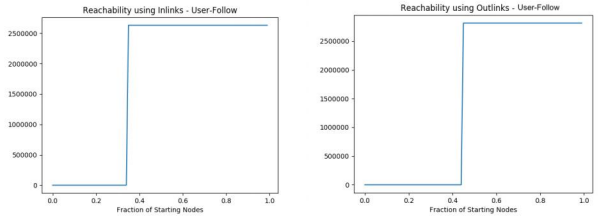


Figure 4. User-Follow Graph Reachability - 100 Node Sample

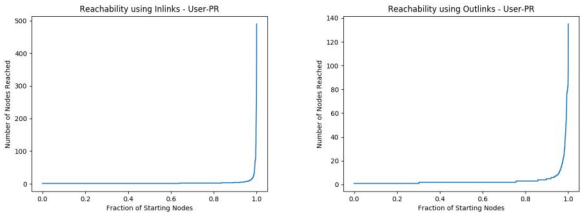


Figure 5. User-PR Graph Reachability - $10,000$ Node Sample

5.4. Role Extraction

The results of role extraction are shown in figures 8 and 9 for the User-Follow and User-PR networks, respectively. For each network, 1000 nodes were randomly selected for the RolX algorithm. Our first observation is that cluster membership changes only slightly when the order of PCA reduction and k-means clustering is switched. This indicates a relative success in our dimensionality reduction; a large amount of the feature vector variance is captured in the first two principal components.

In the User-Follow network, the vast majority of the randomly sampled nodes are clustered together very tightly, with a select few outliers belonging to clusters with only one or two members very far from the central concentration. This seems consistent with our results from the degree distribution; the majority of nodes in the graph have a low degree but are connected to a select few nodes with extremely high degree. These correspond respectively to the large cluster of nodes in the middle left and all of the outliers spread across the rightmost portion of the graph.

The User-PR network paints a different picture due to it's stricter edge criteria. Nodes are distributed rather linearly in two places, eventually converging to a vertex in the upper left corner of the chart. This seems symptomatic of more diverse, varied roles that share more similarity with one another relative to the drastically differing primary roles in the User-Follow network.

We can take this analysis a step further by examining the egonets of a selection of nodes. We can take any selection of nodes corresponding to the dense, tightly-clustered regions in Figures 8 and 9, and find that the result typically matches the form of Figure 6, indicating a very small neighborhood of direct relationships, for both followers and pull-requests.

When we look at the outliers on the PCA plots, we see a very different result. Examining the upper-rightmost node in the User-Follow network results in an egonet that is too dense to easily show. The node in question corresponds to the user `eva1963`, who is a student at Sias University in China. Although she only owns 21 repositories, has only 17 followers, and follows only 42 people, her egonet is quite dense because many of her follows are reciprocated. Follows that are not reciprocated generally belong to very large organizations, like JP Morgan or Tencent.

For the User-PR network, we can look at the outlier in the bottom right corner. We plot the egonet in Figure 7. It turns out that this node corresponds to the user `bnoordhuis` (Ben Noordhuis), who is a member of IBM among other prominent organizations. He owns several widely followed Node.js libraries, and he also is a frequent contributor to other repositories. It seems that bnoordhuis is one of the rare users who contributes at the pull-request level to many projects, and also has many users contribute heavily to his own projects.

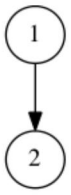
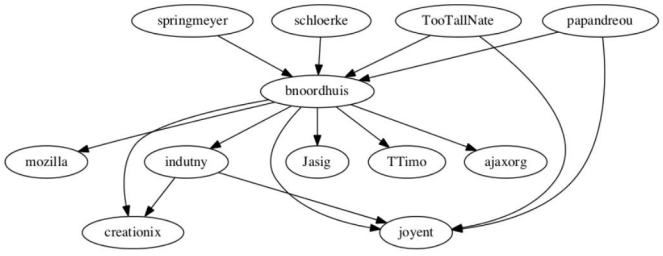


Figure 6. User-Follow and User-PR Networks: Egonet of arbitrary node in central, dense region.



User-PR Outlier Neighborhood

Figure 7. User-PR Network: Egonet of user bnoordhuis, an IBM employee who owns more than 130 repositories.

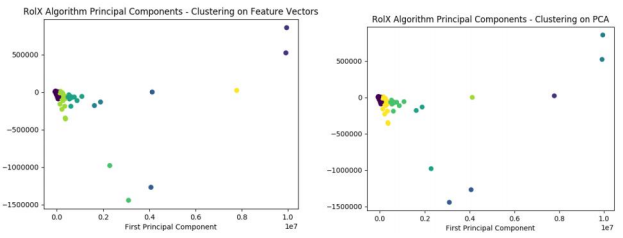


Figure 8. User-Follow Network: Clustering preceded PCA in the leftmost image, while clustering followed PCA in the rightmost image.

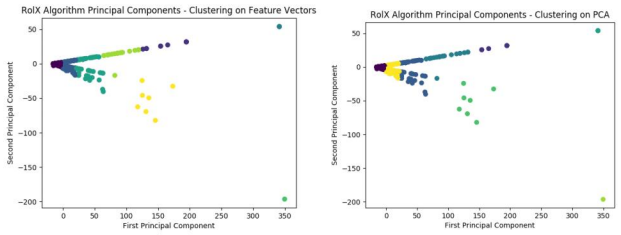


Figure 9. User-PR Network: Clustering preceded PCA in the leftmost image, while clustering followed PCA in the rightmost image.

5.5. Community Detection

5.5.1 User-Follow Graph

The Leiden algorithm gave us 116,986 communities with an average size of 39.32 and a standard deviation of 3011.08. The distribution of community sizes is shown in Figure 10.

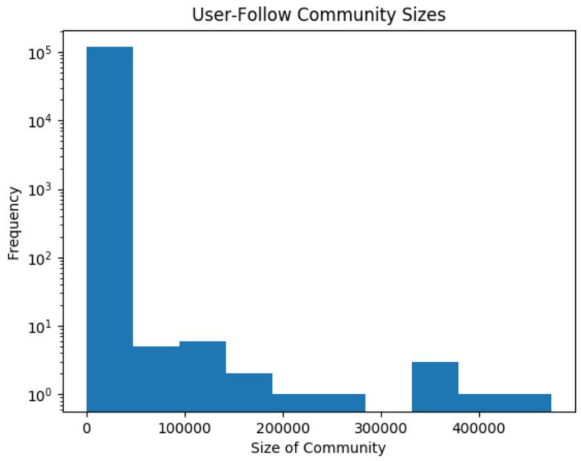


Figure 10. Histogram of User-Follow graph community sizes.

We have a wide spread of community sizes - most communities are very small. Thus, to make further analyses more tractable, we pruned communities smaller than 5 members. We also see that there is one large community of size 474,072. This community contains `daimajia`, a user with many popular Android repos, `mitsuhiko`, a user with many popular Python repos, `LeaVerou`, a user with many popular JavaScript repos, and `matz`, the creator of Ruby. This large community likely represents "modern" app development, which extends from Android and JavaScript front-ends through Python, Ruby, and Node.js back-ends.

The results of homogeneity analysis for each of the three selected features is shown in Figure 11. This shows that many communities are not very homogeneous along either of the features. However, for some features, at least a few communities are more than 50% homogeneous.

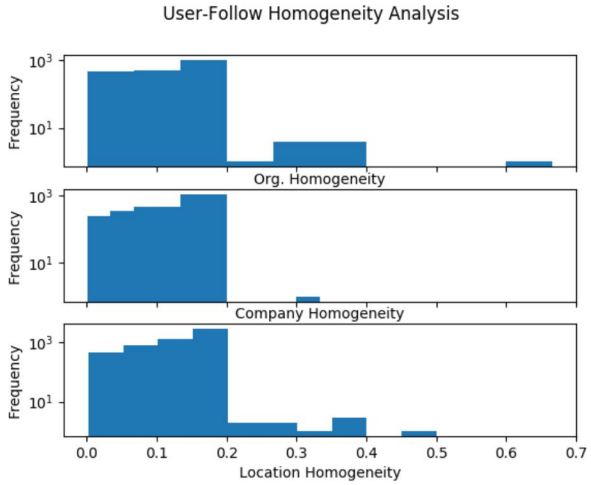


Figure 11. Homogeneity analysis of communities on the User-Follow graph.

In addition to homogeneity analysis, we also looked at the Jaccard similarity between communities and feature-extracted groups. Table 5 shows organizations ranked by their maximum Jaccard similarity to any community. Looking at the largest of these organizations, `Anktech`, we see that it consists of a small amount of users in India who mostly follow each other, and don't follow the GitHub community at large.

Unfortunately, the results for other features degenerated to pairs of small groups and communities with only one user in common. This suggests a low correspondence between real-world groups and the extracted communities.

Table 5. Organizations ranked by maximum Jaccard Similarity to any detected community.

Rank	Organization	Size	Jaccard Sim.
1	BucksCommCSC	8	0.18
2	tangentsnowball	12	0.17
3	Anktech	27	0.14
4	escalation-point	17	0.14
5	InTradeSysTeam	12	0.13

5.5.2 User-PR Graph

Running the Leiden algorithm on the User-PR graph produced 8,188 communities, of mean size 5.44 and a standard deviation of 35.56. Looking at the histogram in Figure 12, we can see a much more even spread of sizes, though we still have many small communities. The largest community has a size of 1,202, and contains `addyosmani`, a prolific member of the web development community, `defunkt` and `kevinsawicki` who work on the Atom text editor, and `matz` the creator of Ruby. Once again, this large community likely corresponds to modern application development which encompasses web and server technologies.

Figure 13 shows the result of homogeneity analysis on the User-PR graph. It demonstrates a more even spread, but at the same time the most homogeneous communities are less homogeneous than the most homogeneous communities in the User-Follow graph.

6. Conclusion

Our analyses have shown that GitHub's graph is highly unequal. Degree distribution, role extraction, and PageRank show that the graph can be

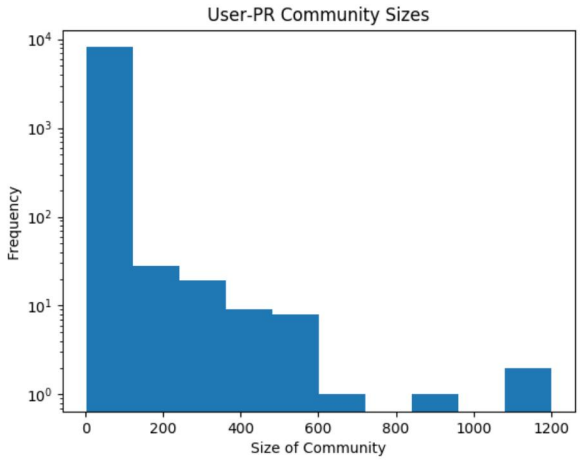


Figure 12. Histogram of User-PR graph community sizes.

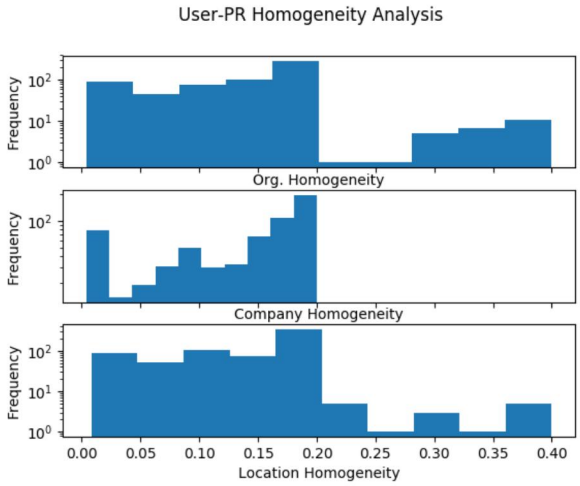


Figure 13. Homogeneity analysis of communities on the User-PR graph.

roughly split into "regular users" who have limited influence and "leaders" who have outsize influence. This is similar to many other social networks like Twitter that also follow a power law distribution. This is highly indicative of the preferential attachment model.

Furthermore, as shown by reachability analysis and community detection, many of GitHub's users exist in one highly-connected community. This large community roughly corresponds with modern application development. It includes Android and browser JavaScript frontends, though

unexpectedly does not include iOS. This may be because the Android open-source community is stronger. A quick search of repos by tag confirms this - 46,257 repos are tagged with "android," but only 17,084 repos are tagged with "ios." This large community also includes server-side languages like Python, Ruby, and Node.js. Many of the other communities are small collections of users that don't connect to the wider GitHub user base.

7. Group Work Breakdown

Varun Ramesh - Set up initial data tables / code, generated User-Follow graph, performed PageRank analysis, and ran community detection / analysis.

Jarrod Cingel - Generated User-PR graph, performed degree distribution analysis, performed reachability analysis, implemented RolX feature extraction, and ran PCA / K-means.

References

[1] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[2] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer networks*, 33(1-6):309–320, 2000.

[3] A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[4] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[5] S. Fortunato and M. Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.

[6] G. Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[7] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239. ACM, 2012.

[8] B. A. Huberman, D. M. Romero, and F. Wu. Social networks that matter: Twitter under the microscope. *arXiv preprint arXiv:0812.1045*, 2008.

[9] J. Leskovec and R. Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

[10] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[11] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang. Network structure of social coding in github. In *Software maintenance and reengineering (csmr), 2013 17th european conference on*, pages 323–326. IEEE, 2013.

[12] V. Traag, L. Waltman, and N. J. van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *ArXiv e-prints*, Oct. 2018.

[13] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011.