Introduction

Open source software is widely used across a variety of fields, so understanding the factors which lead to successful projects is important. This project models contributions on GitHub as a network with two types of nodes: individuals and "repos" (software projects). Individuals can collaborate on repos in a variety of ways; the total number of these collaborations is summed per repo and used as an edge weight. In summary, interactions are modeled as a 2-dimensional weighted digraph. This is a "two-mode" network as edges only connect different types of nodes. We may also consider this as a bipartite graph. More complex network constructions which include repo dependencies or other types of edges are possible but weren't used for simplicity. We construct and analyze this network using data from 2015 through 2020. Statistics are compared to that of a random network generated using an approach similar to the configuration model. We also develop various measures of node connection strength and use these to detect communities, analyze clustering, and generate visualizations.

After discussing the dataset and graph structure, we will first examine various statistics of the network and compare these to the corresponding values for the random graph model. We will then develop a notion of connection strength useful for analyzing the connections of nodes with other nodes of the same type (which aren't directly connected by edges). Finally we derive a method for "normalizing" connection strength in some sense.

Source code for this project can be found at https://github.com/rgreenblatt/apma_1941b_final.

Network Structure

Because of the rich dataset, a variety of different multi-dimensional network constructions are possible. Two node types were used: one dimension for users and another for repos. Contributions were modeled as integer weighted directed edges. While contributions are from users to repos, these edges can also be considered as undirected edges. This is often more convenient.

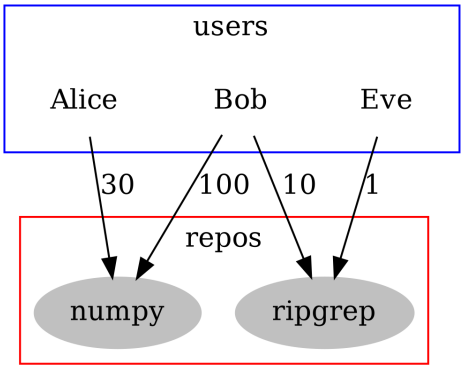


Figure 1: Example network

Data

There are many ways in which individuals may collaborate on a repo: directly committing (updating the code), pull requests, issues, and comments on either of the previous two. Pull requests allow for proposing a number of commits for review, while issues allow for reporting bugs, requesting new features, or general discussion. While all types of events can be accessed on a per repo basis via the API which GitHub makes publicly available, we need to access that data per user. Unfortunately, GitHub only stores user activity going back 300 events, so this approach is insufficient. Traversing over repos to find this information is likely inefficient and incomplete. So, instead we used the data from the GH Archive project which archives all GitHub events starting from 2011 onward. We only used data from 2015 onward.

Including additional information (particularly repo dependencies) was considered, but this data cannot be obtained just using the events data available from GH archive. Unfortunately, the GitHub API is quite slow in practice, so obtaining dependency information for tens of millions of repositories was impractical.

As far as event data is concerned, we counted all events as contributions except for `ForkEvent`, `DeleteEvent`, `MemberEvent`, `SponsorshipEvent`, and `WatchEvent`. For more details about the GitHub event type, see the documentation. We only need the total count of events, the type of event is irrelevant except for filtering. Thus, the dataset consists of a list of users, a list of repos, and a list of contributions. Each contribution entry has a repo, a user, and a total count. To reduce the size of the dataset, we only include repositories which have contributions from more than 1 user. We also exclude users which don't have any contributions to the included repositories and users which have a very large number of total contributions ($> 500,000$). These users with exceptionally large contributions are likely bots or spammers; humans can only do so many things. This is confirmed by looking the accounts of some of these users. There are only 63 such users, but this speeds up analysis and should make results more meaningful.

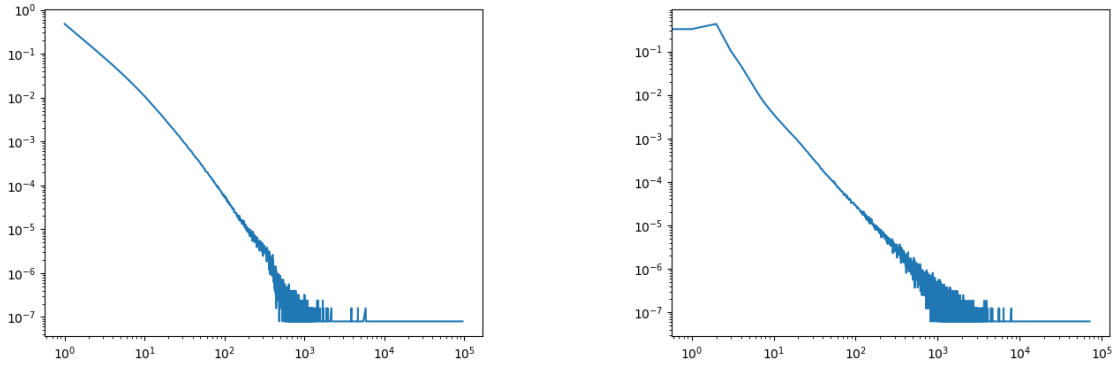
To quickly obtain this data, `BigQuery` was used as the GH Archive data is available there. The exact queries can be found in the source code under the "queries" directory.

Degree distributions (and similar)

We start by examining the degree distributions in figure 2. The repo and user degrees both appear roughly linear on the log-log plot. So, both seem to have power law tails. We do see somewhat of a downward curve on the users degree distribution. This makes sense because users can only make so many contributions which should have a damping effect. The densities of both are near zero at about 1000 or 2000 connections.

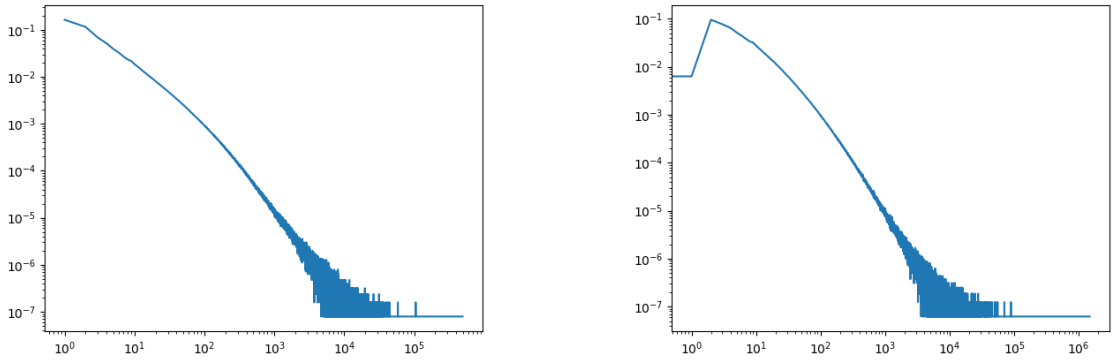
We can also look at the distribution of total contributions (figure 3). The total contribution of a node is just the sum of the weights of its (incoming or outgoing) edges. Both of these distribution have a slight downward curve but are nearly linear in the tails. So, again we see power law tails. We also see that repos typically have at least a few events. This makes sense—having only one commit to a repo should be rare.

Finally, we can look at the distribution of individual contributions (edges weights) in figure 4. Again



(a) Users (b) Repos

Figure 2: Degree distributions



(a) Users (b) Repos

Figure 3: Total contribution distributions

this is slightly downward curving but with a linear (power law) tail. This can also be analyzed on a per user or per repo basis. See figures 5 and 6. Once again, these look linear (power law) though due to the smaller number of samples, they are considerably noisier. It would be possible to rigorously examine the per user and per repo distribution to more accurately determine if they follow a power law distribution. However, that wasn't done here.

Random graph model

In order to analyze which properties of the real network differ from a random network, we use a random graph model very similar to the configuration model. Specifically, we produce a random graph based on the actual graph in which users and repos have the same number of edges and each of these edges has nearly the same contribution number (edge weight). We can't directly use

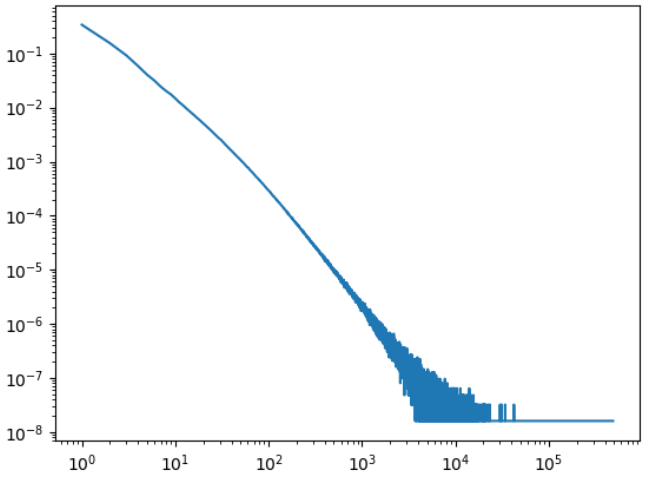


Figure 4: Individual contributions (edge weights) distribution

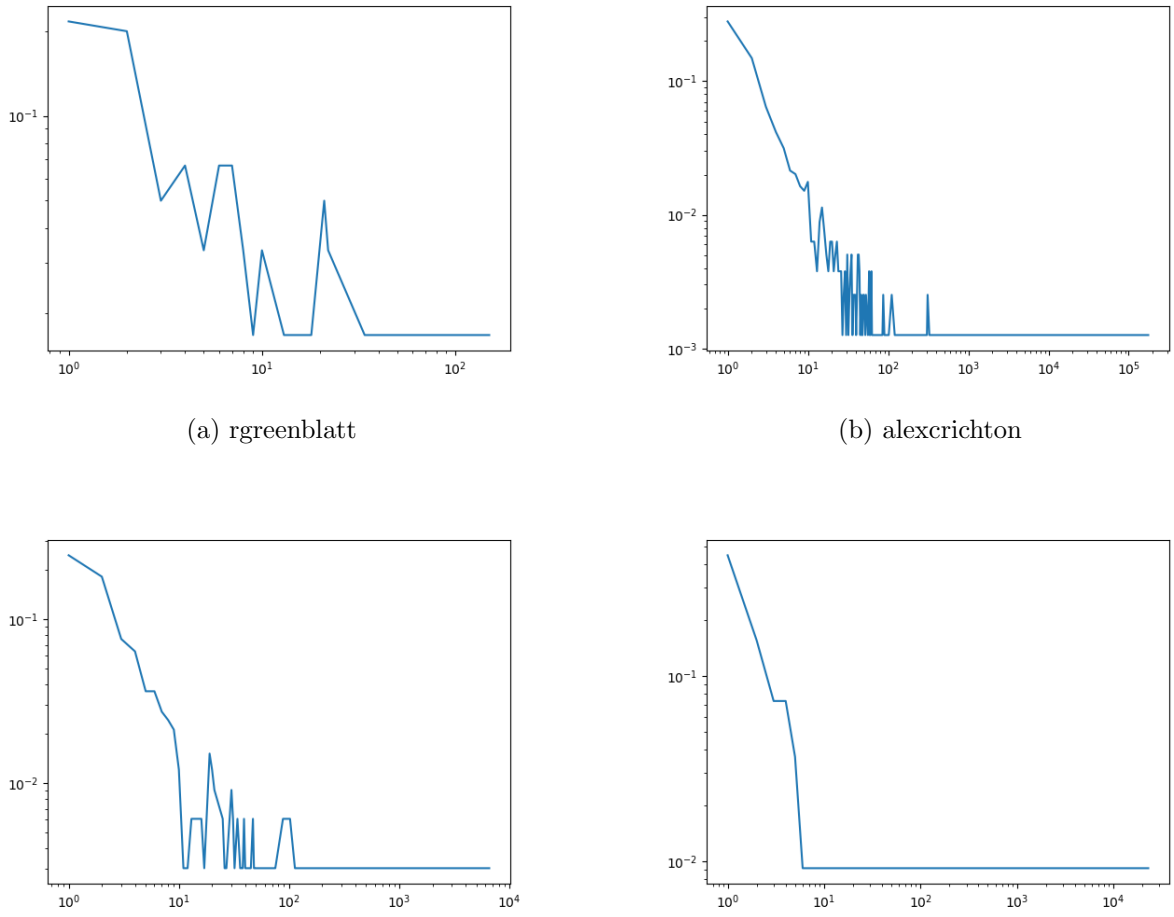
the exact same contribution numbers because some of them are quite rare. For instance, the user "tresf" made 9727 contributions to the repo "LMMS/lmms" and no other user made exactly 9727 contributions to any repo. So, this would force the user "tresf" to be connected to this repo in this random graph model—no other repo exists which can match this edge. To mitigate this while keeping the same contribution counts, we bin contribution counts into bins of at least 1000 users and repos. The bins for small contribution counts contain only one value because these small counts are quite common and bins start and end on integer edge values.

The overall algorithm for constructing a random graph proceeds as follows:

1. Using the number of occurrences of each contribution count, bins of size at least 1000 are constructed.
2. Each edge endpoint is placed into the corresponding bin based on the contribution number. Note that each edge has a user endpoint and a repo endpoint.
3. For each bin, we randomly select pairs of a user endpoint and a repo endpoint. These endpoints form an edge in the new graph. The contribution count of the new edge is selected uniformly from the two endpoint; the endpoints may have different contribution counts due to binning. Selection is repeated until the bin contains no more endpoints at which point we proceed to the next bin.
4. Duplicate edges are removed.

Components and distance

By considering the directed edges as undirected, we can perform traversals on the graph and find component sizes. We find a giant component with 83% of the vertices. Its worthwhile to note that



(a) rgreenblatt

(b) alexcrichton

(c) BurntSushi

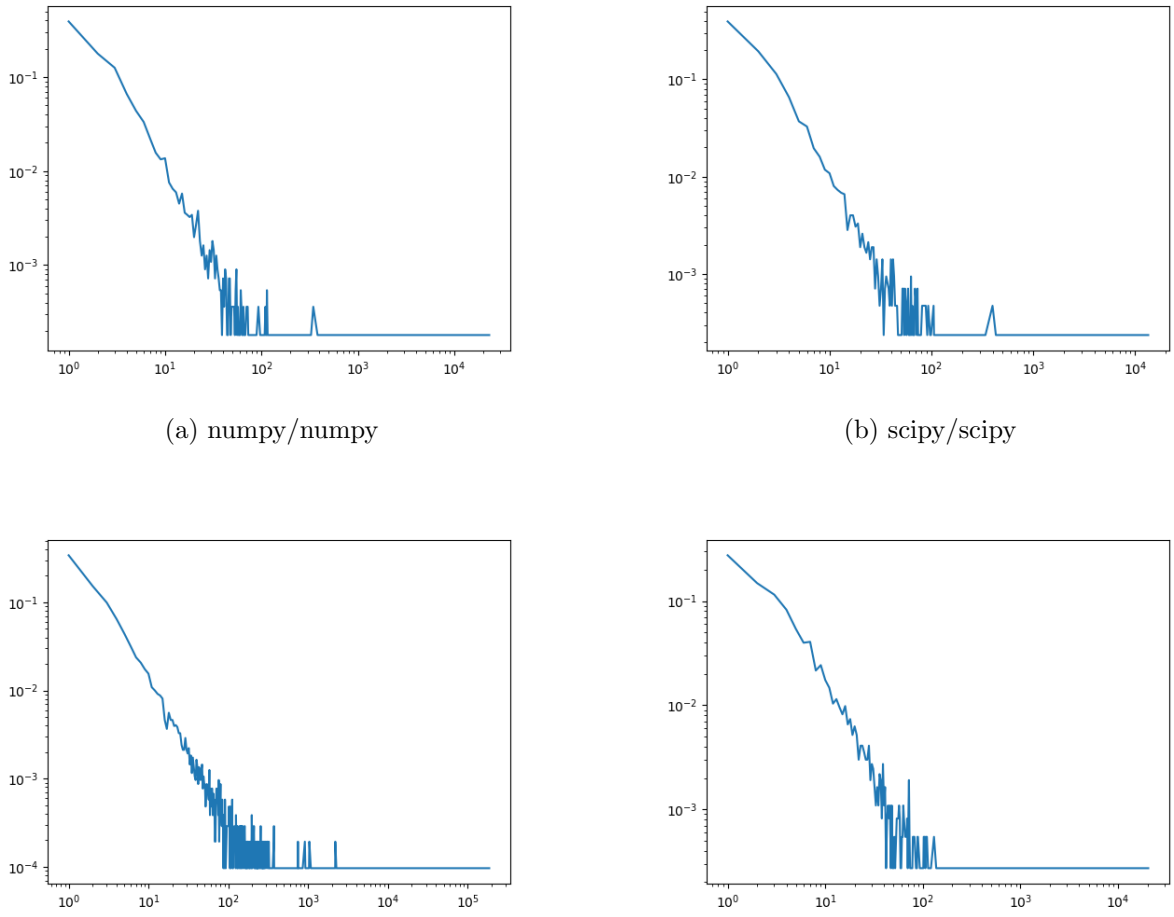
(d) charris

Figure 5: Individual contributions (edge weights) distribution for selected users

removing users (typically bots) with extremely high contributions reduces this size substantially; with these users included, the size is closer to 93%. This differs substantially from the random network which has a giant component with 95% of the vertices. This difference might be due to small local clusters. For instance, maybe a group of users will all contribute to each other's repos, but don't contribute more broadly.

The histogram of component sizes also differs substantially as can be seen in figure 7. We see far more components of moderate size in the actual network. This may be due to the local clustering idea described above.

Within the giant component, the average distance between nodes is 7.34 ± 0.36 using a confidence interval of 2 times the standard error. This is somewhat higher than in the random network which has an average distance of 5.73 ± 0.14. This average distance is approximated by averaging over the average distances of 100 randomly selected nodes. This approximation is needed for computational



(a) numpy/numpy

(b) scipy/scipy

(c) rust-lang/rust

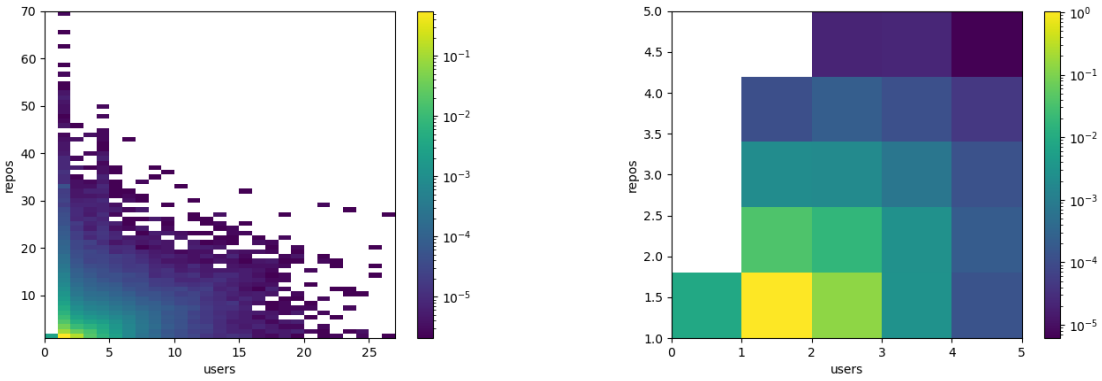
(d) qmk/qmk_firmware

Figure 6: Individual contributions (edge weights) distribution for selected repos

efficiency.

We computed the "pseudo-diameter"[1] of the giant component by starting from a random node, finding the most distant node and then repeating the process from that node until the maximum distance converges. This forms a lower bound of the diameter which should quite close or exact. The pseudo-diameter of the actual network was found to be 71 while the pseudo diameter of the random network was found as 23 (convergence to these values seems to occur consistently). This is a huge difference in the real world network from the expected value. This may indicate the presence of communities which are barely connected to each other and so are relatively far apart. This would also possibly explain the higher average distance.

[1]It seems that the concept of pseudo-diameter originated with Mathematica; I can't seem to find references elsewhere.



(a) Actual network (b) Random network

Figure 7: Component size distribution (excluding giant component)

Projection

In order to construct a one dimensional network from a two-mode network, we can connect nodes which share a common neighbor to form a projected graph for one of the two types of nodes. Consider the example graph in figure 8. If we we apply this projection for users, then we get the graph in figure 9. This allows for applying standard network analysis tools to two-mode networks. Note that in this case we can apply the projection for either user or repos. In the first case, users are connected if they have worked on the same repository while in the second case, repos are connected if they have shared contributors.

As Latapy et al. [1] note, this approach has a number of disadvantages. A large amount of information is lost in the elimination of connections. Also, projection expands the size of the network considerably. The expansion factor is roughly quadratic with the degree of vertices. So if some nodes have very high degree, then the expansion factor will be very large. For this network, naive projection makes the graph impractically large. To develop a more useful notion of projection, we will develop measures for connection strength. Then, the projection can be done using the connection strength as edge weights resulting in a more meaningful graph. Further, we can eliminate edges with weight below some threshold to produce a graph sparse enough for practical use. This naturally leads to an approach for community detection by simply reducing the threshold until distinct communities emerge. We may also use this approach to obtain useful visualizations of parts of the network. Note that these uses extend beyond two-mode networks—just because nodes could be connected doesn't mean that we can't measure connection strength using their neighbors.

Measuring connection strength

We first consider the binary measure of connection strength discussed above: nodes are connected if they share a common neighbor. A natural extension is to instead count the number of neighbors and use these as edges weights, this can be seen in figure 10. Let $N(u)$ be the neighbors of node u. Then

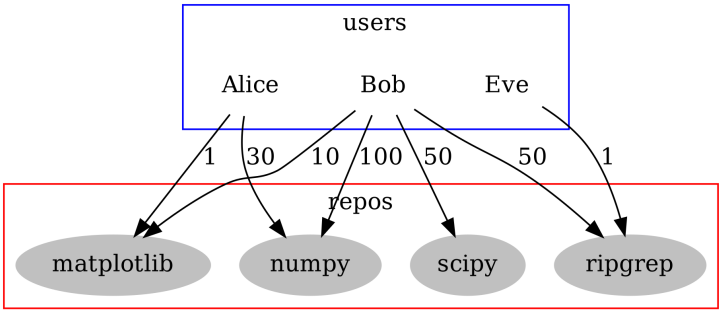


Figure 8: Example graph

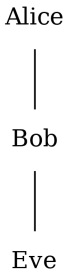


Figure 9: Binary projection of the example graph in figure 8

the count based measure computes connection strength between nodes u and v as $|N(u) \cap N(v)|$. This still disregards the edge weights in the original network. Rather than considering just the number of common neighbors, we can instead apply some function to the edge weights to and from common neighbors. For instance we may wish to consider the geometric mean of the edge weights. Specifically, for a "weight function" f which takes 2 edge weights, we define the connection strength of f between nodes u and v as

$$s_f(u,v) \triangleq \sum_{i \in N(u) \cap N(v)} f(E(\{u,i\}), E(\{v,i\}))$$

where E is the function from edges to their weights. The weight function f should be symmetric over its two input weights. An example for $f(x,y) = \min\{x,y\}$ can be seen in figure 11. Another relevant measure is the geometric mean which is defined over a sequence of numbers $x_1, \ldots, x_n$ to be $\sqrt[n]{x_1 x_2 \cdots x_n}$. So, this in case $f(x,y) = \sqrt{xy}$. The corresponding example can be seen in figure 12. Note that the values are rounded heavily. Also note that counting the common neighbors is equivalent to defining the weight function f to always return 1 (assuming that weight 0 nodes are excluded from the neighborhoods). Alternatively, it is equivalent to the geometric mean when edges have weights in $\{0,1\}$.

Arithmetic mean is a poor choice in this case because it would result in a large jump between a node being disconnected (edge weight 0) and barely connected (edge weight 1). For instance, consider the same example as before with arithmetic mean in figure 13. The connection strength between Alice

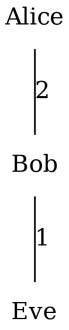


Figure 10: Count projection of the example graph in figure 8

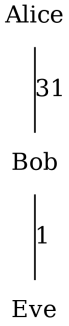


Figure 11: Min edge weight projection of the example graph in figure 8

and Bob is quite large (25.5) given that Alice is barely contributing to the ripgrep repository. If Alice didn't contribute to ripgrep at all, then the connection strength would be 0. If Alice contributed one more time, then the strength would instead be 26 which is barely larger. This is as opposed to geometric mean for which the corresponding progression from 0, 1, and 2 contributions would be strengths of 0, about 7.07, and 10. This is a much more reasonable progression given the meaning of edge weights for this network.

To improve computation speed, edges with weight less than 10 were filtered out of the graph when computing edge strengths. This may not be a good approach to sparsification but it is quite simple to implement.

Distributions of edge strengths using the number of common neighbors, the min, and the geometric mean can be found in figure 14. We can see that the actual distribution of common neighbors is shifted to the right substantially when compared to the random network. Note that this difference is exaggerated by the log-log plot. This indicates that we see far common neighbors than we would expect if all repos and users were treated identically. In other words, this indicates some level of clustering. We also see the actual network is shifted to the right for the geometric mean and min weight functions, but the difference is less extreme. I don't currently have a compelling theory for why this is so much less pronounced.

The corresponding means and variances for repos can also be found in table 1. Note that the

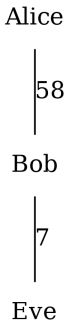


Figure 12: Geometric mean projection of the example graph in figure 8

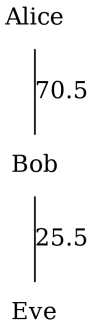


Figure 13: Arithmetic mean projection of the example graph in figure 8

variances are considerable higher in the actual network—this is due to clustering.

With these measures of strength, we can now consider thresholded projection graphs. For instance, see figures 15, 16, and 17. Note that the number of nodes in these figures vary greatly despite using the same thresholds. That's (in part) because the degree distributions of the starting repos and their surrounding repos vary. We could produce more consistent figures simply by adjusting the threshold per repo, but this approach has a number of disadvantages. Specifically, it requires manual tuning and can't be used effectively when low degree clusters connect to high degree clusters. Instead, we will consider a more general approach to normalizing these strength measures.

Measures of clustering

Prior to discussing our notion of normalized strength, we will first consider some of the measures of clustering for two-mode networks from the literature. Various notions of clustering have been defined on pairs of nodes often using the number of common neighbors. For instance, we can consider the number of common nodes between two vertices divided by the total degree of the two vertices. We could instead divide by the min or max degree of the two vertices to obtain somewhat different measures which weight vertices with far apart degrees differently [2].

weight function	number of common nodes	geometric mean	min
actual mean	1.06171	43.413	23.486
configuration mean	1.00030	41.381	24.802
variance	0.27688	22798.278	16171.743
configuration variance	0.00042	12482.147	4394.633

Table 1: Strength mean and variance for a variety of weight functions for repos

Another common approach is to consider a notion similar to the clustering coefficient defined on 1 dimensional networks. Instead of considering the number of completed triangles vs the number of 2 paths, we instead consider the number of completed 4 cycles vs the number of 3 paths [4]. This is because 4 cycles are the smallest possible cycle in a two-mode graph. We can define a local coefficient by dividing these numbers per node. A global coefficient can be constructed by averaging the local coefficients of all nodes. We can also define a number analogous to transitivity by summing each of the these values across the entire graph. It is also possible to extend this notions to weighted graphs by assigning each 2 path a weight and summing over those weights instead of just counting [3].

Normalizing connection strength

A measure of clustering between 2 nodes can be used for community detection and measuring overall clustering in the graph. We will obtain this measure by "normalizing" a connection strength measure in some sense. We aim to do this (roughly speaking) by dividing by the expected strength measure under the configuration model[2] when conditioning on variety of random variables.

Consider distinct nodes u and v of same type (for instance repos). These nodes have neighborhoods $N(u)$ and $N(v)$. We will consider the configuration model in which edge endpoints are paired uniformly at random. For the purposes of this derivation, the effect of edge weights on these pairings is irrelevant and will be ignored. We condition the configuration model on the following values:

- The degrees of u and v: $|N(u)|$ and $|N(v)|$.
- The degrees of the neighbors of u: $|N(i)|$ for all $i \in N(u)$.
- The weights of the edges of u: $E(\{u, i\})$ for all $i \in N(u)$ where E is the function from edges to their weights.
- The weights of the edges of the neighbors of u: $E(\{w, i\})$ for all $w \in N(i)$ for all $i \in N(u)$.

So, in this derivation, the distribution under consideration is that of the configuration model, but with those values fixed (conditioned out). This must be kept in mind when considering expectations and probabilities.

[2]After presenting you asked me what random graph model I was considering and I said preferential attachment (instead of the configuration model). However, thinking about this in terms of the configuration model makes more sense (the formulation is equivalent).

We will first consider the special case when the strength metric is the total number of common edges: $|N(u) \cap N(v)|$. Let Y_i be the indicator random variable that node $i \in N(u)$ connects to node v. By linearity of expectation and the fact that $|N(u)|$ is fixed,

$$\mathbb{E}[|N(u) \cap N(v)|] = \sum_{i \in N(u)} \mathbb{E}[Y_i]$$

Let n be the sum of the degrees of nodes with the same type as u and v. Note that because the network is a two mode network, this is the same as the total number of edges in the graph. Then each of the edges of node i has an approximately $p_v \triangleq |N(v)|/n$ probability of connecting to v. Note that p_v is a fixed value because we condition on the value of $|N(v)|$. We know that i connects to u which removes one of its edges from contention. The probability that i connects to v is just $\approx 1 - (1 - p_v)^{|N(i)|-1}$. Recall that $|N(i)|$ is fixed, so this is a valid probability. So we get that

$$\mathbb{E}[|N(u) \cap N(v)|] \approx \sum_{i \in N(u)} \left(1 - (1 - p_v)^{|N(i)|-1}\right)$$

We will only measure this expectation when 2 nodes are connected in the projection—this is when there is at least one common node, so $|N(u) \cap N(v)| \geq 1$. Thus, we actually need the expectation conditioned on $|N(u) \cap N(v)| \geq 1$.

$$\begin{aligned}
\mathbb{E}[Y_i \mid |N(u) \cap N(v)| \geq 1] &= \frac{\mathbb{P}(Y_i = 1 \cap |N(u) \cap N(v)| \geq 1)}{\mathbb{P}(|N(u) \cap N(v)| \geq 1)} \\
&= \frac{\mathbb{P}(Y_i = 1)}{\mathbb{P}(|N(u) \cap N(v)| \geq 1)} && Y_i = 1 \Rightarrow |N(u) \cap N(v)| \geq 1 \\
&\approx \frac{\sum_{i \in N(u)} \left(1 - (1 - p_v)^{|N(i)|-1}\right)}{\left(1 - (1 - p_v)^{\sum_{i \in N(u)} (|N(i)|-1)}\right)}
\end{aligned}$$

Now we extend this to arbitrary weight functions. Given that node i connects to v, each of its edges has equal probability of being the sole connecting edge under the configuration model. Now recall that we fix the weights of the edges of u and the weights of the edges of i for all $i \in N(u)$. So using the definition of s_f from above:

$$\mathbb{E}[s_f(u, v)] \approx \sum_{i \in N(u)} \left(1 - (1 - p_v)^{|N(i)|-1}\right) \frac{1}{|N(i)| - 1} \sum_{w \in N(i) \mid w \neq u} f(E(\{u, i\}), E(\{w, i\}))$$

Again we must condition this on $|N(u) \cap N(v)| \geq 1$. We can just divide by the same denominator. So the overall approximation is

$$\frac{\sum_{i \in N(u)} \left(1 - (1 - p_v)^{|N(i)|-1}\right) \frac{1}{|N(i)|-1} \sum_{w \in N(i) \mid w \neq u} f(E(\{u, i\}), E(\{w, i\}))}{\left(1 - (1 - p_v)^{\sum_{i \in N(u)} (|N(i)|-1)}\right)}$$

Note that this approximation is asymmetric, it doesn't condition on the degrees of the neighbors of v or the edge weights of these neighbors. So we compute the same approximation considering the neighborhood of v instead of u and average these two values (using linearity of expectation once again). Note that we could instead fix all of the known values at once and compute a single expectation from that. However, this is considerably harder to compute and requires reasoning about weighted edges in the configuration model.

An alternative approach for thinking about this approximation is to consider this as approximating any other random graph model in which edges are independently selected. In that case, this is equivalent to using the empirical values as an estimate for the actual values. For instance, p_v is the empirical probability and

$$\frac{1}{|N(i)|-1}\sum_{w\in N(i)\mid w\neq u} f(E(\{u,i\}),E(\{w,i\}))$$

is the empirical expectation (mean).

This approximation can be computed reasonably efficiently by pre-computing the values of the sum of the weight function. We may also combine terms which have the same value of $|N(i)|$ by summing their two weight function sums. Only the value of p_v is unknown for each u for the purposes of pre-computation.

There are several approaches to analyze clustering based on normalized strength measures. We could use the sum of the unnormalized strengths (the numerator) divided by the sum of the expectations (the denominator). This approach has the advantage of having a value greater than 1 indicate clustering regardless of the degree distribution or size of the graph. We could also consider the average normalized strength or the average squared normalized strength; however, the thresholds for these will depend on the graph, so comparison to a random model would be needed.

Distributions of normalized strength can be found in figure 18. The corresponding means and variances for repos can also be found in table 2. As expected, the mean of the expectation is very close to the mean of the strength for the configuration model. In the actual network, the mean of the strength is considerably larger than the mean of the expectation which indicates clustering (the ratio is larger than 1). We can also see this in the distribution plots.

weight function	number of common nodes	geometric mean	min
actual strength mean	1.06171	43.413	23.486
actual expectation mean	1.00025	39.556	23.923
configuration strength mean	1.00030	41.381	24.802
configuration expectation mean	1.00035	41.388	24.805

Table 2: Strength (unnormalized) mean and the mean of the estimated expectation for a variety of weight functions for repos

With these normalized measures of strength, we can now consider thresholded projection graphs which are based on clustering and should be partially scale invariant. For instance, see figures 19, 20, and 21. We can see that the number of repos reached remains more constant for each of these starting points (numpy, rust, and neovim) than when using unnormalized measures of connection strength (as in figures 15, 16, and 17). When we do see large clusters, that seems to be due to actual graph

structure. For instance, 19 contains a cluster around the pandas (middle left) which seems to consist of repos related directly to pandas based on the names.

Interestingly, this measure of clustering results in forks which are only used for pull requests being strongly clustered which may or may not be desirable. Specifically, a common pattern on GitHub is to fork a repo to your account, make commits to the fork, and then pull request the repo using the fork. Typically, the forked repo will only be committed to by the owner and possibly a few other users. We can see that this pattern can result in high clustering as it shows up in these sampled subgraphs. For instance, consider the charris/numpy repo which is connected to the numpy/numpy repo. This repo is where the user charris commits when making a pull request to the actual numpy repo. So, the connection strength between these repos will depend on how much charris contributes to numpy. They are a major contributor, so this connection strength is reasonably large (it is around 9538 when using the geometric mean). However, the connection strength using the neighbors of charris/numpy will mostly depend on the typical edge strength of charris (only a few other users are even weakly connected to this repo). This value will be much lower as charris contributes much less to other repos. The expected connection strength using the neighbors of numpy/numpy will depend on the typical edge strength of all of its contributors. So this will also be much lower than the actual connection strength from charris/numpy. Thus, both of these expectations will greatly underestimate the true connection strength resulting in high clustering.

Normalized connection strength seems to be a decent measure of clustering. This metric allows for distinguishing between real networks with considerable clustering and random networks. It also seems to reflect actual underlying clustering based on looking at the projected subgraphs which threshold edges by normalized connection strength.

Conclusion

The GitHub network differs substantially from a (naive) random network model. It has greater average distance and substantially higher diameter in the giant component than in the random case. We also see moderate sized components which aren't present in the random model. Further, the giant component is smaller. However, the actual network has higher clustering than the random model as seen in a variety of ways. This could be explained by densely connected communities which are more sparsely connected with each other leading to high distances.

We also investigate approaches for measuring edge strength and create a measure of normalized edge strength. We find that this measure of normalized edge strength is a useful measure of clustering and can also be used to produce meaningful figures showing how repos are connected.

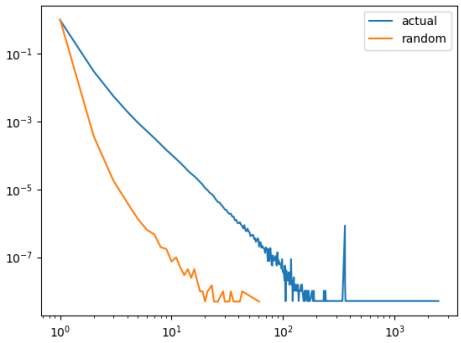
References

[1] Matthieu Latapy, Clémence Magnien, and Nathalie Del Vecchio. "Basic notions for the analysis of large two-mode networks". In: *Social Networks* 30.1 (2008), pp. 31–48. ISSN: 0378-8733. DOI: https://doi.org/10.1016/j.socnet.2007.04.006. URL: https://www.sciencedirect.com/science/article/pii/S0378873307000494.

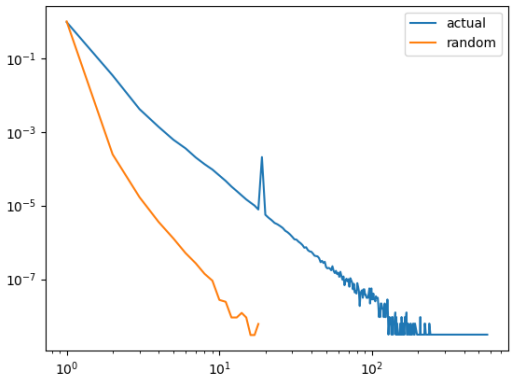
[2] Stevens Le Blond, Jean-Loup Guillaume, and Matthieu Latapy. "Clustering in P2P Exchanges and Consequences on Performances". In: *Peer-to-Peer Systems IV*. Ed. by Miguel Castro and Robbert van Renesse. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 193–204. ISBN: 978-3-540-31906-1.

[3] Tore Opsahl and Pietro Panzarasa. "Clustering in Weighted Networks". In: *Social Networks* 31 (May 2009), pp. 155–163. DOI: 10.1016/j.socnet.2009.02.002.

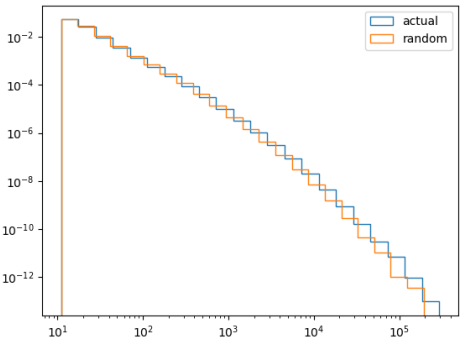
[4] Garry Robins and Malcolm Alexander. "Small Worlds Among Interlocking Directors: Network Structure and Distance in Bipartite Graphs". In: *Computational & Mathematical Organization Theory* 10 (May 2004), pp. 69–94. DOI: 10.1023/B%3ACMOT.0000032580.12184.c0.



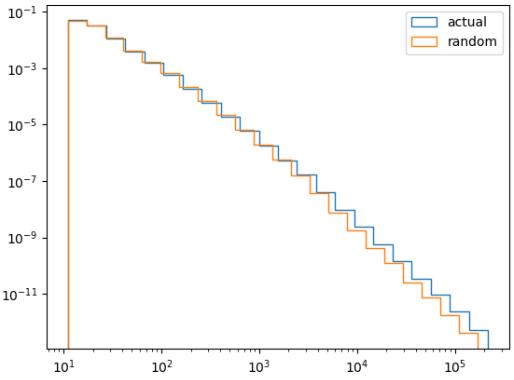
(a) Users, number of common neighbors (count)



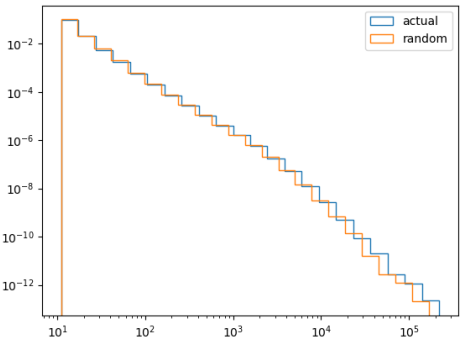
(b) Repos, number of common neighbors (count)



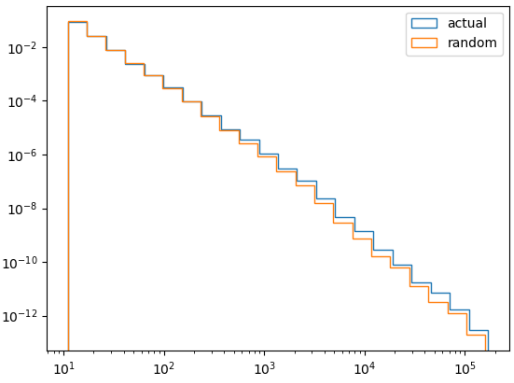
(c) Users, geometric mean



(d) Repos, geometric mean



(e) Users, min



(f) Repos, min

Figure 14: Strength distribution for a variety of weight functions for both users and repos

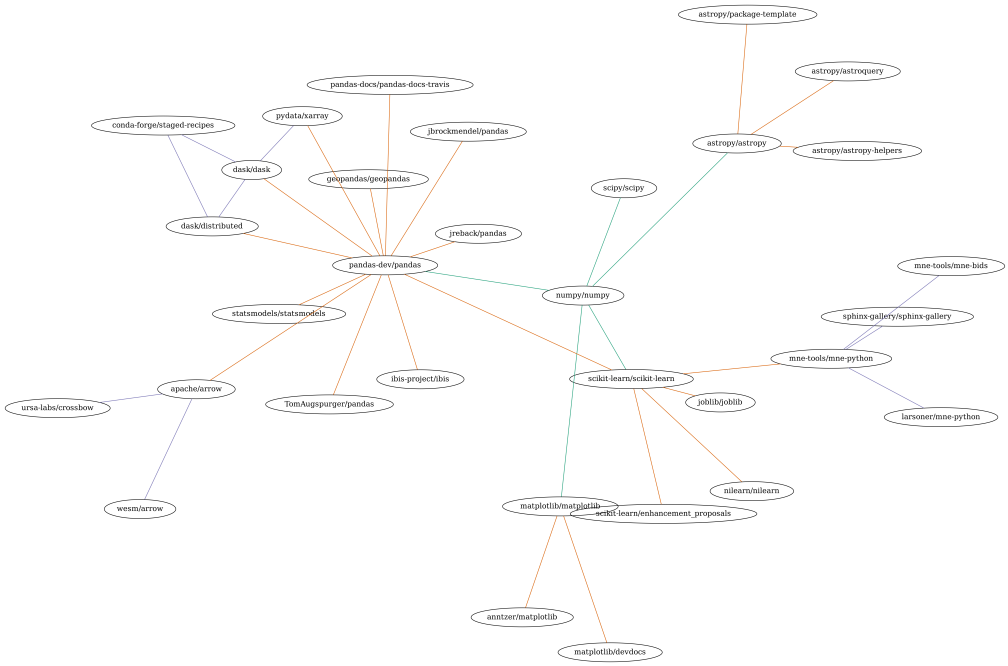


Figure 15: Subgraph found by traversing outward 3 steps from the numpy repo with geometric mean edge strength thresholded to be above 10000

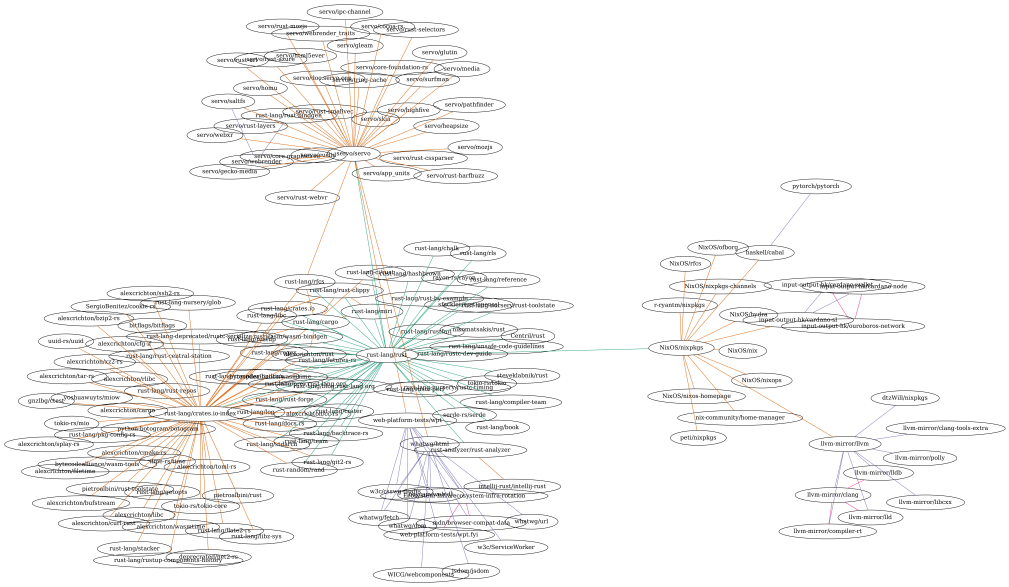
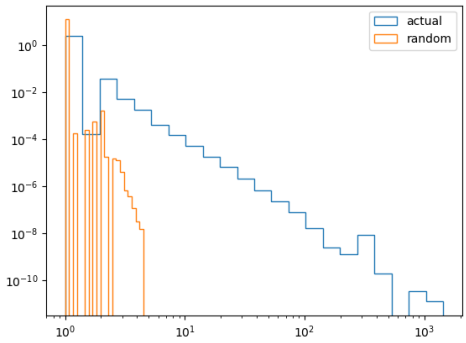
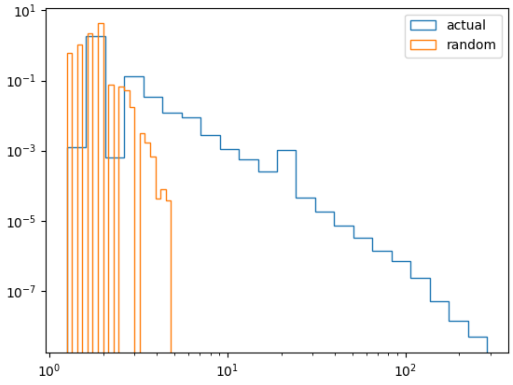


Figure 16: Subgraph found by traversing outward 3 steps from the rust repo with the geometric mean edge strength thresholded to be above 10000

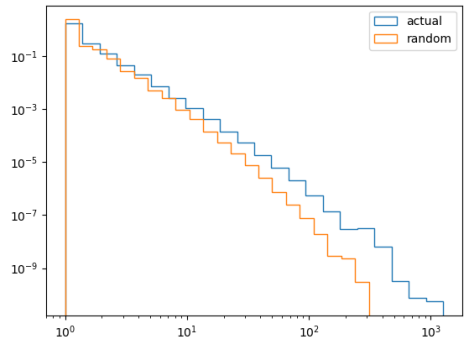
Figure 17: Subgraph found by traversing outward 3 steps from the neovim repo with the geometric mean edge strength thresholded to be above 10000



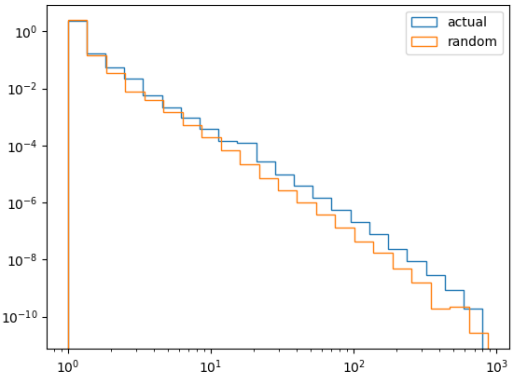
(a) Users, number of common neighbors (count)



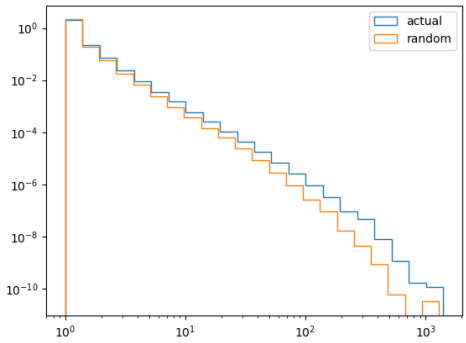
(b) Repos, number of common neighbors (count)



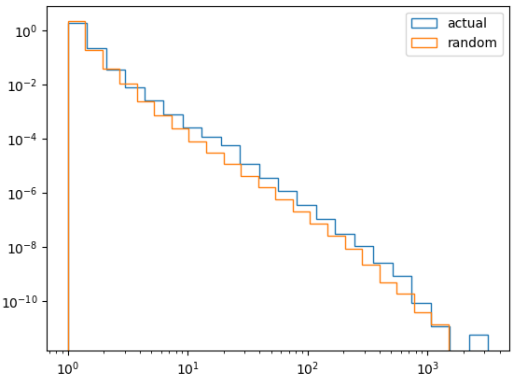
(c) Users, geometric mean



(d) Repos, geometric mean



(e) Users, min



(f) Repos, min

Figure 18: The distribution of normalized strength for a variety of weight functions for both users and repos

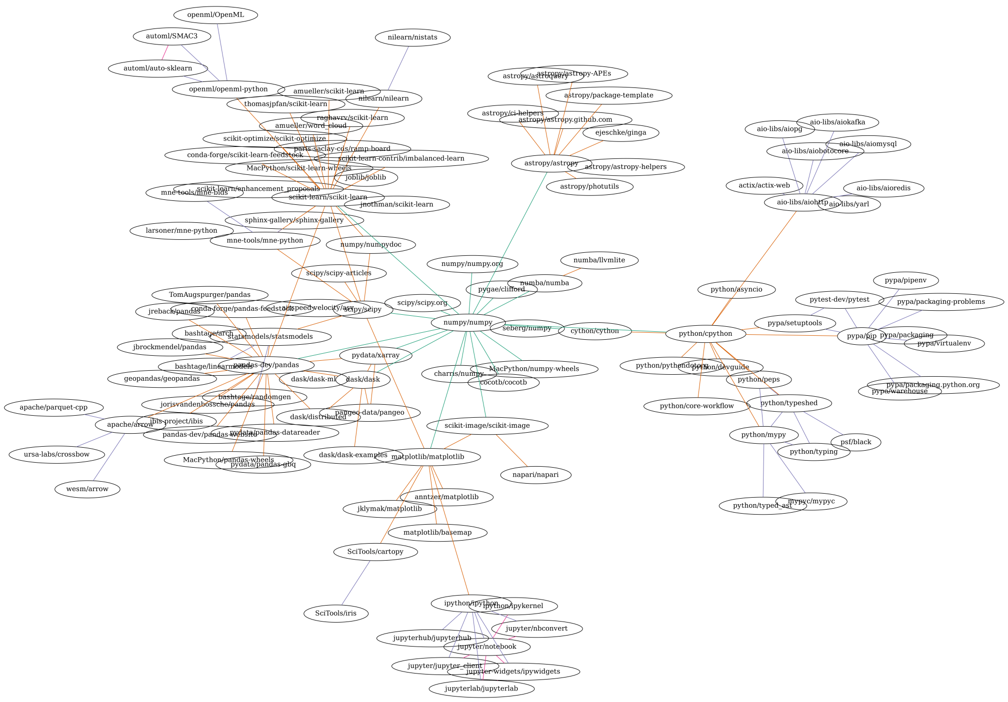


Figure 19: Subgraph found by traversing outward 3 steps from the numpy repo with normalized geometric mean edge strength thresholded to be above 70

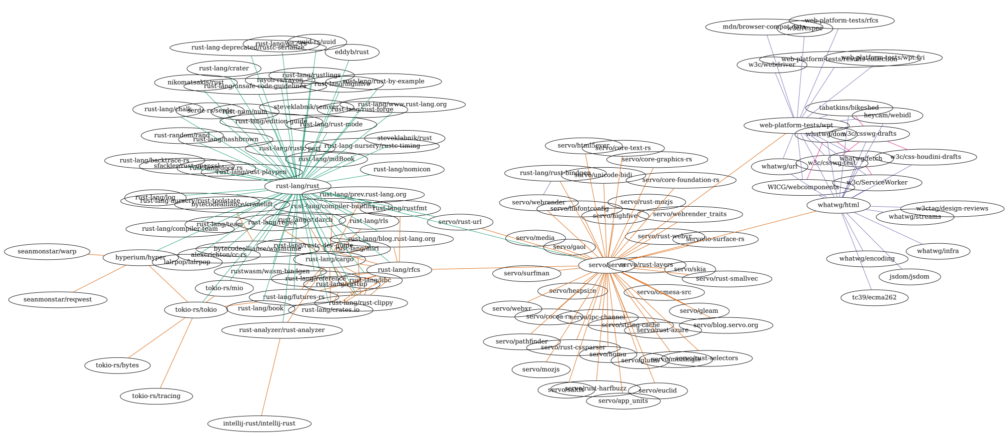


Figure 20: Subgraph found by traversing outward 3 steps from the rust repo with the geometric mean edge strength thresholded to be above 70

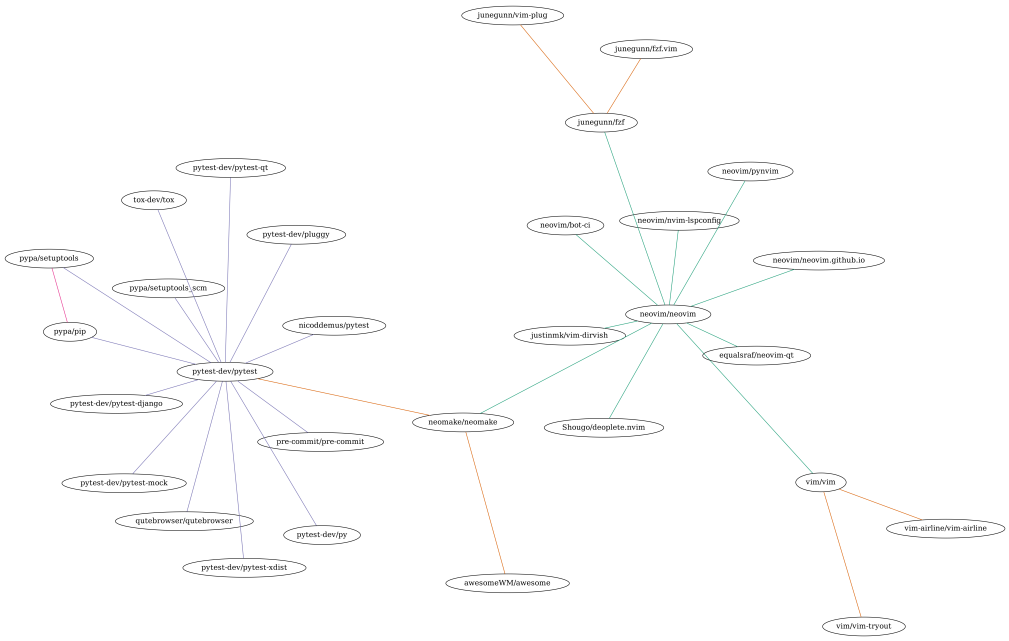


Figure 21: Subgraph found by traversing outward 3 steps from the neovim repo with the geometric mean edge strength thresholded to be above 70