Network Analysis on GitHub programming languages

Isabella Marasco 1040993
Francesco Saverio Beccafichi 1052139

Master's Degree in Computer Science
A.A. 2022/2023

Abstract

Github is one of the largest social coding communities on the internet, hosting millions of programming projects in a wide range of languages. This paper proposes a study aimed at understanding which programming languages are most commonly used on the platform and the most common combinations. Additionally, we explore users' propensity to experiment with new technologies and form communities that share similar behaviors and aspects.
The analysis results show that the most commonly used programming languages are Ruby and JavaScript, which also represent one of the most commonly used combinations. Furthermore, there is a tendency among programmers to experiment with various technologies and form communities.

1 Context

The analysis of this project is based on Github a collaborative development platform for versioning and hosting source code. Developers can host their projects, collaborate with other developers, and contribute to open-source projects. As one of the largest social coding communities on the Internet, GitHub hosts millions of projects covering a wide range of programming languages.
The platform also provides tools for version control, issue and pull request management, code documentation, and automated testing.
The analysis of this project aims to understand which programming languages are most widely used on the platform and how one language is combined with others to complete various projects, as well as to be interested in other aspects such as the propensity of users who frequent the platform in experimenting with new technologies that are proposed or, to try to understand if there are any repositories that most influence personal projects.
Thus, the main context of the study is the field of computer science, especially the programming languages that are most widely used by the computer science community.

2 Problem and Motivation

The problem concerning the network analysis is understanding which programming languages are most popular and most used within the platform. Such a study can be of interest in understanding which languages are currently the most widely used and probably most in demand in the job market, to target one's training on the languages used in the job areas of greatest interest.

Lastly, it can indicate whether there is an active community in certain languages rather than others.
Starting from this problem, we want to go into the analysis of how the use of one language can influence or can be influenced by the use of other languages, and this can provide a deeper understanding of the dynamics of developer communities on GitHub. For example, if several projects use Python and JavaScript together, this could indicate that these languages are often used together for web application development.
Our research aims to explore whether programmers tend to specialize in a particular programming language or use multiple languages. This information can provide valuable insights into developers' habits and help us better understand the job market. By identifying the skills that are most in demand, we can guide people who want to stay competitive in the industry. Our research has led us to identify two main types of developers: those who specialize in one or a few programming languages, resulting in an active and highly specialized community, and those who explore a variety of languages, creating communities that care about the latest technologies.
A further study we want to do is to understand if there is a correlation in repositories between the most used programming languages and the number of stars and forks a project has.
Finally, we want to understand if there is a motivation for a low presence of topics associated with repositories, perhaps due to their widespread use predominantly in certain domains, and whether their use by one user may affect their use for other users who may be collaborated on a project.

3 Datasets

The dataset used was constructed using Github's REST API [1] Python [4] and NetworkX [2] will be used for data manipulation and representation. The dataset will consist of three types of nodes, the first one will identify the repositories that are created by the users for the implementation of their projects, the second one will represent the various programming languages identified in the platform, and the last type of node will represent the users who frequent the platform. The nodes representing repositories will have two attributes, the first will indicate the number of stars the project has received from external users who have viewed it, and the second will serve us to understand the number of forks that have been created for a given repository. These two attributes will be critical for making assessments within the network so that we can understand the importance of repositories.
The arcs that will connect our nodes represent two types of relationships, the first connects the repository nodes to the programming language nodes according to the relationship: *"in the repository, the language was used"* while the user nodes and the repository nodes will be connected using the relationship: *"created the repository"*.
The dataset used will be only one, in a table format, but it will be considered in subsections according to the type of analysis to be performed.

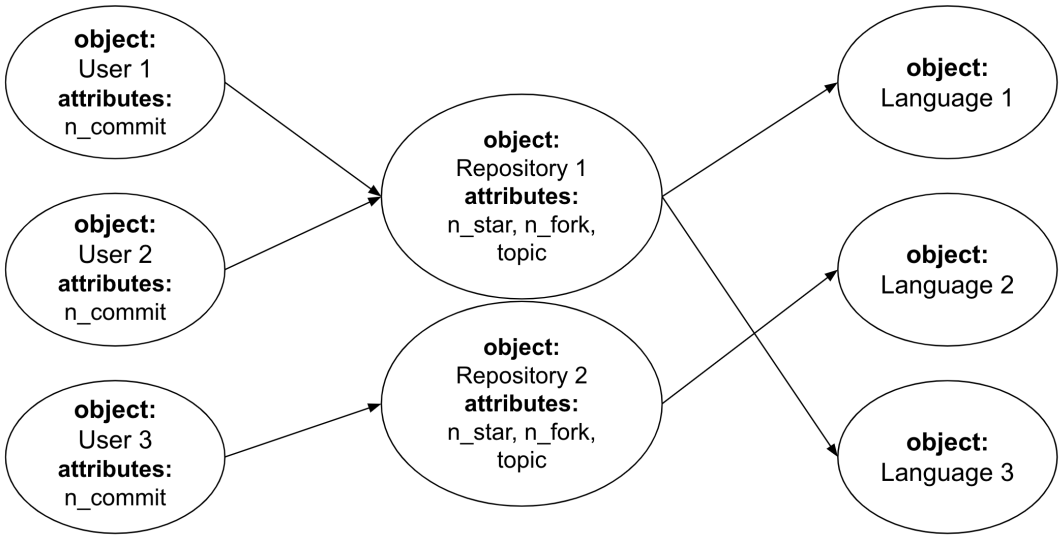


Figure 1: Small representation of the dataset

4 Validity and Reliability

The reliability and validity of the dataset should be reasonably consistent since it was obtained by downloading data from GitHub using the PyGitHub[3] framework, which simplifies the interaction with GitHub API[1] and allows downloading of the data needed to perform this study.
In addition, to reinforce the reliability and validity of the dataset, the selection of repositories, was realized by setting a seed equal to 33, which was used by the GitHub API to extract 5000 repositories. Later, from these 5000 repositories we decided to extract, again randomly, a subset of 2000 repositories using the same seed set initially. The use of the seed, ensures us that the same random selection of repositories is generated each time the code is run, this aspect is important for replicability and consistency in the search.
So, the dataset used within this study consists of 2000 repositories. The network consists of 1499 nodes and 2369 edges, a very small portion if we compare it to all the content available on GitHub. The latter issue we could have solved by downloading more data but, we decided not to do so to avoid the problem of network explosion, which in turn could have led to problems during network analysis. This is because if networks are too large they can lead to insignificant or even misleading results, as there may be too many connections and patterns to fully analyze and understand.

5 Measures

The measures applicable to our dataset could be numerous; the following are the ones we will apply for our analysis:

- **Degree centrality**: measures the number of incoming and outgoing connections of a node within the network. Nodes with a higher degree of centrality are considered more important within the network as they have more connections to other nodes. In the case of the GitHub repository dataset, degree centrality can be used to identify the most popular

programming languages and the most active and influential users on the platform. In particular, it can identify:

  - *The most important nodes in the network*: that is the one with the most connections to other nodes. These nodes can be considered influential within the GitHub platform, indicating which programming languages are most widely used in the platform.
  - *Border nodes*: the nodes with low degree centrality are less connected to other nodes in the network. However, they can be important because they can represent the boundaries between different clusters or communities within the network. In the case of the GitHub repository dataset, boundary nodes may represent programming languages or communities that are less connected to the rest of the platform.
  - *Identifying communities*: degree centrality can be used to identify communities of users and repositories on the GitHub platform. Communities are groups of nodes that are highly connected to each other and with fewer connections to nodes outside the group. In the case of GitHub's repository dataset, degree centrality can help identify communities of users and repositories that deal with similar projects or use the same programming languages.

- **Betweenness centrality**: is defined as the number of shortest paths between all pairs of nodes in the network that pass through that node. This metric can be used to identify users who may have a high potential for information dissemination within the network.

- **Modularity**: this metric indicates how much the network is divided into distinct groups (modules), each of which contains nodes that are highly connected to each other and less connected to nodes in other groups.
  Users who contribute to repositories of only one programming language and who are grouped into distinct modules will have a high degree of modularity, while those who contribute to repositories of different programming languages and who are distributed among different modules will have a low degree of modularity. This value may indicate to us, if high, that the network is divided into distinct communities that are loosely interconnected with each other, otherwise, the network will be less structured and divided.

- **K-means clustering**: groups programming languages into clusters based on the number of contributors or the number of users following the repository.
  This allows us to identify programming languages that are most popular or growing rapidly in popularity on the GitHub platform. In addition, it could be useful to identify communities of users who use the same programming languages or collaborate on similar projects.

- **Structural equivalence**: a measure of similarity between nodes in a graph based on their structural position within the graph. Specifically, two nodes are structurally equivalent if they are connected to similar nodes in the graph, regardless of the nature of the nodes themselves.
  In the context of programming languages on Github, it could be used to identify programming languages that are often used together in Github projects, suggesting some affinity between them. This information can be useful in understanding trends and dynamics in the programming world and in identifying relationships between different programming languages.

- **Cliques**: are groups of highly interconnected nodes, where each node is connected to all other nodes in the group, so they are used to identify groups of nodes that are closely related to each other.
  In our particular study, it can be used to understand the intensity of interactions between programming languages. In addition, clique counts can help us identify the most important languages and their combinations on the platform

- **Cores**: are groups of nodes in which each node has a specified minimum degree, for example, a degree three core is a group of nodes in which each node has at least three incident arcs. Cores are used to identify the most central and influential nodes in the graph, and are often used for centrality analysis.
  In the case of Github, we could use them to identify the most central projects that tend to have more stars and forks than others.

- **Components**: are sets of nodes that are all connected, but not necessarily connected to the rest of the graph. This measure is used to identify groups of nodes that are isolated from the rest of the graph, and may represent parts of the graph that are distinct and separate from the rest of the graph. Used to identify groups of projects that are isolated or distant from the rest of the graph.
  In this case we will want to use it to see if there is any correlation between the few users who use topics within their repositories.

- **Homophily**: is a concept used to describe the tendency of some individuals to associate with other individuals similar to them based on characteristics they have in common.
  In this study, it will be used to understand whether users who frequent the platform tend to collaborate with other users who develop projects similar to each other, e.g., use the same language, and also to see if they are influenced in using parameters provided by GitHub for repositories, so in our case try to understand whether connected users use topics in repositories.

- **Statistical homogeneity**: calculates how similar the objects within a cluster are to each other. In the case of programming languages used on GitHub, it might be useful to identify the percentage of repositories that use the same combination of programming languages.

- **Small-worldness**: is a property of complex networks that occurs when a graph exhibits both strong localization (clustering) and a short average distance between nodes (short path length).
  In the context of programming language studies, it can be used to identify whether there are programmers within the network who are more central than others and whether they are exclusively so for a single project or a group of projects. In the case where we find people who turn out to be central to several projects, it will be possible to extend the study to the type of programming languages being used, by going to see whether the projects that are taken care of by a programmer have similar characteristics or tend to vary.

- **Scale-free**: a network is considered scale-free if it follows a power-law degree distribution, that is, if there are few highly connected nodes (hubs) and many nodes with few connections. Calculating the scale-free distribution can be useful in understanding the structure of the network and its organization, it is a probability distribution in which some entities have many connections and others have few connections. In a network, this means that some repositories or programming languages might be very popular and

have many connections to other nodes, while others might be less popular and have fewer connections.
Understanding the scale-free structure of the network can help identify important and influential nodes within the network, such as repositories that have been forked or starred many times or programming languages that have been used in many repositories. This can help identify trends or usage patterns of programming languages and repositories.

6 Results

In this section we are going to illustrate all the results obtained from the analyses performed with the measures, as described in Section 5.
During analysis, we assigned different colors to the user, repository, and languages nodes yellow, blue, and light blue respectively.

6.1 Degree centrality

We calculated degree centrality only on the language and repository nodes since our goal was to understand which programming languages are most widely used in GitHub projects. The programming languages were sorted in descending order based on their usage within the repositories. The figure 2 and Table 1 below shows the top 10 most used languages within the repositories.

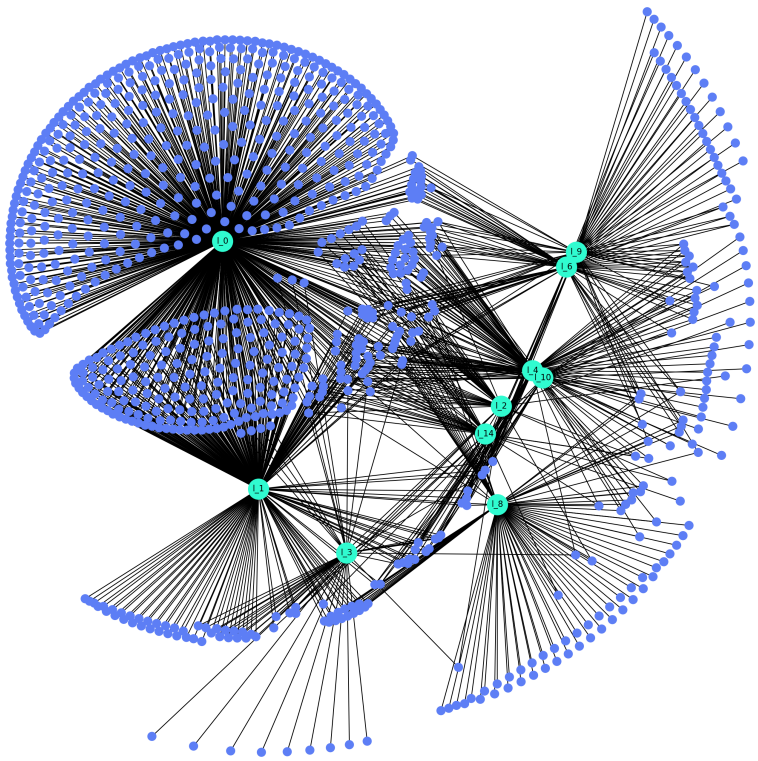


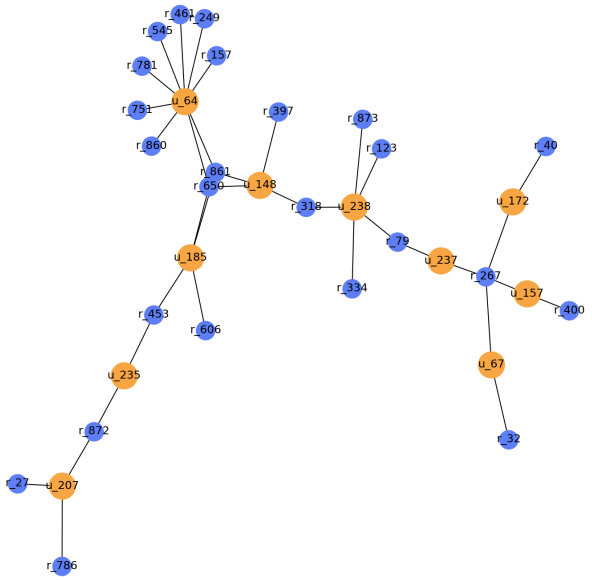
Figure 2: Degree centrality of languages

id	Language	Degree Centrality
l_0	Ruby	0.48664886515353806
l_1	JavaScript	0.2096128170894526
l_4	Shell	0.0801068090787717
l_8	Python	0.06475300400534045
l_6	C	0.058077436582109475
l_9	C++	0.03271028037383177
l_10	Perl	0.032042723631508674
l_14	CSS	0.030707610146862484
l_2	HTML	0.030040053404539385
l_3	PHP	0.028037383177570093
Mean		0.013930662128258938

Table 1: Degree Centrality of Languages

6.2 Betweennes centrality

For calculate betweennes centrality, we created a subgraph that included only user nodes and repository nodes. The latter were connected only if two users had at least one shared repository between them. This subgraph was created to understand which users could have a high potential for information diffusion in the network.
The results obtained were sorted in descending order to understand which users could have a greater influence within the network. Figure 3 shows the top 10 users who obtained the highest value of betweenness centrality.



id	Betweennes Centrality
u_238	0.01396
u_148	0.01338
u_237	0.01145
u_185	0.01029
u_235	0.008921
u_64	0.005672
u_67	0.005507
u_157	0.003469
u_172	0.002368
u_207	0.001211

Figure 3: Betweenness centrality of users

6.3 Modularity

We calculated the modularity using the entire graph, therefore all the nodes (users, repositories and languages) to understand if users tend to always use the same languages or to vary, indicating how much the division into communities of the graph is better than what we would casually expect.
This metric returns a value between -1 and 1 and during our analysis we obtained a value of 0.55, which indicates the presence of some structure within the graph that suggests users' specialization in certain programming languages.

6.4 K-Mean Clustering

Regarding the k-means clustering, we used the same subgraph generated to calculate the betweenness centrality, as our goal was to identify communities of users who tend to collaborate on similar projects, for example, using the same programming languages. The value of k used was 3, allowing us to generate three different clusters in which to group the values of the dataset. Before populating the clusters, it was necessary to calculate the adjacency matrix, which represents the relationships between the various nodes of the network, indicating whether two nodes are connected or not. The adjacency matrix was calculated because our data is represented by a graph, so the distance between nodes is not directly defined but is defined based on the structure of the network. Finally, we populated the three clusters created through the *kMeans* function.
In the figure 4, we can see an initial version of the clusters found by the function, which identifies us users collaborating in similar projects.The presence of many users without connections demonstrates that there are several cases where programmers who use uncommon programming languages or tend to work individually exist. This identifies users who collaborate on similar projects.

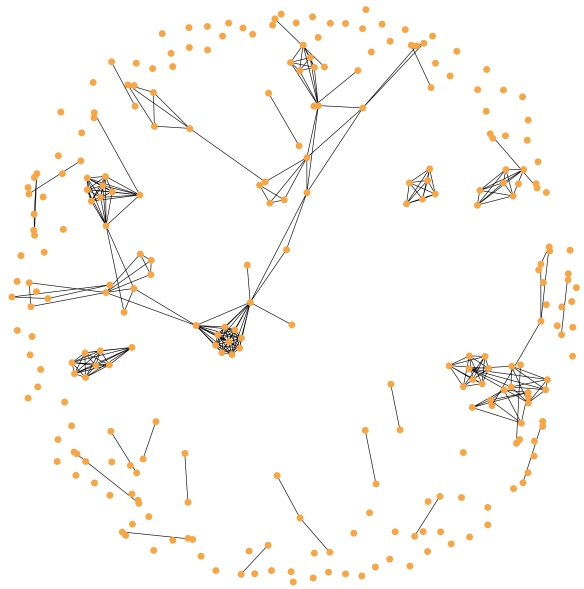
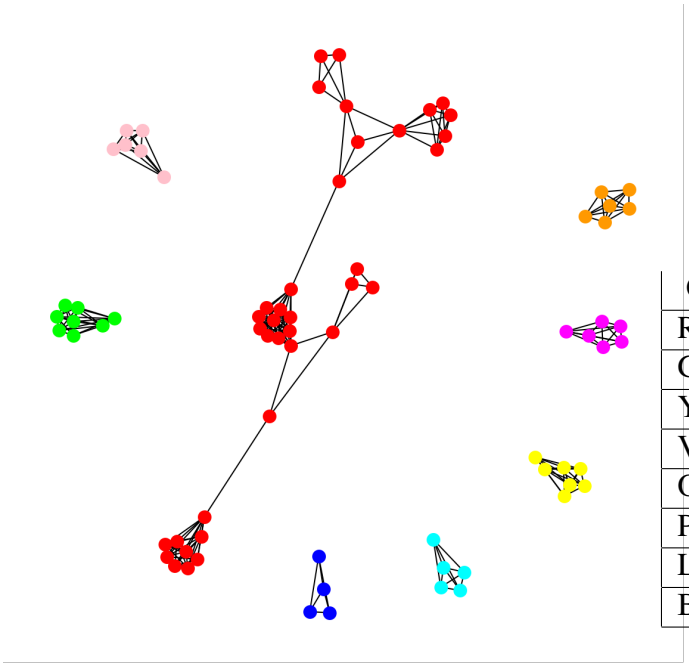


Figure 4: Communities of users collaborating in similar projects.

In this case, as we want to find communities of users who collaborate on similar projects, we decided to refine our clusters by selecting only nodes with a degree greater than or equal to three, thus excluding users who collaborate only in one repository, as they do not create a community but only a collaboration.
In the figure 5, we can identify 8 different communities (identified by different colors), most of them consisting of about 6 users, so this leads us to think that these users share one or more repositories concerning similar projects, this aspect is also suggested by the fact that each node is strongly connected with the rest of the nodes in the subgroup. On the other hand, the situation is different for the largest community in our network (red component) which is made up of 37 nodes and not all of them are strongly connected, this aspect shows us that the communities formed in the network are not only small groups of users but also users collaborating on independent projects but using similar languages, in fact in this component we can see some users allowing different groups to join together and form a larger community.

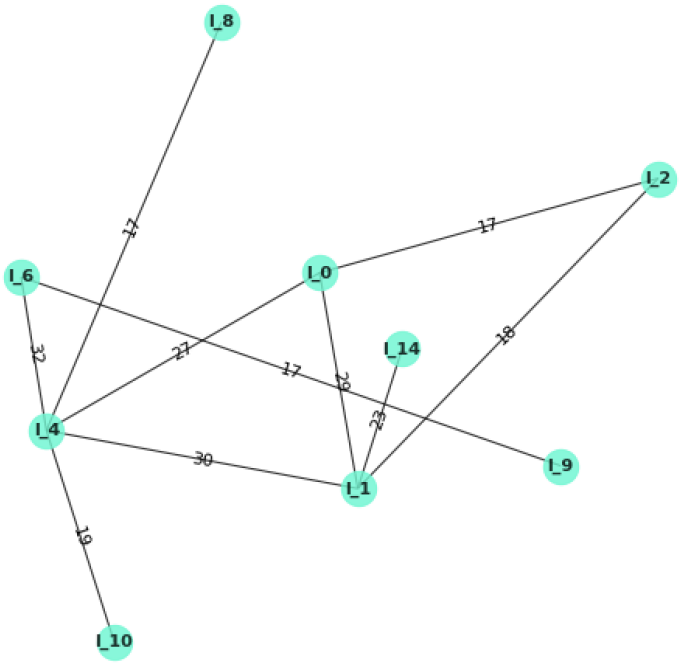


Color	Number of user node
Red	37
Green	8
Yellow	7
Violet	6
Orange	6
Pink	6
Light Blue	5
Blue	4

Figure 5: Clusters formed by users collaborating in similar projects, with node degree greater than three

6.5 Structural Equivalence

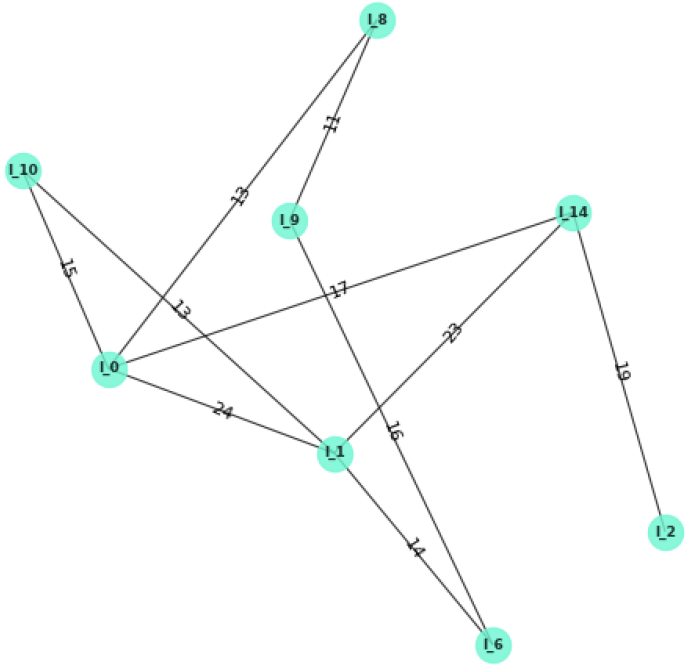
Structural equivalence was calculated to understand which programming languages are often used together in different projects. Thus, if the same languages are used within two or more repositories then the repositories are considered structurally equivalent.
In the Figure 6, we can look the 10 most used combinations of programming languages in different repositories and on the edges we can see the number of times that combination of languages has been used together.



id	Language
l_0	Ruby
l_1	JavaScript
l_2	HTML
l_4	Shell
l_6	C
l_8	Python
l_9	C++
l_10	Perl
l_14	CSS

Figure 6: Structural equivalence of languages

In Figure 7, we have the graph created without taking Shell into account, we decided to do this because being a scripting language it is mainly used to automate commands and set up the working environment.



id	Language
l_0	Ruby
l_1	JavaScript
l_2	HTML
l_6	C
l_8	Python
l_9	C++
l_10	Perl
l_14	CSS

Figure 7: Structural equivalence of languages without Shell

6.6 Cliques

To understand which language combinations were most commonly used in our dataset, we used this metric because it can identify a group of nodes that are all connected to each other. By applying this metric to the same subgraph used for structural equivalence, we were able to identify which types of programming language combinations were most commonly used in the data we have available. In the figure 8, we can see various combinations of languages.

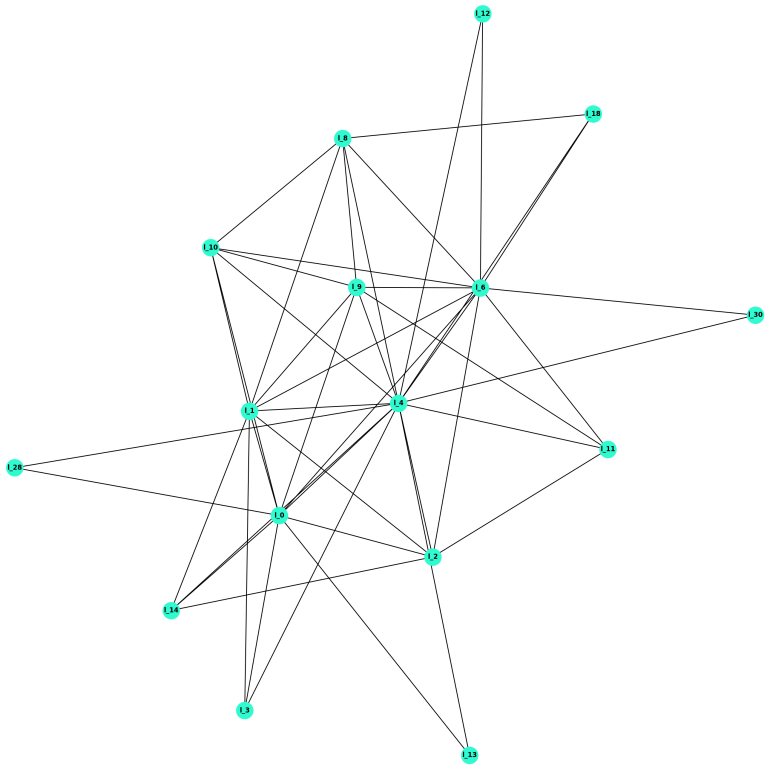


Figure 8: Cliques for languages combinations

We can easily notice that the 'l_4' language is used in all the language combinations we can identify in our graph. However, since this is a scripting language, mainly used to set up the working environment correctly, we decided to exclude it to see how the cliques vary. In the figure 9, we can see the new result obtained.

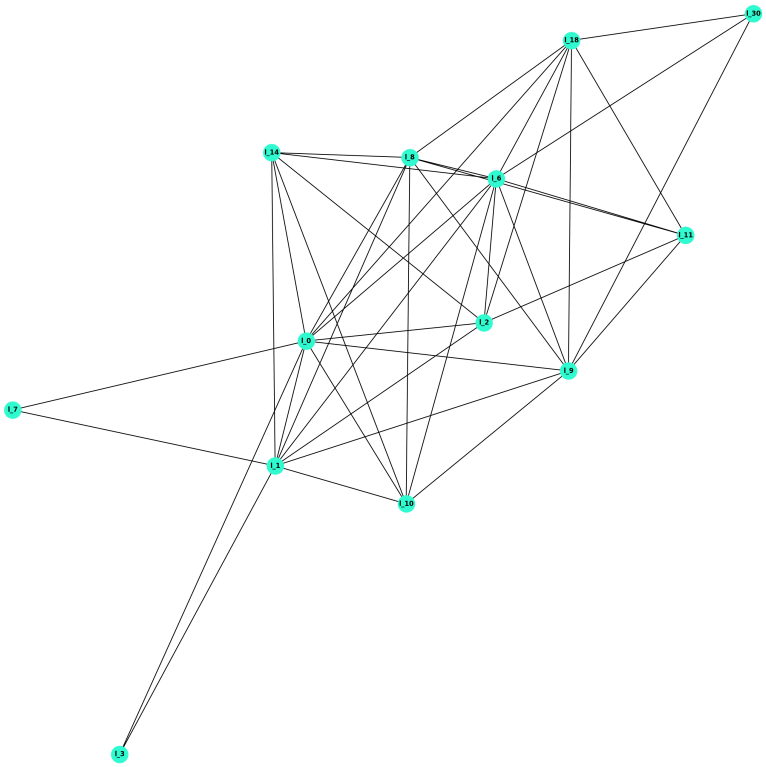


Figure 9: Cliques for languages combinations without languages node 'l_4'

6.7 Cores

A core is a maximal subgraph in which each node has at least the rank of the core, but not necessarily the same rank as the other nodes within the core. In this analysis, we computed it, using a subgraph consisting of only user nodes linked together if they have the same number of stars, to identify groups of repositories that are similar in terms of popularity.
In Figures 10 and 11, we can visualize the graphs that are obtained with the first two most interconnected k-cores. On each graph is indicated the number of stars that have all repositories connected to each other.
In Figure 10, where nodes with cores with k=260 are visible, we can see the presence of a single graph composed of several nodes related to each other by at least 260 edges where, all mutually connected repositories have a number of stars equal to 3. Next to the graph, in Table 2 we can visualize the 5 most used languages within the repositories.

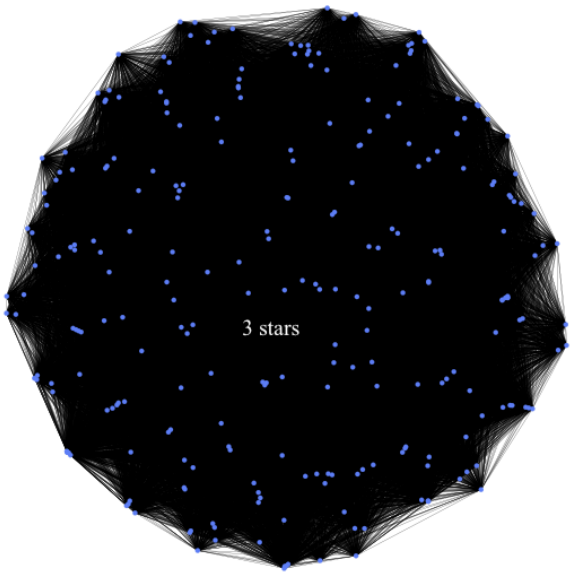


Figure 10: Core with k = 260

id	Language
l_0	Ruby
l_1	JavaScript
l_4	Shell
l_8	Python
l_6	C

Table 2: Top 5 most used languages

In Figure 11, where nodes with cores having k=113 are visible, we can see the presence of three subgraphs with different star numbers. Thus, we can see that there are several groups of highly interconnected repositories within the network with different star numbers but, visible in Table 3, using the same languages. Furthermore, we can see that they are also the same programming languages used in the repositories found with cores with k=260.

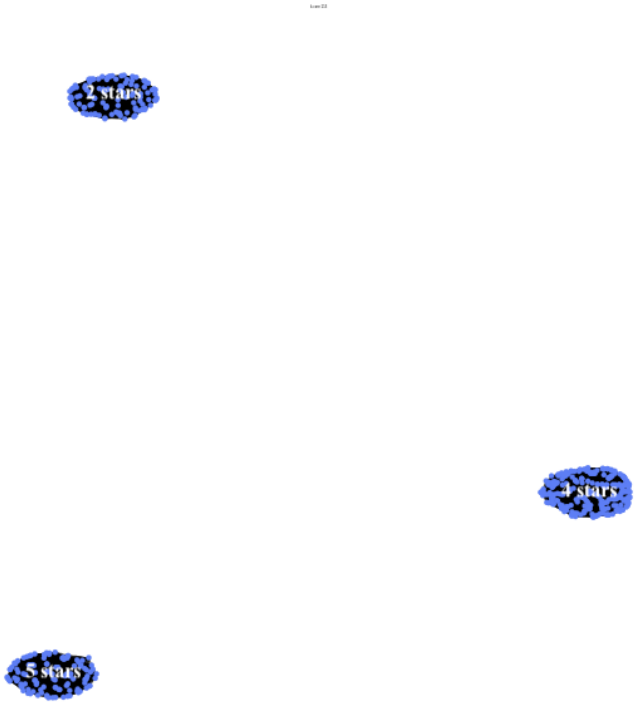


Figure 11: Cores with k = 113

id	Language
l_0	Ruby
l_1	JavaScript
l_4	Shell
l_8	Python
l_6	C

Table 3: Top 5 most used languages

From these results, we can say that there are several groups of highly interconnected repositories within the network with different numbers of stars but using the same languages. Thus, we can infer that there is not a strong correlation between the number of stars a repository has and the programming language used.

6.8 Components

We used this metric based on the same subgraph considered for clustering and betweenness centrality as we wanted to understand if there were any correlations between users who use topics in the repositories they work on or not. We decided to tackle this problem by calculating components as they indicate sets of nodes all connected to each other or isolated nodes from the rest of the network. Therefore, first, we found the connected components for the subgraph used for clustering, and the result is shown in the figures 12.

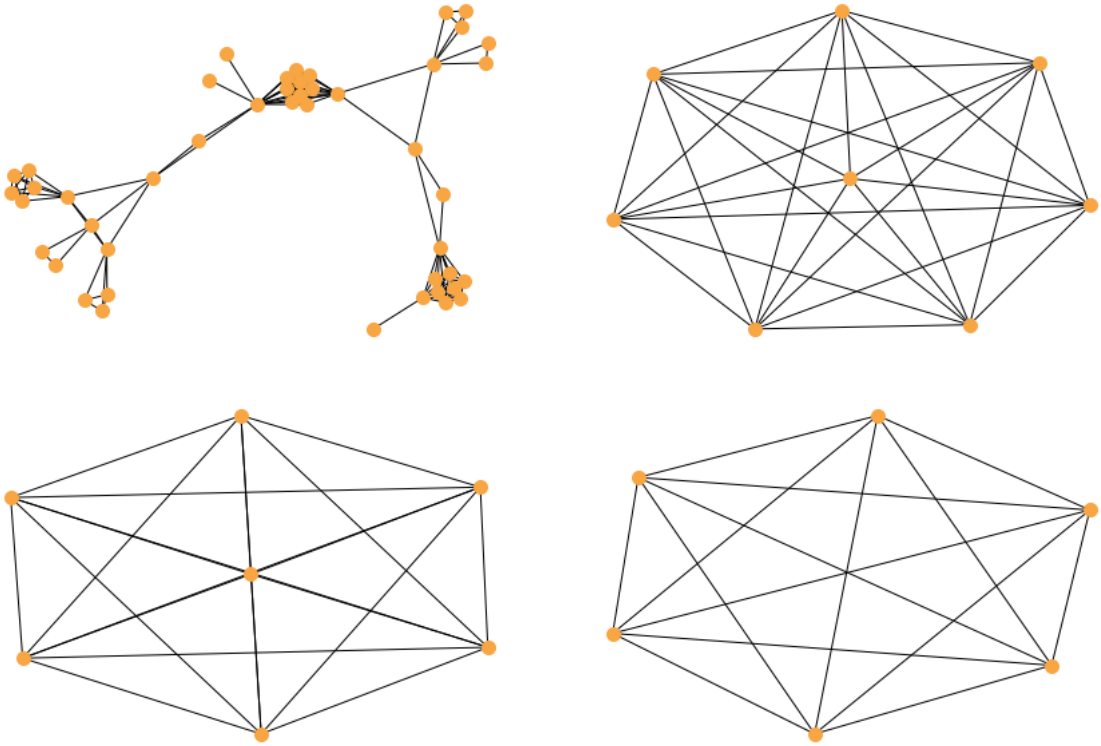


Figure 12: First 4 components formed by users collaborating in similar projects

Secondly, we decided to create a new subgraph that included only the repositories that did not have an empty topics field and the users who collaborated on these repositories. The resulting subgraph is very small compared to the rest of the dataset, as only a few repositories use topics, but it was extremely useful once the components were calculated. The result is shown in the figure 13.

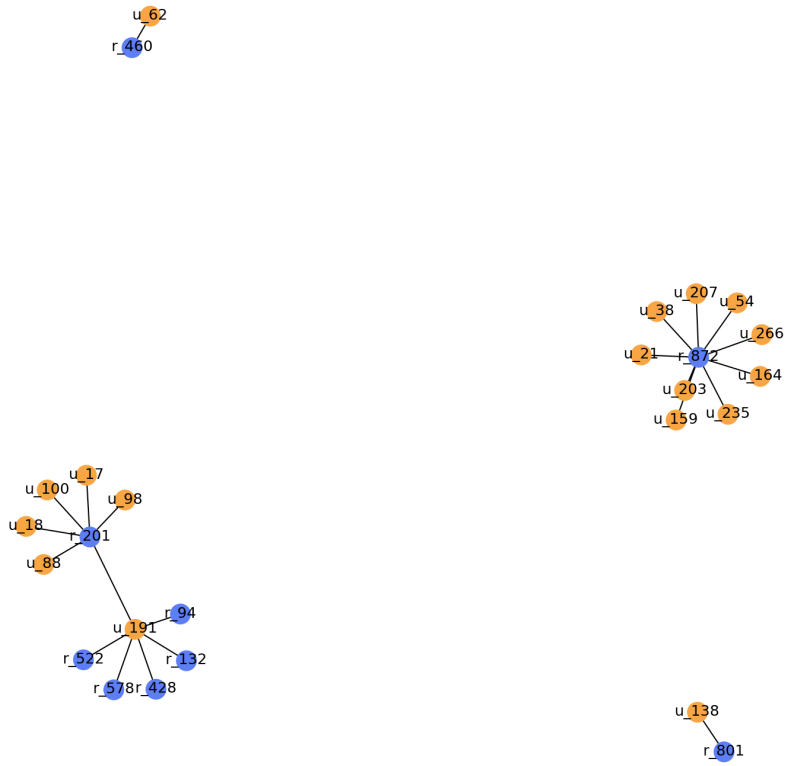


Figure 13: Subgraph of users collaborating in common repositories using topics

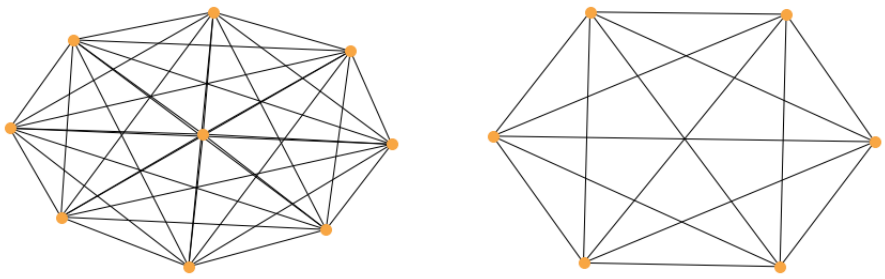


Figure 14: First two components of users using topics in common repositories

To understand whether users who use topics are the same as those in the connected components found in the graph, we compared the results of the two component analyses. We found that there is no correlation between the two graphs, indicating that users who tend to use topics in their repositories are a small and isolated community compared to other users on the platform.

6.9 Homophily

To compute the homophily, we modified the topology of the graph, creating one consisting only of user nodes, connected by an arc only if they collaborated on at least one common project. In Figure 15 the new topology of the graph can be seen.

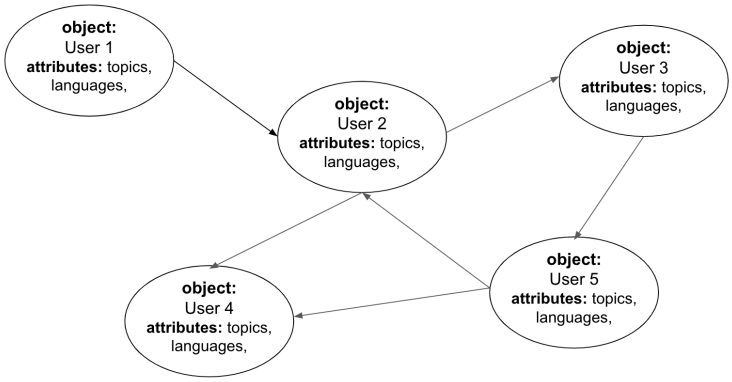


Figure 15: Rapresentation of graph

We used homophily to conduct two studies, one focused on GitHub users to understand how much they tend to collaborate with other users who use the same programming languages, to do this we selected only the language used with a higher percentage in each project, obtaining a result of 0.89.
The second analysis focused on seeing whether users are influenced to use topics by collaborating with other users who use them, obtaining a value of 0.80.

6.10 Statistical homogeneity

This type of study was used to evaluate the goodness of the clusters created, so it can tell us how similar the nodes that make up the clusters are to each other. To calculate this value, we used a combination of measures in order to evaluate different aspects of the clusters, but overall they give us a general picture of the situation. The measures used are: Average clustering coefficient: This coefficient measures the probability that a node's neighbors are also neighbors to each other. A high clustering coefficient indicates greater statistical homogeneity:

- ***Average clustering coefficient***: This coefficient measures the probability that a node's neighbors are also neighbors to each other. A high clustering coefficient indicates greater statistical homogeneity
- ***Assortativity coefficient***: This coefficient measures the tendency of similar nodes to associate with each other. A high assortativity coefficient indicates greater statistical homogeneity.

The results obtained were calculated from the same subgraphs used for k-mean clustering and are 0.93 and 0.85, respectively.

6.11 Small-worldness

Small-worldness occurs when nodes in a network are connected to each other through few intermediaries.
Several metrics are used to determine whether a network is characterized by the phenomenon of small-worldness:

- ***Clustering coefficient***: measures the number of triples of nodes that are mutually connected. A network with a high clustering coefficient indicates that nodes tend to be strongly interconnected, which may indicate the presence of small worldness.
- ***Average shortest path length***: measures the minimum number of nodes that must be traversed to reach one node from another node in the network. If this distance is relatively small compared to the total size of the network, this suggests the presence of small worldness.

One way to quantify the small-worldness of the network can be to use the coefficient σ, calculated by comparing clustering and path length of a given network to an equivalent random network with same degree on average.

$$\sigma = \frac{\frac{C}{C_r}}{\frac{L}{L_r}}$$

If $\sigma > 1$ and C » C_r and L≈L_r network is small-world. Then, we went to calculate the clustering coefficient and shortest path length for both our starting graph and the Erdos-Renyi graph, the results can be seen in the table below.

Clustering coefficient	**Shortest path lengh**	**Clustering coefficient ER**	**Shortest path lengh ER**
0.62638	1.11748	0.02803	1.209479

Table 4: σ values

Next we went to calculate the sigma which returned a value of 7.15.
So, from the results obtained we can see that we have a $\sigma > 1$, a higher starting graph clustering coefficient than C_r and the two values of shortest path length are similar therefore, we can say that we are in a small-world network.

The sigma value, however, is not always considered a reliable value because it is sensible to network size and density. For this reason we went to calculate ω which estimates the tendency of a network to resemble either a random or a regular one.

$$\omega = \frac{L_r}{L} - \frac{C}{C_l}$$

Where the characteristic path length L and clustering coefficient C are calculated from the network you are testing, C_l is the clustering coefficient for an equivalent lattice network and L_r is the characteristic path length for an equivalent random network.
For $\omega < 0$ the network is closer to a lattice, $\omega \approx 0$ it is small worldness, for $\omega > 0$ it is closer to a random graph. The value of our omega came out equal to -1.56 so, this means that ours is not small-world network but it is closer to a lattice.

6.12 Scale-free

A scale-free network is a type of network in which the degree distribution follows a power law, meaning that the probability of a node having a certain number of connections (degree) decreases exponentially as the degree increases. In other words, there are a few highly connected nodes, known as "hubs", and many poorly connected nodes.
To understand whether a network is scale-free, you examine the degree distribution of the network. If the degree distribution follows a power law, meaning that the probability of a node having a certain number of connections (degree) decreases exponentially as the degree increases, then the network is likely to be scale-free.
In addition, the value of alpha can be calculated to get an accurate measure and understanding of how close the distribution is to a power-law distribution. Empirically, in a power-law distribution the value of alpha is between 2 and 3. The formula for alpha is:

$$\alpha = 1 + n \left(\sum_i \ln \frac{d_i}{d_{\min} - \frac{1}{2}} \right)^{-1} \tag{1}$$

where n is the number of nodes in the network, d_i is the degree of node i, and $d_{\min}$ is the minimum degree in the network.
The alpha value that we obtained in our specific case is equal to 24.13. This value indicates a distribution with a very high slope and the presence of an above-average number of hubs.
In the Figure 16 below, we can see the degree of distribution of users into project in the network.

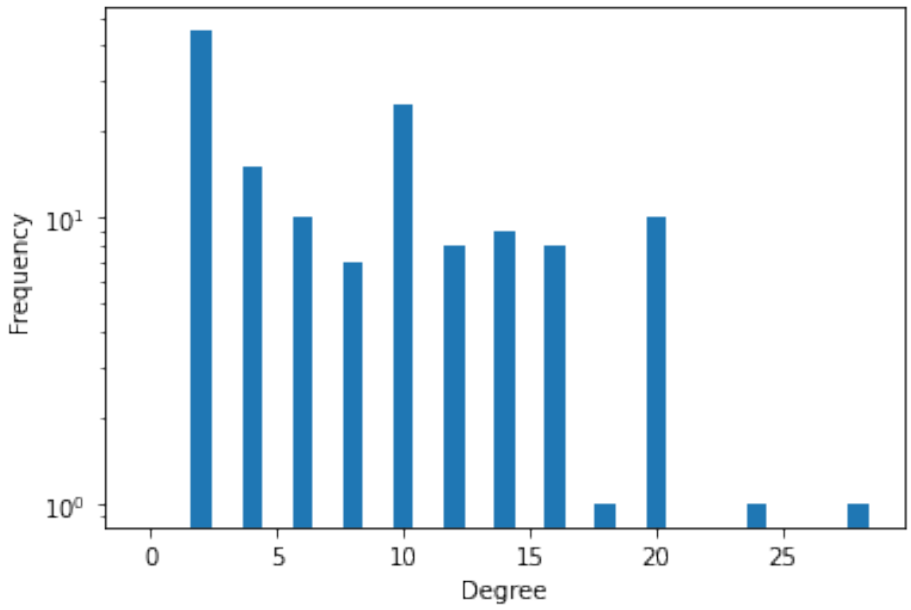


Figure 16: Degree distribution of users into project

From the alpha value obtained and the observed graph we can see that our subgraph, used to check whether our network is small-world, is not scale-free because it does not follow the power law for the degree distribution.

7 Conclusion

The metrics used to analyze our GitHub network were different, below we will take a qualitative analysis of the results obtained from the metrics used.
From the results obtained through *degree centrality*, we can see the presence of more central programming languages within the network than others, such as Ruby (l_0) and JavaScript (l_1).
The calculation of *betweenness centrality* tells us which users within our network can be used to best disseminate information. For example, user u_238 appears to be more central than the others, but users u_148 and u_237 also play a very important role in the network. Having these 3 nodes in the first three positions is what we expected, as by taking only the 10 nodes that scored higher, these nodes are more likely to be in the shortest path between a pair of nodes.
The value obtained by calculating *modularity* is 0.55, this result indicates to us that programmers, although they tend to use some languages more frequently than others, do not always focus only on one language but, at times, vary.
The results of *k-mean clustering* allowed us to identify user communities that tend to collaborate in similar projects. In this case, we did not go to verify the target of each repository but simply evaluated repositories that use the same programming languages. The communities we identified, using clustering, were then verified using *components* so that we could precisely identify each group that formed in our network. In this way, we were able to identify larger communities that based on the previous results, obtained from the programming languages used, we can assume are made up of repositories that use web programming languages, and more niche communities, thus with few repositories making them up, which could be those who use languages for purposes other than web development.
The *clusters* we found were evaluated to see if their component nodes were similar to each other. To carry out this task we decided to calculate two metrics that using them together allow us to evaluate the goodness of the clusters obtained. The metrics used were the average clustering coefficient and the assortativity coefficient, having obtained very good results, 0.93 and 0.85 respectively, leads us to think that the clusters created have similar nodes to each other.
To understand which language combinations are most used we used *structural equivalence*, directly considering the results obtained by excluding Shell (l_4), it can be seen that the language present in the majority of combinations is Ruby (l_0), which together with JavaScript (l_1), represent the most used combination within our network. This aspect is also highlighted by the study carried out to go and identify the *cliques*. The languages that create the most cliques are Ruby (l_0) and JavaScript (l_1), but Perl (l_10), Objective-C (l_18) and CSS (l_14) also turn out to be languages used in many repositories. These combinations are particularly interesting since they are all programming languages used primarily for web development this may indicate that they are particularly useful and interesting languages to learn for those who want to go into this field.
From the results obtained using *homophily*, calculated on programming languages, we obtained a value of 0.80, this indicates to us that there is a good tendency for programmers to collaborate in projects with other users using the same programming language. While calculating it on topics the value returned was 0.89, this indicates a fairly high tendency to start using topics after collaborating with other users who have used them. In addition, we went to check the connected components so as to understand which users are using them in the repositories. The results showed us two connected components that included some users and some repositories, but also components that consisted of only one component and one repository. Thus, the use of topics within our network affects only a small part of the community that has no correlation with the rest of the users.

As a final aspect, we can state that our network does not exhibit the characteristics of a *small-worldness* network, as the omega value turns out to be less than zero, indicating that our network is more like a lattice network. This suggests that users in our network tend to collaborate mainly with other users developing similar projects. Also, looking at the degree distribution graph and considering the value of alpha equal to 24.13, we can see that our network is not a *scale-free* network because it does not follow the degree distribution. This implies that there are few programmers within the network who are more active in the repositories than others.

8 Critique

Our study has provided interesting insights on the most used programming languages in GitHub projects, as well as on user trends in using not only programming languages, but also topics and factors that influence programmers' tendency to collaborate.
Although the metrics used have provided satisfactory results for our objectives, there is always room for improvement. For example, we could conduct the same study using alternative metrics to obtain further information and perspectives. Additionally, we could expand our data sample by including a greater number of repositories that use topics to obtain even more reliable results and, potentially, different from those currently reported in our analyses.

References

[1] *GitHub REST API*. URL: https://docs.github.com/en/rest?apiVersion=2022-11-28.

[2] *NetworkX*. URL: https://networkx.org/documentation/stable/index.html.

[3] *PyGitHub*. URL: https://pygithub.readthedocs.io/en/latest/index.html#.

[4] *Python*. URL: https://www.python.org.