Social Network Analysis: Project Report

Ananya Sharma(2023EET2186) Kareem Shaik(20223EET2194)

November 29, 2023

1 Introduction

StackOverflow has become a primary source of learning as it covers a wide range of programming-related topics. By building a **tag-tag relationship graph**, the activities and interests of StackOverflow users in the field of computer science are analyzed. The goal of these analyses is to highlight the evolution of the network over time, as opposed to static approaches, and to incorporate link prediction methods in network analysis.

StackOverflow is one of the largest Community Question Answering communities for programmers, boasting over three million users. It has become a primary learning source, covering various programming-related topics. This forum serves various purposes, such as obtaining generic information about programming languages or solving programming problems. Many community-based information websites, like StackOverflow, rely on **metadata called tags**, aiding in indexing, categorization, and content search. Users explicitly request tags when posting a question. In StackOverflow, tagging questions is mandatory (at least one tag per question), and each question can have up to five tags.

The purpose of this study is to analyze StackOverflow tags to gain insights into the user community by representing tags and their occurrences as a graph. Graph nodes represent tags, and there is a common edge between two nodes A and B if there is at least one question where they coexist. The edges of the resulting network are weighted based on the number of times the two connected tags appear together in a question. The study explores the features of this tag-tag network, comparing it with some standard models: **Barabási-Albert, Erdös–Rényi, Watts–Strogatz, and Configuration Model**.

Notably, the network was not intentionally constructed by users to form social relationships but reflects shared interests within the community. One goal of the study is to identify computer science areas where users pose more questions.

The report is organized as follows: Section [2] briefly describes the data collection and Preliminary Analysis provides some summary statistics, and details about network construction. Section [3] follows with an analysis of network characteristics. Subsequently, Sections [4 –6- 7] discuss community discovery, link prediction, and spreading approaches. Section [5] presents a dynamic analysis of the network and communities over the months, incorporating a temporal analysis of the entire dataset.

Tags	Set of tags associated with the question (max 5)
Question link	Link to the question
View count	No of users who saw the question
Down vote count	Number of down votes
Up vote count	Number of positive votes
Is Answered	Indicates whether the question has answers
Answer count	Number of answers
Score	Overall score of the question
Creation date	Creation date of the question
Question id	associated with the question from the forum

Table 1: Description of dataset

Number of questions	50000
Number of tags	164520
Number of unique tags	13848
Average number of tags per question	3.2904

Table 2: Statistics of dataset

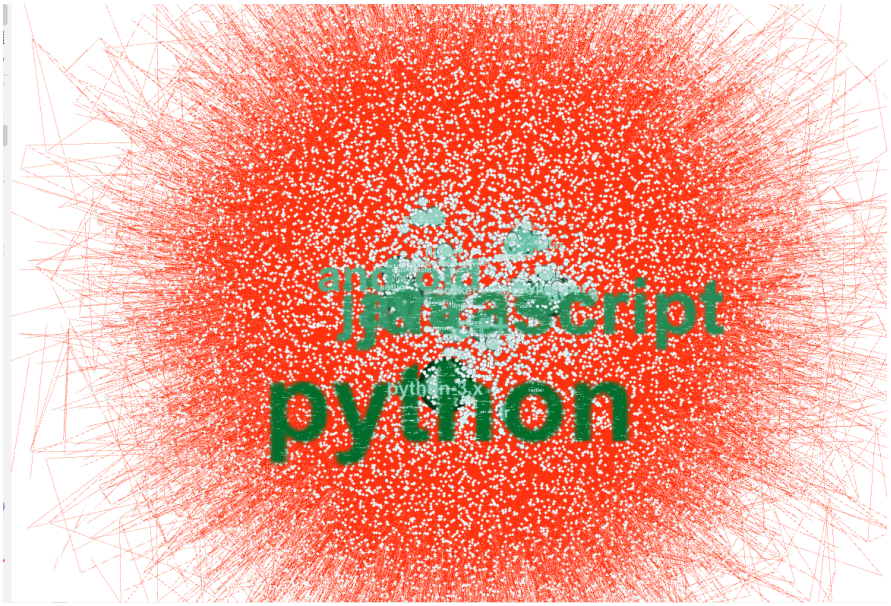

Figure 1: Real-world Network Model

2 Data Collection and Preliminary analysis

The network data set was taken from the public network data sets provided by Stanford, which can be found here dataset. From the dataset, a **Tag-to-Tag network** was created using some pre-processing. Tags as nodes and creating an edge between two nodes if their respective tags were present together in at least one question. The weight of the edge was set as the number of questions in which the tags were present simultaneously.

Table [1] and [2] illustrates the description and statistics of the dataset.

Due to the increasing specificity of tags entered by users, reaching the maximum threshold of 5, it was found (as shown in the word cloud in Figure [2]). The histogram in Figure[3], shows a representation of the frequency of tags. Another representation is with a bubble plot as shown in Figure[4]. Gephi's Visualization of Real-world network is represented in Figure[1].

Some tags had a frequency exceeding 2500 occurrences (especially tags like **python, javascript, and java**), while others, highly specific, had a frequency of 1 (such as google-nearby-connections, build numbers, SilverStripe). About 98% of the tags appear fewer than 100 times, compared to the top 19 most frequent tags that appear more than 1000 times. These results suggest a **highly skewed distribution**, with relatively few frequent tags dominating the dataset.

To visualize the graph more clearly and conduct analyses, the tags were grouped into the following

python	6827
javascript	3203
android	13848
java	2996
reactjs	2904
C++	2801

Table 3: Frequency of dominating tags

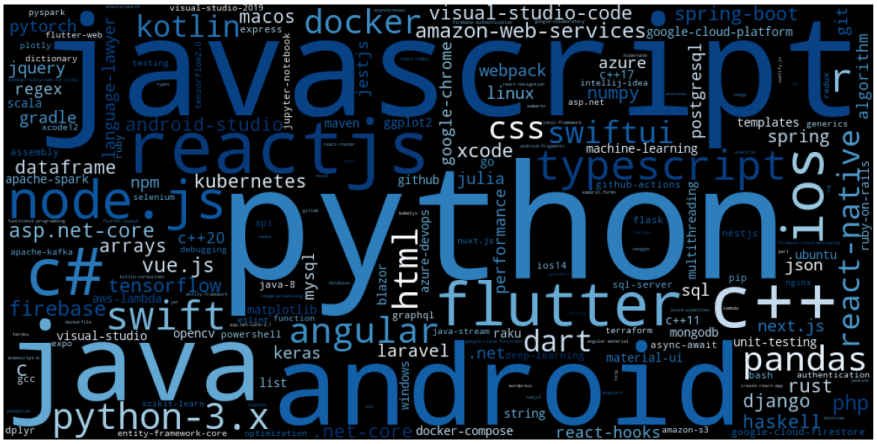

Figure 2: Tag Cloud

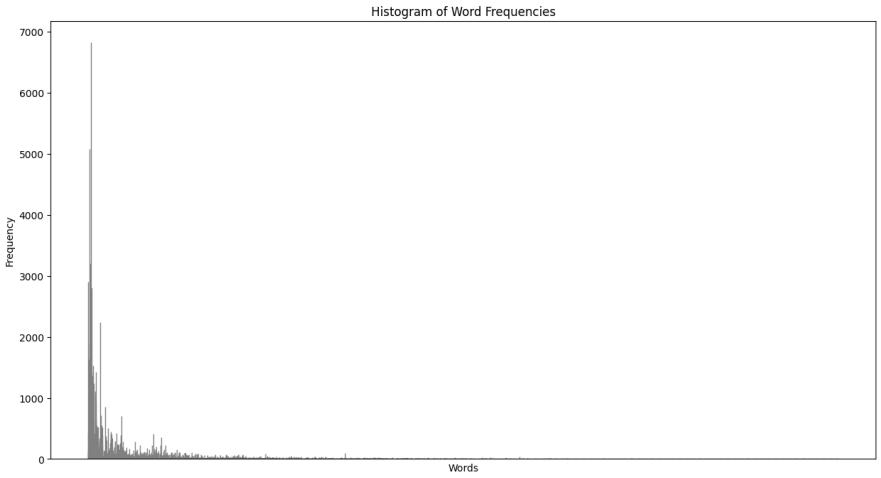

Figure 3: Histogram for frequency of tags

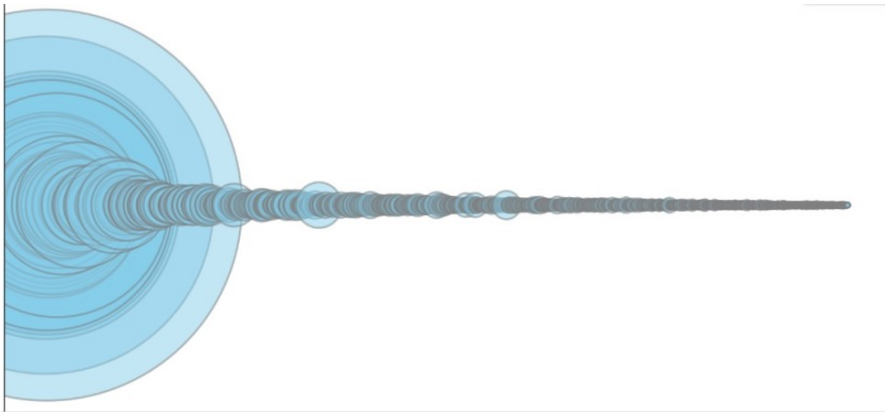

Figure 4: Bubble plot representing the frequency of tags

Google nearby connections	1
silverstripe-4	1
SilverStripe	1

Table 4: Frequency of less occurring tags

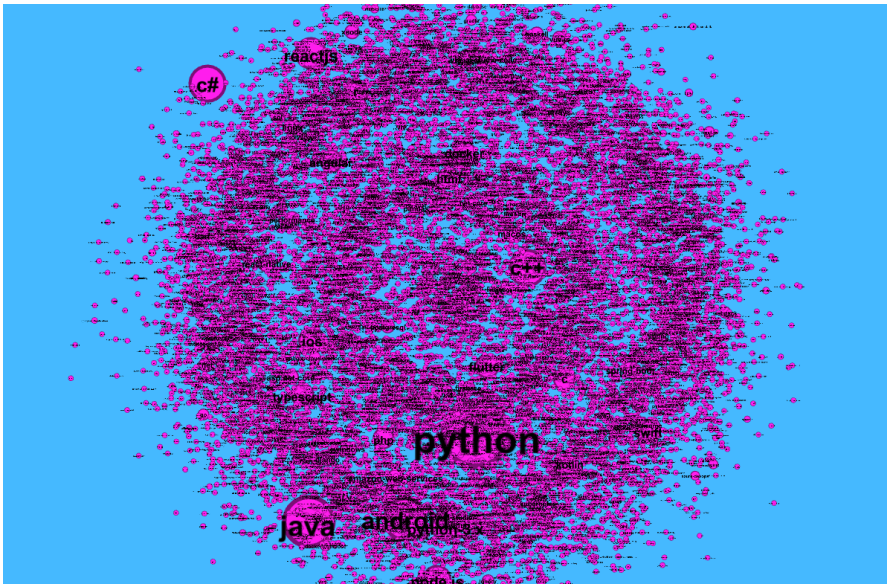

Figure 5: Nodes tagged by Language

Number of nodes N	13780
Number of edges E	97320
Maximum no of edges	94937310
Average clustering	14.208
Average Clustering Coefficient	0.6988
Density d(G)	0.001031

Table 5: Specifications of network

7 macro-areas based on their meaning in the field of computer science: **Programming Languages:** containing tags related to programming languages and their libraries. **Database:** including tags related to databases. **Development Tools:** encompassing development environments and cloud-related tags. **Web & Application Servers:** tags referring to network protocols, security, and web development. **Operating Systems:** related to tags tied to operating systems, computer peripherals, components, and file extensions. **Data Structures & Functions:** a collection of tags related to data structures and functions.

The resulting network, shown in Table [5], is a bidirectional network with 13,780 nodes and 97,892 edges, as seen in Table [5], which outlines some specifications of the network. Figure[5] represents Gephi's visualization of nodes tagged by language.

In particular, it can be observed that this is a sparse graph. If the graph were complete, we would have E = N(N-1)/2, an average degree equal to N - 1 = 13,779, and network density d(G) = 1. Furthermore, the network falls into the connected regime, as the average degree is greater than ln(N) = 9.53. Additional analyses will be presented in the following sections.

3 Network Characterization

The constructed network i.e. a **Real World Network** was compared with **four other synthetic models** created using predefined algorithms. The parameters of these models were set to have the same number of nodes as the original network and a similar quantity of edges, providing a deeper perspective on the composition of the network and analyzing its similarities and differences with standard models.

The considered models, with their node and edge values, are as follows (as shown in Table [6]):

- Barabási-Albert (BA) of n nodes is grown by attaching new nodes each with m=7 edges that are preferentially attached to existing nodes with a high degree, where $1 \leq m \leq n$. The network visualization with Gephi is represented in Figure[6].

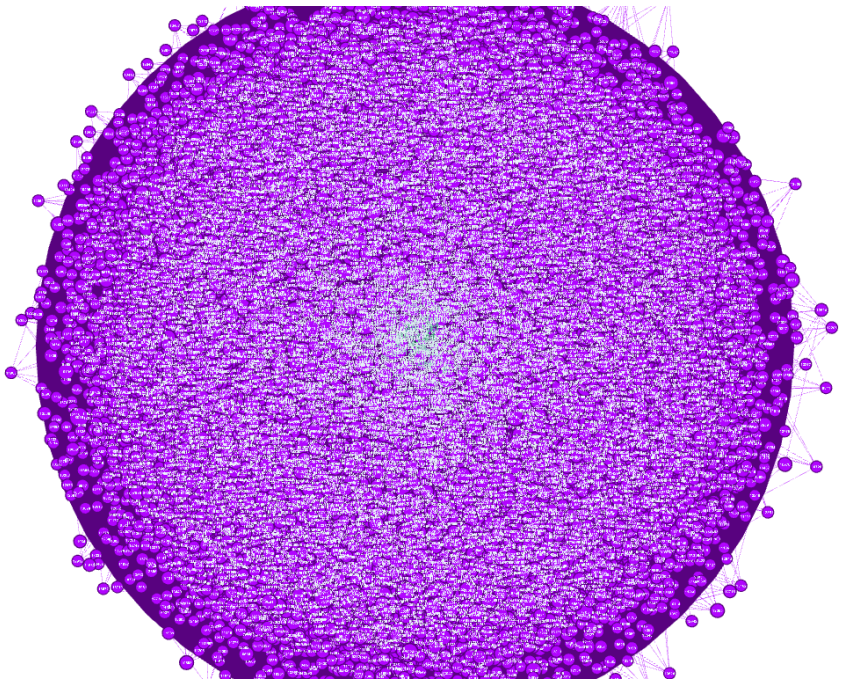

Figure 6: Barabási-Albert Model

	Barabási-Albert	Configuration model	Erdös–Rényi	Watts-Strogatz
Number of nodes N	13780	13780	13780	13780
Number of edges E	96411	97892	97895	96460

Table 6: Synthetic networks

- Configuration model (CM) constructed following the degree distribution of the real-world network. The network visualization with Gephi is represented in Figure[7].
- Erdös–Rényi (ER) chooses each of the possible edges with probability p = 0.0010337. The network visualization with Gephi is represented in Figure[8].
- Watts-Strogatz (WS) where k = Each node is connected to k nearest neighbors in a ring topology and p = The probability of rewiring each edge, here k = 14 and p = 0.1. The network visualization with Gephi is represented in Figure[9].

3.1 Degree Distribution Analysis

Observing the degree distribution of the networks in Figure [10], it can be stated that the constructed real-world network is **similar to the Configuration Model**, as expected. This is because the Configuration Model is a random network model that relies entirely on maintaining the same degree as the Real world network. The network is also similar to the Barabási-Albert model while appearing **very different from the Erdös-Rényi and Watts-Strogatz networks**.

The presence of hubs in the network is noticeable, as the creation of connections between tags follows a preferential attachment to these hubs. The three tags with the highest degrees, exceeding 2000, are **python (2784), javascript (2291), and java (2036), corresponding to three of the main programming languages.** The minimum degree is 1, for extremely rare and specific tags such as silverstripe, geode, gemfire, indicating no presence of isolated nodes (tags with a degree of 0).

The **network's distribution follows a power-law distribution,** as there is an inversely proportional relationship between the number of nodes and their degree.

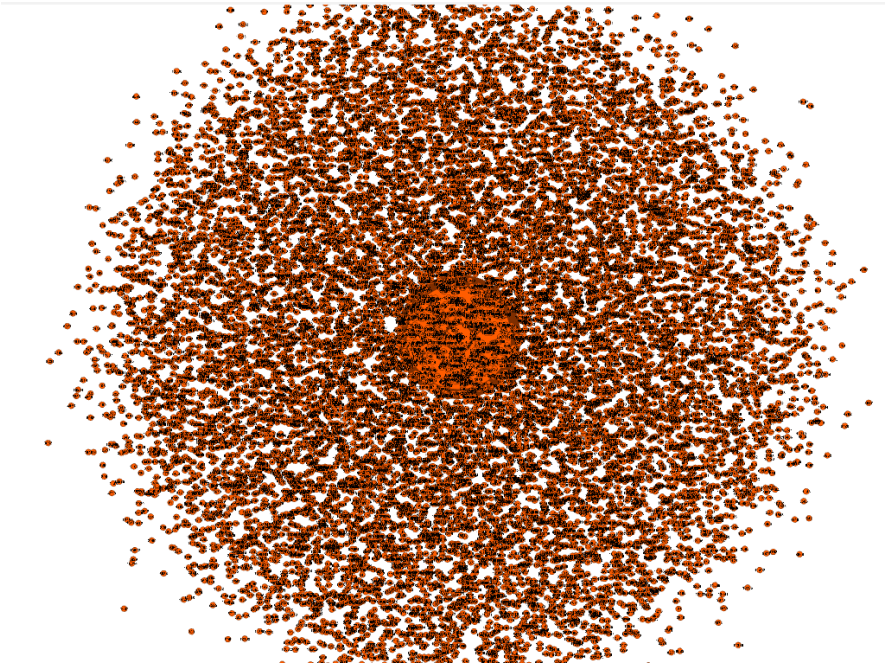

Figure 7: Configuration model

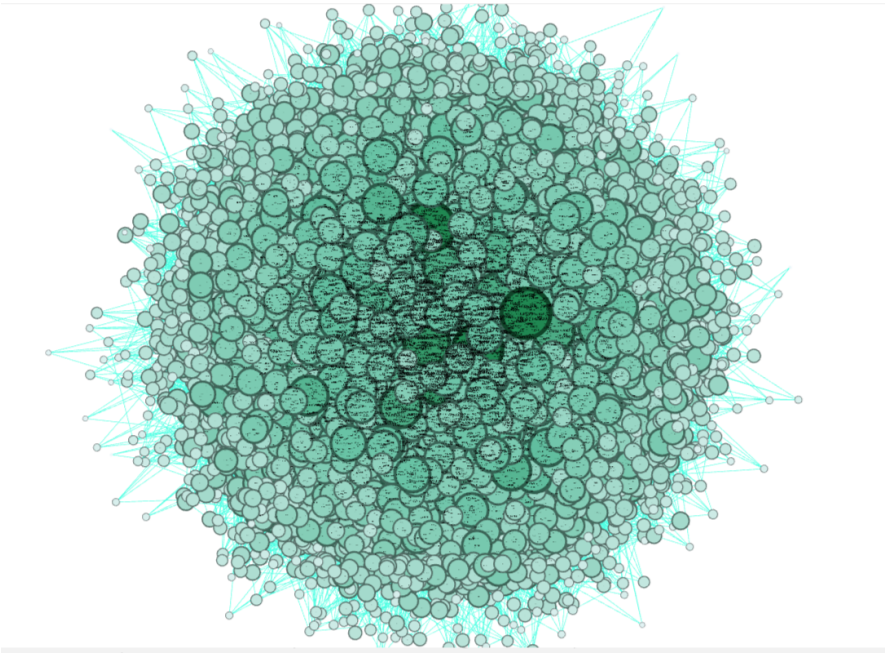

Figure 8: Erdös–Rényi Model

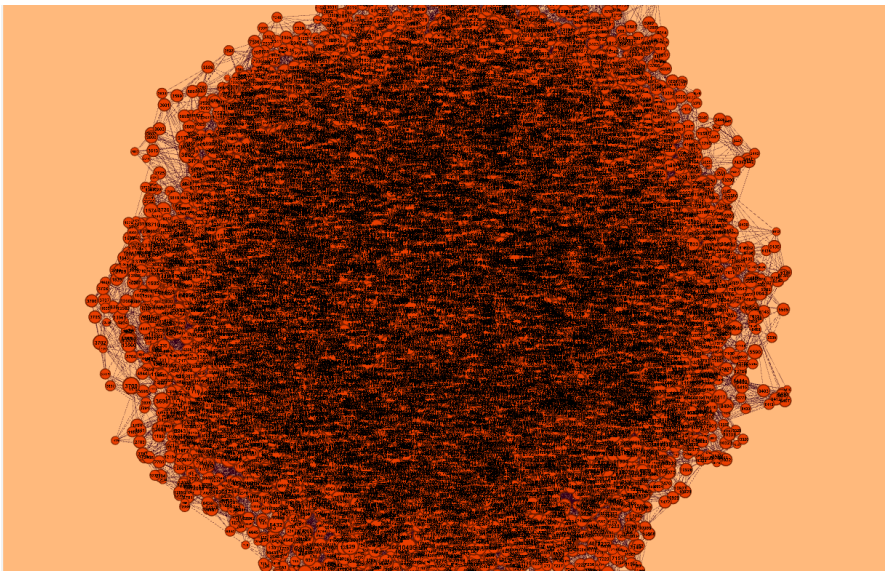

Figure 9: Watts-Strogatz Model

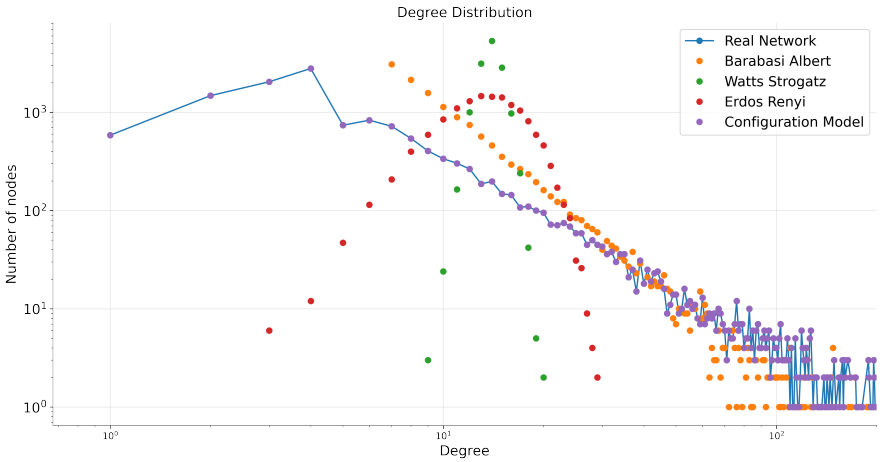

Figure 10: Degree Distribution

	Barabási-Albert	Configuration model	Erdös–Rényi	Watts-Strogatz
Number of Connected Components	13780	13778	13780	13780
component length	1	2	1	1

Table 7: Connected Components

	Real World	Barabási-Albert	Configuration model	Erdös–Rényi	Watts-Strogatz
Diameter	7	5	7	6	8
Average shortest path	2.97	3.445	3.148	3.861	5.3136

Table 8: Path Statistics

3.2 Connected Components Analysis

The analyzed network is composed of **a total of 32 connected components, among which a giant component emerged consisting of 13,699 nodes** (99% of the entire network), while each other component included at most 4 nodes. The synthetic models obtained a single connected component, except for the Configuration Model, for which 2 connected components were discovered, one of which was a giant component composed of 13,778 nodes, as shown in table[7].

3.3 Path Analysis

For the largest component of each network, average shortest paths and diameters were calculated, as reported in Table [8]. The utilized network exhibited **diameter values similar to synthetic networks** (the highest, equal to 8, was **observed in the analysis of the Watts-Strogatz network).** However, in terms of the average shortest path, it had a value lower than that of the other networks and lower than $\ln(N) = 9.53$. Considering the weights on the links for the real network, the average shortest path increased slightly, amounting to 3.3240.

Since a **Scale-Free network is resilient to random node removals but vulnerable if these nodes are hubs** for the network, specific node removals of sets were carried out. In particular, groups of 10 nodes with higher Degree Centrality were removed in a test consisting of 9 iterations, evaluating in each the number of connected components, the diameter, and the average shortest path of the Giant Component. As observed in Table [9], the removal of hubs fragments the network into numerous connected components and increases the average path length between two nodes in the Giant Component; **a behavior expected in Scale-Free networks.**

3.4 Clustering coefficient - Density analysis

The global clustering coefficient of the graph is 0.6988, a value higher than those obtained in synthetic networks (as observed in Table [10]). As expected, the average clustering coefficient of the ER network is equal to the probability p, with an order of magnitude significantly lower than the value for the real network. This demonstrates that the **random model not only fails to capture the degree distribution of the real network but also the clustering coefficient.** Additionally, by observing

	No. of Connected Components	Diameter	Avg Shortest path
1	215	8	3.36
2	411	9	3.63
3	482	9	3.72
4	565	9	3.81
5	602	9	3.85
6	650	10	3.90
7	722	10	3.96
8	767	10	4.02
9	811	10	4.06

Table 9: Path Statistics

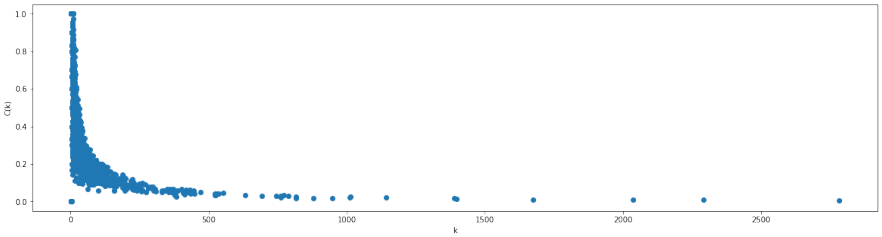

Figure 11: Local clustering coefficient vs Node degree

	Real World	Barabási-Albert	Erdös–Rényi	Watts-Strogatz
Clustering coefficients	0.699	0.00681	0.00103	0.50661

Table 10: Clustering coefficients

the ratio between the local clustering coefficient and the node degree, it was highlighted that nodes with a higher degree had a very low clustering coefficient, while nodes with lower degrees tended to have clustering coefficient values throughout the entire range (Figure [11]).

The density of synthetic networks turned out to be very similar (with variations after the 4th decimal place) to the density of the StackOverflow network. The very low-density value (close to 0.00103) for all networks is justified by the presence of a much lower number of edges than the maximum possible number (N(N-1)/2), which is close to 94 million.

3.5 Centrality analysis

The network was analyzed using methods based on different centrality definitions: **degree-based, connectivity-based (Eigenvector, PageRank), and geometric-based (closeness, harmonic, betweenness).**

In Figures [12] and [13], you can observe the results of these analyses for the top 15 nodes, particularly the difference between the two geometric-based methods **(Closeness and Harmonic)** and the other methods in terms of the most central tags. Indeed, with the Closeness and Harmonic methods, tags such as **visual-studio-code, amazon-web-services, and macOS emerged as central,** which were not present in the top 15 of the other methods. These two methods reported centrality values that were almost constant, indicating how these centrality definitions are less interesting for the analyzed network.

Almost all methods identified **python, javascript, and java as the most central nodes**, representing, in relation to what was reported earlier. A substantial difference was found with the Eigenvector method, which identified **javascript as the most central node** and subsequently three tags semantically related to the language: **reactjs, typescript, and node.js.** The Eigenvector Centrality measures the reputation of a node and the recognition it receives from other nodes. In this context, nodes like reactjs, typescript, and node.js provide greater recognition to javascript (compared to python) because they are languages/frameworks based on the javascript language.

4 Static Community Discovery

In this task, the goal is to identify hidden communities within the network using four methods: **K-Clique, Label Propagation, Louvain, and Demon.**

For computational reasons, the K-Clique algorithm was executed on a random sample of 8,500 nodes from the original network. Where possible, an optimization phase of hyperparameters was conducted

	Barabási-Albert	Configuration model	Erdös–Rényi	Watts-Strogatz
Density	0.00101	0.00103	0.00103	0.001016

Table 11: Density Analysis

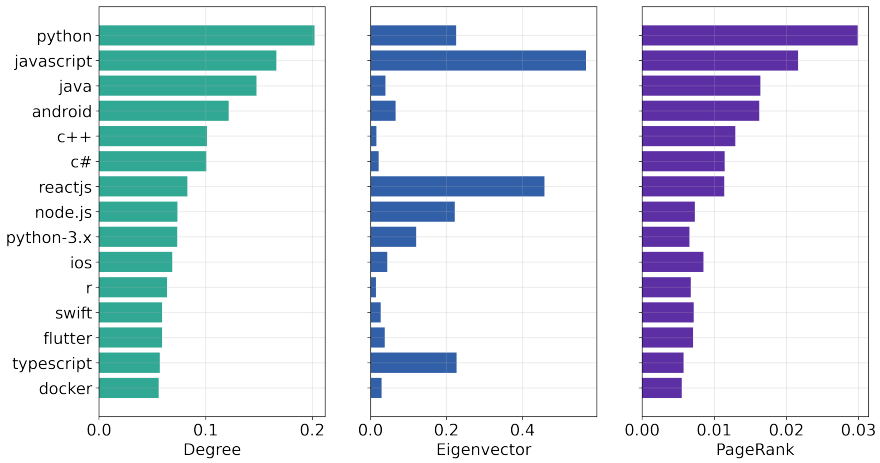

Figure 12: Top 15 tags by centrality

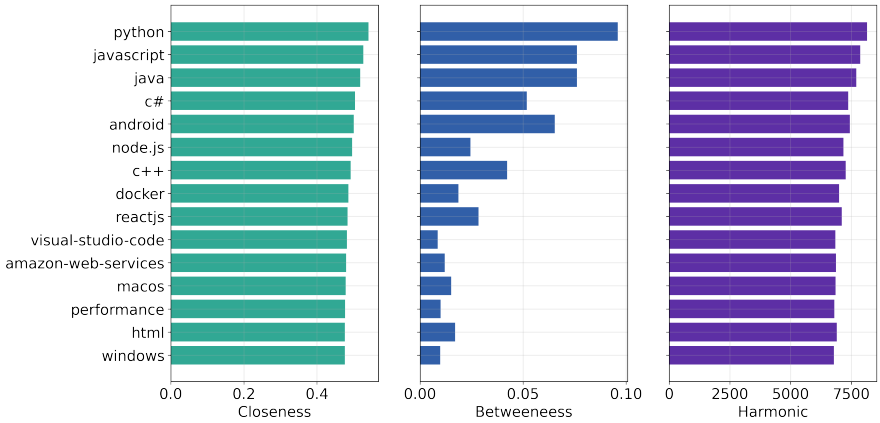

Figure 13: Top 15 tags by centrality

Algorithm	Hyperparameters	Optimal
K-Clique	k ϵ [2,8]	k=4
Louvain	resolution ϵ [0.1,1]	0.9
Demon	epsilon ϵ[0.1,0.6] min_com_size ϵ[3,5]	ϵ=0.4 min_com_size=3

Table 12: Hyperparameters

resolution	0.1	0.3	0.7	0.9
AID	3.5	3.74	3.31	3.21
EID	0.27	0.42	0.61	0.67
Conductance	0.56	0.38	0.20	0.14
Modularity	0.34	0.41	0.47	0.47
No of Community	216	90	57	44
Max dim community	412	801	2027	2263

Table 13: Louvain Results

through random search (an optimization strategy provided by cdlib), with the aim of maximizing Modularity.

While **Louvain and Label Propagation identify crisp communities** where each node belongs to at most one community, **Demon and K-Clique identify overlapping communities,** allowing each node to belong to more than one community. Table [12] shows, for each algorithm, the hyperparameter groups used during the optimization phase and their corresponding optimal values.

The **optimal hyperparameters were further evaluated** to maximize Average Internal Degree and Internal Edge Density while minimizing Conductance, as shown in Table [13]. For the Louvain algorithm, the table displays the number of communities obtained, the size of the largest community, and metric results for various resolutions. The choice of 0.9 as the optimal value for the resolution parameter is justified by the number of communities found, considering the values of **AID, EID, Conductance, and Modularity, which are overall better compared to other parameter values.**

The results obtained from the utilized implementations for the previously described metrics are presented in Table [14]. The K-Clique algorithm achieved the lowest modularity value, although it's not directly comparable to the other models due to simplifications made before its execution and covering only 78% of the total nodes. On the other hand, **the Demon model, an algorithm with overlapping communities like K-Clique, covered 94% of the nodes.**

Figure [14] displays the boxplot of the distribution of community sizes produced by the algorithms. **Label Propagation exhibited one large community (comprising 96% of the nodes) and many small communities,** resulting in an **unbalanced outcome** quite different from the other algorithms. The Demon algorithm produced more large-sized communities compared to Louvain as shown in Figure[15].

Indeed, the Demon algorithm has produced overlaps between communities, as evident in Figure [19], which displays WordClouds of some communities identified by the algorithm. From the figure, one can **observe an example of the overlaps, particularly among the first 6 communities**, all semantically related to tags concerning .NET programming.

The best results, considering the metrics and the number of partitions, were obtained by the Louvain algorithm. Although the number of communities obtained by Louvain was close to that of the Demon algorithm, Louvain had the absolute best values for Conductance and Modularity. **Louvain achieved**

	AID	EID	Conductance	Modularity	No of Communities
K-Clique	2.16	0.99	0.75	-0.012	332
Label propagation	1.68	0.93	0.37	0.01	188
Louvain	3.21	0.67	0.14	0.48	47
Demon	10.20	0.54	0.61	0.09	73

Table 14: Community discovery algorithm results

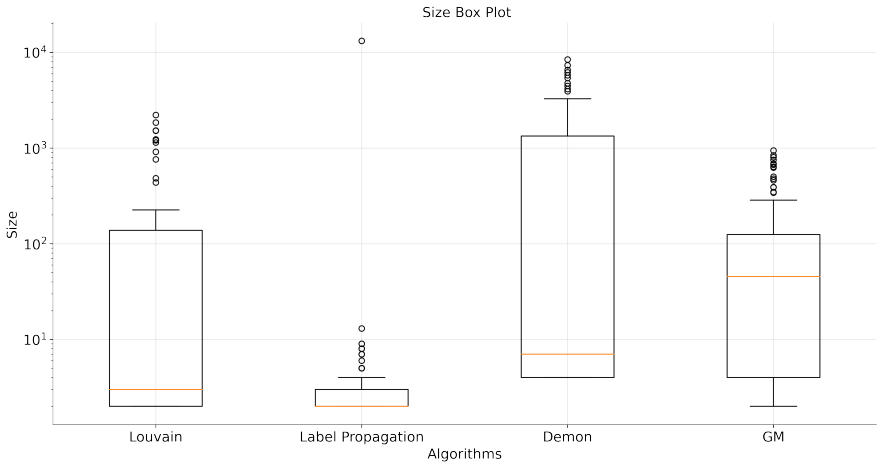

Figure 14: Community box plot

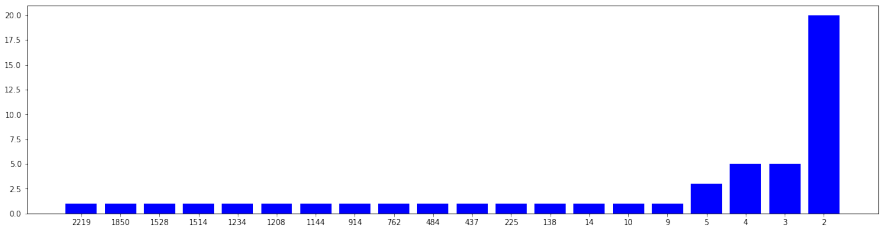

Figure 15: Cardinality of Louvain community

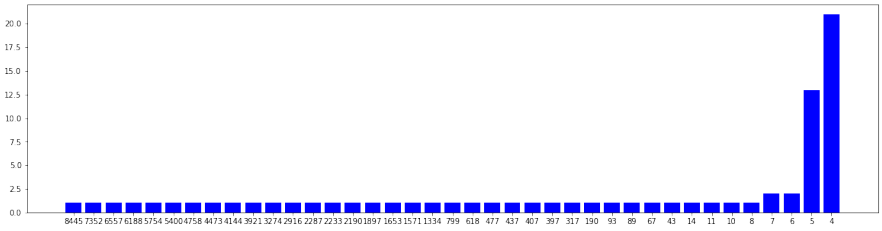

Figure 16: Cardinality of Demon community

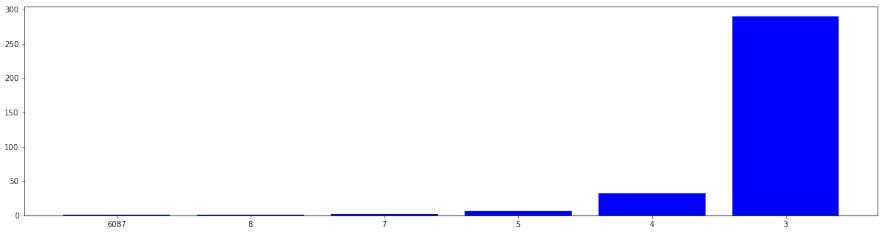

Figure 17: Cardinality of K-Clique community

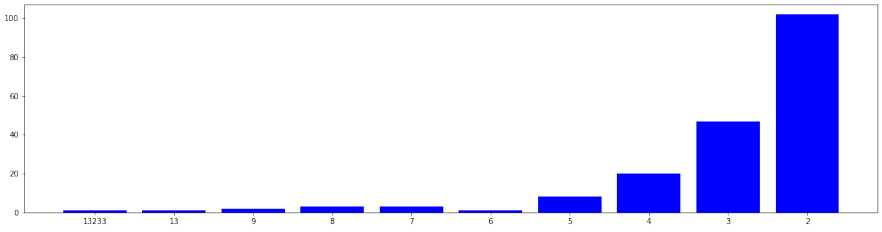

Figure 18: Cardinality of Label Propagation community

Figure 19: WordClouds Community 0-5 Demon

not only fewer large partitions than Demon (Figure [15]) and smaller-sized partitions (given the non-overlapping characteristics of the model) but also semantically more meaningful partitions.

In Figures [20 - 23], WordClouds of four communities obtained by the Louvain model are shown, where semantic divisions between the communities are noticeable:

- **Community 0** (Figure [20]) concerns **frontend development**, with terms such as react, javascript, angular, and css. - **Community 1** (Figure [21]) relates to **development in Python and Julia**, with terms associated with these languages, including frameworks and libraries such as Django, Flask, and Matplotlib. - **Community 3** (Figure [22]) revolves around **version control**, with terms related to debuggers, web services, and GitHub. - **Community 12** (Figure [23]) pertains to **development in Ruby,** with frameworks and keywords linked to the language, such as bundle etc.

The main characteristics of these four communities (treated as subgraphs of the original network), where it's evident that the number of nodes and edges tends to decrease from a community with a low index (e.g., Community 0) to a community with a high index (e.g., Community 12), leading to a decrease in related parameters (LMAX, Avg degree, Avg Clustering Coefficient). This demonstrates that the **Louvain algorithm identifies communities with increasingly smaller and denser**

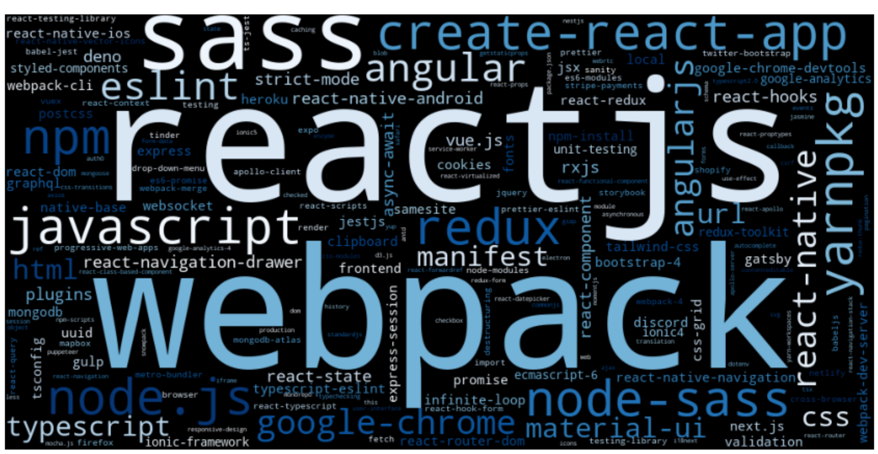

Figure 20: Community 0 Front-end Developers

Figure 21: Community 1 Python Developers

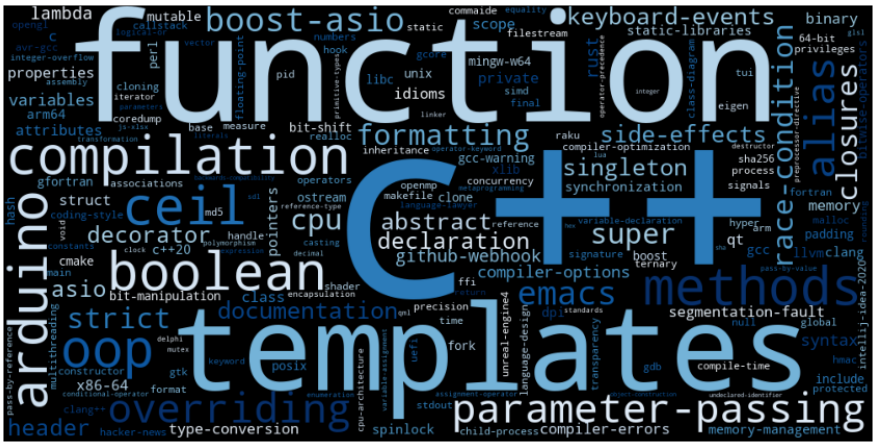

Figure 22: Community 3 Version Control

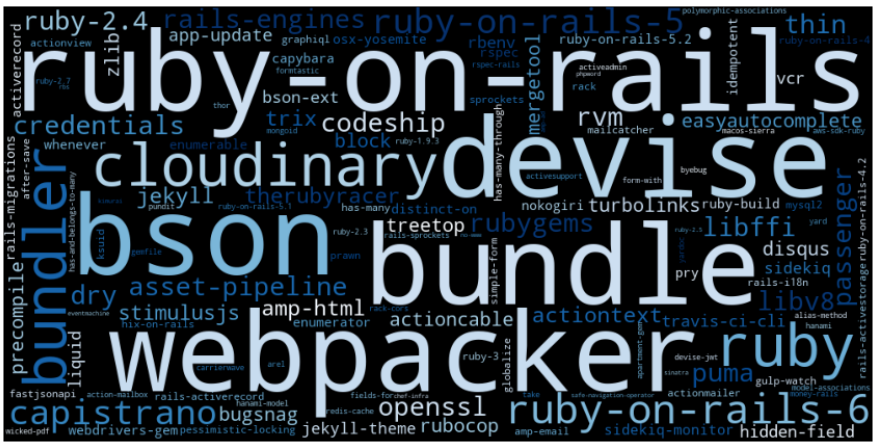

Figure 23: Community 12 Ruby Developers

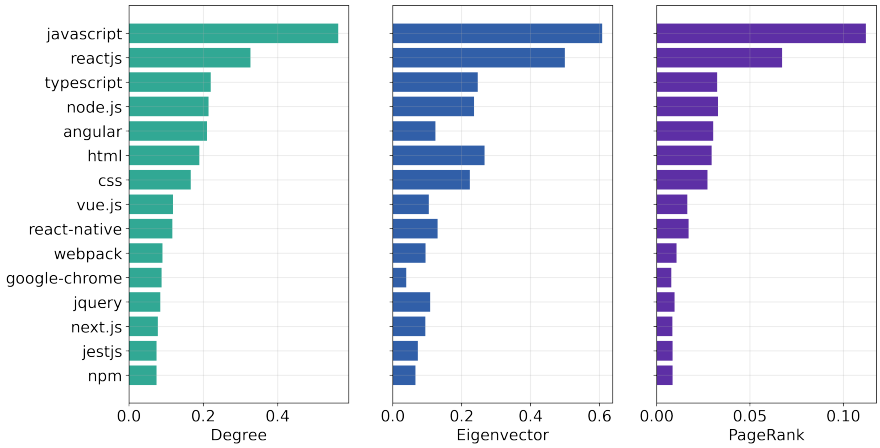

Figure 24: Centrality community 0

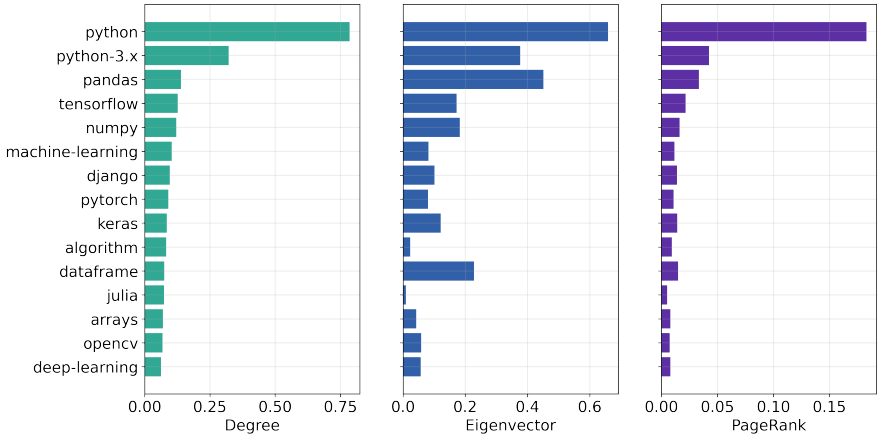

Figure 25: Centrality community 1

structures rather than communities with an even distribution of nodes.

The same centrality analyses reported earlier were conducted on the communities. For each community, observing the most central nodes with various measures, shown in Figures [24 - 25], no substantial differences in terms of central tags were found, except for their order. However, these results allowed the validation of communities concerning the semantic divisions made in the previous analysis. For example, for **Community 0** (Figure [24]), the most central nodes were found to be **javascript, reactjs, nodejs, html**, while for **Community 1** (Figure [27]), the central nodes were **python, pandas, tensorflow, dataframe, julia.**

Community	0	1	3	12
No of nodes N	2219	1850	1514	138
No of edges E	11561	9273	5741	334
Emax	2460871	1710325	1145341	9453
Average degree	10.42	10.02	7.58	4.84
Avg Clustering coeff	0.746	0.752	0.651	0.678
Density d(G)	0.005	0.005	0.005	0.035

Table 15: Community Statistics

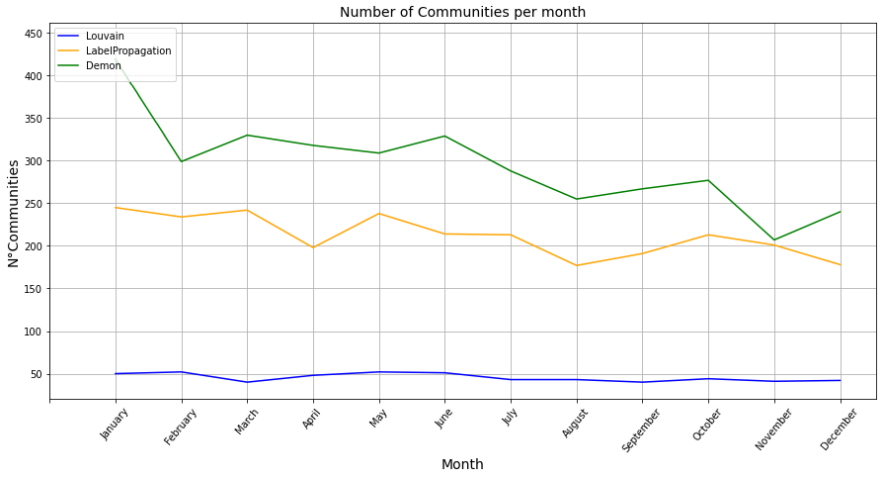

Figure 26: Comparison of algorithms by community

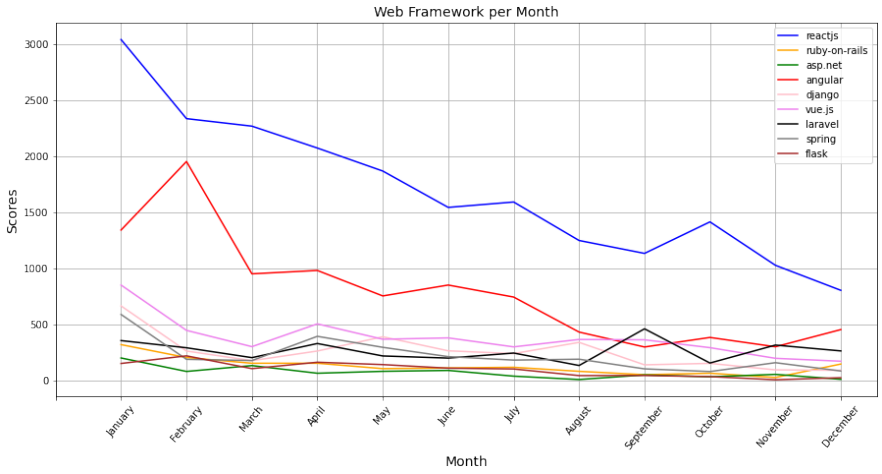

Figure 27: Tag distributions by month for Web Frameworks

5 Dynamic Community Discovery

In this section, a dynamic analysis of the created network is presented, contrasting it with the static results obtained. This dynamic analysis is placed within the context of a **temporal analysis of the dataset collected**. Our aim is to observe the evolutions in the computer-related interests of forum users throughout the year.

To observe trends in the popularity of different software tools and technologies, **scores for tags were aggregated monthly for three tag categories: Programming Languages, Web Frameworks, and Operating Systems**. Through these scores, it's possible to monitor which tags received more positive attention from the Stack Overflow community throughout the year.

Trends in web frameworks (Figure [27]) indicate greater community interest in Reactjs and Angular, possibly reflecting their widespread adoption by most web-oriented companies in the past year.

Python and Javascript remain the top two programming languages, as shown in Figure [28], as reported in Stack Overflow Trends18. Figure [29] **displays tags related to operating systems, where Android is the most dominant**, followed by iOS. This trend indicates a bias in the community towards discussions about programming for mobile platforms.

Having access to the temporal information provided in the stack overflow dataset. We can perform a dynamic community discovery.

A methodology of **Instant Optimal community discovery** was implemented, considering the intuitive division into monthly snapshots of temporal traces and utilizing the **Temporal Clustering**

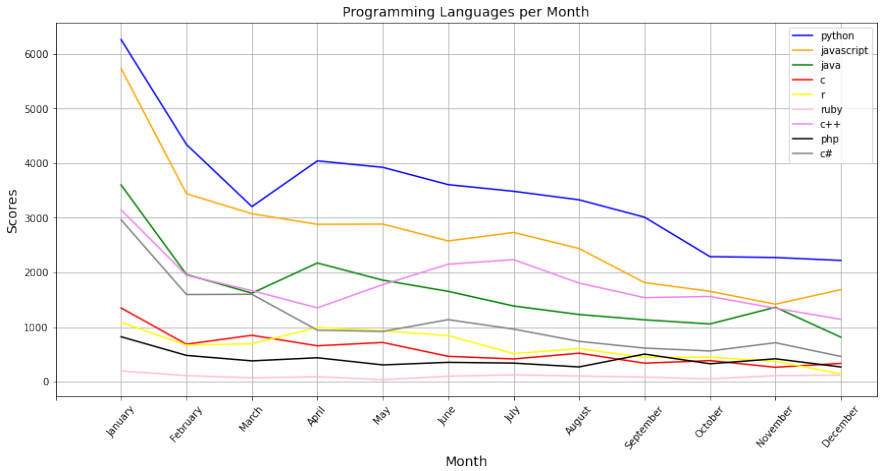

Figure 28: Tag distributions by month for programming languages

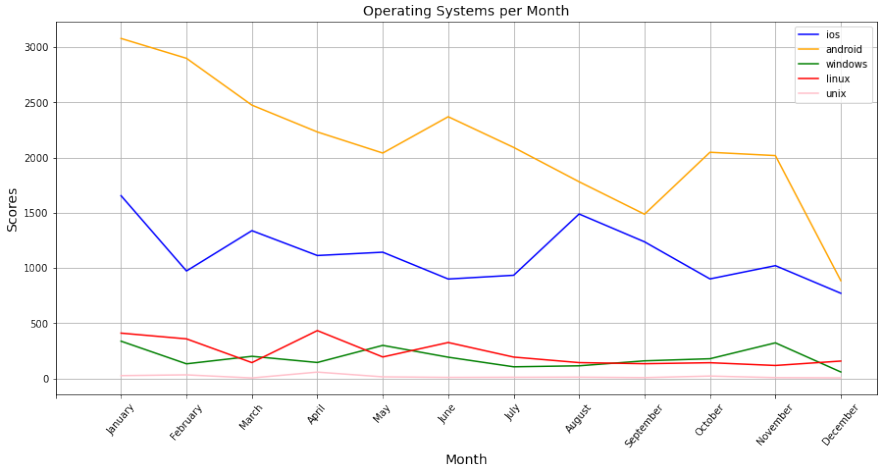

Figure 29: Tag distributions by month for Operating Systems

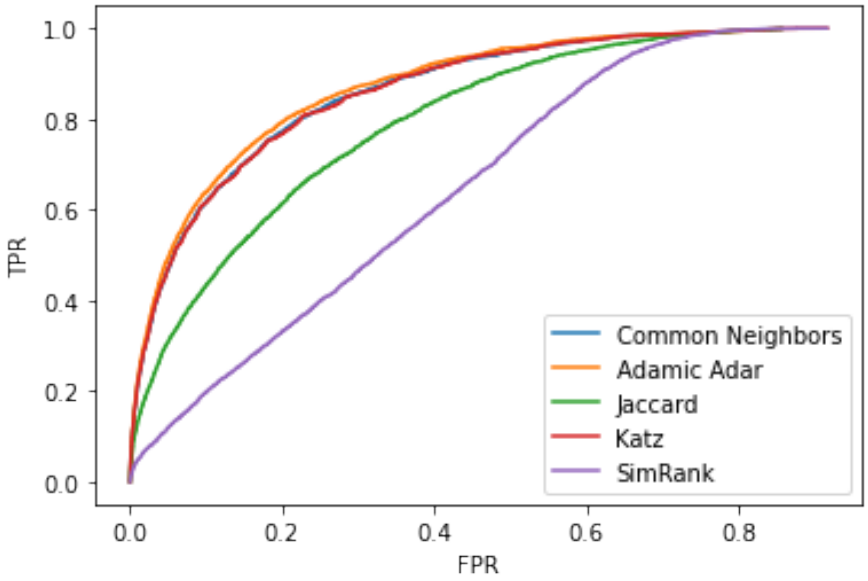

Figure 30: : ROC curve on G1 subgraph

structure provided by the cdlib library.

Before applying the standard methodology, a phase of choosing the static community discovery algorithm was carried out. This involved comparing the number of communities obtained for each snapshot by the models. Specifically, the **Louvain algorithm was chosen** because, as shown in Figure [26], it **identified a significantly lower number of communities than the others**, with a **very stable distribution throughout** the year.

From the obtained communities, **the life cycle graph** for the 12 considered months was calculated using the **Jaccard similarity function** and a cut on the graph to consider only edges whose communities have a similarity greater than 0. Cuts greater than 0 were not considered due to the low similarity values, which generally did not exceed 33%.

6 Link Prediction

The execution of this task, aiming to predict links between nodes in the network, was carried out using two subgraphs of the network due to computational constraints. T**he two subgraphs, G1 and G2, were obtained by selecting nodes with degrees** greater than 50 and 30, respectively. In G1, there are 597 nodes and 18,056 edges, while in G2, there are 1,057 nodes and 28,301 edges. This **sampling choice was made to retain only the most important nodes,** which are more likely to be connected, **thus reducing the potential for false positives.** Unsupervised prediction methods were evaluated by dividing the subgraphs into training and test sets with an 80% - 20% split.

Figures [30 - 31] show the ROC curves of the used methods, which can be categorized into three types:

- **Neighborhood-based:** Measures in this category assign a score between two nodes x and y, determined by the neighbors they have in common. **Common Neighbours, AdamicAdar, and Jaccard** were tested in this category.

- **Path-based: Katz**, a measure considering the number of paths between two nodes, falls into this category.

- **Ranking: SimRank,** which assigns a score based on the similarity of neighbors, is considered in this category.

The **AUC (Area Under the Curve**) of the ROC curve obtained by the methods for the two graphs is reported in Table [16].

It can be observed that the predictors based on neighborhood have obtained lower values when transitioning from the G1 sample to the G2 sample. This difference is likely due, by construction, to

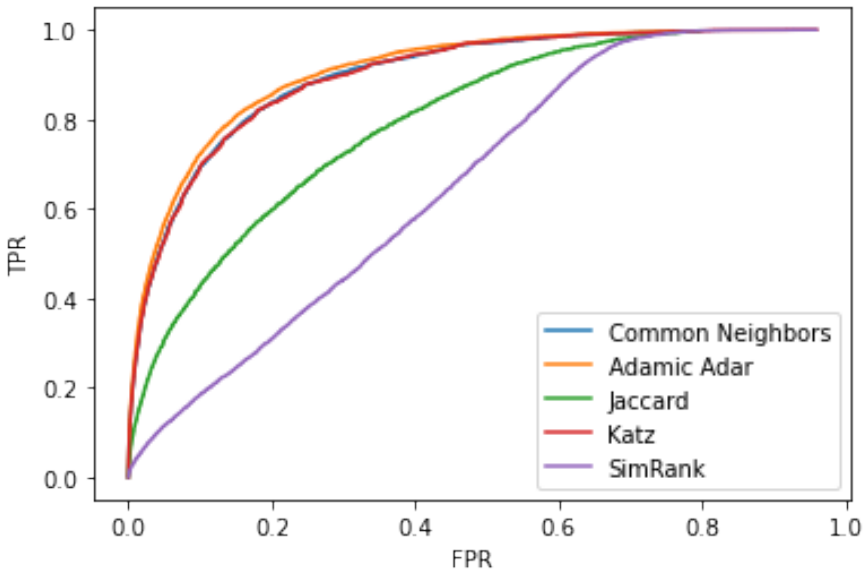

Figure 31: : ROC curve on G2 subgraph

	AUC for G1	AUC for G2
Common neighbors	0.73	0.69
Adamic adar	0.74	0.70
Jaccard	0.66	0.59
Katz	0.79	0.86
Sim Rank	0.59	0.62

Table 16: AUC method Link Prediction

the characteristic of G2, which has almost half of the nodes with a lower degree compared to the nodes in G1.

The other two methods, based on paths and ranking, obtained higher values for G2 compared to the previous test, particularly Katz. The increase in performance of methods not based on neighborhood, at the expense of Neighborhood-based methods, as the sample size increased, might not be unexpected. In fact, the connections between nodes in the constructed network are not driven by friendships and personal connections between users but rather by shared computer-related interests. Overall, excellent results were obtained, likely due to the **characteristics of the network being strongly connected** earlier.

Further explorations, such as the intersection of the top 100 links predicted by each method, did not yield additional information, except for the presence of nodes with tags related to Python for both the methods applied to G1 and those applied to G2.

The same Link Prediction methods were tested on subgraphs obtained from the first 8 communities reported by the Louvain method, considering a random sample of 597 nodes for each community. Results showed AUC values significantly lower than the previous tests, as visible in Table [17]. In these tests, nodes were not randomly sampled but considered based on their degree. However, even in this case, the Katz method (together with the SimRank method) obtained higher values than the other methods, once again proving to be the best for the type of data considered.

7 Spreading

In this section, spreading is analyzed, which allows for determining the characteristics of the network and, more importantly, evaluating the role that certain nodes play within it. Some nodes can accelerate or decelerate the spread of an idea/infection. **Two mathematical diffusion models (SI and Profile) are analyzed**, comparing the results obtained on the Stack Overflow network with those

Community	0	1	2	3	4	5	6	7
Common neighbors	0.015	0.044	0.152	0.047	0.017	0.020	0.024	0.006
Adamic adar	0.019	0.055	0.183	0.056	0.021	0.024	0.029	0.008
Jaccard	0.005	0.014	0.040	0.022	0.010	0.008	0.010	0.006
Katz	0.126	0.241	0.470	0.393	0.160	0.166	0.178	0.177
Sim Rank	0.108	0.183	0.314	0.355	0.147	0.156	0.158	0.231

Table 17: AUC Communities

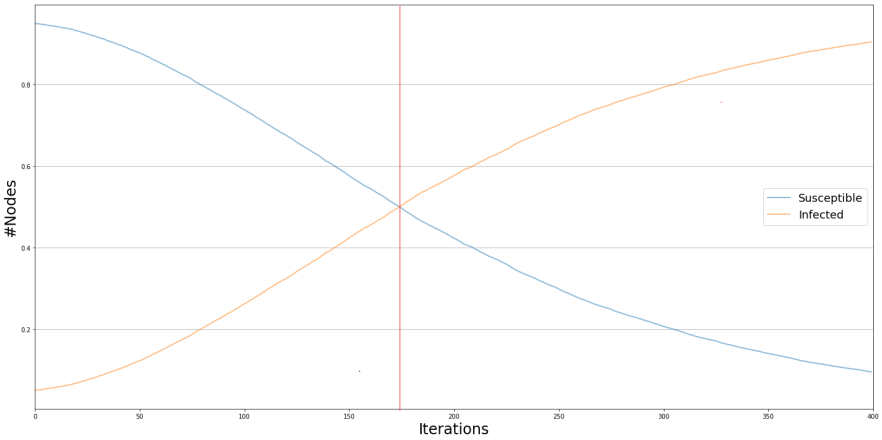

Figure 32: Diffusion BA Model

obtained on synthetic models.

For the network used, **it wasn't possible to consider spreading as the classic epidemic diffusion** but rather as the dissemination of information, such as the introduction of a new framework initially associated only with certain tags and gradually linked to other tags (languages, frameworks, libraries) through the development of the framework itself in terms of portability, integrity, and extensions. For this reason, the SI model (Susceptible-Infected) was preferred over its extensions SIS and SIR because the **states of healing and removal would be meaningless in this analysis.** The model considers two states for an individual: Susceptible (S) or Infected (I), without the possibility of being in the Removed (R) state. The model was executed using an initial percentage of infected individuals equal to 5% of the population with β (beta) equal to 0.001.

In Figure [32], it is shown that the spreading speed in a random network of the same size as the Stack Overflow network is lower compared to a Scale-free network. A ratio of healthy to infected individuals of about 50% is achieved at iteration 224 for the random network (highlighted in red in

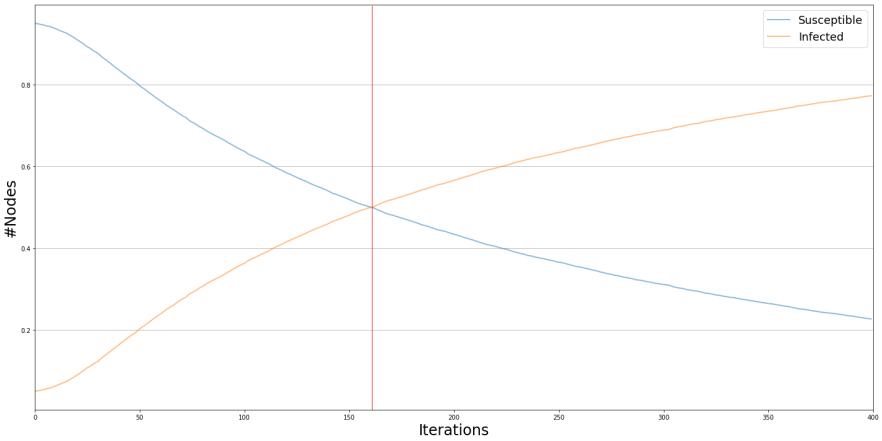

Figure 33: Diffusion RN Model

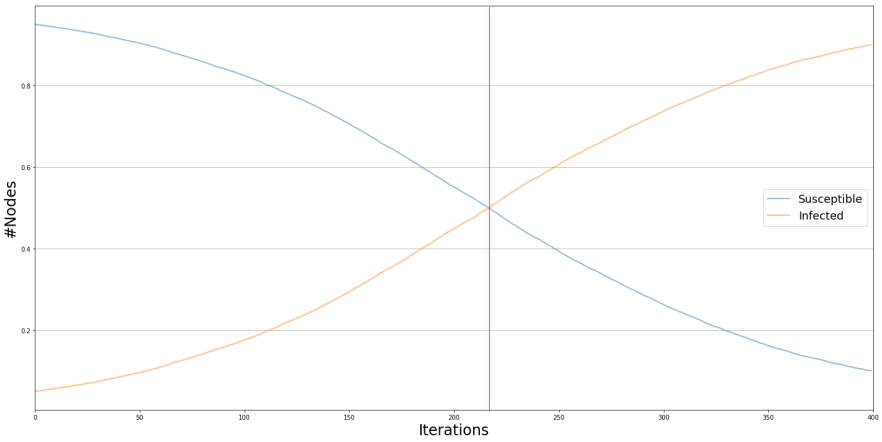

Figure 34: Diffusion ER Model

Figure [33]), while for the Stack Overflow network, this ratio is reached at iteration 163. The spreading dynamics for the Barabàsi-Albert network are similar to the expected behavior for the random network (but faster).

As observable in Figure [32], **information spreads through the BA network more quickly than the ER network** but more slowly than the Stack Overflow network, reaching a 50-50 ratio between healthy and infected individuals at iteration 180.

It has been observed (as expected) that as β increases, the spreading speed increases, as shown in Figure [34]. In this figure, it is demonstrated that running the model on the Stack Overflow network with$\beta = 0.01$ results in a healthy-to-infected ratio of about 50% as early as iteration 17, reaching network saturation at iteration 350. The presence of hubs significantly influences the speed of information spreading because hubs, having more opportunities to come into contact with an infected individual, are more likely to become infected and infect others. In the examined network, **hubs, represented by major programming languages (such as Python, JavaScript, Java)**, would likely be among the first to be involved in creating or extending a framework. Figure [35] illustrates the information spreading in the Stack Overflow network without hubs, demonstrating that the removal of hubs (at the same $/beta$) results in a slower spreading speed compared to the implementation in Figure [32]. In this case, a parity between the set of healthy and the set of infected individuals is reached at iteration 252, 89 iterations later than the network with hubs.

The second spreading technique considered was the **Profile model**, which assumes that the spreading process is only apparent, giving each node the **ability to choose whether to adopt a certain behavior based on their interests.** This deviates from the idea of classic epidemic spreading (not suitable for this network). Since the spreading process, in this context, starts from a set of infected nodes that have already adopted a certain behavior (S), four trials were conducted, differing in how this set is chosen: - **Selecting the most central nodes**, as shown in Figure [36]. - **Selecting marginal nodes**, i.e., those not belonging to the giant component, as shown in Figure [37]. - **Selecting the most central node from each community** identified by the Louvain algorithm, as shown in Figure [39]. - **Selecting random node**s (choosing an initial fraction of nodes equal to 0.06% of the sample), as shown in Figure [38].

For each susceptible node near a node in state S, a biased coin is flipped, with the bias set at 10% for each node in this implementation. The node will adopt the behavior if the coin flip is positive. If the adoption is rejected, a node enters the blocked state with a fixed probability of 10%. The adopter rate was set to 0.001, so at each iteration, 0.1% of nodes adopt the behavior spontaneously due to endogenous effects. This configuration of values was chosen because increasing the bias would have slowed down the speed at which nodes would have adopted the behavior, while increasing the adopter rate would have resulted in faster behavior diffusion.

Figure [36]-[39] shows the **diffusion behaviors for the four trials conducted.** As observed for the previous spreading model, it was also noted for the Profile model that the i**ncrease in nodes adopting the behavior and the consequent decrease in susceptible nodes are slower** when starting from random nodes compared to when the set S is initialized with central nodes.

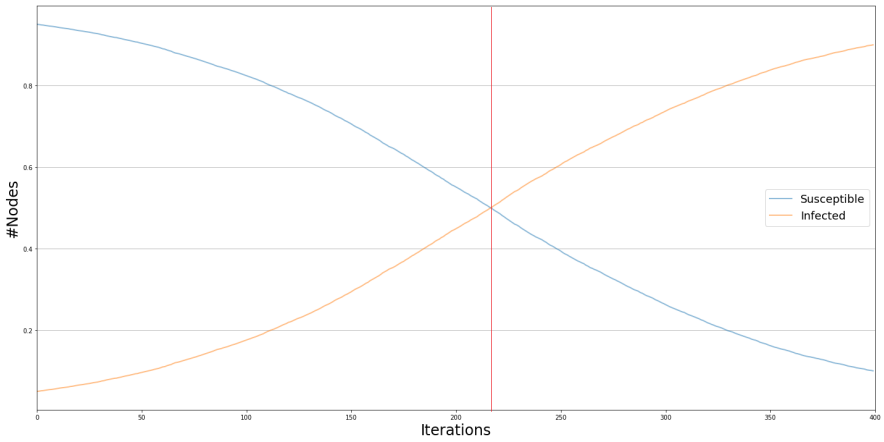

Figure 35: StackOverflow without hubs

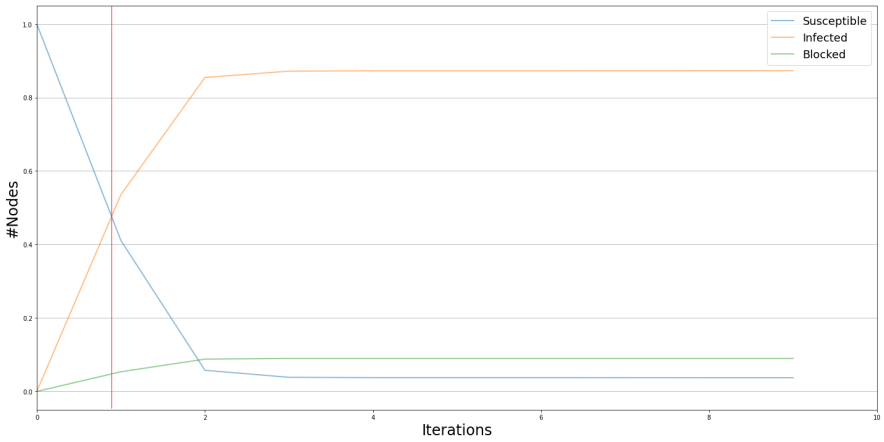

Figure 36: Profile: central nodes

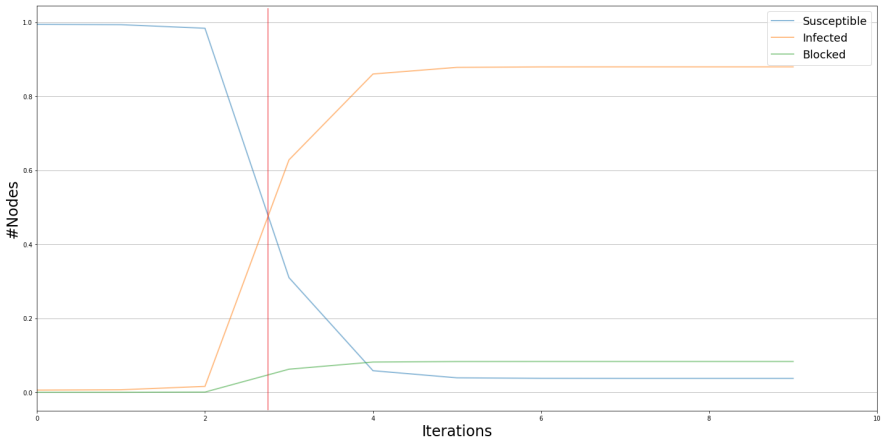

Figure 37: Profile: marginal nodes

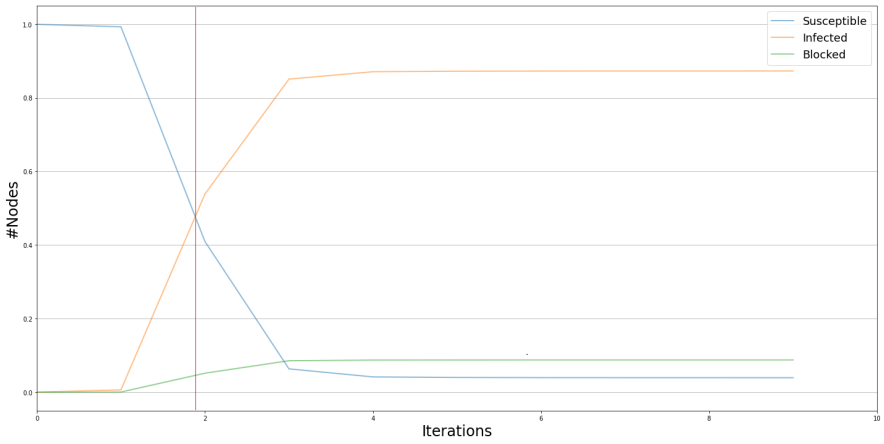

Figure 38: Profile: random nodes

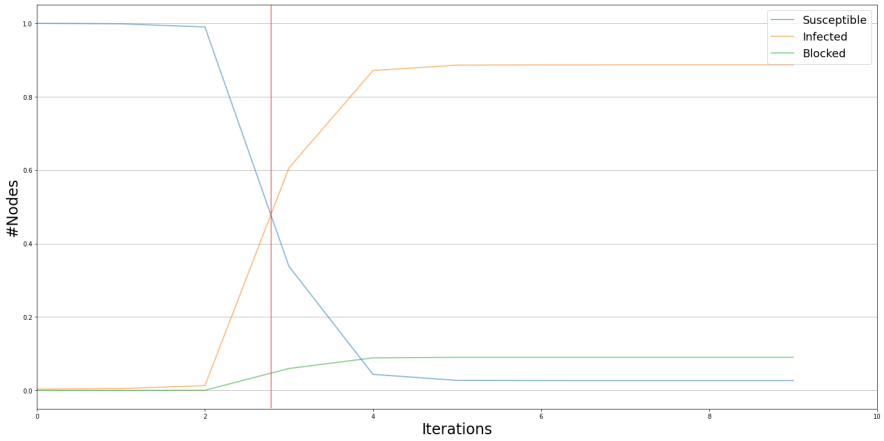

Figure 39: Profile: Louvain community nodes

8 Conclusions

The project successfully achieved its goal of analyzing the interactions between StackOverflow tags, especially in the tasks of Community Discovery and Link Prediction.

Further analyses could be conducted by considering a temporal window exceeding one year to gain a more comprehensive understanding of tag interactions and the evolving interests of forum users in the field of computer science. The work provides a foundational study that could potentially contribute to the improvement of the tag autocompletion system during question creation in the future.

9 References

[1] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. 855–864.

[2] Letizia Milli, Giulio Rossetti, Dino Pedreschi, and Fosca Giannotti. 2017. Information diffusion in complex networks: The active/passive conundrum. In International Conference on Complex Networks and their Applications. Springer, 305–313