Exploring Language Communities on GitHub

Antigoni M. Founta
Department of Computer Science
Aristotle University of Thessaloniki
Thessaloniki,https://www.sharelatex.com/project Greece
founanti@csd.auth.gr
UID: 647

Abstract

Graph analysis is a significantly interesting domain that can be applied to a great variety of subjects in order to produce zestful results. On the other hand, GitHub is a very rich source of data, with users, repositories and language annotation on these repositories. This study applies graph analysis on user-based language features derived from GitHub, in order to detect communities of languages that are frequently used together and attempt to identify some patterns on the usage of programming languages. There is a worldwide analysis graph as well as two location-specific graphs, so as to identify some location-based trends, as well. It is also shown that depending on the purpose of the research there are two types of schemes that might be suitable, the user-based and the repository-based.

1 Introduction

For the past decades, computer use has changed from mainly professional to heavily personal. In the same way computers have penetrated our everyday life, the use of programming has also dramatically increased both by professionals and as a hobby. That led to an upsurge of new programming languages that is increasing by the day. According to Wikipedia, there are thousands of different programming languages and many more are created every year. The classification of these languages is rather complicated, as most of the languages fall under more than one categories, adding to the fact that the very nature of those categories is controversial. Nonetheless, the most widespread ways to group programming languages is either by their programming paradigm (object-oriented, procedural, functional, logic etc.) or the intended domain of use (web programming, mobile-specific and even languages of general purpose).

Apart from the advances in the field of programming languages, the entire practice of programming has completely changed. In the beginning of the Information Age, IT professionals used to perform their tasks mostly alone. Nowadays, collaboration is key for the success of any software venture. Of course, new tools, methods and processes had to be introduced; most importantly version control systems. Today the most famous one is Git. It allows developers to write code on the same project at the same time, while tracking every change. It is widely used by a large number of independent developers, small start-ups and even seasoned corporations.

Users can access git through a server, such as Github, a web-based git repository hosting page. Github offers distributed revision control and source code management (SCM) and is considered the leading host of source code in the world. Social coding was popularised by the GitHub platform, which means that developers and coding contributors are able to interact with each other on the same platform they interact with their programming projects.

Many attempts have been made in order to analyze the GitHub data, both in Web Mining and Network Analysis. However, even though there are several studies regarding the users and the repositories of the platform, there is no official research considering using the Languages feature of GitHub. This study is attempting to fill this gap, by collecting and analyzing a number of GitHub users in order to summarise the languages they use and the co-occurrences of the languages used. The main goal of the current paper is to identify hidden patterns between the use of programming languages and discriminate the languages in distinct categories, using graph analysis and community detection. In other words, the main goal is to understand how languages are practically grouped in terms of the way developers use them. One more side task is to identify trends on programming languages, either worldwide or on specific locations. The locations chosen to be analysed below are California and Greece.

The main challenge that rises from this task is the ambiguity of the programming languages' categorization, meaning that many programming languages are multi-purpose and have more than one paradigms. This is a difficult challenge to overcome, but this study might actually help on this issue as the clustering of the languages is purely based on the way developers use them. On the other hand there is another important challenge raised, which comes from the own nature of the data. GitHub users are mostly web developers and thus the data derived from the platform are heavily biased towards web developing languages. Despite those challenges, however, the task of categorizing programming languages and understanding how they are combined is very interesting and can be proven very beneficial, especially on the developers scope.

2 Related Work

Github, as the leading social coding platform, has attracted plenty of researchers that have experimented with analyzing its data. As Kalliamvakou et al[1] mention, the introduction of social features in a code hosting site has drawn particular attention from researchers while the integrated social features, and the availability of metadata through an accessible API have made GitHub very attractive for software engineering researchers.

On the graph analysis domain, there is a number of contributions that have been made regarding GitHub; Thung et al[2], for example, explore the GitHub network structure by creating a developers' relationship graph and a projects' relationship graph and use PageRank in order to identify influential developers and projects. In this way, all of the network analysis research existing is focusing on either the users or the repositories of the users, with a view to understanding the social patterns that lie underneath, possible influencers etc.

This study, however, focuses on the patterns that exist between the use of programming languages on GitHub and the detection of language communities. That way, developers will get a hint of which languages are used jointly, and thus perhaps serve the same purpose, and language creators will get a hint of what their audience prefer and trust.

On the other hand, the location feature of GitHub users has also been previously seen, both on web mining researches[3] and network analysis studies[4], but always from the scope of the developers. Here, it is used to get a better sense of what are the languages used in some countries as well as what communities of languages, ergo developers, are identified, in order to mine some location-specific language trends.

Finally, when it comes to languages, there is a graph database centered around Ruby on Rails that is provided by Wagstrom et al[5] and consisted of 1166 Github projects. The authors' goal was to assist on understanding the context of relationships between projects, users and their activities.

3 Fundamentals

In this section there is an analysis of the dataset and the network construction and structure. The features used in the dataset, the dataset and some descriptive statistics as well as the structure of the network are described extensively below.

3.1 Data Collection

GitHub API provides a great variety of features about the users, such as their username, location, public repositories, number of people they follow or who follow them etc. In this study, the focus is concentrated mainly around the programming languages that the user has written and secondary on the user location. The library used for the collection of the data is PyGithub[6].

Languages is not a direct user feature but can be derived from the user's public repositories. Repositories are automatically annotated by GitHub with language labels and a weight named "Bytes of Code" for every language. Thereby, the final Languages feature per user is constructed by summarizing all the languages from all the public repositories of the user as well as the bytes of code per language. Forks are also considered public repositories, and thus appear in a user profile, but in this study are not participating to the language summarization process. This is primarily due to the fact that forks might be very large projects with plenty of languages and a user contribution can be very little, or even no at all, comparing to the whole project. Therefore, adding such an information might eventually complicate instead of accommodate the final results.

One major challenge regarding languages, that rises from the availabilty of GitHub data, is that private repositories are not accessible and so their languages cannot be summarized. Even though public repositories are more than enough in order to get a sense of correlations between languages, most users work on private repositories and thus there is a major information loss. Having said that, there is actually little one can do regarding this challenge and so from this point forward, this study considers that the knowledge derived from public repositories is as good as it gets.

The Location feature, on the other hand, is manually provided by the user and thus brings a number of different challenges to be considered, especially when it comes to pre-processing and matching. Except from the fact that location is not a mandatory field on a user profile, and is hence most of the times empty, it also faces issues that are commonly risen on any manually added field; many locations are not real or are not written precisely or even correctly, and even if the location is precise and clear, the structure of it is not always the same which makes matching a very hard task.

The final features for each user are:

$$\{\textsf{ID, username, location, public repositories, languages: bytes of code}\}$$

3.2 Dataset

The initial data collected were consisted of the 4000 first GitHub users, since the foundation of the company, as well as 150.000 extra users from 2012, a period in which GitHub was vastly growing in matters of users and repositories. The reason behind the choice of these two time periods is mostly due to the interest on active users versus the interest on recent trends and languages. The first users of GitHub are experienced and active while later subscribed users use more up-to-date languages, thus the trends derived from such a dataset can be considered contemporary. The next step was to filter the data based on the Location feature, where only users with declared locations participated in the final dataset, which is presented in Table 1 below.

Final Dataset			
	All Data	California	Greece
Users	39427	3456	103
Repositories	638547	70801	1755
Languages	289	123	58
Locations	∞	60	8

Table 1: Final dataset for each case

Figures 1, 2 and 3 present some descriptive statistics regarding the dataset. More specifically, in Fig.1 there is a pie with the top 10 programming languages in the entire dataset, where C is dominating almost half of the languages of the dataset with 52.1% against the second language, JavaScript, which follows with barely 12.3%. Figure 2 presents the edge distribution of the entire dataset, in other words the frequency of the co-occurrence of languages in user profiles, which seems to be power-law, as the almost linear distribution in the log-log figure shows. This means that most of the pairs of languages co-occur rather rarely, while there are a few languages whose occurence on the dataset is almost always paired. Finally Fig.3 (a) and (b) show the frequencies of all the places of the location-specific datasets. What is rather interesting in both figures is that the most frequent place is a very large city of the current location and then follows the location itself, meaning that many users prefer not to specify their location.

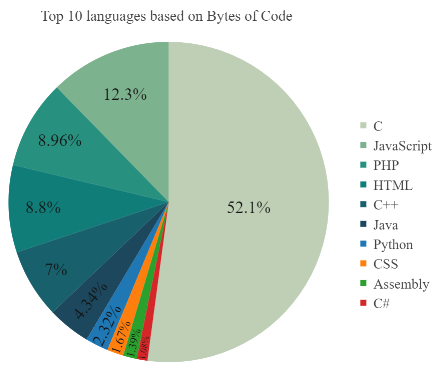


Figure 1: Top 10 programming languages in the entire dataset

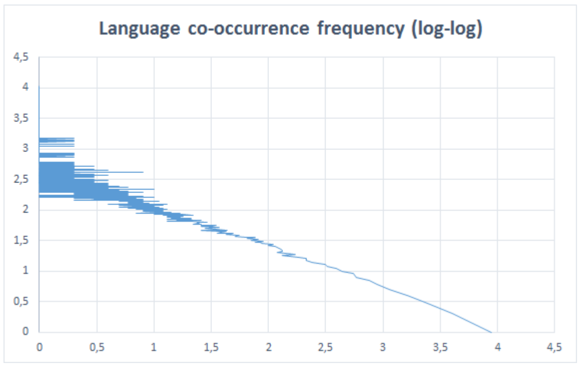
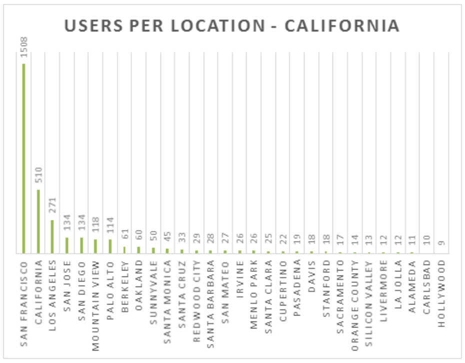
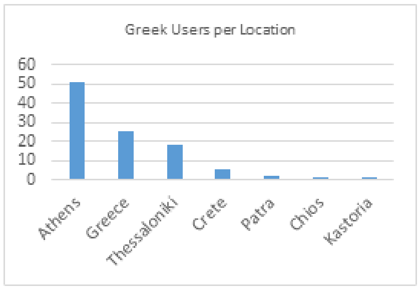


Figure 2: Edge distribution in the entire dataset



(a)



(b)

Figure 3: Location frequencies for California (a) and Greece (b)

3.3 Network Construction

As mentioned at the Introduction, there are four graphs constructed, two using the entire dataset and two location-specific. The reason behind this separation is that by visualising the entire graph, the reader will only get a sense of what underlying patterns lie between the use of languages, while in a location-specific graph there are also some language trends that can be identified and connected to the location. Furthermore, in most cases top languages are used instead of all the languages; three out of the four schemes make use of the top languages of the users. That is happening due to the fact that a user cannot be considered significantly proficient in more than three languages, but most of developers attempt to write some code using various languages, during their carreer or as a hobby. As a consequence, if all languages are considered, most of the languages will be paired one way or another.

All four graphs have the same structure and were built using the same layouts and filters. The total scheme with the four graphs is shown in Figure 4.

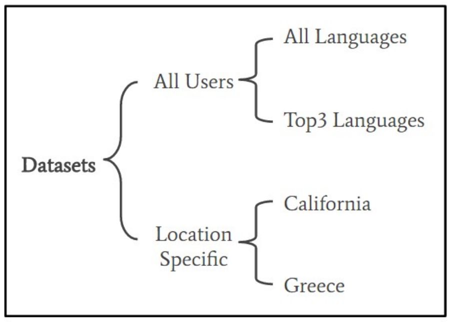


Figure 4: Total graphs scheme

The final graphs were created using Gephi[7], a graph visualization platform. In the first case, where all users are considered, the amount of languages as well as the connections was significantly high. To solve that, a Degree Range filter was used, where the threshold of the minimum degree allowed depended on the size of the graph. Finally, after experimenting with a number of layouts, the best one to visualize the data was Force Atlas 2, thus it is used in all four graphs.The structure of the graphs goes as follows:

- **Nodes**: Languages
  - Attribute: *Total Bytes of Code per Language*
  - Label Size: *Degree Range*
  - Node Size: *Bytes of Code Attribute*
- **Edges**: Pairs of Languages that co-occurred in at least one user profile
  - Weight: *Amount of users that use both languages*

Lastly, in order to detect communities from the graphs, modularity was computed using the default algorithm from gephi[8], which measures how well a network decomposes into modular communities. The higher the score, the better the results for the communities. The ideal values for an appropriate decomposition, according to current bibliography, lies between 0.3 and 0.7. In the Results section, modularity score is reported for each one of the graphs.

In addition, the randomization parameter of the modularity algorithm might produce better results, depending on the size of the graph. The default value of randomization is 1.0 and for the purposes of this paper the values varied from 0.8 to 1.2, depending on the number of communities identified to the default value. Last but not least, one more parameter that needs tuning and sometimes also helps increasing the modularity score is the use of edge weights, if existing. During the experimentation on the graphs for the resulting communities, weights were also considered but did not always improve the value of modularity, so in some cases were used while in others not.

The expected outcome of the community detection is a number of groups that divide languages effectively. The interpretation of those groups is expected to be related to either the purpose of use of the languages participating, or the programming paradigm. However, because of the great ambiguity that surrounds programming languages categorization, no standard groups are expected. The only exception in that is the Web-oriented group of languages, which is expected to be identified in most if not all cases. This is due to the bias of GitHub to web developers reported earlier.

4 Results

4.1 All Users - All Languages

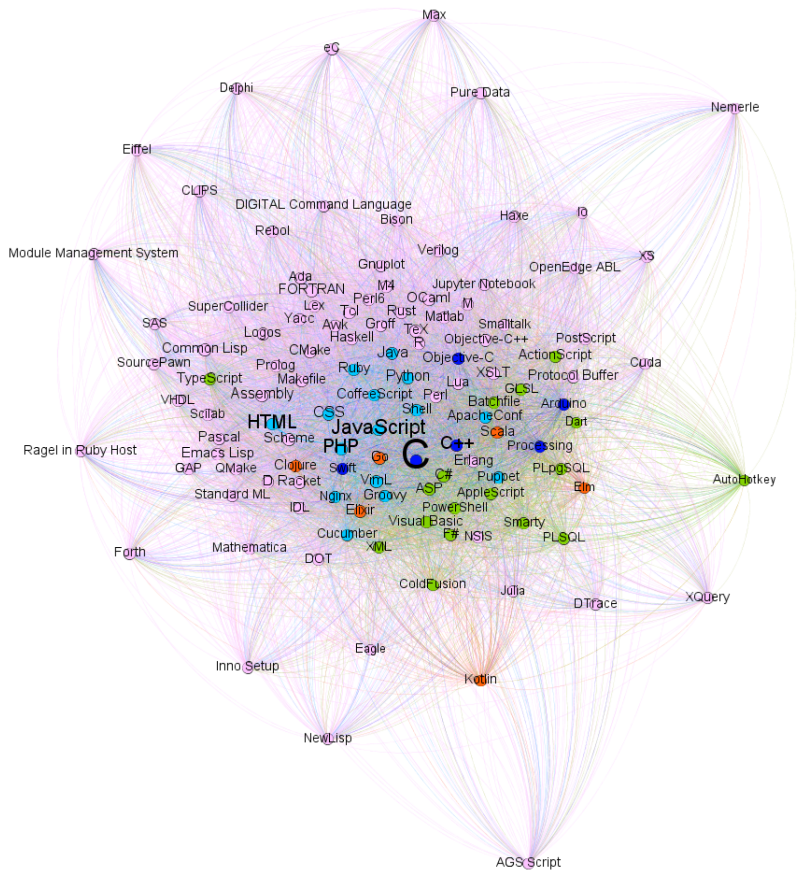


Figure 5: All Users - All Languages, User-based graph

Figures 5, 6 and 7 and Table 2 present the results of the All-Languages user-based graph. More specifically, Figure 5 is the graph itself, color coded based on the community detection, Figure 6 is an attempt to interpret the resulting communities and Figure 7 is consisted of the 10 most frequent sets of languages as well as the 10 most frequently used languages. Table 2 shows some metrics and properties of the graph such as the Avg. Degree, the Network Diameter, the Avg. Path Length etc.

There is only one Connected component and the value of the network's diameter is 2, which means that the network is very well connected, and almost all languages are connected to each other. Furthermore, the average degree value is 139, which compared to the number of nodes shows that indeed the vast majority of languages is well connected with most of the other languages. In addition, the average clustering coefficient value is also very high and finally the value of Modularity is significantly low which means that the knowledge we get from the communities of the graph is no better than that of a random graph.

An explanation for this phenomenon lies at what was mentioned earlier regarding developers and languages; developers tend to get in touch with various languages during time but they do not get proficient to many of these.

For all the abovementioned reasons, the graph that follows on the next section considers using only the top languages for each user in order to try and resolve this issue.

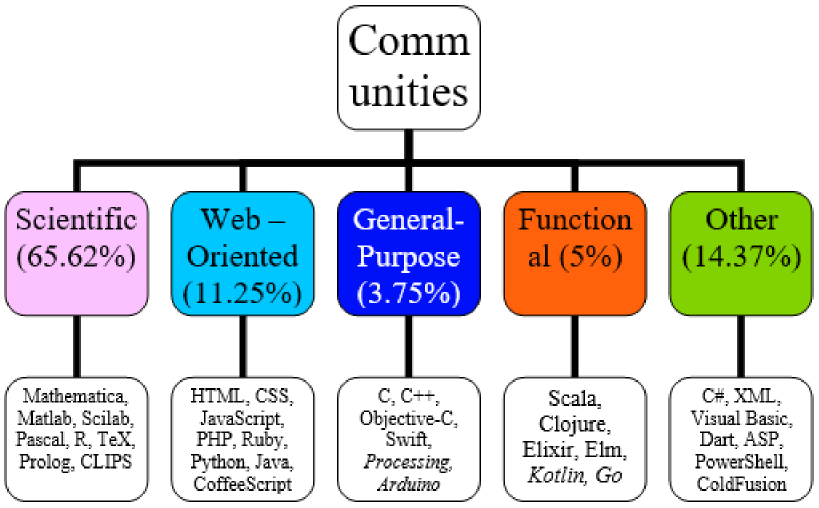
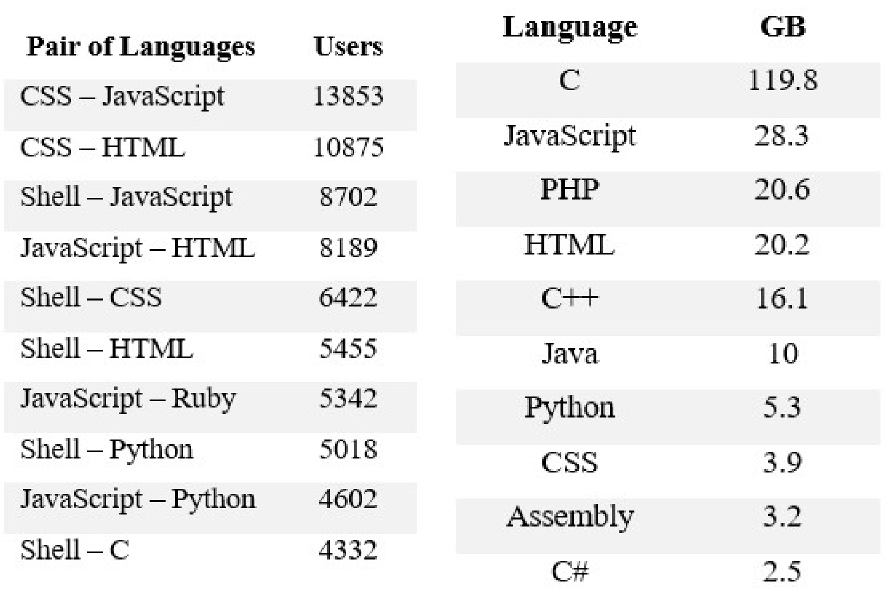

Communities
Scientific (65.62%)
Web – Oriented (11.25%)
General-Purpose (3.75%)
Functional (5%)
Other (14.37%)
Mathematica, Matlab, Scilab, Pascal, R, TeX, Prolog, CLIPS
HTML, CSS, JavaScript, PHP, Ruby, Python, Java, CoffeeScript
C, C++, Objective-C, Swift, Processing, Arduino
Scala, Clojure, Elixir, Elm, Kotlin, Go
C#, XML, Visual Basic, Dart, ASP, PowerShell, ColdFusion

Figure 6: Interpretation of the detected communities

Nodes: 160	Edges: 11138	Type: Undirected	Filter: Degree >100
Avg. Degree	139.225	Avg. Path Length	1.124
Diameter	2	Avg. Clustering Coefficient	0.913
WCC	1	Modularity (using weights)	0.068

Table 2: Metrics and Properties of the Graph

In this graph, the most commonly connected language is Shell scripting, which is seen in 5 out of 10 language co-occurrences (Fig. 7a), followed by CSS and JavaScript. However, despite the popularity of Shell between users, the pairs of the basic web languages (meaning HTML, CSS and JavaScript) still surpassed this, proving once again that in most cases (meaning most User profiles) they co-occur. Moreover, as it seems, even in this very tightly connected graph the web-oriented community is far better distinguished than the rest of the communities, with less outliers.



Pair of Languages	Users
CSS – JavaScript	13853
CSS – HTML	10875
Shell – JavaScript	8702
JavaScript – HTML	8189
Shell – CSS	6422
Shell – HTML	5455
JavaScript – Ruby	5342
Shell – Python	5018
JavaScript – Python	4602
Shell – C	4332

(a)

Language	GB
C	119.8
JavaScript	28.3
PHP	20.6
HTML	20.2
C++	16.1
Java	10
Python	5.3
CSS	3.9
Assembly	3.2
C#	2.5

(b)

Figure 7: Top Pairs of Languages (a) and Top Languages (b)

4.2 All Users - Top3 Languages

The results of this Top-Languages user-based graph are presented at Figures 8, 9 10 and Table 3. More specifically, Figure 8 is the graph itself, color coded based on the community detection and Figure 9 is an attempt to interpret the resulting communities. Figure 10 is consisted of the 10 most frequent sets of languages identified in the data as well as an image showing how strongly JavaScript and CSS are connected. The reason there is no Figure regarding the 10 most frequent languages identified in the dataset, is that there are almost no changes compared to Figure 7b. Finally, Table 3 shows some metrics and properties of the graph.

As it happened with the previous graph, there is only one Connected Component and the value of the network's diameter is 2, which means that this network is also very well connected, and almost all languages are connected to each other. Furthermore, the average degree value is 34, which is also very close to the number of nodes and the average clustering coefficient value is still very high, even though it's lower compared to the previous graph. Finally the value of Modularity is still rather low but with an important increase compared to the All-Languages graph. This could probably mean that the decrease on the number of the languages helped improving the results. However, the results are still not quite satisfactory. Some possible ideas on what could help resolve these issues and improve the results are discussed on the Future Work section, by the end of this study.

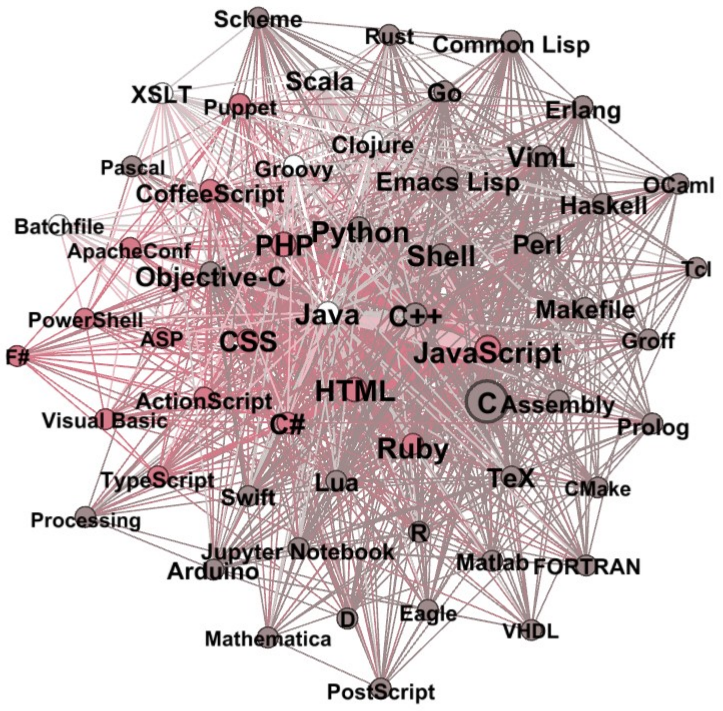


Figure 8: All Users - Top3 Languages, User-based graph

The most commonly connected language this time is JavaScript, which is dominating the top 10 pairs of Languages as seen in Figure 10a, followed by CSS. The edges connecting JavaScript with some of the other languages can also be seen in Figure 10b, where it is made clear that by far the most common combination of languages in the dataset is JavaScript and CSS. As discussed earlier, that result is more or less expected. It is interesting though that JavaScript overpassed Shell, that was the most common language met in pairs on the previous results. What is interesting about that is that it seems like Shell was a "noisy" language that users did not code a lot into, but most of the users have actually written once or twice. Thus, this is showing that the filtering of the languages actually improved the quality of the results, even though it did not help improve the results of the community detection. Last but not least, the web-oriented community is once again far better distinguished than the rest of the communities, with less outliers.

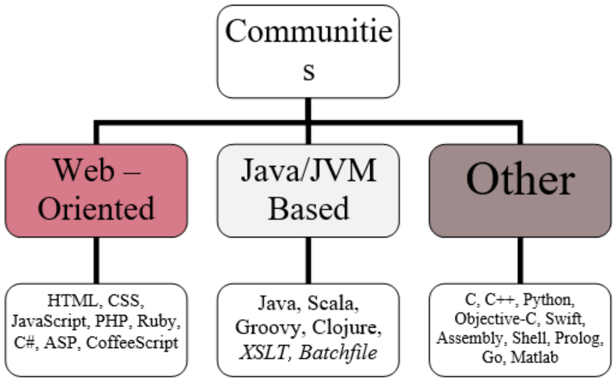
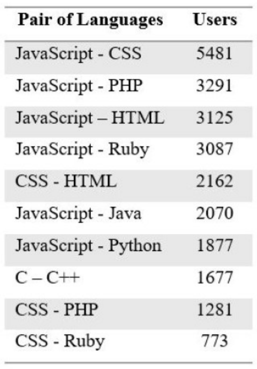

Communitie s
Web – Oriented
Java/JVM Based
Other
HTML, CSS, JavaScript, PHP, Ruby, C#, ASP, CoffeeScript
Java, Scala, Groovy, Clojure, *XSLT, Batchfile*
C, C++, Python, Objective-C, Swift, Assembly, Shell, Prolog, Go, Matlab

Figure 9: Interpretation of the detected communities

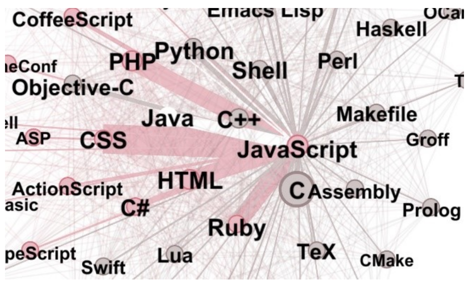
Nodes: 57 Edges: 969		Type: Undirected Filter: Degree >20	
Avg. Degree	34	Avg. Path Length	1.393
Diameter	2	Avg. Clustering Coefficient	0.801
WCC	1	Modularity (using weights)	0.139

Table 3: Metrics and Properties of the Graph



Pair of Languages	Users
JavaScript - CSS	5481
JavaScript - PHP	3291
JavaScript – HTML	3125
JavaScript - Ruby	3087
CSS - HTML	2162
JavaScript - Java	2070
JavaScript - Python	1877
C – C++	1677
CSS - PHP	1281
CSS - Ruby	773

(a)



(b)

Figure 10: Top Pairs of Languages (a) and JavaScript edge weight (b)

4.3 California - Top3 Languages

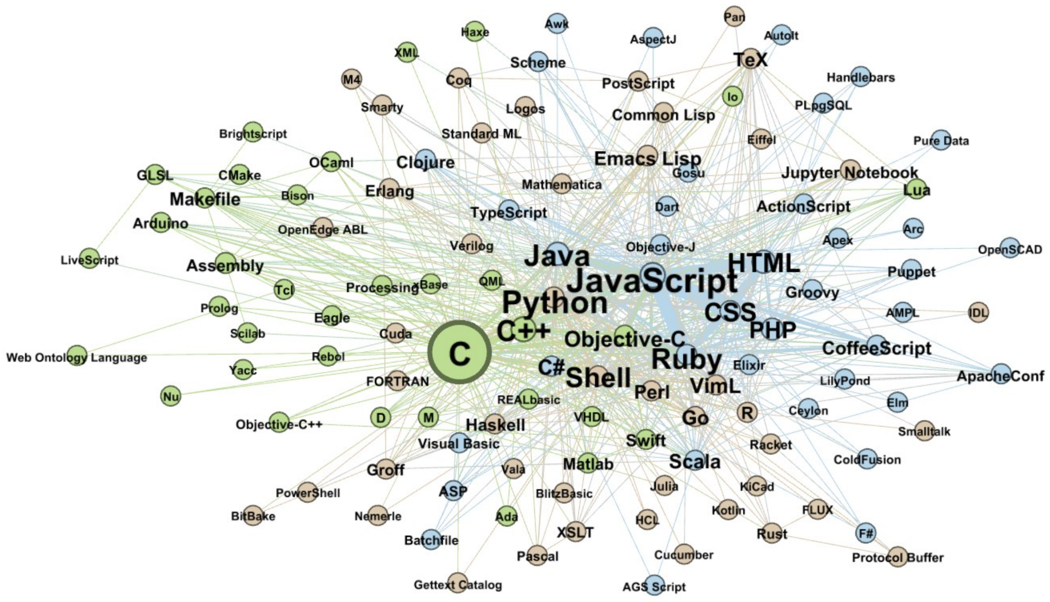


Figure 11: Users from California - Top3 Languages, User-based graph

Figures 11, 12 and 13 and Table 4 present the results of the Location-Specific user-based graph for California. More specifically, Figure 11 is the graph itself, color coded based on the community detection, Figure 12 is an attempt to interpret the resulting communities and Figure 13 is consisted of the 10 most frequent sets of languages as well as the 10 most frequently used languages. Table 4 shows the metrics and properties of the graph.

Unlike the rest of the graphs analysed previously, this graph is consisted of 3 Connected Components and the value of the network's diameter is 4, which means that the network is still strongly connected but not as tight as the previous ones. Also, the appearance of 3 components could indicate a successful discrimination between groups, ergo the improvement of the detected communities. Moreover, the average degree value is 11, which is much lower than the number of nodes. All these are indications that the languages might be quite separated. On the other hand, the average path length as well as the average clustering coefficient value are both still not improved, i.e. an average path length of 2 indicates strong connections. Finally the value of Modularity is not improved compared to the previous graph, even though the data are much less.

Once again, the most commonly connected language is JavaScript, as seen in Figure 13a, followed by CSS, HTML and Ruby, and the Web-oriented community is still very well discriminated, exactly like before.

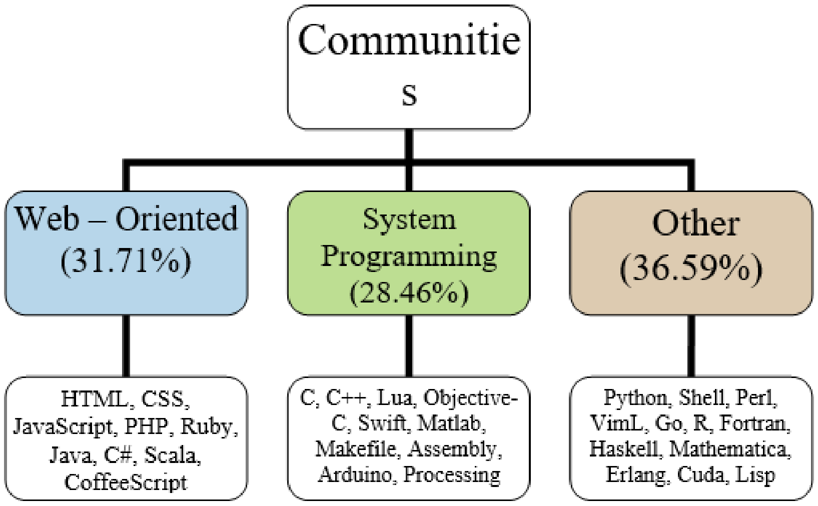
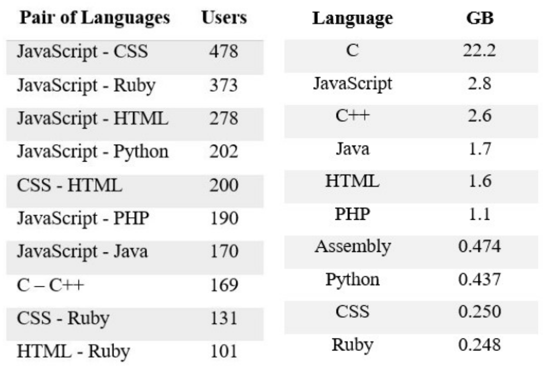

Communitie
s
Web – Oriented (31.71%)
System Programming (28.46%)
Other (36.59%)
HTML, CSS, JavaScript, PHP, Ruby, Java, C#, Scala, CoffeeScript
C, C++, Lua, Objective-C, Swift, Matlab, Makefile, Assembly, Arduino, Processing
Python, Shell, Perl, VimL, Go, R, Fortran, Haskell, Mathematica, Erlang, Cuda, Lisp

Figure 12: Interpretation of the detected communities

Nodes: 123 Edges: 686		Type: Undirected Filter: —	
Avg. Degree	11.154	Avg. Path Length	2.219
Diameter	4	Avg. Clustering Coefficient	0.833
WCC	3	Modularity (using weights)	0.130

Table 4: Metrics and Properties of the Graph



Pair of Languages	Users
JavaScript - CSS	478
JavaScript - Ruby	373
JavaScript - HTML	278
JavaScript - Python	202
CSS - HTML	200
JavaScript - PHP	190
JavaScript - Java	170
C – C++	169
CSS - Ruby	131
HTML - Ruby	101

(a)

Language	GB
C	22.2
JavaScript	2.8
C++	2.6
Java	1.7
HTML	1.6
PHP	1.1
Assembly	0.474
Python	0.437
CSS	0.250
Ruby	0.248

(b)

Figure 13: Top Pairs of Languages (a) and Top Languages (b)

4.4 Greece - Top3 Languages

Finally, the results for the last graph are presented on Figures 14, 15 16 and Table 5. More specifically, Figure 14 is the graph itself, which is significantly smaller than all the previous once and is color coded based on the community detection. Figure 15 is an attempt to interpret the resulting communities and Figure 16 is, just like before, consisted of the 10 most frequent sets of languages as well as the 10 most frequently used languages. Table 4 shows the metrics and properties of the graph.

This graph is consisted of one Connected Component, which can also be seen visually, and the value of the network's diameter is 4, which means that the network is rather strongly connected but if analysed in proportion the graph size, this diameter seems quite normal. Also, the average degree value is 5, which is lower than the number of nodes and the average clustering coefficient value seems fairly decreased. Last but not least, Modularity value is improved compared to all previous graphs, but all these improvements could actually happen due to the fact that there are so little data.

The detected communities include a relatively distinct web-oriented community, but also include another well identified community, regarding iOS development. Even though there is a small amount of languages assigned to this cluster, both languages are used on the same purpose. Even the orange community, regarding System Programming languages, can be considered more robust.

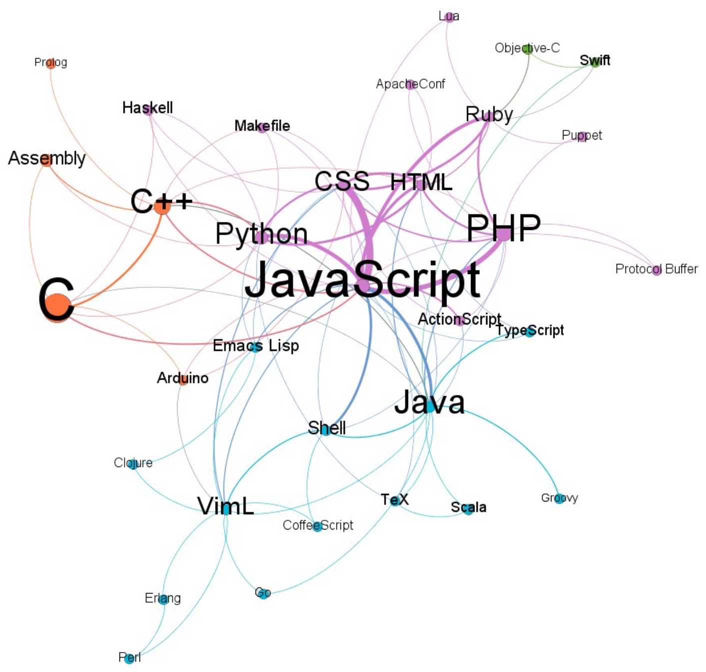


Figure 14: Users from Greece - Top3 Languages, User-based graph

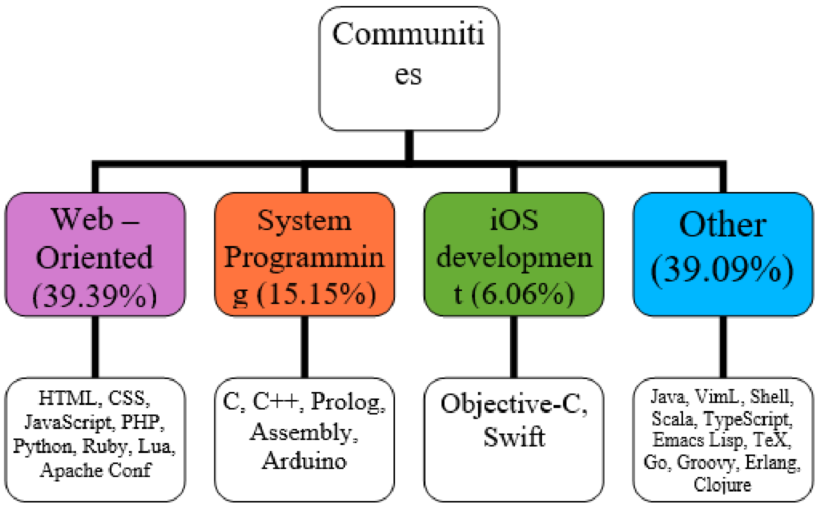
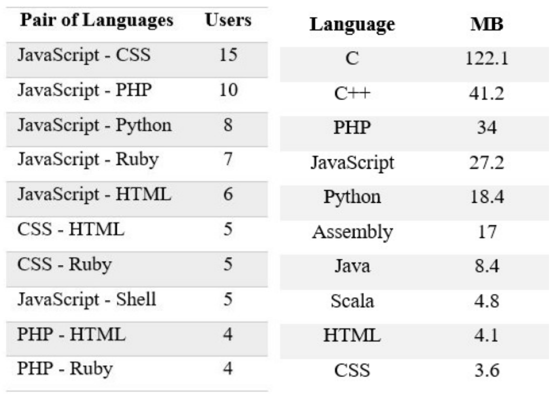

Communiti es
Web – Oriented (39.39%)
System Programmin g (15.15%)
iOS developmen t (6.06%)
Other (39.09%)
HTML, CSS, JavaScript, PHP, Python, Ruby, Lua, Apache Conf
C, C++, Prolog, Assembly, Arduino
Objective-C, Swift
Java, VimL, Shell, Scala, TypeScript, Emacs Lisp, TeX, Go, Groovy, Erlang, Clojure

Figure 15: Interpretation of the detected communities

Nodes: 33	Edges: 87	Type: Undirected	Filter: —
Avg. Degree	5,273	Avg. Path Length	2.218
Diameter	4	Avg. Clustering Coefficient	0.706
WCC	1	Modularity (using weights)	0.203

Table 5: Metrics and Properties of the Graph



Pair of Languages	Users
JavaScript - CSS	15
JavaScript - PHP	10
JavaScript - Python	8
JavaScript - Ruby	7
JavaScript - HTML	6
CSS - HTML	5
CSS - Ruby	5
JavaScript - Shell	5
PHP - HTML	4
PHP - Ruby	4

(a)

Language	MB
C	122.1
C++	41.2
PHP	34
JavaScript	27.2
Python	18.4
Assembly	17
Java	8.4
Scala	4.8
HTML	4.1
CSS	3.6

(b)

Figure 16: Top Pairs of Languages (a) and Top Languages (b)

Because of the problems identified earlier regarding the matching of languages to create pairs, there is one last graph presented in this study which was experimentally constructed in order to check whether or not the results will improve. The difference in this graph emanates from the way the language co-occurrences are counted. More specifically, instead of using user profiles to summarize when languages are often used together, in this graph the co-occurrences are measured based on each repository of each user. In other words, for every user's public repository, if two languages are used in the same repository they make a pair. This aims at dramatically decreasing the connections between languages in order to clarify which languages are actually strongly connected.

Even though the experimental graph is based on the Greek Users dataset that is very small, it seems like the connections between the languages are decreased and the communities are somewhat improved. The modularity value for this graph is 0.23, the highest modularity value reached by all the graphs.

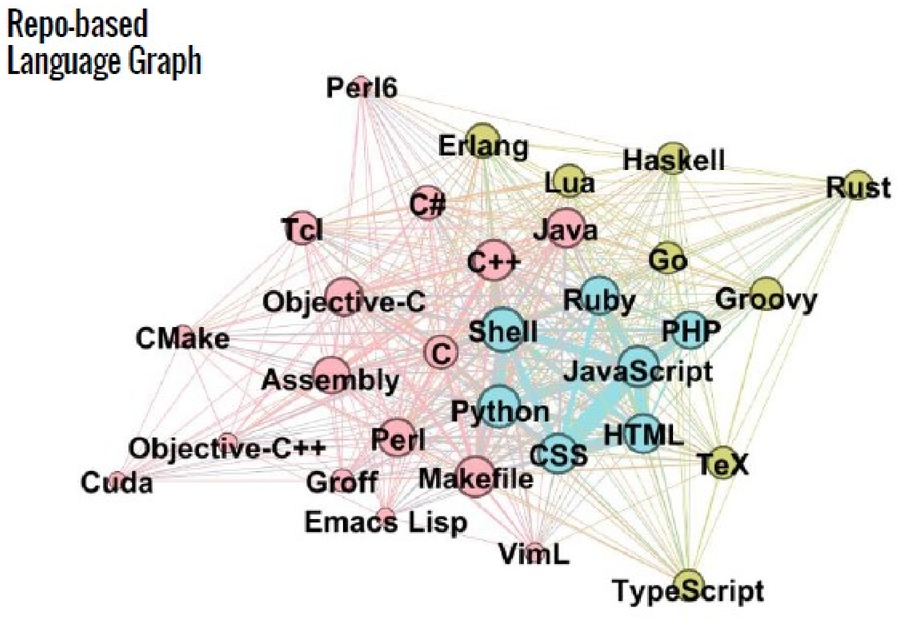


Figure 17: Repository-based graph on Language Co-Occurency

5 Conclusion

This study focuses on the detection of underlying patterns and communities between programming languages, in order to get a better understanding of which languages are used together, and show some location-specific trends. The locations considered are chosen because of the great difference between them regarding the amount of developers living in these locations.

Regarding the community detection, the final communities resulted from the graph analysis are not very well formed and do not discriminate languages profoundly. However, in some cases such as the Web-oriented community, the algorithm always succeeded in recognizing that group. This is probably due to the fact that there is a heavy bias on GitHub data about web development, as the majority of GitHub users are web developers. Despite that, it is still important that both the web community as well as the most important web languages were always presented together both in graphs and in pairs of languages. This means that, with the correct amount of data and some necessary tuning, the algorithm can succeed in discriminating language groups. Another conclusion that can be made about the web-oriented community is that some languages were always assigned to the same community, like HTML, CSS, JavaScript and PHP, while others like Python and Java did not always belong to the same cluster, in each graph. This is due to the fact that such languages are multi-purpose and are very difficult to be classified in one standard category.

An interesting result regarding the language co-occurrences and language frequencies is that in almost all cases JavaScript was the most "matched" language, meaning in most of the pairs one of the two languages was JavaScript. On the other hand, C was the most popular language in matters of Bytes of Code written, in all four Language Frequency Figures, while there was only one or two pairs including C. What could be an explanation for that is that perhaps C users are not interested in writing or learning any other language, they might be "language-introverts". On the other hand, users that are proficient in JavaScript are most probably proficient in a number of other languages, including the top web-oriented, so can be referred as "language-extroverts".

Some scheme-oriented conclusions that show some interest are those discussed during the results, regarding the amount of edges connecting the languages. More specifically, with the user-based language graphs the reader can understand some general preferences of the developers, while repo-based language graphs are probably a better suit to understand patterns in use of languages and detect communities. Thus, a suggestion for future researches that might be interested on this subject is that if the point of the research is to generally understand patterns of languages then a repo-based scheme will probably yield better results, while for language trends' detection in specific locations user-based schemes can work rather effectively.

Finally some suggestions for future work would be to work on plenty more data and locations, in order to make some comparisons between locations and get more accurate results. Some other features provided by GitHub can also be used, such as the followers of the users or the stars of the repositories, in order to keep only the most influential users or repositories and create a graph with only their languages. Last but not least, a nice idea would be to apply association rules on languages for community detection.

References

[1] Kalliamvakou, Eirini, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. "The promises and perils of mining GitHub." In Proceedings of the 11th working conference on mining software repositories, pp. 92-101. ACM, 2014.

[2] Thung, Ferdian, Tegawende F. Bissyande, David Lo, and Lingxiao Jiang. "Network structure of social coding in github." In Software maintenance and reengineering (csmr), 2013 17th european conference on, pp. 323-326. IEEE, 2013.

[3] Takhteyev, Yuri, and Andrew Hilts. "Investigating the geography of open source software through GitHub." (2010).

[4] Figueira Filho, Fernando, Marcelo Gattermann Perin, Christoph Treude, Sabrina Marczak, Leandro Melo, Igor Marques Da Silva, and Lucas Bibiano Dos Santos. "A study on the geographical distribution of Brazil's prestigious software developers." Journal of Internet Services and Applications 6, no. 1 (2015): 1.

[5] Wagstrom, Patrick, Corey Jergensen, and Anita Sarma. "A network of Rails a graph dataset of Ruby on Rails and associated projects." In Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on, pp. 229-232. IEEE, 2013.

[6] PyGithub library, https://pypi.python.org/pypi/PyGithub

[7] Gephi Software, https://gephi.org/

[8] Blondel, Vincent D., Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. "Fast unfolding of communities in large networks." Journal of statistical mechanics: theory and experiment 2008, no. 10 (2008): P10008.