Bilkent University

Department of Computer Engineering

CS425 Project Report

Community Detection Implementation on GitHub

Oğuz Demir
Mehmet Furkan Şahin
İlteriş Tabak
Süleyman Can Özülkü

Instructor: Assoc. Prof. M.Mustafa Özdal

Project Report
December 21, 2016

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS425

Table of Contents

Table of Figures

1. Introduction

Social networks are all around us in today's world. People are always connected to each other online and they maintain their relationship through social media tools. Therefore, social networks give us some clues about the real world networks between communities. The tools such as Facebook, Instagram, Snapchat are massively used and they are used as the backbone of today's social life. Hence, for example, it wouldn't be wrong to assume the networks in Facebook give us some general understanding on the social relationships between people. In our project, we wanted to examine the relationships between programmers just like us and GitHub is the best environment to analyze such a network since it is a massively used tool among any type of programmer (software developers, computer scientists, white hat hackers etc.).

As an output of this project we wanted to give a brief summary on main characteristics about the communities in the network of a person's professional life. We specifically examine the network for a person and examine his own relationships not the whole GitHub community. Through his connections in GitHub we are able to get information on his favorite programming languages, his main field of interest and the connections in the industry. While determining the possible variables on the relationships between people, we decided to consider follower/following, company and the organization information.

The results we get from the implementation were pretty convincing on our side. To be able to judge the results, we used Mehmet Furkan Sahin's account[1] as a starting point we were able to detect communities from his school, working life, and social life as you can see in the analysis and conclusion parts.

[1] https://github.com/furkansahin

2. Collecting Dataset

Since we aimed to make our analysis based on our profiles, no existing datasets have been used in this project and we formed a dataset for one of group members. Collecting the dataset from GitHub was very challenging process of the project because of 2 reasons: Firstly, the GitHub's API allows only 5000 requests per hour from single IP address and with authentication of single GitHub account. Second reason which complicates the data collecting process is that the connections to the GitHub's API is very expensive, hence we decided to put the collected data into a database instead of downloading the data for each execution of analysis. For these reasons, we decided to distribute the data collection process over a master server and bunch of workers.

In our system, the master server mainly holds the queue for breadth first search algorithm, and the workers download the information for users taken from the master's queue and save the information into their local databases. We used PostgreSQL for database system, which is very easy to use and open-source system. Also, it allows us to convert the dataset to a CSV file easily, which is simple to read programmatically. For the data collection, we also need the level data of users according to our start point because we should know the point where the search should be stopped (i.e. for 3 level search, start->a follower-> a follower -> a follower is a 3rd level user, whose followers and followings should not be included in database.). Therefore, we made some enhancements for breadth first search algorithm in order to make it concurrent.

For single threaded breadth first search algorithm, indicating the search level is relatively easy, and can be solved by putting separator markers to the queue at the end of each level. However, this is not possible for a multi-threaded search since a separator can only be taken

from one-thread, where the other threads miss the level information. In order to solve this problem, we separated the searching queue into 2 queues such that the first queue holds the users for currently searched level and the second queue hold the users for next level. Also, the workers check for previously processed users which are held in a set by master, in order not to reprocess. The workers and master's algorithms is changed to following way:

The workers:

1- take a user from the first queue

2- process the user, find its followers and followings

3- put the followers and followings to second queue for next level of search

4- repeat until first queue becomes empty or exit code is taken from first queue.

(when exit code is taken, process of worker is terminated)

The master:

1- start the search process by putting the first user into first queue

2- wait until completion of a level of search.

3- wait 15 seconds to allow the workers to complete their current user processes

(This time can be arranged according to processing power of workers.)

4- increase the level, is maximum level is reached:

4.a- put exit code to first queue until no worker stays alive.

If maximum level is not reached:

4.b – copy the elements of second queue to first queue, clear second queue.

For these processes, another queue where login credentials is stored is held in master, a newly staring worker takes a login details from this queue and makes the authentication.

Also, distributed lock is used among master and workers, which prevents workers from manipulating the queues during copy process of master.

The data collections is held with the help of Hazelcast. It provides various data types and collections which can be stored distributed and accessed concurrently from different clients. Since master does not need much storage to hold the search queues and other information, we used just one server, so the data itself is not distributed. The Hazelcast clients can access and manipulate the data on servers, so we used this feature for establish the communication between master and workers and distribute the data fetching process.

3. Algorithm

The algorithm we used in our project is based on the paper [1] with the header *"Computing communities in large networks using random walks"*[2]. The paper introduces the algorithm called *Walktrap.* Walktrap is a random walking based algorithm which is already discussed in the class by Prof. Mustafa Ozdal.

3.1 Initial Graph

Our graph is weighted and undirected. We start with weighing the edges according to the relationship between two nodes. One node is used to indicate one person. Following relationship of a person adds the weight 0.25 and follower adds 0.25 more to the same edge if two people are in the same company or organization we assume that the relationship is stronger than a regular follower/following relationship and add 2 more to the same edge. After the initialization of the graph with the indicated parameter above *r distance* is

[2] https://arxiv.org/pdf/physics/0512106v1

calculated. *r distance* is another metric defined in the same paper and we are going to discuss in the next chapter.

3.2 Calculating Similarity

To find the communities, we need to be able to measure the similarities of people in the network according to the possibility of being in a specific node *k* after starting the random walk from the node *i*. As we discussed in the class, the mentioned possibility can be found by taking the *n*th. power of the weight matrix *P* for *n* being the number of steps during the random walk. After finding the probability matrix, the paper defines a new metric called *r distance. r distance* is calculated for each pair of nodes in the network according to their possibility of being in the node k in *n*th. step. Thus, memory requirement is O(n^2) to store *r distance*s. Mathematical definition of *r distance* can be seen in the following formula;

$$r_{ij} = \sqrt{\sum_{k=1}^{n} \frac{(P_{ik}^t - P_{jk}^t)^2}{d(k)}}$$

For the next steps of the algorithm, the same distance definition is transformed for the communities as well in a straightforward way. The probability to be in the node j in nth. step

$$P_{Cj}^t = \frac{1}{|C|} \sum_{i \in C} P_{ij}^t$$

after starting from the community C is defined as follows;

The r distance formula for communities can also be seen below;

$$r_{C1C2} = \sqrt{\sum_{k=1}^{n} \frac{(P_{C_1k}^t - P_{C_2k}^t)^2}{d(k)}}$$

3.3 Detecting Communities

In this step, the nodes in the graph are collected together in communities according to their similarity ratios formerly calculated as *r distance.* In this step, the algorithm requires to find the adjacent communities to merge which have the closest r distance difference. The two communities are found and merged according to Ward's method[3]. As it is stated in [1]

"At each step k, we merge the two communities that minimize the mean σ_k of the squared distances between each vertex and its community."

$$\sigma_k = \frac{1}{n} \sum_{C \in \mathcal{P}_k} \sum_{i \in C} r_{iC}^2$$

We iteratively go through the similarity rankings and calculate σ_k n-1 times and each time we calculate a new metric to detect the increase ratio on σ. The metric η_k is defined as;

$$\eta_k = \frac{\Delta\sigma_k}{\Delta\sigma_{k-1}} = \frac{\sigma_{k+1} - \sigma_k}{\sigma_k - \sigma_{k-1}}$$

[3] https://en.wikipedia.org/wiki/Ward's_method

If η_k is large it means that the communities in step k-1 are largely relevant. After calculating η_k for each step we pick the maximal set and the output gives the communities on that step. For example, the best partition contains 2 different communities in the following example

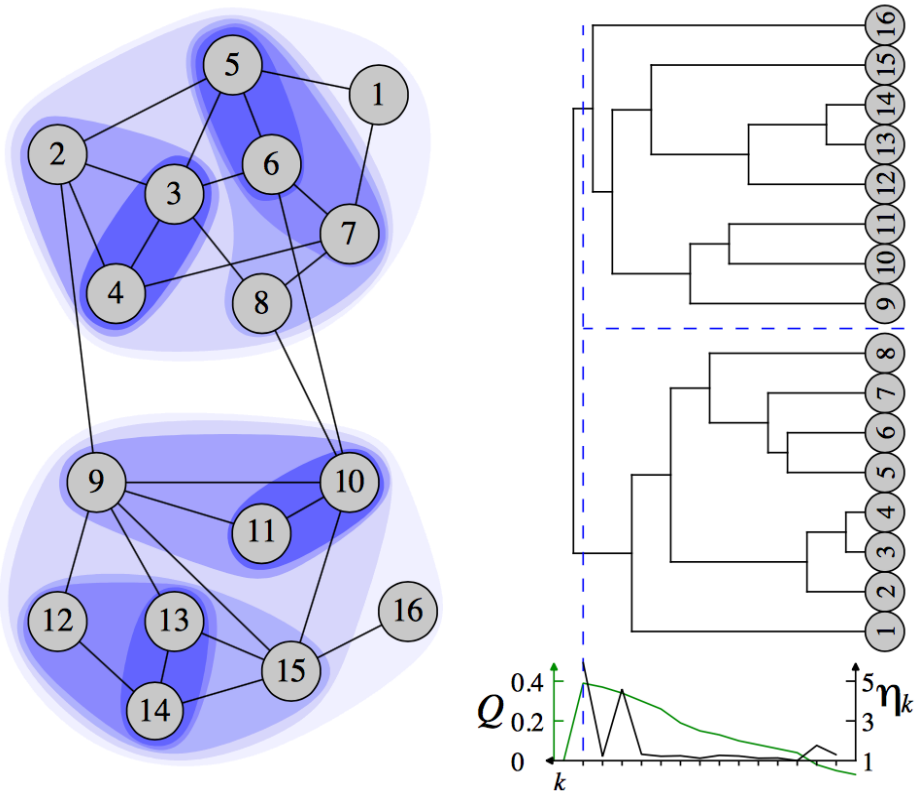

Figure 1: Example of community partition

4. Analysis

The analysis has been made for different length of random walks, from 2 to 6. Here is the screenshots for 2 step and 6 step. As reminder, one of group members, Furkan Sahin's

GitHub network is used.

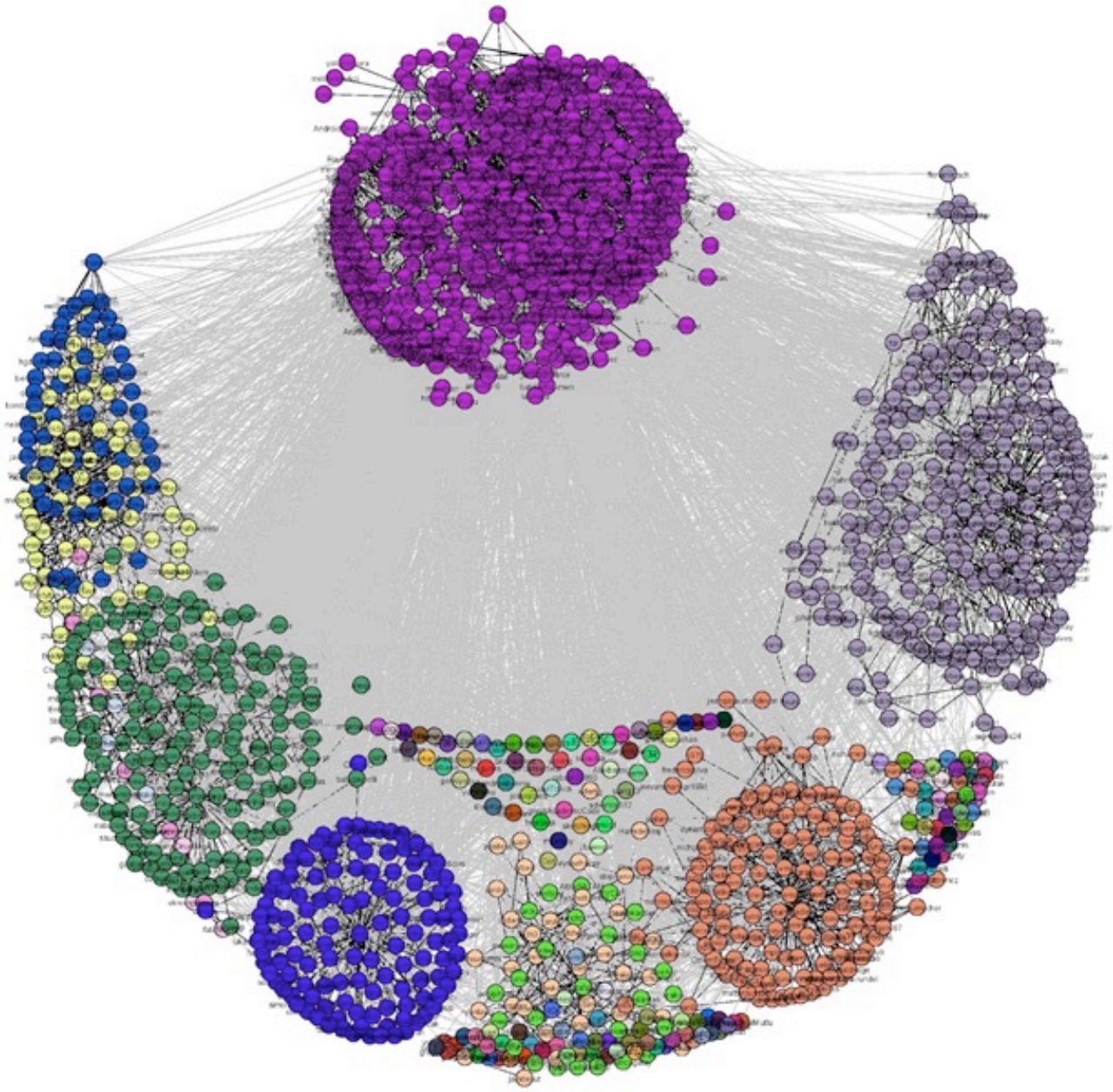

Figure 2: Analysis Result with 2 Step

In this figure, there are 139 communities, and only the users who have very close relationship are gathered in same community.

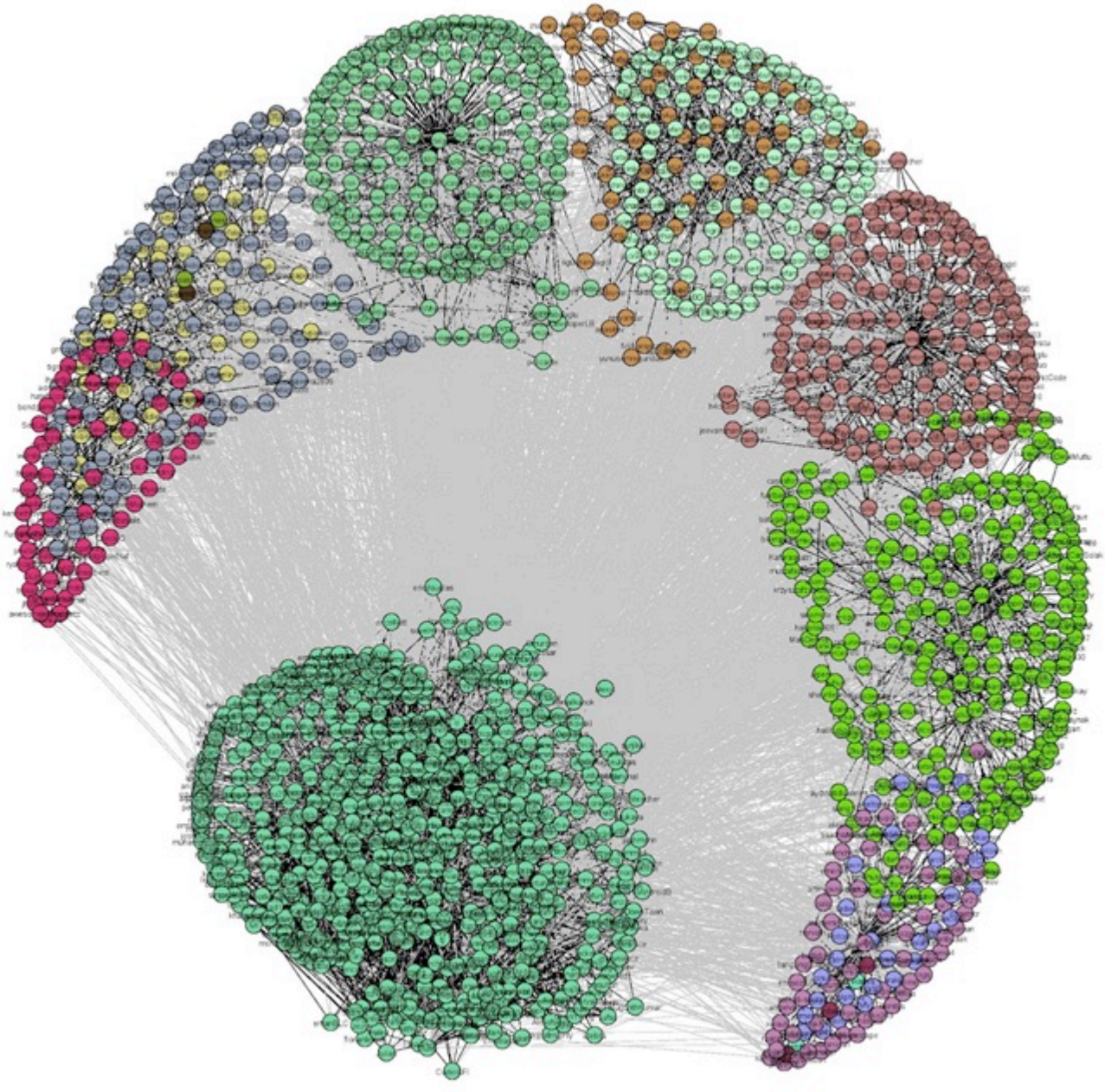

Figure 3: Analysis Result with 6 Step

In this figure, 1688 users are gathered in 14 communities, and the users who have very weak relationship are also put in same community.

When we run the code in 2 steps we see the communities which are connected very closely by the follower/following or company/organization relationship. If we increase the number of

steps, the effect of further relationships like the follower of followers becomes effective and gets included in the community according to their r values.

When we analyze the graphs with step 2 to 6, we intuitively decided on using the length of random walks as 3, because the most meaningful graph was obtained with 3 step. There is no automatic way of deciding on the random walk length, since the relations between users are not systematic.

With 3 step random walk analysis, we also analyzed the most common organizations and companies among the communities and most favorite programming languages that the community uses. Here is a section from 3 step analysis screenshot.

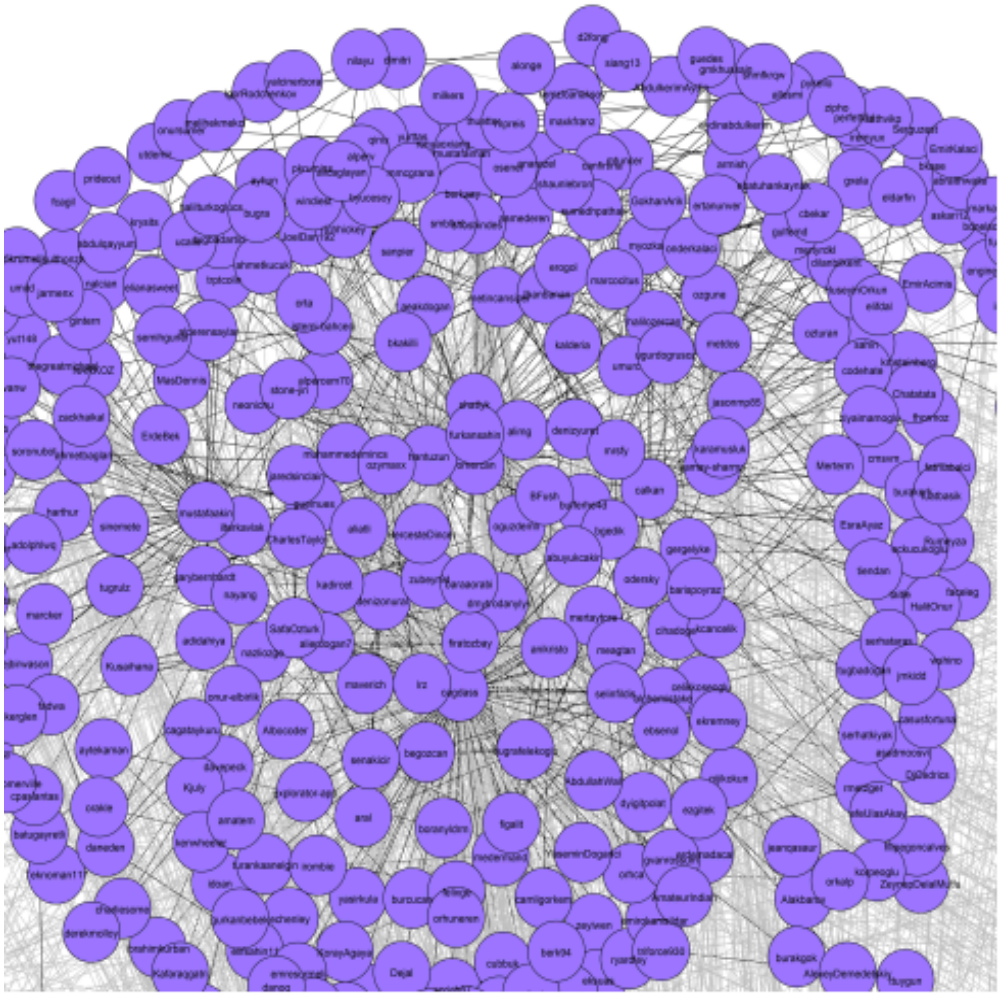

Figure 4: Bilkent and Citus Data Communities from 3 Step Analysis

With this image, we observed that this community includes the students from Bilkent and workers from Citus Data Company, which a Furkan (start point of dataset) also works for. Since many workers in Citus Data are former Bilkent University students, and there are also workers who are making their PhDs in Bilkent University, this community is very accurate. The number analysis for this community is following (these are ratios to 1):

Organizations/Companies

1. BilkentUniversity 0.0418
2. "<Organization[citusdata:]>" 0.028
3. CitusData 0.023

Languages

1. Java 0.290
2. Python 0.233
3. JavaScript 0.228

Unfortunately, most of the people using GitHub do not properly fill the Company/School section in GitHub. Therefore, most of the data about company and school is missing. Also, since we are considering both Company section in GitHub profiles of users and companies that users are belong to, there are 2 different organization names in most common list as "<Organization[citusdata:]>" and CitusData. Due to the missing information on Company/School data, the analysis says we have only 4% Bilkent University people in this community, whereas the Bilkent University people constitutes more than 95% of the community.

On the other hand, language analysis is very accurate since they are taken from the repositories that users had committed. Since every engineering student in Bilkent University

start his programming life with Java, the top of the list with %29 percent is Java. With 2nd and 3rd positions, Python and JavaScript are other most popular languages.

5. Conclusion

For Walktrap Community Detection algorithm, the length of random walks is a decisive factor on determining communities, since it basically determines a threshold for people's relationships such that how strong relationship should be taken into consideration.

For GitHub, we chose the best length for random walks as 3, intuitively. The closures of the communities is very accurate with 3 step random walks but there is still significant information about users companies and schools is missing. After determining the communities, the analysis we made on most common company/school information was accurate but their percentage was not due to the previously mentioned problem. However, the language analysis was very accurate with its percentage because this data is taken from repositories of users directly.

For future work, the dimension of the GitHub community analysis can be increased by taking the repositories committed, stars given, issues commented, languages used etc. into consideration. With these new dimensions, new analysis could also be done on these.

6. References

[1] *Computing communities in large networks using random walks*, Pascal Pons, Matthieu Latapy, 2005, France