Developer-Repository Recommender System: An Ensemble Way of Clustering, Community Detection and Graph Projection

7/26/17

**Nan Wang; Evangelos Katsamakas**

**Gabelli School of Business, Fordham University**

**WORKING PAPER**

**DO NOT CITE/DISTRIBUTE**

# Abstract

…….

**Keywords:** Recommender System, Software Development, Workload Assignment, Weighted Bipartite Graph, Graph Projection, Clustering, Community Detection

<http://snap.stanford.edu/class/cs224w-2013/projects2013/cs224w-038-final.pdf>

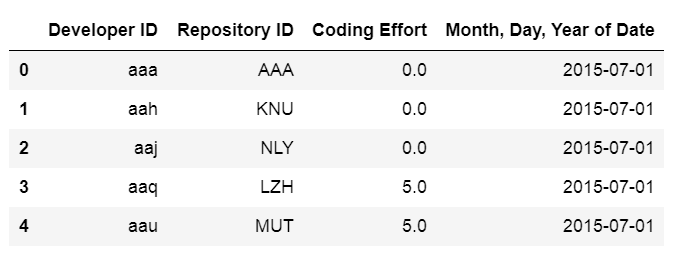
# Introduction

………

Recommender systems have been created for various data science applications in a variety of areas (Ricci et al 2011). For example, Facebook utilizes recommender systems to suggest friends to users; iTunes and YouTube utilize similar machine learning and recommendation algorithms to suggest songs, videos and movies. Given this general theme, we attempt to create a recommender system to suggest the assignment of developers to repositories.

# Dataset and Graph

The dataset used in this paper describes contribution of 2621 developers to 1705 repositories in a real-life IT company in a period of 92 days, from July 1, 2015 to September 30, 2015. This panel data has four variables, namely date, Developer ID, *Coding Effort Analytics™* (see appendix), Repository ID. There are 172,354 records in total, where “Developer ID” and “Repository ID” respectively identify each unique developer and repository. Data is gathered regarding developers’ past software development activity on a daily base via source code repositories like Subversion and Git, and task tracking systems such as Jira. This dataset is kindly provided by the company BlueOptima. Table 1 shows a subset from the dataset.



**Table 1.** **Sample of original dataset. *Coding Effort Analytics™* is a proprietary software measurement and is calculated through evaluating every change that software developers contribute to repositories in terms of a series of metrics, such as volume, complexity and interrelatedness of codes (Newswire 2013)**

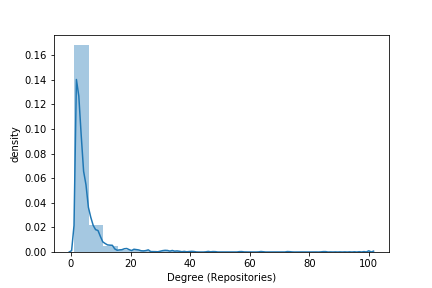
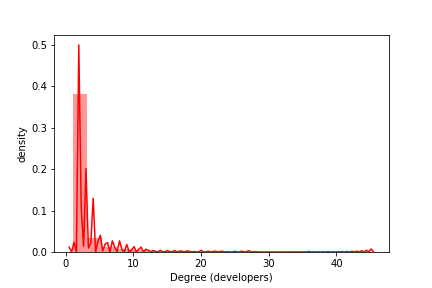
There are many records with zero *Coding Effort Analytics™* in the data. This is due to the fact that if a developer is involved in a repository, *Coding Effort Analytics™* is still recorded regardless of the absence of contribution. Therefore, to keep the information of developers’ involvement, we retain all data records.

The data is later represented by a bipartite weighted graph with developers being one type of nodes and repositories another. We aggregate each developer’s contribution to every repository across the period. The graph therefore stands for a static developer-to-repository network, with edges representing developers’ contribution to repositories and weighted by *Coding Effort Analytics™.*

The density of degree centrality (the number of ties that a node has) is shown in Figure X. As we can see, degree of developer nodes and repository nodes converges around one, which leads to our initial conclusion that the data is sparse, constituting a potential problem for collaborative filtering.

Regarding nodes degree, one more thing worth mentioning is that there exist 61 cases when “exclusive developers” work on “individual repository”, which means that during the period in concern, 61 developers have no interaction with others and 61 repositories share no resource with others.

Table X is the summary of the properties of the developer-repository graph.



**Figure 6.** Node degree distribution for developer graph

|  |  |
| --- | --- |
| Number of developer nodes | 2621 |
| Number of repository nodes | 1705 |
| Number of edges | 6414 |
| Average weight for edges | 60 |
| Average degree of developer nodes | 2.45 |
| Average degree of repository nodes | 3.76 |

# Literature Review

* 1. **Collaborative filtering**

Collaborative filtering (CF), one of the most popular ways to implement recommender system, is a method of making [predictions](https://en.wikipedia.org/wiki/Prediction) or recommendations (filtering) about the interests of a user by collecting preferences or [taste](https://en.wikipedia.org/wiki/Taste_(sociology)) information from [many users](https://en.wikipedia.org/wiki/Crowdsourcing) (collaborative). There are three major types of collaborative filtering algorithms in the recommender system literature, namely memory based approaches (uses user rating data to compute the similarity between users or items), model based approaches (uses [data mining](https://en.wikipedia.org/wiki/Data_mining), [machine learning](https://en.wikipedia.org/wiki/Machine_learning) algorithms to predict users' rating of unrated items), as well as hybrid approaches (combines the memory-based and the model-based algorithms). Each of those approaches has its own advantages and limitations (Ricci et al 2011; Breese et al 1998). And the choice of modeling methods is usually affected by the data available and the purpose of recommender systems.

In this paper, we employ memory based approaches. There are two main types of memory based approaches: user-based and item based, with the former calculating the similarity between all pairs of users according to a pre-defined metric and then predicting the rating of a user for an item by summarizing the ratings of the user’s “neighbors”, and the latter calculating the similarity between all pairs of items and then predicting the rating of a user for an item by summarizing the ratings of the item’s “neighbors”.

It is therefore reasonable to say that a well-defined metric to calculate similarity is of paramount significance to the effectiveness and accuracy of memory based approaches. There are abundant similarity functions available thanks to people’s long time research, among which Pearson correlation coefficient, Cosine similarity and Jaccardsimilarity are popular.

* Pearson correlation coefficient equals to the [covariance](https://en.wikipedia.org/wiki/Covariance) of the two variables divided by the product of [standard deviations](https://en.wikipedia.org/wiki/Standard_deviations). It tries to find vectors’ derivations from their average while recognizing linear adjustment in between.
* Cosine similarity, because of its computational efficiency is most commonly used in high-dimensional positive spaces. For example, in the field of text mining and information retrieval, Cosine similarity is widely applied to calculate the similarity between two documents. It is computed by the dot product of two vectors divided by their magnitude product. Results of Cosine similarity ranges from -1 (exactly opposite) to 1 (exactly same), with zero indicating orthogonality (decorrelation).
* Jaccard coefficient similarity (also known as Tanimoto coefficient similarity) measures similarity by calculating the intersection of two sets divided by their union sets. This metric intuitively cannot be greater than one.

However, not until a few years ago, recommender systems literature started to turn to graph for similarity implementation (……). Therefore, topics in graph analysis such as graph projection, graph properties and community detection haven’t been researched as exhaustively as household ways such as Matrix Factorization in literature (Gábor Takács et al (2008). Matrix factorization and neighbor based algorithms for the Netflix prize problem. In: Proceedings of the 2008 ACM Conference on Recommender Systems, Lausanne, Switzerland, October 23 - 25, 267-274.)

In the paper, our embracement of varied similarity functions is in a manner of applying varied weight allocation methods in the process of bipartite graph projection.

* 1. **Graph Projection**

Bipartite graph projection is applied when a one-mode graph is needed while a two-mode is provided. It is an extensively used way to compress and extract information from graph data. As most of real-life graph data is weighted, an appropriate way for edge weight re-allocation is thus important. An optimal weighting method should reflect the structure and properties of the graph, conform to the projection objectives, as well as minimize information loss.

Popular weighting methods include simply weighting (edges are weighted by the number of times the common association is repeated), Hyperbolic weighting (adds a scaling factor to simply weighting to weaken the connection between nodes with popular common matches), and weighting based on resource allocation (Zhou, T., et al., Bipartite network projection and personal recommendation, Physical Review E, 2007. 76(046115) which introduces weight allocation across neighbors.

In this paper, to better accord with the underlying logic of repository assignment recommender systems, we apply two customized weighting methods as ways of calculating similarity, and compare the two methods based on recommender system performance.

* 1. **Graph Properties**

Statistical metrics to describe graph properties include degree centrality, closeness centrality, betweenness centrality and eigenvector centrality. By studying those metrics, we can derive a broad range of insights regarding the rules of nodes (developers in our case). Bearing this in mind, we can utilize those metrics to group developers together and thus find “neighbors” for developers.

Closeness centrality, which uses the reciprocal of the average shortest distance to other nodes, is a measure of the degree to which an individual is near all other individuals in a network (Newman 2010). High closeness centrality therefore indicates close access to the resources in the network.

Betweenness centrality for each [node](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) is the number of these shortest paths that pass through the node (Freeman 1977). Nodes with higher betweenness scores therefore have more control of other nodes in terms of communication access.

Eigenvector Centrality is a measurement revealing neighbors’ quantity and quality (Newman 2001). A high eigenvector score is resulted from a large number of high-quality neighbors. High-quality developers can be described as those involving in one or several principal repositories, or participating in diversified repositories, by which they interact with many people in the network. Correspondingly, low-quality developers are those engaging in unitary or individual repositories, by which they collaborate with few others in the network.

* 1. **Community Detection and Clustering**

Communities in a network are groups of nodes internally connected or nodes sharing attributes (Girvan and Newman 2002). Detecting communities provides insights regarding the overall network structure, behavioral patterns of nodes and their relations (Fortunato 2010, Malliaros and Vazirgiannis 2013). For the developer-developer network, it supports findings about developers’ connections and collaboration preference. (Porter, Onnela and Mucha 2009)

Clustering based on graph properties, on the other hand, reveals insights of developers’ rule and functionality. It has been also proved to be able to improve the results of collaborative filtering especially when data is sparse.

In order to capture the underlying information of developers’ relationships (graph structure) and roles (graph properties), we employ both community detection and clustering as mechanisms in identifying “neighbors” for collaborative filtering.

* + 1. **Algorithms for Community Detection**

In order to recognize graph structure from different perspectives, we apply algorithms of mixed methodology on the developer-repository bipartite graph. Algorithms we apply include greedy optimization of modularity, label propagation, multi-level optimization of modularity, optimal community structure and walk trap.

Label propagation method works by labeling nodes with unique labels and then updating the labels by majority voting in the neighborhood of the vertex. Walk trap method, while based on the idea that short random walks tend to stay in the same community, tries to find densely connected subgraphs (communities) in a graph via random walks. Greedy optimization of modularity, multi-level optimization of modularity and optimal community structure are all aimed at optimizing modularity.

* + 1. **Algorithms for Clustering**

K-means is one of the most popular clustering algorithm. The methodology behind is firstly to initialize cluster centroids by randomly picking data points, and then to assign data points to the closed centroid based on some similarity metric, and afterwards to re-set each centroid to be the average of assigned data points. The algorithm repeats those three steps until convergence (no data points can be moved from one to another). We apply K-means to group developers together based on their properties.

In unsupervised learning problems, cluster numbers are usually implied by domain knowledge. In our case, because we attempt to group developers based on their functionality or roles, we may get hints about cluster number from job types in software development organizations. Here alternatively, we turn to Elbow method to determine cluster number.

Elbow method looks at the percentage of variance explained as a function of the number of clusters (XXX). Percentage of variance explained is the ratio of the between-group variance to the total variance. It is recommended to decide the number of clusters when variance has been largely explained by exiting clusters and is no longer obviously reduced by adding clusters.

# Methodology

* 1. **Train Test Split**
  2. **Baseline Model**
  3. **Similarity Metrics by Customized Weighting**

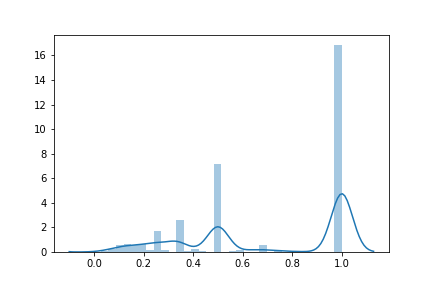
In our developer-repository case, one interpretation of similarity is the collaboration frequency. For example, if a large ratio of one developer’s repositories overlaps another’s, it is reasonable to infer that they use the same programming language or they share skillsets and thus they are similar. The similarity function that embodies this logic can be defined as follows. The similarity of ui to u1 is the ratio of the number of repositories they have in common over the number of repositories ui works on.

**Sim (u1, ui) =**

Number of Common Repositories between u1, ui

Number of Repositories of ui

Figure X shows the density of weight in the projected developer-developer graph:



Another interpretation of similarity is the “recommendation power” that each user gives to others, in other words, how powerful or influential others’ behavior could be to a certain user. In order to formalize this assumption, we consider a two-step random walk on the developer-repository bipartite graph: Developer u -> Repository b -> Developer v. For each step, we calculate the transition probability of occurring. Intuitively, for the first step, weight of edges (*Coding Effort Analytics™)* can be translated as how likely a developer is willing to give a repository his or her effort. This transition probability (TP) therefore can be captured as follows**:**

**TP (u, b) =**

**ru, b** (*Coding Effort* developer u to repository b)

**Ru** (Total *Coding Effort* of developer u)

Similarly, the transition probability from repository b to developer v is:

**TP (b, v) =**

**rv, b** (*Coding Effort* developer v to repository b)

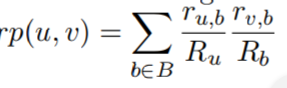
**Rb** (Total *Coding Effort* Repository b receives)

As mentioned earlier, we retain records of zero *Coding Effort*. Therefore, Rhas probability to be zero. To make transition probability mathematically feasible, we apply a normalization function to Rwhich changes the value of Rto one when R is zero.

Furthermore, the transition probability from Developer u to Repository b to Developer v is intuitively as follows:

**TP (u-b-v) = TP (u, b) \* TP (b, v)**

If we sum up transition probability across all repositories between Developer u and Developer v, we get the transition probability from Developer u to Developer v directly:

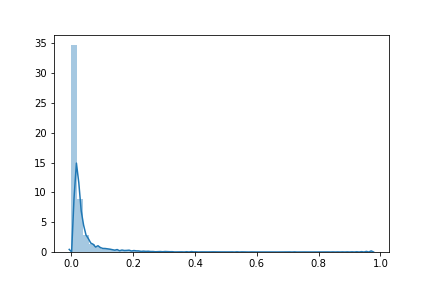


**TP (u, v)**

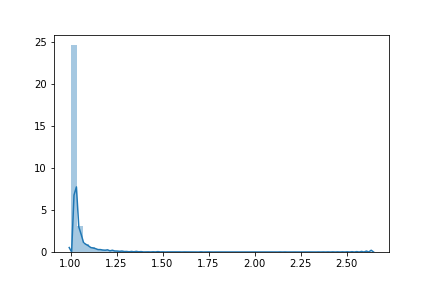
The resulted transition probability can therefore be seen as the “recommendation power” which Developer u gives to Developer v. As discussed before, “recommendation power” can be considered as similarity. Accordingly, we conclude with the formula below:

**Sim (u1, ui) = TP (u, v)**

Figure X shows the density of weight in the projected developer-developer graph:



As we can see, there are a large quantity of recommendation power being zero. It may cause problems when we assign weights to neighbors’ repositories. Therefore, we apply the exponential function to rescale the weights. Figure X shows the results:



During the implementation of projecting weighted bipartite graph, those two weight allocation methods will be applied. The resulting weights for edges in the one-mode developer-developer graph thus equal to the similarity between every two developers.

* 1. **Community Detection and Clustering**
  2. **Ensemble**

# Results

# Conclusion

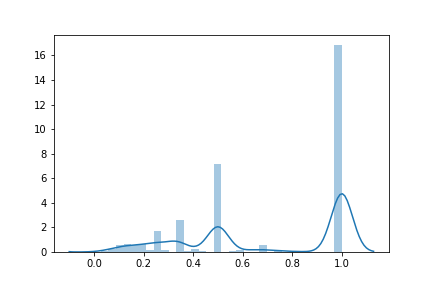
In our developer-repository case, one interpretation of similarity is the collaboration frequency. For example, if a large ratio of one developer’s repositories overlaps another’s, it is reasonable to infer that they use the same programming language or they share skillsets and thus they are similar. The similarity function that embodies this logic can be defined as follows. The similarity of ui to u1 is the ratio of the number of repositories they have in common over the number of repositories ui works on.

**Sim (u1, ui) =**

Number of Common Repositories between u1, ui

Number of Repositories of ui

Figure X shows the density of weight in the projected developer-developer graph:



Another interpretation of similarity is the “recommendation power” that each user gives to others: how powerful or influential others’ behavior could be to a certain user. In order to formalize this “recommendation power”, we consider a two-step random walk on the developer-repository bipartite graph: Developer u -> Repository b -> Developer v. For each step, we calculate the transition probability of occurring. Intuitively, for the first step, weight of edges (*Coding Effort Analytics™)* can be thought as how much a developer is willing to give a repository his or her effort. This transition probability (TP) can be captured as follows**:**

**TP (u, b) =**

**ru, b** (*Coding Effort* developer u to repository b)

**Ru** (Total *Coding Effort* of developer u)

Similarly, the transition probability from repository b to developer v is:

**TP (b, v) =**

**rv, b** (*Coding Effort* developer v to repository b)

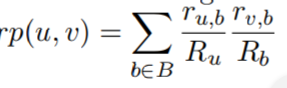
**Rb** (Total *Coding Effort* Repository b receives)

As mentioned earlier, we retain records of zero *Coding Effort*. Therefore, Rhas probability to be zero. To make transition probability mathematically feasible, we apply a normalization function to Rwhich changes the value of Rto one when R is zero.

Furthermore, we define the transition probability from Developer u to Repository b to Developer v as:

**TP (u-b-v) = TP (u, b) \* TP (b, v)**

In order to get the transition probability from Developer u to Developer v directly, we sum up transition probability across all repositories between Developer u and Developer v.

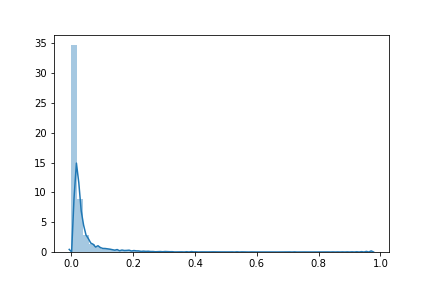


**TP (u, v)**

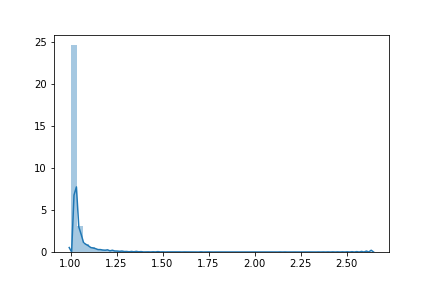
The resulted transition probability can therefore be seen as the recommendation power which Developer u gives to Developer v. As discussed above that recommendation power can be considered as similarity, we conclude with the formula below:

**Sim (u1, ui) = TP (u, v)**

Figure X shows the density of weight in the projected developer-developer graph:



As we can see, there are a large quantity of recommendation power being zero. It may cause problems when we try to find neighbors. Therefore, we apply the exponential function to rescale the weights. Figure X shows the results:



During the implementation of projecting weighted bipartite graph, those two weight allocation methods will be applied. The resulting weights for edges in the one-mode developer-developer graph thus equal to the similarity between every two developers.

Advantages of memory based approaches include content-independence, easy implementation and intuitive interpretation (Breese et al 1998). However, those approaches are sensitive and vulnerable to large sparsity data. For example, if there are limited number of instances where a pair of users rate the same item, memory based approaches will not be feasible because of the problem that many users cannot find neighbors.

One of the common ways to address sparsity is to apply hybrid approaches. For example, use principle component analysis to compress a high dimensional user-item matrix containing abundant number of missing values into a much smaller matrix in lower-dimensional space. With the ability of effectively overcoming the problems of sparsity and loss of information, this method is widely applied in commercial recommender systems (Das et al 2007). In the paper, we turn to clustering to address data sparsity problem.

# Methodology

In the situation of developers’ contributing to repositories, developers can be modeled as users in the recommender systems. Similarly, repositories can be modeled as items while *Coding Effort Analytics™* are ratings.

When the developer-repository graph is projected, edge weights in the resulted one-mode graph will be considered as the similarity measure between the two associated nodes, leading to the linkage between recommender systems and weighted bipartite graph projection. (Zhou, T., et al., Bipartite network projection and personal recommendation, Physical Review E, 2007. 76(046115). Weights (similarity measures) are derived from different weight allocation process. We will evaluate and compare those process based on recommender system performance.

# Community Detection and Clustering

* 1. **Community Detection**

Community detection, however, is an ill-defined problem. There is no universal definition of the methodology that one should follow. Consequently, there are no clear-cut guidelines on how to evaluate and compare the performance of different algorithms ([Fortunato](https://arxiv.org/find/physics/1/au:+Fortunato_S/0/1/0/all/0/1) and [Hric](https://arxiv.org/find/physics/1/au:+Hric_D/0/1/0/all/0/1) 2016).

In order to recognize graph structure from different perspectives, we apply algorithms of mixed methodology on the developer-repository bipartite graph. Algorithms we apply can fall into two categories: optimization based and dynamics based.

* + 1. **Optimization Based**

The idea of optimization based techniques is to optimize a quality function which measures the goodness of a clustering over all possibilities (Fortunato 2010). The most widely-used quality function is modularity by Newman and Girvan (Girvan and Newman 2006). It was designed to measure the strength of division of a network into communities. Networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules.

Based on the modularity measure, the multi-level modularity optimization algorithm is a hierarchical approach (Blondel et al 2008). It works according to the following steps: first, each node is assigned to a community independently. Afterwards, each node is moved to the community in a local, greedy way, where it achieves the highest contribution to modularity. When no node can be reassigned, each community is considered as a node on its own, and repeat the second step. The process stops either when there is only one node left or when the modularity cannot be increased any more.

After applied to the developer-repository bipartite graph, the Multi-Level model generates a modularity of 0.98, grouping 3,826 nodes into 642 communities. Communities contain 6 nodes on average. The largest community in the network, according to the model, contains 146 nodes, and there are 331 ungrouped nodes.

* + 1. **Dynamics Based**

Other than optimization methods, communities can also be identified by methods of running dynamical processes on the network (Fortunato 2010), among which Random walk dynamics is by far the most exploited.

Walktrap and Infomap are two algorithms both applying random walk, with the former based on an observation that short random walks have the tendency of staying in the same community, while the latter aimed at minimizing the expected description length of a random walk trajectory (Pons and Latapy 2006). Walktrap is a hierarchical model while Infomap is not.

Table 10 shows the comparison of the model performance between Walktrap and Infomap.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Modularity** | **Group Number** | **Average Group size** | **Largest Group size** | **Ungrouped Node Size** |
| Walktrap | 0.84 | 252 | 15 | 1057 | 0 |
| Infomap | 0.95 | 815 | 5 | 65 | 311 |

**Table 10:** Comparison of Walktrap and Infomap

## Summary

We save and merge the community assignment which three models (Multi-level, Walktrap and Infomap) generate, and will utilize the nodes’ community membership to identify their “neighbors” for building collaborative filtering models.

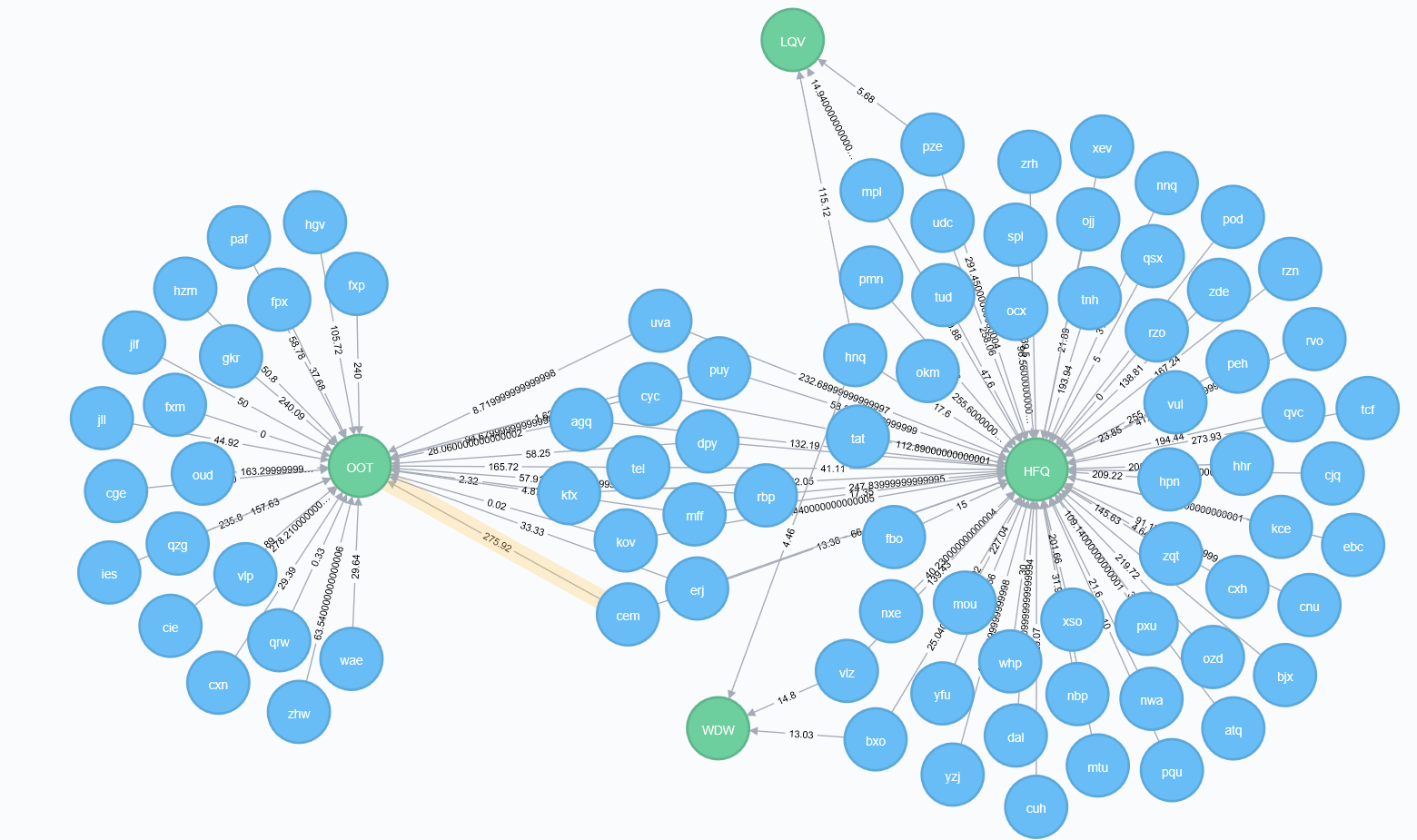
* 1. **Clustering**

Statistical metrics to describe graph properties include degree centrality, closeness centrality, betweenness centrality and eigenvector centrality. By studying those metrics, we can derive a broad range of insights regarding the rules of nodes (developers in our case). Bearing this in mind, we can utilize those metrics to group developers together and thus find “neighbors” for developers.

* + 1. **Graph Properties**

Closeness centrality, which uses the reciprocal of the average shortest distance to other nodes, is a measure of the degree to which an individual is near all other individuals in a network (Newman 2010). High closeness centrality therefore indicates close access to the resources in the network.

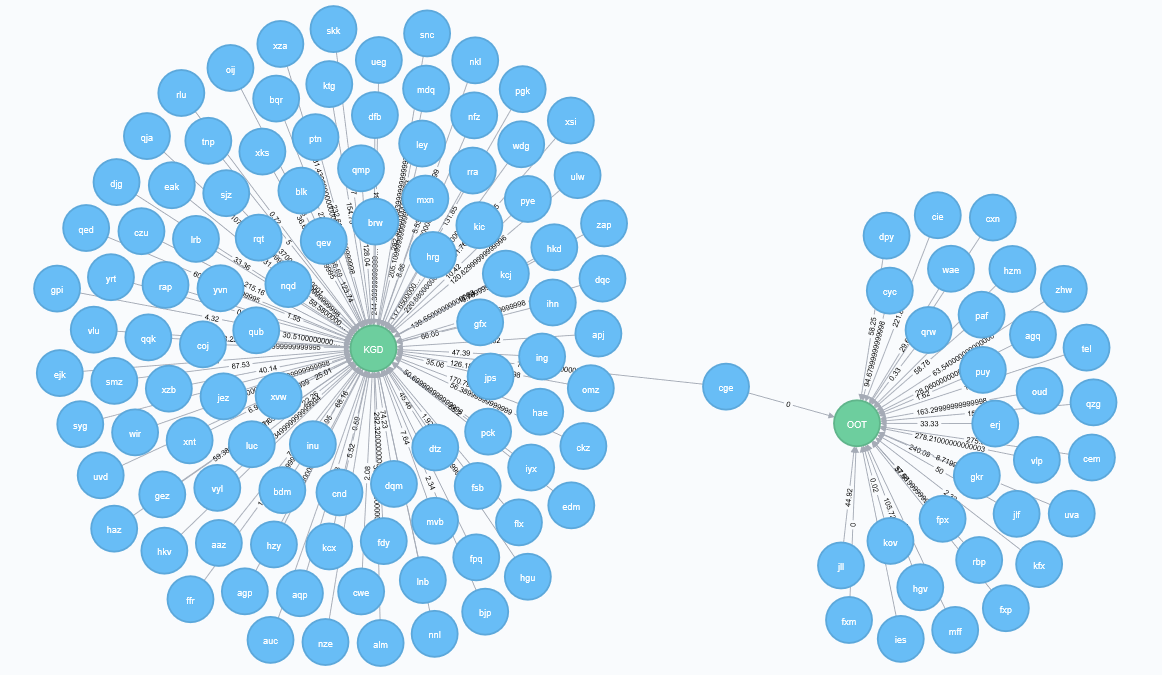
For example, in the developer-repository network, developer 'rbp', 'kfx', 'agq', 'tel', 'erj', 'kov', 'cyc', 'uva', 'dpy', 'mff', 'puy' and 'cem' are top twelve individuals having closest access. Furthermore, as it shown in Figure 9, all of them work on repository ‘OOT’ and ‘HFQ’. It can be therefore implied that repository ‘OOT’ and ‘HFQ’ are repositories gathering massive resource and by working in both, those twelve developers are able to get closer access to recourse.



**Figure 9.** Repository ‘OOT’ and ‘HFQ’ are shared by twelve developers who have the highest closeness centrality.

Betweenness centrality for each [node](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) is the number of these shortest paths that pass through the node (Freeman 1977). Nodes with higher betweenness scores therefore have more control of other nodes in terms of communication access.

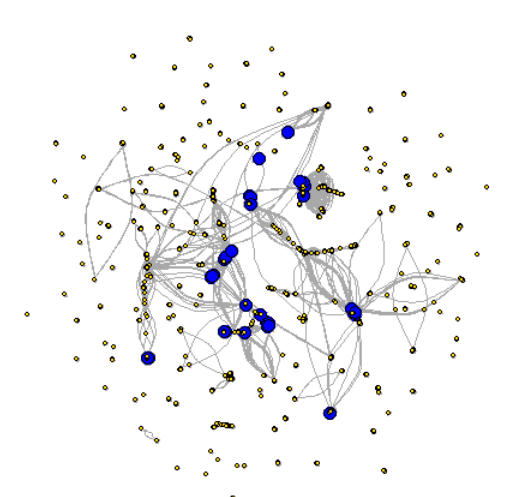
In the develop-developer network, developer ‘cge’ has the highest betweenness centrality. As shown in Figure 11, ‘cge’ is the only person working in both ‘OOT’ and ‘KGD’, two principal repositories in the company. Thus, it is reasonable to infer that developer ‘cge’ plays a significant role for the information sharing in between.



**Figure 11.** Developer ‘cge’ and corresponding connections

Eigenvector Centrality is a measurement revealing neighbors’ quantity and quality (Newman 2001). A high eigenvector score is resulted from a large number of high-quality neighbors. High-quality developers can be described as those involving in one or several principal repositories, or participating in diversified repositories, by which they interact with many people in the network. Correspondingly, low-quality developers are those engaging in unitary or individual repositories, by which they collaborate with few others in the network.

As it shown in Figure 12, nodes colored in blue are those with highest Eigenvector. We find that they are widely positioned among different subcomponents in the graph and are connected with different sets of nodes.



**Figure 12.** Visualization of developers with high eigenvector scores

* + 1. **K-Means**

We apply K-means to group developers together based on their properties. K-means is one of the most popular clustering algorithm. The methodology behind is firstly to initialize cluster centroids by randomly picking data points, and then to assign data points to the closed centroid based on some similarity metric, and afterwards to re-set each centroid to be the average of assigned data points. The algorithm repeats those three steps until convergence (no data points can be moved from one to another).

In unsupervised learning problems, cluster numbers are usually implied by domain knowledge. In our case, because we attempt to group developers based on their functionality or roles, we may get hints of cluster number from job types in software development organizations. Here alternatively, we turn to Elbow method to determine cluster number.

Elbow method looks at the percentage of variance explained as a function of the number of clusters (XXX). Percentage of variance explained is the ratio of the between-group variance to the total variance. It is recommended to decide the number of clusters when variance has been largely explained by exiting clusters and is no longer obviously reduced by adding clusters.

# Hybrid Collaborative Filtering Model

With similarity function, community detection methods and clustering algorithm clarified, we are ready to build a neighborhood-based collaborative filtering model which makes use of graph properties and structure, as well as graph projection.

We use the ratio of 80% to 20% to split data into training set and test set, with the former used to structure graph, calculate similarity, generate communities and form groups, and the latter used to validate the model.

Another prominent merit that graph provides is a more intuitive and sensible similarity metric. A

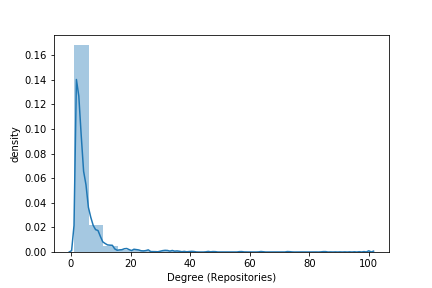
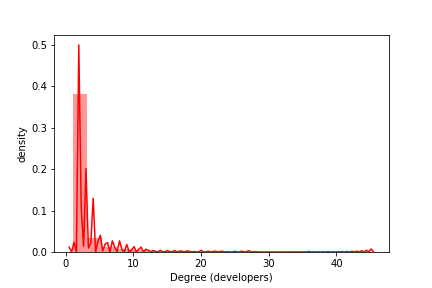
Graph projection is to transform the graph from two-mode into one-mode (Zhou et al 2007). The process works by selecting one set of nodes, and linking two nodes if they are connected to the same node of the other set. However, since one-mode projection is always less informative than the bipartite representation and the redistribution of weights would strongly affect the community structure, a proper weighting method is required to better retain the original information (Fan et al 2007).

In this paper, Newman’s weighted projection is applied, which adopts the weighting scheme presented by Newman. The Newman’s collaboration model (Newman 2001) is as follows. U and v are nodes of the same type (for example, developer), and w is a node of the other type (for example, repository). The value kw is the degree of node win the bipartite network.

Before building recommender systems, we apply several statistical measurements, namely degree, XXXXX, to the graph to better understand the properties of this network,

Degree is the number of ties that a node has. In the developer-repository situation, nodes with degree of one can be interpreted as “exclusive developers” who work on only one repository, or individual repository which contain only one developer. It is found that during the entire period in concern, near half of developers (1440 out of 2621) is responsible for only one repository and about 43% (725 out of 1705) are individual repositories. Figure X shows that the degree distribution for developer and repository nodes. One more thing worth mentioning is that there exist 61 cases when “exclusive developers” work on “individual repository”, which means that 61 developers in the organization have no interaction with others.

In addition, the maximum degree is 45 for developer nodes and 100 for repository nodes, meaning that the most multi-tasking developer (Developer “ouj”) is involved in 39 repositories and the largest repository (Repository “KGD”) is contributed by 100 developers. Also, the average degree is 2.45 for Developer nodes and 3.76 for repository, meaning that, overall, each Developer works on two repositories and each repository contains four Developers.



The distribution appears to follow a power-law distribution (Clauset et al 2009). In the network theory, a network is named scale-free if its degree distribution follows a mathematical function called a power law (Albert and Barabasi 2002). Unlike a random graph (i.e., new nodes attach to existing nodes with uniform probabilities), a scale-free graph displays preferential attachment of new nodes (i.e., some nodes have higher probability of attachment than others).

In our developer-repository case, it is intuitive that the network is not randomly formed. It is also in accordance with expectation that a developer, if having diversified skillsets or senior experience, is more likely to engage in heterogeneous tasks or collaborate with mixed-background colleagues.

To validate our assumption that nodes in the graph have preferential attachment, we apply a statistical analysis called Kolmogorov-Smirnov test (Hazewinkel and Michiel 2001). In many real-world cases, the power-law behavior kicks in only above a threshold value (Xmin) for the input vector. We apply maximum likelihood principle to find an optimized Xmin value for which the p-value of a Kolmogorov-Smirnov test is the largest (Clauset et al 2009).

Test results for developer nodes are as follows. The combination of small test statistic (KS.stat) and bigger p-value (KS.p) indicates a good fit of power-law distribution. As it shown in Table X, KS.stat is 0. 0319 and KS.p is 0. 498. It therefore proves that the developer degree distribution has the property of scale-free (Marsaglia et al 2003). The same methodology applies to

|  |  |  |
| --- | --- | --- |
| **Measurement** | **Definition** | **Value** |
| Alpha | The exponent of the fitted power-law distribution. | 4.48 |
| xmin | The minimum value from which the power-law distribution was fitted. | 70 |
| logLik | The log-likelihood of the fitted parameters. | -2899.41 |
| KS.stat | The test statistic of a Kolmogorov-Smirnov test that compares the fitted distribution with the input vector. **Smaller scores denote better fit.** | 0.0319 |
| KS.p | The p-value of the Kolmogorov-Smirnov test. Small p-values (less than 0.05) indicate that the test rejected the hypothesis that the original data could have been drawn from the fitted power-law distribution. | 0.498 |

**Table X** Statistical tests for power-laws

**Figure 1.** Implications of roles and productivity based on node degree and edge weight

* 1. **A Graph Recommender System**

Recommender systems have been created for various data science applications in a variety of areas (Ricci et al 2011). For example, Facebook utilizes recommender systems to suggest friends to users; iTunes and YouTube utilize similar machine learning and recommendation algorithms to suggest songs, videos and movies. Given this general theme, we attempt to create a recommender system to suggest the assignment of developers to repositories.

Collaborative filtering (CF), one of the most popular ways to implement recommender system, is a method of making [predictions](https://en.wikipedia.org/wiki/Prediction) or recommendations (filtering) about the interests of a user by collecting preferences or [taste](https://en.wikipedia.org/wiki/Taste_(sociology)) information from [many users](https://en.wikipedia.org/wiki/Crowdsourcing) (collaborative). There are three major types of collaborative filtering algorithms in the recommender system literature, namely memory based approaches (uses user rating data to compute the similarity between users or items), model based approaches (uses [data mining](https://en.wikipedia.org/wiki/Data_mining), [machine learning](https://en.wikipedia.org/wiki/Machine_learning) algorithms to predict users' rating of unrated items), as well as hybrid approaches (combines the memory-based and the model-based algorithms). Each of those approaches has its own advantages and limitations (Ricci et al 2011; Breese et al 1998).

In the situation of developers’ contributing to repositories, developers can be modeled as users in the recommender systems. Similarly, repositories can be modeled as items and *Coding Effort Analytics™* are ratings. The choice of modeling methods is usually affected by the data available and the purpose of recommender systems. Considering that our recommender system is to make recommendations rather than predictions, and content data (data describing users or items) is unavailable, we choose to apply memory based approaches.

There are two main types of memory based approaches: user-based and item based, with the former calculating the similarity between all pairs of users according to a pre-defined metric and then predicting the rating of a user for an item by summarizing the ratings of the user’s “neighbors”, and the latter calculating the similarity between all pairs of items and then predicting the rating of a user for an item by summarizing the ratings of the item’s “neighbors”.

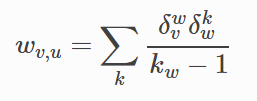
Advantages of memory based approaches include content-independence, easy implementation and intuitive interpretation (Breese et al 1998). However, those approaches are sensitive and vulnerable to large sparsity data. Given the fact that in our case, there are 2621 users (developers) and 1705 items (repositories), and over half of users rate only one item, the data is expected to be sparse.

One of the common ways to address sparsity is to apply hybrid approaches. For example, use principle component analysis to compress a high dimensional user-item matrix containing abundant number of missing values into a much smaller matrix in lower-dimensional space. With the ability of effectively overcoming the problems of sparsity and loss of information, this method is widely applied in commercial recommender systems (Das et al 2007). However, it is complex and expensive to implement. Therefore, instead of hybrid approaches, we turn to graph to address the problem of data sparsity.

Another prominent merit that graph provides is a more intuitive and sensible similarity metric. A well-defined metric to calculate similarity is of paramount significance to the effectiveness and accuracy of memory based approaches. Traditionally, popular similarity functions include the Pearson correlation coefficient, Cosine similarity and Jaccardsimilarity**.** Here we employ the method of clustered weighted bipartite projection to achieve user similarity (XXX).

Graph projection is to transform the graph from two-mode into one-mode (Zhou et al 2007). The process works by selecting one set of nodes, and linking two nodes if they are connected to the same node of the other set. However, since one-mode projection is always less informative than the bipartite representation and the redistribution of weights would strongly affect the community structure, a proper weighting method is required to better retain the original information (Fan et al 2007).

In this paper, Newman’s weighted projection is applied, which adopts the weighting scheme presented by Newman. The Newman’s collaboration model (Newman 2001) is as follows. U and v are nodes of the same type (for example, developer), and w is a node of the other type (for example, repository). The value kw is the degree of node win the bipartite network.



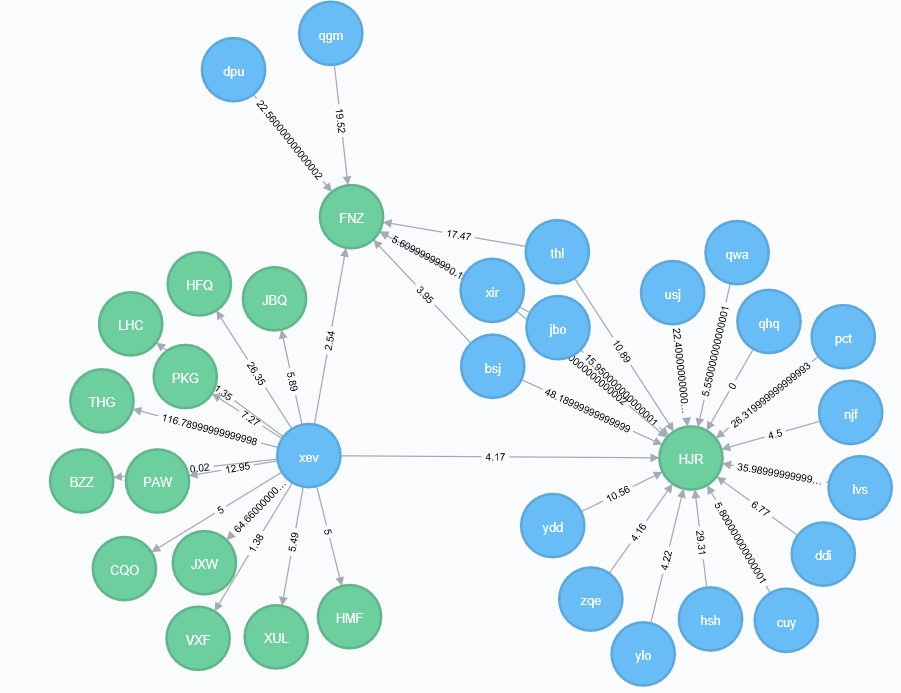


: 1 if node w and v are connected on the bipartite graph else zero.

: Edge degree between node v and u on the projected graph.



As the first attempt, we define that two users are neighbors if they share at least one item. In other words, two developers are neighbors if they work on at least one repository. Instead of computing similarity of every two users, we only do calculations between neighbors.



**Figure 2.** Developer‘xev’ is neighbored with both ‘dpu’ and ‘bsj’, by sharing the one repository with the former and two with the latter.

To define the similarity of ui to u1, the following function is applied:

**Sim(u1, ui) =**

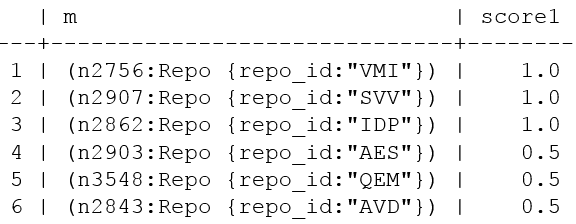
Number of Common Repositories between u1, ui

Number of Repositories of ui

The similarity of ui to u1 is the ratio of the number of repositories they have in common over the number of repositories ui works on. We further set a threshold of 0.5 for similarity, which means that developer ui is considered to be similar to developer u1 only if u1 work on at lea st half of ui’s repositories.

We can afterwards make recommendations by answering either the question of what repositories popularly shared by similar neighbors, or what repositories actively contributed by similar neighbors.

Table 2 is a list of repositories recommended by the system to the developer ‘xev’:



**Table 2.** Recommender system with similarity threshold being 0.5

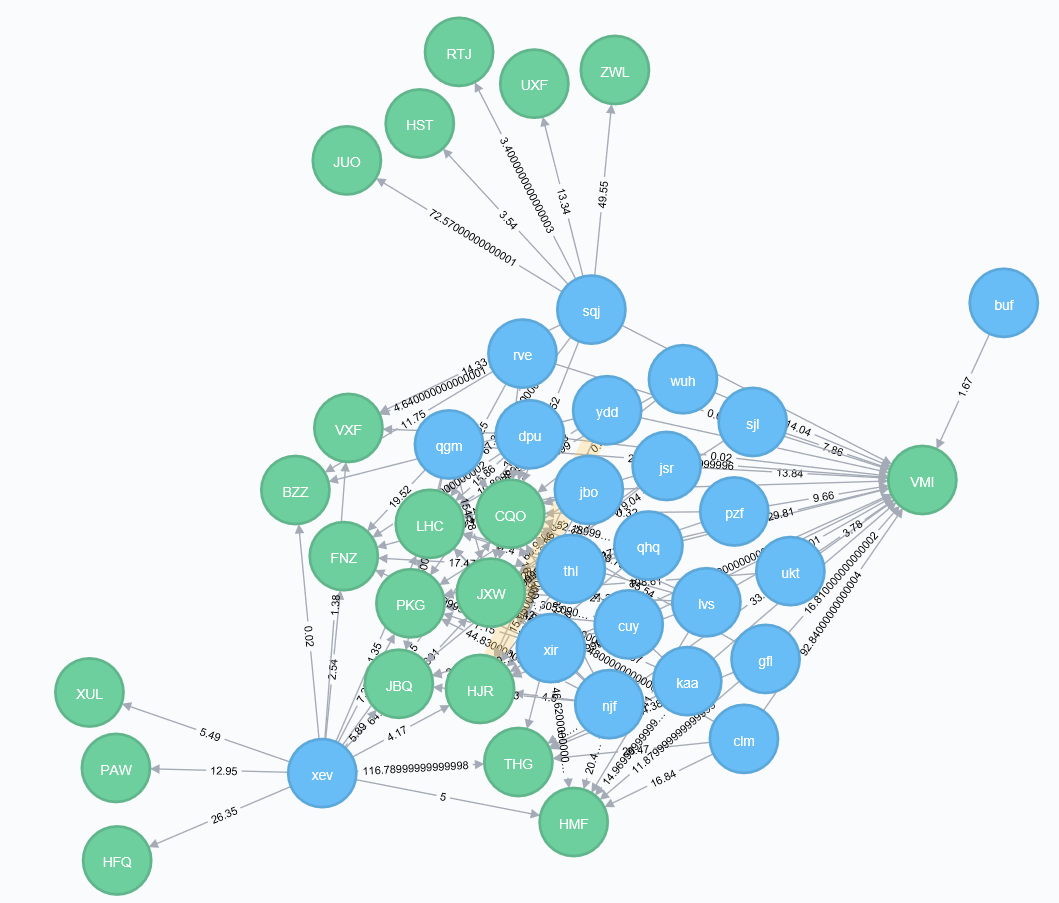
The scoring function calculates the ratio of similar neighbors who contribute to the recommended repository.

**Score 1** **=**

Number of contributing similar neighbors

Number of similar neighbors

Assuming a manager is about to assign the developer ‘xev’ a familiar repository, ‘VMI’, ‘SVV’ and ‘IDP’ would be worthy of consideration, given the fact that all xev’s nearest neighbors have worked on them, as it shown in Figure 3.



‘xev’ individual repositories

‘xev’ similar neighbors

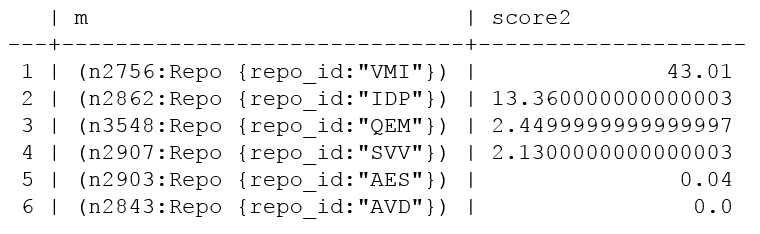
**Figure 3.** All xev’s nearest neighbors work on repository ‘VMI’.

Table 3 is another list of repositories recommended by the system to the developer ‘xev’. Different from the previous system, the scoring function has changed into the sum of similar neighbors’ contribution (Score 2). Therefore, the system recommends new repositories to the developer ‘xev’ in an order from ‘contributed the hardest by similar neighbors’ to ‘contributed the least by similar neighbors’.

**Score 2 =**

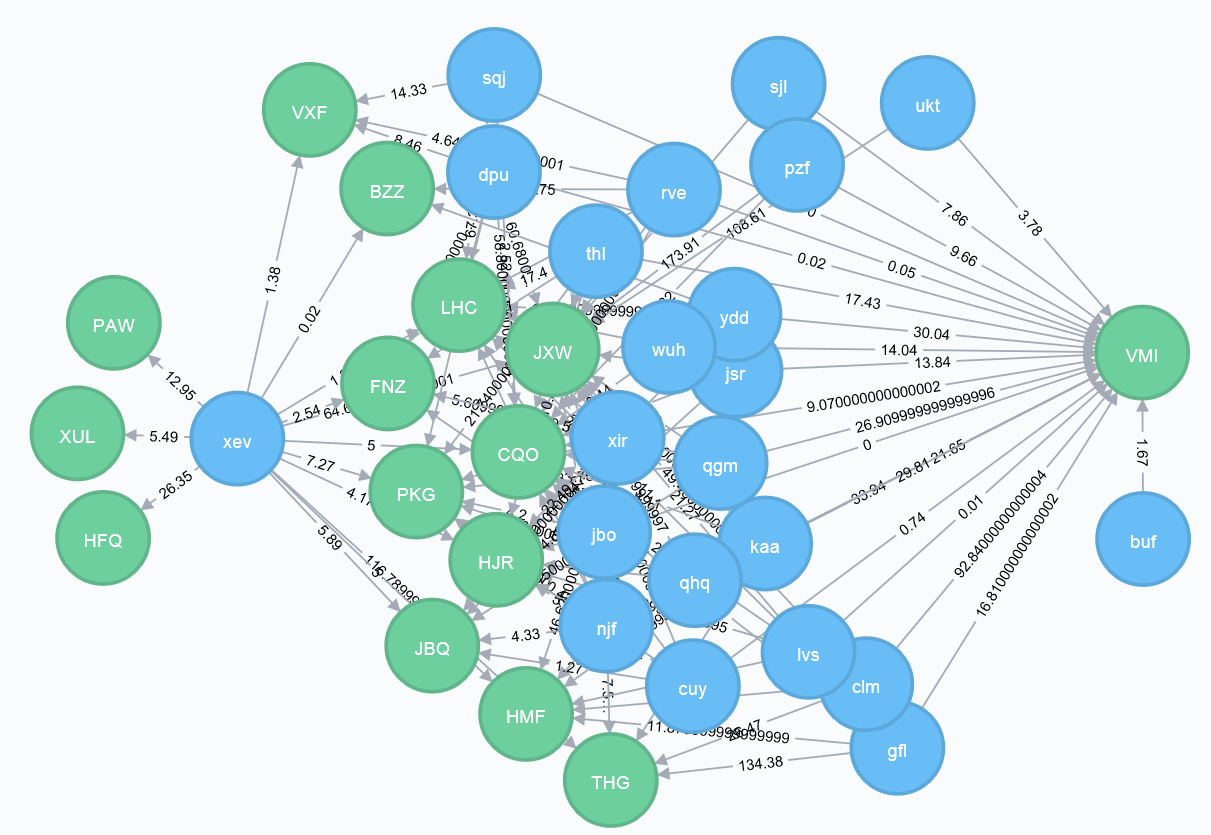
Sum (similar neighbors’ contribution)

If a manager wants to ensure developers’ contribution when assigning new repositories, he probably prefers this recommendation scheme based on the assumption that a developer will contribute to a repository in a comparable way as his similar developers. If nearest neighbors unanimously work hard on a repository, the developer is likely to devote too.



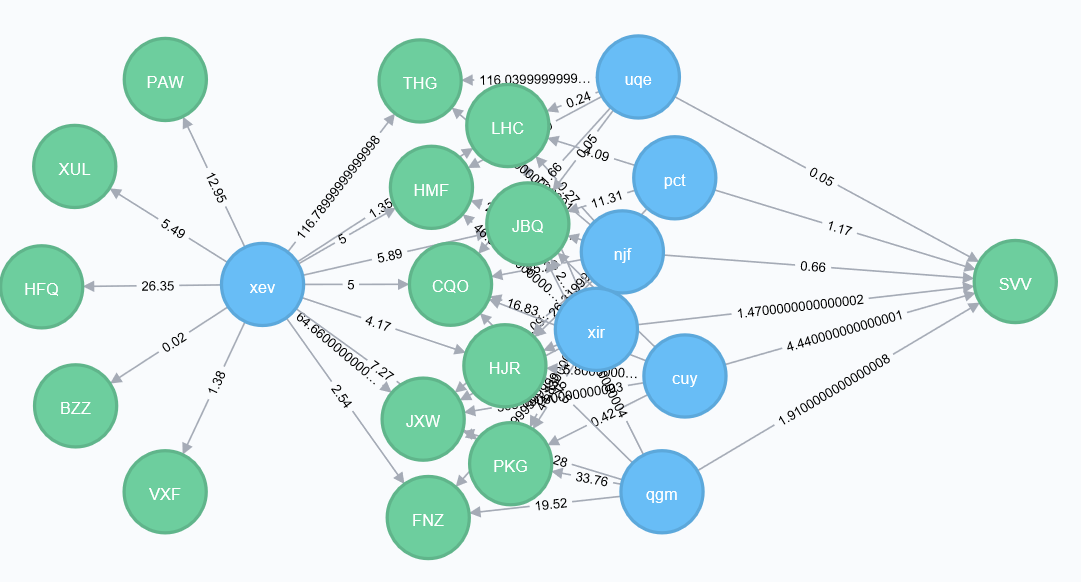
**Table 3.** AnotherRecommender system with similarity threshold being 0.5

More insights can be retrieved if the results of two recommender systems are combined. For example, although both repository ‘VMI’ and ‘SVV’ are participated by all the nearest neighbors of developer ‘xev’, the former receives massive contribution (as it shown in Figure 4) while the latter has little (as it shown in Figure 5). Also, although half of nearest neighbors participate in the repository ‘AVD’, none of them contribute anything in the period of analysis.



Large contribution

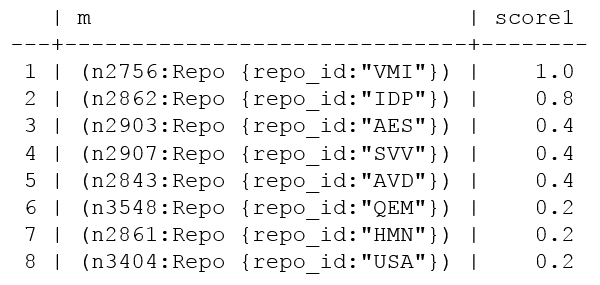
**Figure 4.** Most of xev’s nearest neighbors work hard on repository ‘VMI’, especially developer ‘clm’ whose *Coding Effort Analytics™* is about 93.



Small contribution

**Figure 5.** None of xev’s nearest neighbors work hard on repository ‘VMI’.

Furthermore, we find that a different similarity threshold would affect the recommendation results. For example, if we change the similarity threshold to 0.4, which means that neighbors sharing over 40% of repositories are seen as similar neighbors (or nearest neighbors), the results would be as follows (Table 4). ‘VMI’ is the only repository shared by all the developer’s nearest neighbors.



**Table 4.** AnotherRecommender system with similarity threshold being 0.4

An optimized graph-based recommendation system is beyond scope of this paper. Here we just implement a general-purpose way to demonstrate the simplicity and scalability of a graph-based approach. It doesn’t necessarily require user features or large volume of data for training. Moreover, it helps solve the data sparsity and computation problem of memory-based approaches.

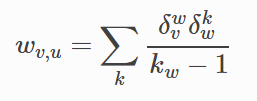
* 1. **Summary of lessons from developers-repositories graph**

With the weighted bipartite developer-repository graph, we gain an overview of the structure and workforce allocation of the company’s software development business, as well as build an efficient graph-based recommender system which facilitates the repository assignments of different purposes.

# Modeling Collaboration: Developers-Developers Graph

When relations among only one class of objects in a bipartite graph are needed, projection is usually applied to transform the graph from two-mode into one-mode (Zhou et al 2007). The process works by selecting one set of nodes, and linking two nodes if they are connected to the same node of the other set. However, since one-mode projection is always less informative than the bipartite representation and the redistribution of weights would strongly affect the community structure, a proper weighting method is required to better retain the original information (Fan et al 2007).

In this paper, Newman’s weighted projection is applied, which adopts the weighting scheme presented by Newman. The Newman’s collaboration model (Newman 2001) is as follows. U and v are nodes of the same type (for example, developer), and w is a node of the other type (for example, repository). The value kw is the degree of node win the bipartite network.





: 1 if node w and v are connected on the bipartite graph else zero.

: Edge degree between node v and u on the projected graph.

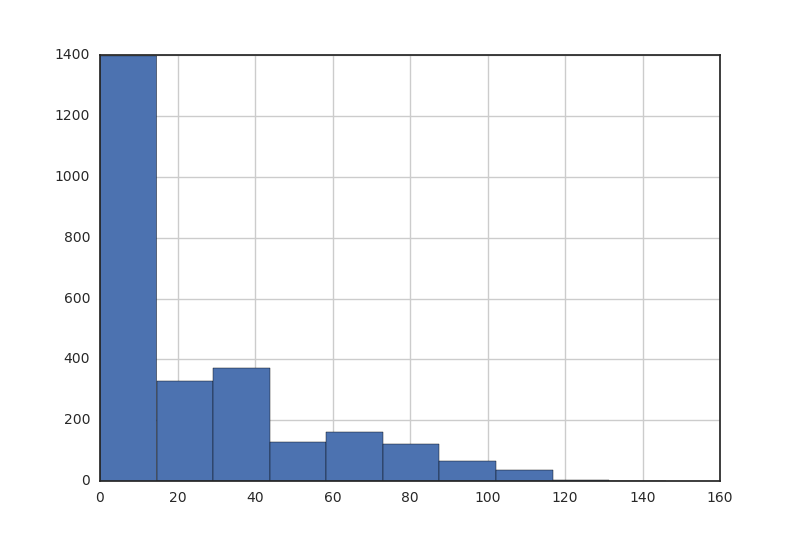


Accordingly, the developer-to-repository graph is repositoryed into two separate one-mode graphs. One is a developer-to-developer graph with 2459 nodes and 61378 edges where each developer is a node and an edge exists between two nodes if both developers are on the same repository, and another is a repository-to-repository graph with 1496 nodes and 16764 edges where repositories are nodes and edges represent shared developers.

* 1. **Graph Properties**

With the one-mode developer-developer graph, we can then elaborate on developers and their connections through studying properties of the developer-developer graph.

* + 1. **Degree Centrality**



**Figure 6.** Node degree distribution for developer graph

To validate our assumption that nodes in the graph have preferential attachment, we apply a statistical analysis called Kolmogorov-Smirnov test (Hazewinkel and Michiel 2001). In many real-world cases, the power-law behavior kicks in only above a threshold value (Xmin) for the input vector. We apply maximum likelihood principle to find an optimized Xmin value for which the p-value of a Kolmogorov-Smirnov test is the largest (Clauset et al 2009).

Test results from igraph, open source network analysis tools (Csárdi G. et al 2017), are as follows. The combination of small test statistic (KS.stat) and bigger p-value (KS.p) indicates a good fit of power-law distribution. As it shown in Table 5, KS.stat is 0.0346 and KS.p is 0.229. It therefore proves that the developer-to-developer network has the property of scale-free (Marsaglia et al 2003).

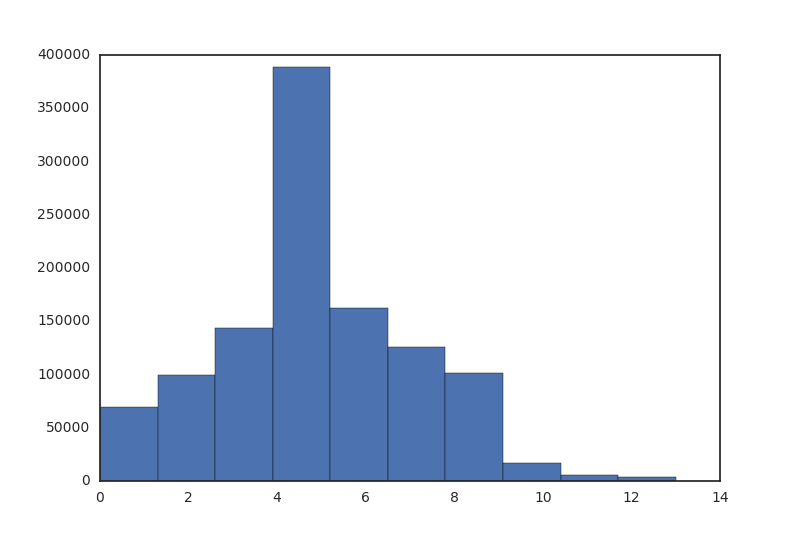
|  |  |  |
| --- | --- | --- |
| **Measurement** | **Definition** | **Value** |
| Alpha | The exponent of the fitted power-law distribution. | 4.48 |
| xmin | The minimum value from which the power-law distribution was fitted. | 70 |
| logLik | The log-likelihood of the fitted parameters. | -2899.41 |
| KS.stat | The test statistic of a Kolmogorov-Smirnov test that compares the fitted distribution with the input vector. **Smaller scores denote better fit.** | 0.0319 |
| KS.p | The p-value of the Kolmogorov-Smirnov test. Small p-values (less than 0.05) indicate that the test rejected the hypothesis that the original data could have been drawn from the fitted power-law distribution. | 0.498 |

**Table 5.** Statistical tests for power-laws

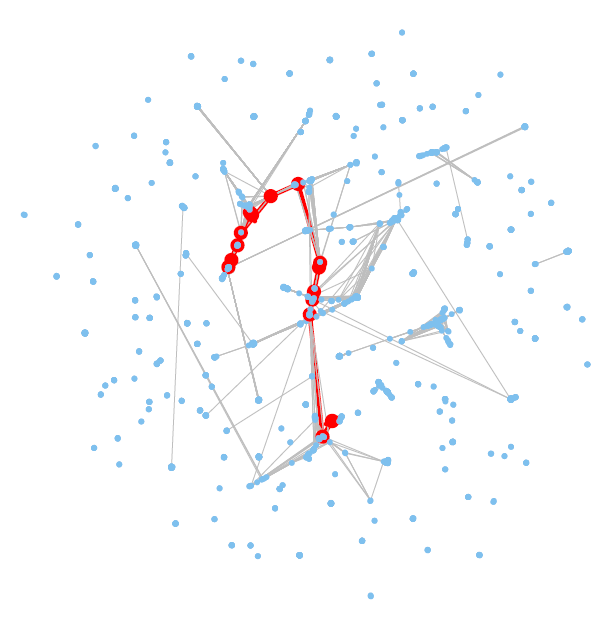
* + 1. **Shortest Path Length and Closeness Centrality**

Shortest path length is the number of steps along the shortest paths for all possible pairs of network [nodes](https://en.wikipedia.org/wiki/Node_(networking)) (Zhan et al 1998). Figure 7 shows the distribution of shortest path length in the graph.

Averaged at 4.86, length of the shortest path ranges from one to thirteen, meaning that a developer might expect to reach a randomly-selected developer in a typical distance of five, and the distance could be as small as one and as large as thirteen (as it shown in Figure 8) in the company of 2,459 developers.



**Figure 7.** Distribution of Shortest Paths in the developers’ network

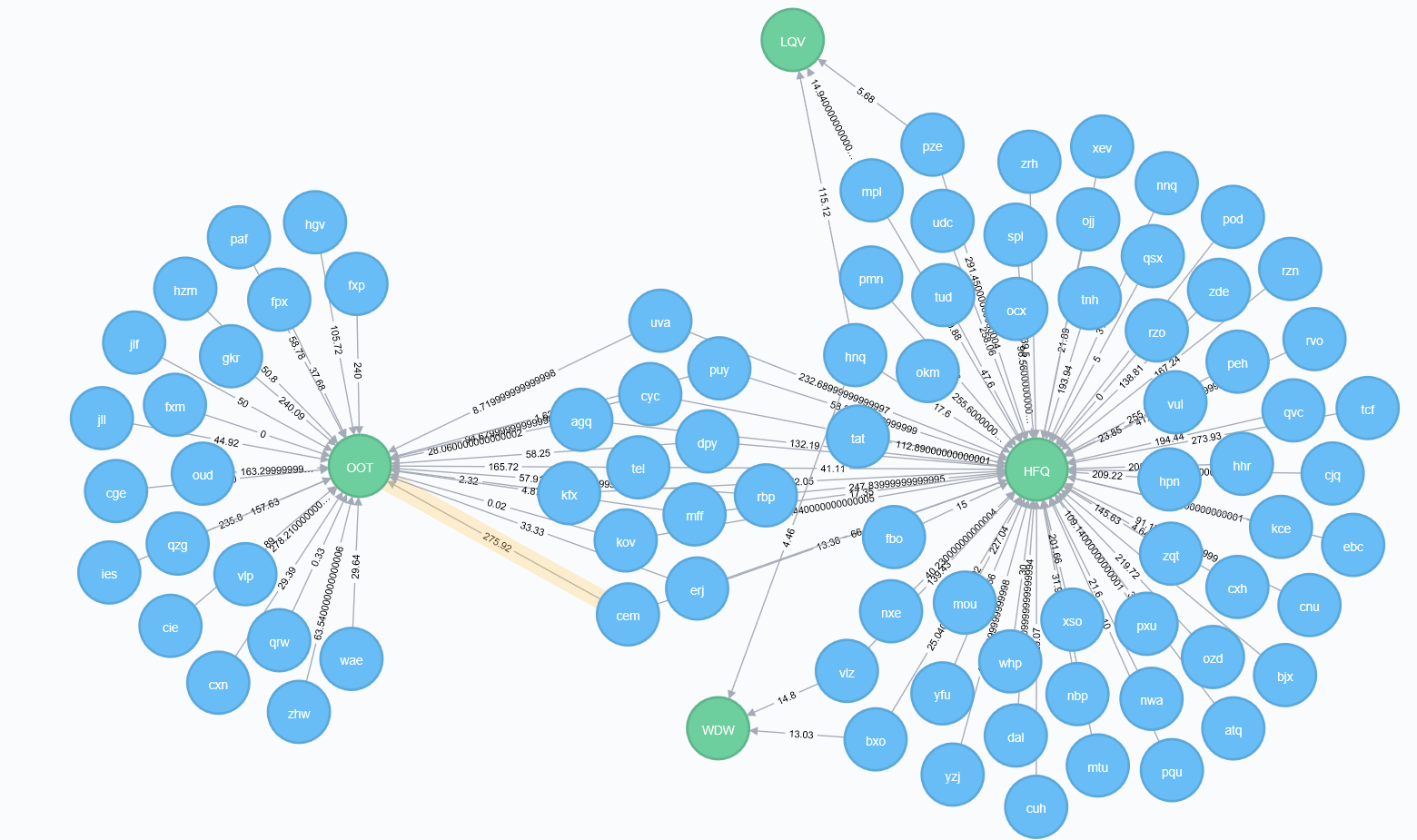


**Diameter: 13**

**Figure 8.** Visualization of developer network diameter

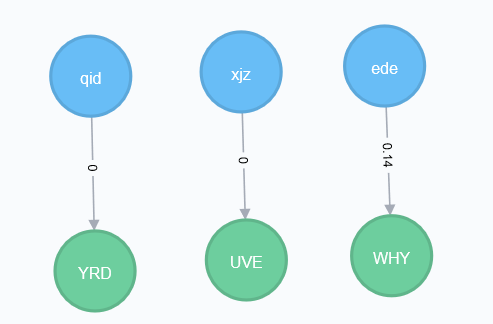
Closeness centrality, which uses the reciprocal of the average shortest distance to other nodes, is a measure of the degree to which an individual is near all other individuals in a network (Newman 2010). High closeness centrality therefore indicates close access to the resources in the network.

In the developer-repository network, developer 'rbp', 'kfx', 'agq', 'tel', 'erj', 'kov', 'cyc', 'uva', 'dpy', 'mff', 'puy' and 'cem' are top twelve individuals having closest access. Furthermore, as it shown in Figure 9, all of them work on repository ‘OOT’ and ‘HFQ’. It can be therefore implied that repository ‘OOT’ and ‘HFQ’ are repositories gathering massive resource and by working in both, those twelve developers are able to get closer access to recourse.



**Figure 9.** Repository ‘OOT’ and ‘HFQ’ are shared by twelve developers who have the highest closeness centrality.

In terms of nodes with smallest closeness centrality, for example, developer ‘qid’, ‘xjz’, ‘ede’, as it shown in Figure 10, they all exclusively engage in their own individual repository.

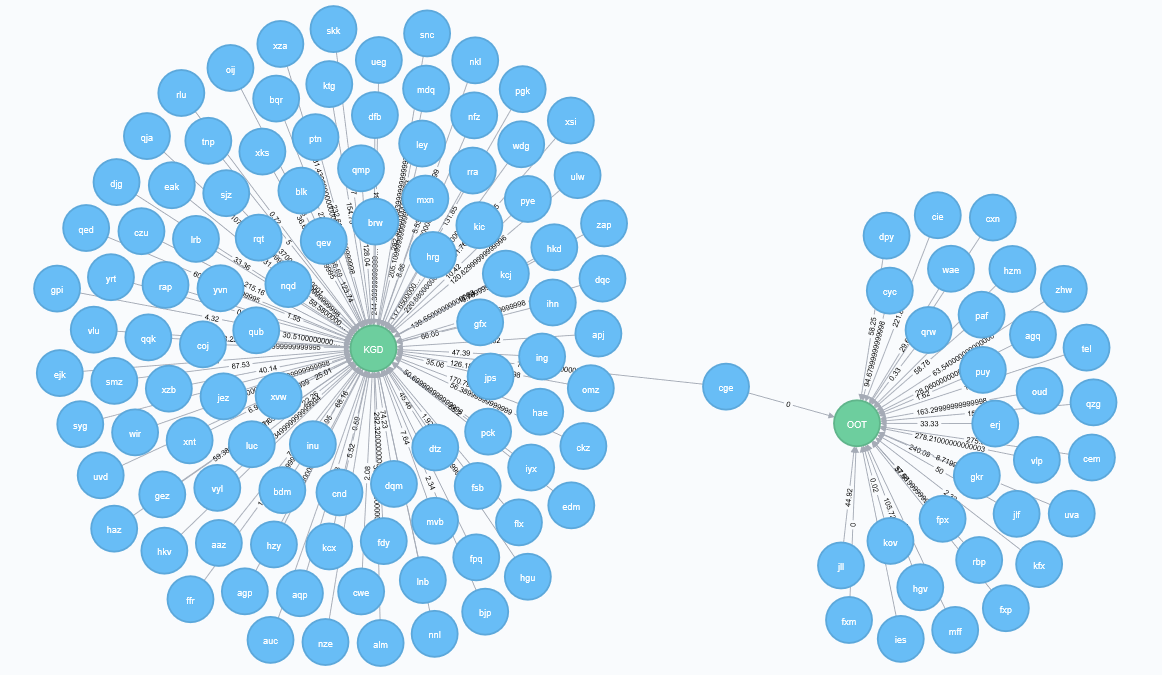


**Figure 10.** Developers with lowest closeness centrality and their contributions

* + 1. **Betweenness Centrality**

Betweenness centrality for each [node](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) is the number of these shortest paths that pass through the node (Freeman 1977). Nodes with higher betweenness scores therefore have more control of other nodes in terms of communication access.

In the develop-developer network, developer ‘cge’ has the highest betweenness centrality. As shown in Figure 11, ‘cge’ is the only person working in both ‘OOT’ and ‘KGD’, two principal repositories in the company. Thus, it is reasonable to infer that developer ‘cge’ plays a significant role for the information sharing in between.

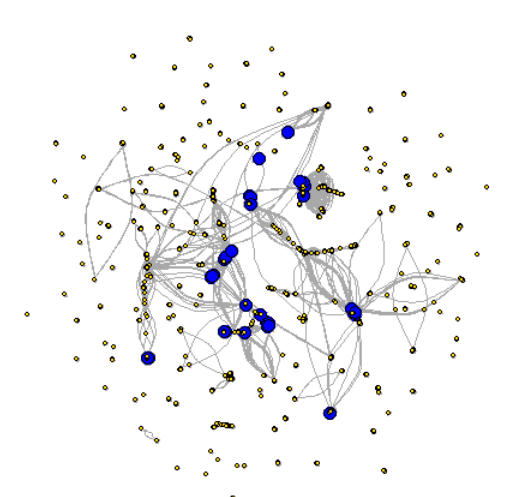


**Figure 11.** Developer ‘cge’ and corresponding connections

* + 1. **Eigenvector Centrality**

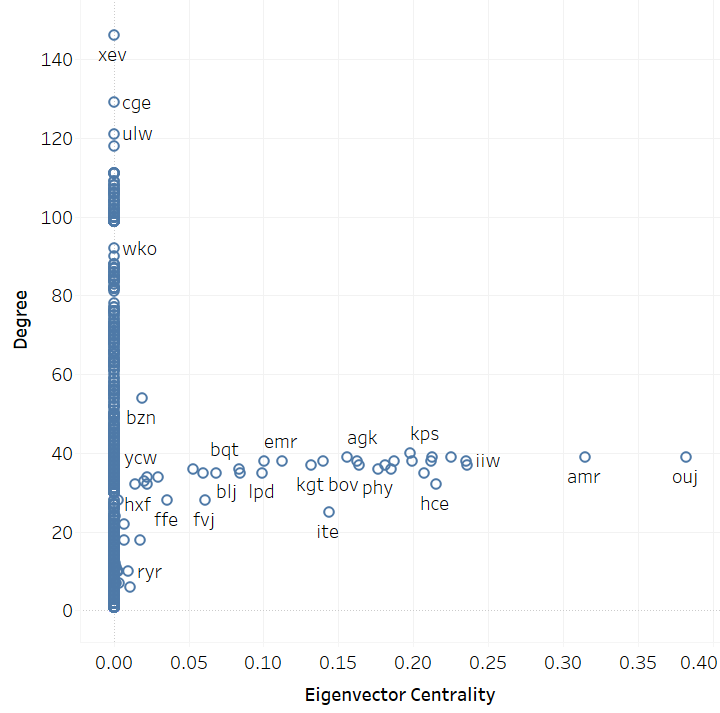
Eigenvector Centrality is a measurement revealing neighbors’ quantity and quality (Newman 2001). A high eigenvector score is resulted from a large number of high-quality neighbors. High-quality developers can be described as those involving in one or several principal repositories, or participating in diversified repositories, by which they interact with many people in the network. Correspondingly, low-quality developers are those engaging in unitary or individual repositories, by which they collaborate with few others in the network.

As it shown in Figure 12, nodes colored in blue are those with highest Eigenvector. We find that they are widely positioned among different subcomponents in the graph and are connected with different sets of nodes.



**Figure 12.** Visualization of developers with high eigenvector scores

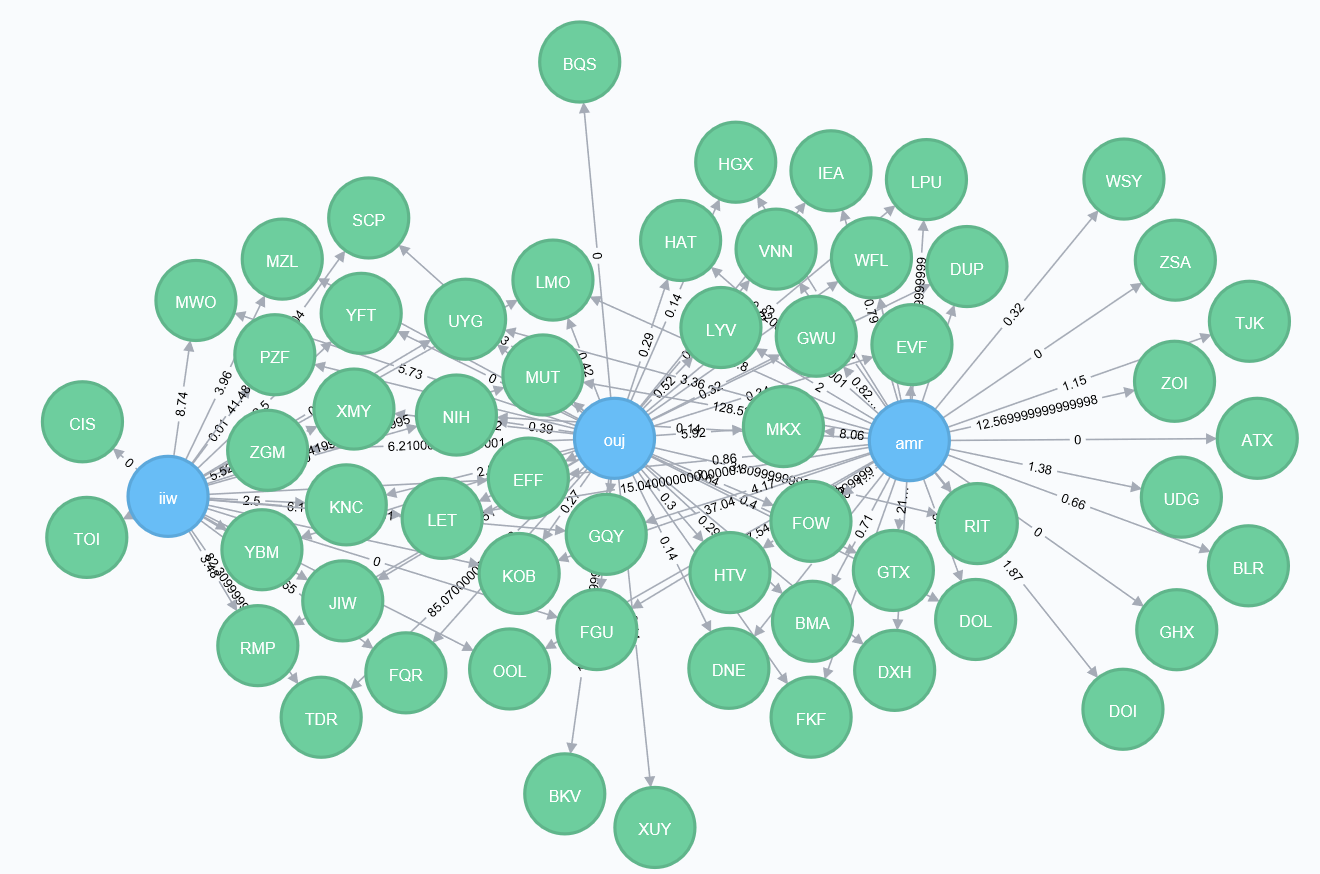
If looking at eigenvector and degree centrality together, as it shown in Figure 13, we find that most nodes have an eigenvector centrality of about zero. Also, although that ‘xev’ has more than three times of degree than ‘ouj’, ‘ouj’ has the largest eigenvector centrality while ‘xev’ the least. In addition, other than ‘ouj’, developer ‘amr’ and ‘iiw’ are another two developers having high eigenvector centrality scores, while developer ‘xev, ‘cge’ and ‘ulw’ have the lowest.



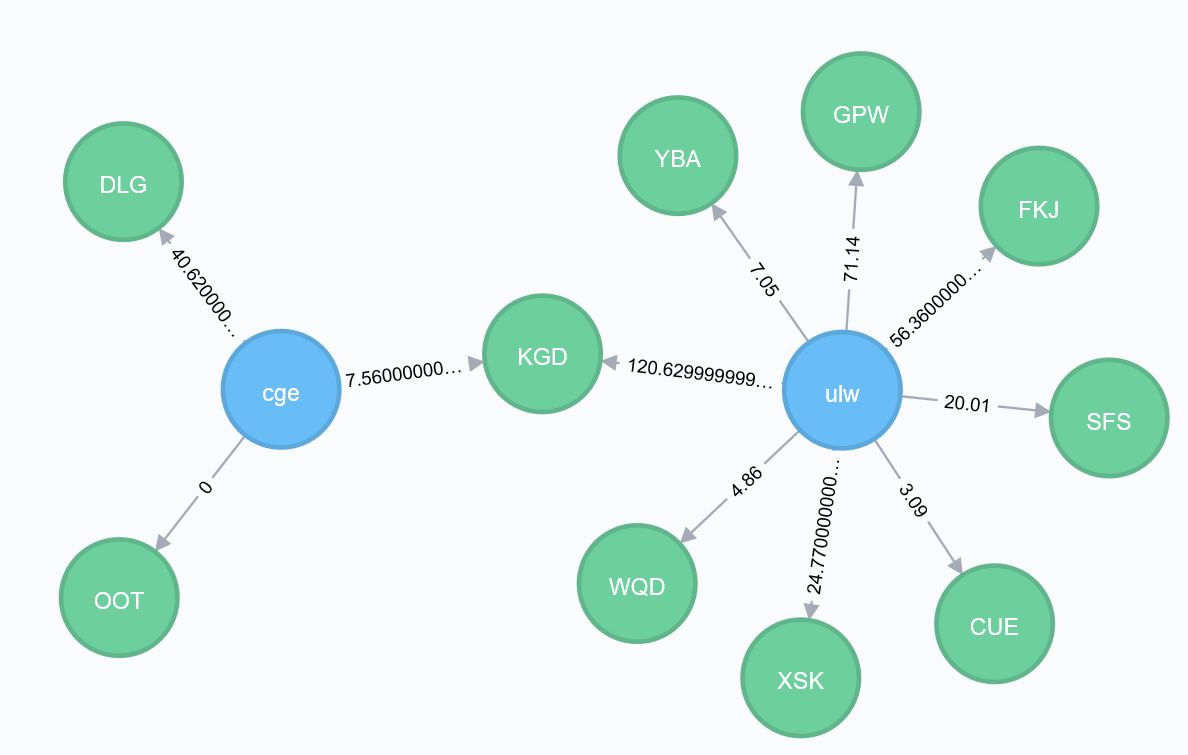
**Figure 13.** Scatter plot of degree and eigenvector centrality

Furthermore, as it shown in Figure 14, developer ‘ouj’,’amr’ and ‘iiw’ intensively share repositories. It can be therefore implied that those shared repositories are worked by a lot of high-quality developers, bringing those three developers high-quality neighbors and thus high eigenvector centrality.

On the contrary, as it shown in Figure 15, developer ‘cge’ and ‘ulw’ share only one repository, ‘KGD’. Chances are that ‘KGD’ is a repository worked by plenty of low-quality developers, giving both ‘cge’ and ‘ulw’ low-quality neighbors and thus low eigenvector centrality.



**Figure 14.** Developer ‘ouj’,’amr’ and ‘iiw’ share a lot of repositories.



**Figure 15.** Developer ‘cge’ and ‘ulw’ share one large repository ‘KGD’.

* 1. **Community Detection**

Communities in a network are groups of nodes internally connected or nodes sharing attributes (Girvan and Newman 2002). Detecting communities provides insights regarding the overall network structure, behavioral patterns of nodes and their relations (Fortunato 2010, Malliaros and Vazirgiannis 2013). For the developer-developer network, it supports findings about developers’ roles, collaboration preference as well as corporate cohesion. (Porter, Onnela and Mucha 2009)

However, community detection is an ill-defined problem. There is no universal definition of the methodology that one should follow. Consequently, there are no clear-cut guidelines on how to evaluate and compare the performance of different algorithms ([Fortunato](https://arxiv.org/find/physics/1/au:+Fortunato_S/0/1/0/all/0/1) and [Hric](https://arxiv.org/find/physics/1/au:+Hric_D/0/1/0/all/0/1) 2016).

Here we apply some popular and easy-accessible algorithms in open source tools, such as igraph and graph-tool (Csárdi 2017; Peixoto 2014), and compare their performance. More details on network clustering can be found in several review articles (Chakraborty et al 2016; Fortunato 2010; Malliaros and Vazirgiannis 2013; Satuluri et al 2011; Porter et al 2009; Xie et al 2013).

Algorithms we apply can fall into three categories: optimization based, statistical inference based and dynamics based.

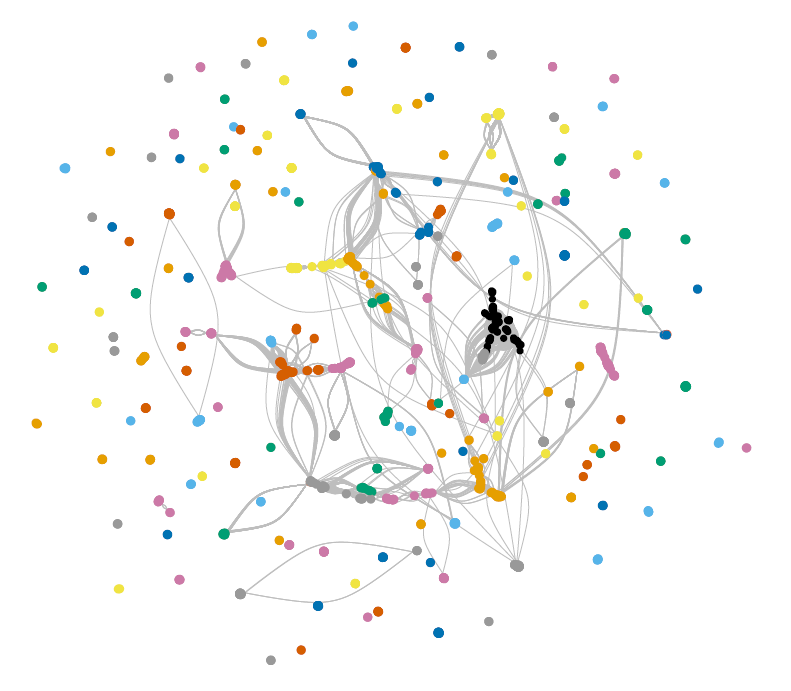
* + 1. **Optimization Based**

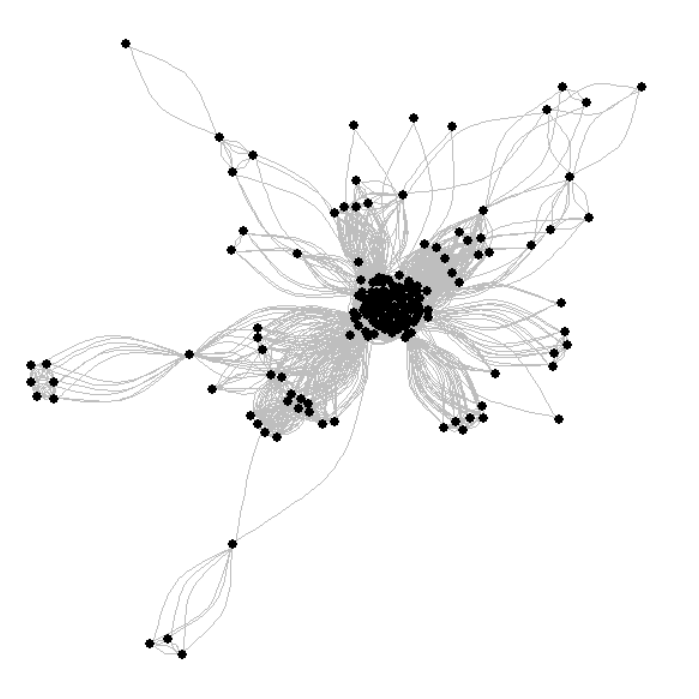
The idea of optimization based techniques is to optimize a quality function which measures the goodness of a clustering over all possibilities (Fortunato 2010). The most widely-used quality function is modularity by Newman and Girvan (Girvan and Newman 2006). It was designed to measure the strength of division of a network into communities. Networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules.

Based on the modularity measure, the multi-level modularity optimization algorithm is a hierarchical approach (Blondel et al 2008). It works according to the following steps: first, each node is assigned to a community independently. Afterwards, each node is moved to the community in a local, greedy way, where it achieves the highest contribution to modularity. When no node can be reassigned, each community is considered as a node on its own, and repeat the second step. The process stops either when there is only one node left or when the modularity cannot be increased any more.

The algorithm can be applied to weighted undirected graphs. The Multi-Level model generates a modularity of 0.946. Groups (or communities) contain 12 developers on average. The largest community in the network, according to the model, contains 196 developers, and there are 73 ungrouped developers.

Figure 16 displays 223 groups with different colors, among which the largest group is colored in black.





**Figure 16.** Visualization of community detection in the developer’s network

The main problem with optimization based approach is that it ignores the nature of the considered community structures. In addition, as we can see, there are 73 communities consists of only one developer. It is because suffering a resolution limit, modularity optimization methods are usually unable to detect small communities (Fortunato 2007).

* + 1. **Statistical Inference Based**

Unlike optimization based method, Statistical inference based method is a popular community detection way to generalize graph structures, by fitting [generative models](https://en.wikipedia.org/wiki/Generative_model) where parameters are inferred from data (Ball et al 2011; Guimer`a and Sales-Pardo 2009; Hastings 2006; Karrer and Newman 2011).

To find parameters for a best-fitted model, we use Minimum description length (MDL), which measures the amount of [information](https://en.wikipedia.org/wiki/Information_theory) required to describe the data. This approach corresponds to an implementation of [Occam’s razor](https://en.wikipedia.org/wiki/Occam%27s_razor), where the simplest model is selected, among all possibilities with the same explanatory power (Rissanen 1978).

The [stochastic block model](https://en.wikipedia.org/wiki/Stochastic_block_model) (SBM) is arguably the simplest generative process based on the notion of groups of nodes (Holland et al 1983). There are two types of [stochastic block model](https://en.wikipedia.org/wiki/Stochastic_block_model): non-degree-corrected and degree-corrected.

Non-degree-corrected model assumes that the edges are placed randomly inside each group, and as such the nodes that belong to the same group have very similar degrees. Therefore, the model usually performs bad on networks possessing highly heterogeneous degree distributions, while the degree-corrected model, on the other hand, provides a better fit for many empirical networks (Karrer and Newman 2011).

Table 6 shows the performance of two stochastic block models on the developer-developer network.

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Minimum description length** | **Group Number** |
| *Non-degree-correlated stochastic block model* | 68796.50778086044 | 52 |
| *Degree-correlated stochastic block model* | 67503.250630552488 | 61 |

**Table 6:** Degree correlated stochastic block model performs better than non-degree-correlated model based on minimum description length

A regular SBM has a drawback when applied to very large networks: it can’t find relatively small groups in very large networks. To address this problem, the nested stochastic block model (Peixoto 2014) was introduced. It not only finds small groups in large networks, but also provides a multilevel hierarchical description of the network. Table 7 demonstrates the general description of nested stochastic block models on the developer-developer network, while Table 8 shows the group membership of developer ‘xon’ in different hierarchies.

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Minimum description length** | **Inferred Hierarchy** |
| *Non degree-correlated nested stochastic block model* | 58349.392019931613 | 7 |
| *Degree-correlated nested stochastic block model* | 58447.470967214511 | 6 |

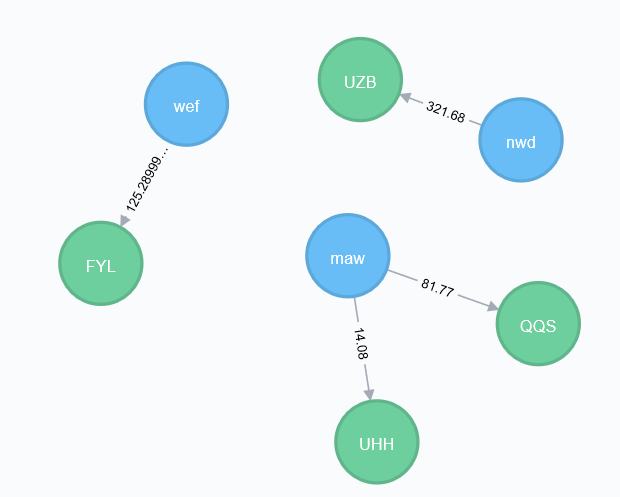
**Table 7:** Although it is often true that the degree-corrected model provides a better fit for many empirical networks, there are also exceptions. As it shown in Table 7, non-degree-correlated nested stochastic block model generates lower minimum description length on the developer-developer network than the degree-correlated nested stochastic block model.

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Hierarchy** | **Group Membership** |
| *Non degree-correlated nested stochastic block model* | 0 | 14 |
| 1 | 8 |
| 2 | 7 |
| 3 | 6 |
| 4 | 5 |
| 5 | 4 |
| 6 | 3 |
| *Degree-correlated nested stochastic block model* | 0 | 13 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |

**Table 8:** Developer ‘xon’ membership in different hierarchies

With the ability of generalizing graph structures, statistical inference based methods can be furthermore applied to predict missing or spurious edges, or potential edges in the network. The idea is generally that, based on the prior of observed graph, compute the posterior of graph with potentially connected edges, by sampling partitions from the posterior, then simulating the delete and insert of edges and eventually updating the likelihood (Clause 2008; Guimera 2005).

In the case of developer-developer network, detecting potential edges can lead to implications about which two developers are more likely to collaborate in the future. For example, there are no existing records about developer ‘maw’ collaborating with either ‘nwd’ or ‘wef’, as it shown in Figure 17. With statistical inference model however, we can then get a probability for the collaboration (as it shown in Table 9) and thus predict the likelihood of their future connection.



**Figure 17.** Developer ‘maw’ has no collaboration with either ‘nwd’ or ‘wef’.

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Potential collaboration** | **Probability** |
| *Degree-correlated nested stochastic block model* | ‘maw’ and ‘ nwd’ | 55.55% |
| ‘maw’ and ‘wef’ | 44.45% |

**Table 9:** ‘maw’ is slightly more likely to collaborate with ‘nwd’, rather than ‘wef’, based on the degree-correlated nested stochastic model.

Here we apply degree-correlated nested stochastic block model to demonstrate general nonparametric statistical models’ ability to infer missing or potential edges. More details on network completion problem can be found in several review articles (Rodriguez et al 2010; Guimer´a et al 2005; Hanneke et al 2009).

* + 1. **Dynamics Based**

Other than statistical and optimization, communities can also be identified by methods of running dynamical processes on the network (Fortunato 2010), among which Random walk dynamics is by far the most exploited.

Walktrap and Infomap are two algorithms both applying random walk, with the former based on an observation that short random walks have the tendency of staying in the same community, while the latter aimed at minimizing the expected description length of a random walk trajectory (Pons and Latapy 2006).

Table 10 shows the comparison of the model performance between Walktrap and Infomap.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Modularity** | **Group Number** | **Average Group size** | **Largest Group size** | **Ungrouped Developer Number** |
| Walktrap | 0.925 | 277 | 9.46 | 152 | 73 |
| Infomap | 0.901 | 303 | 8.65 | 77 | 73 |

**Table 10:** Comparison of Walktrap and Infomap

Infomap and walktrap model have the same ungrouped developer number, as well as similar modularity and average group size. However, the largest community in walktrap model contains 152 developers, twice of the counterpart in Infomap model.

A comprehensive and optimized way of inferring network structure through community detection is beyond the scope of this paper. Here we apply some easy-accessible algorithms, namely multi-level model, [stochastic block model](https://en.wikipedia.org/wiki/Stochastic_block_model)s, walktrap and Infomap, which can be categorized as optimization based, statistical inference based and dynamics based methods to the developer-developer network. When comparing model performance, we turn to modularity, minimum description length, as well as group number, average group size and ungrouped node number for reference.

## Summary of lessons from developers-developers graph

In order to understand the interaction among developers, we create a weighted developer-developer graph by repositorying the weighted bipartite developer-repository graph. Four graph properties, namely degree, closeness, betweenness and eigenvector centrality, are studied, from which we achieve implications about developers’ functionality, roles and collaboration. In addition, we compare and implement different community detection algorithms to categorize and capture the network structure.

# Modeling Dynamics: Evolving Network

Analyzing network dynamics has gained popularity and significance as longitudinal network data becomes more available. Organizations study corporate network in a dynamic way in order to discover the decision cycle of major events, track the evolutions of corporate finances, operations and culture, as well as detect changes in the organizational behavior. Being able to identify a change in network dynamics can enable managers to better prepare for the coming change and prevent potential problems (Doreian and Stokman 1997).

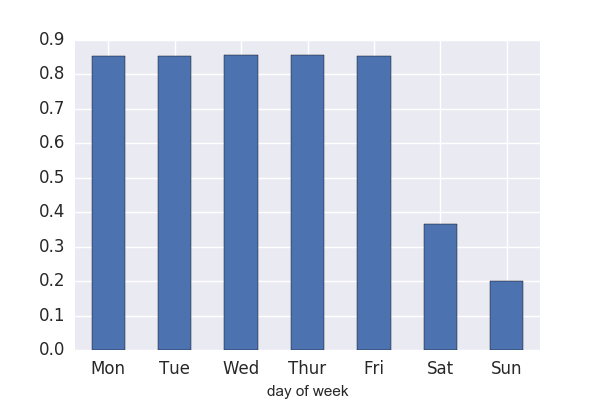
Dynamic network analysis (DNA) brings together traditional social network analysis, link analysis, social simulation and multi-agent systems within network science and network theory.

(Kathleen 2014). There are two aspects of this field. The first is the statistical analysis of DNA data. The second is the utilization of simulation to address issues of network dynamics. Dominant modeling methods include Markov chain models, multi-agent simulation models, and statistical models (McCulloh and Carley 2009). A complete application of dynamic network modeling is beyond the scope of this paper. Considering the nature of the given problem, we choose statistical models as our modeling approach. In other words, we compare the properties of networks at different points in time.

Clustering coefficient is a common measure of the degree to which nodes tend to cluster together (Kaiser 2008; Opsahl and Panzarasa 2009). While zero means no clustering, one indicates maximal clustering. There are two types of clustering coefficient: local and global. When applied to a single node, it is a measure of how complete the neighborhood of a node is (local). When applied to an entire network, it is the average clustering coefficient over all of the nodes in the network (global).

Figure 18 shows the global coefficient clustering for the developer-developer graph in different days of week. We can see that the global coefficient clustering stays high (about 0.85) and constant from Monday through Friday and decrease largely in weekends.

Figure 19 shows the clusters of developer-developer graph on different days of week. Clusters are generated by the multi-level algorithm. Despite of having similar attributes, such as number of nodes and communities, graphs are distinctive in terms of structures, indicating that the collaboration among developers varies daily from Monday through Friday.



Coefficient clustering

**Figure 18.** Global coefficient clustering for the developer-developer graph by day of week

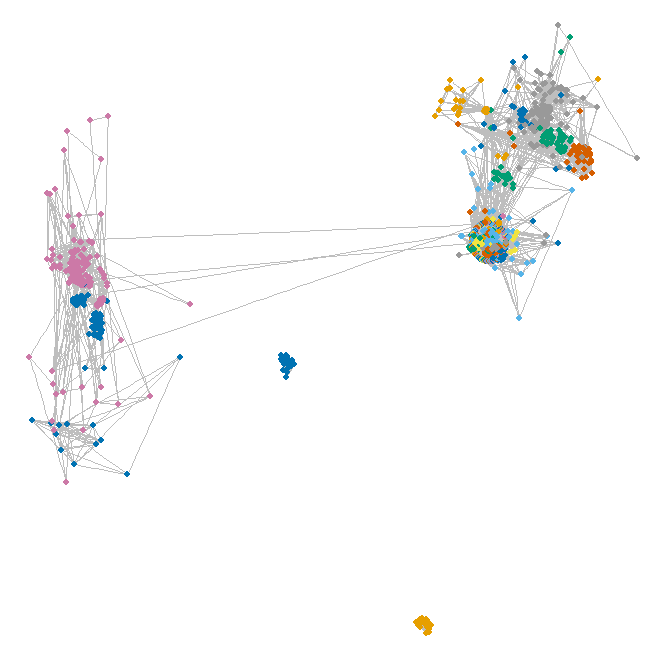
**Monday**

Nodes number:2574

Edges number: 29963

Community number: 241

Ungrouped developer number: 78



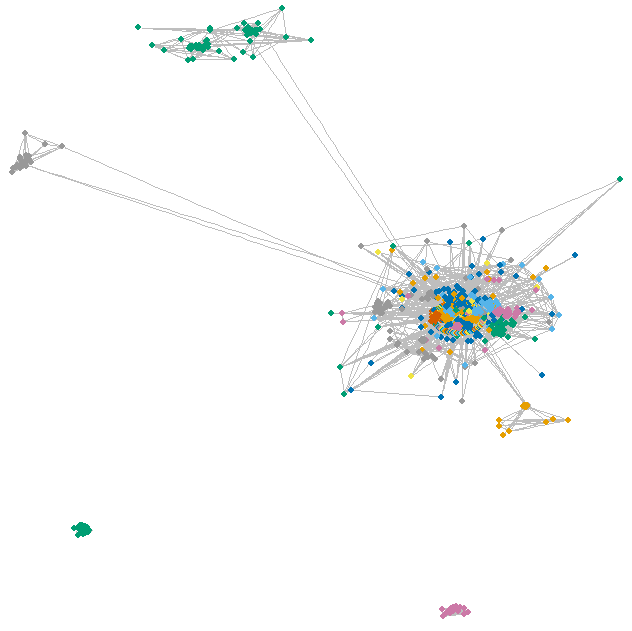
**Tuesday**

Nodes number:2572

Edges number: 30356

Community number: 248

Ungrouped developer number: 81



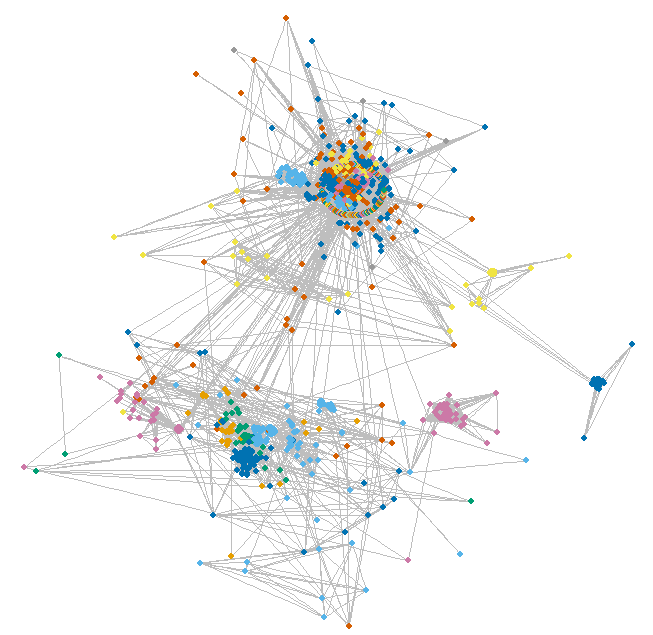
**Wednesday**

Nodes number:2592

Edges number: 31592

Community number: 239

Ungrouped developer number: 77



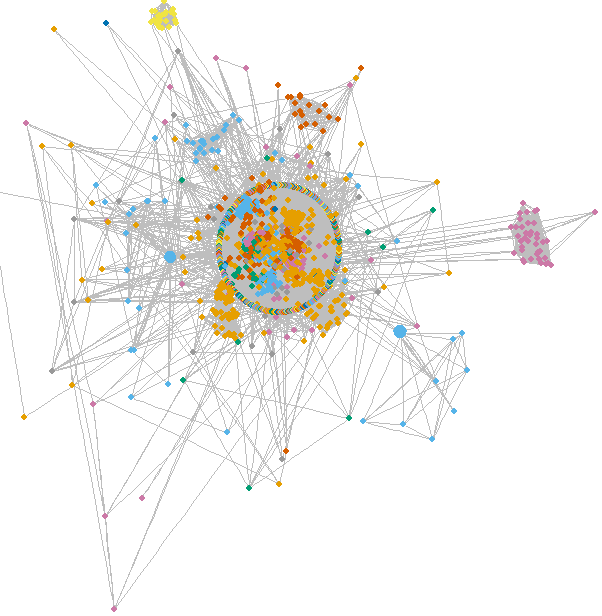
**Thursday**

Nodes number:2576

Edges number: 30697

Community number: 251

Ungrouped developer number: 85

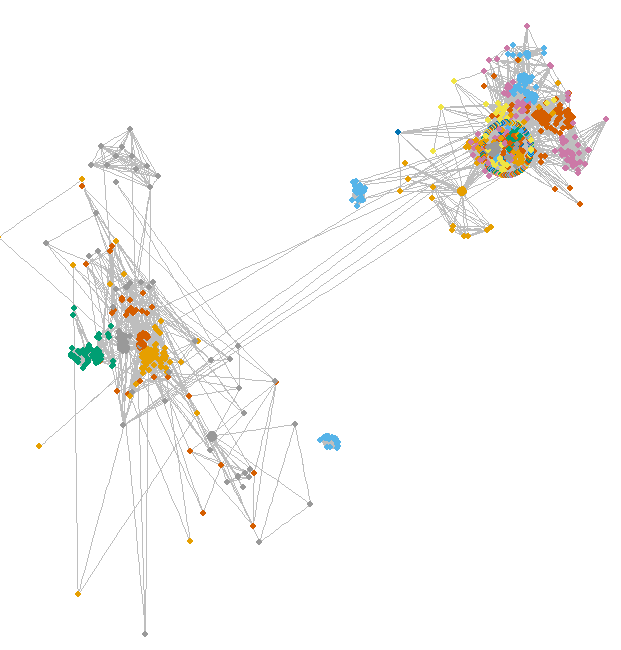


**Friday**

Nodes number:2573

Edges number: 30664

Community number: 251

Ungrouped developer number: 86

、

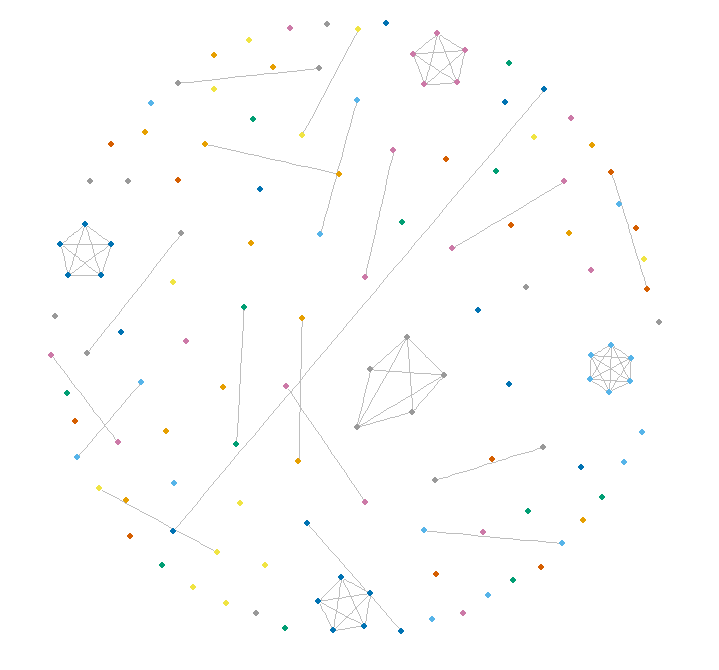
**Sunday**

Nodes number:128

Edges number:72

Community number: 89

Ungrouped developer number: 66



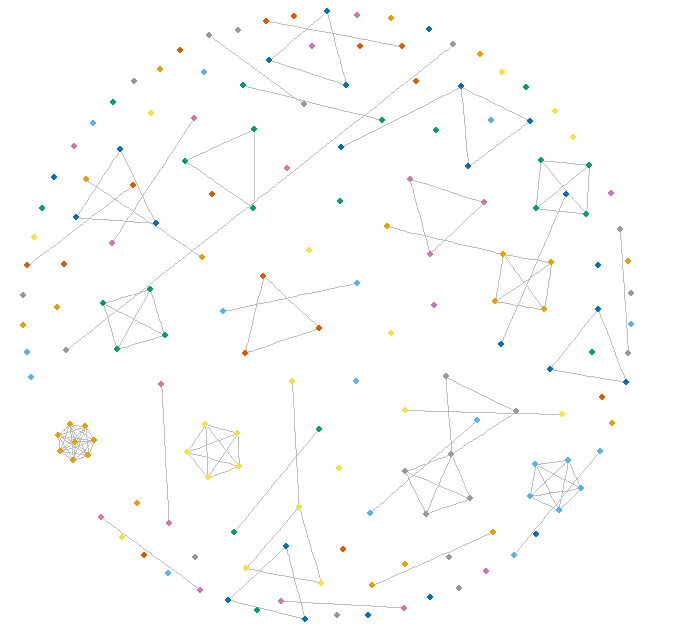
**Saturday**

Nodes number:165

Edges number:123

Community number: 97

Ungrouped developer number: 63



**Figure 19.** Clusters of developer-developer graph on different days of week

# Discussion and concluding remarks

Network data science is the use of data science methods, tools and algorithms in the modeling and analysis of network (graph) data. We demonstrate how a network data science approach, using a variety of algorithms and methods and open source tools can help companies gain rich insights about employee effort, contribution, and collaboration. These insights are useful in order to optimize work patterns and productivity in organizations.

# A study of the proprietary software development started with constructing weighted graphs. Initial insights about developers’ role (managers or engineers), productivity and individual influence were achieved through analyzing network properties of the developers-repositories graph. We proposed a recommender system that managers could consult to assign employees to repositories. Implications about software development cohesion and culture were developed through the analysis of the developers-developers graph.

# In addition to the analysis of a snapshot or aggregated version of the software development network, we did a longitudinal and dynamic analysis. This provides better understanding of how node attach and detach from the network (developers dropping off or initiating repositories), and thus lead to a more thorough understanding of developers’ working pattern.

There are many opportunities for future research to extend the methodology proposed in this paper. For example, future research could apply and extend the methodology in multiple other organizational contexts beyond software development. Other research could combine graph theoretic data and insights with other contextual data to help organizations achieve a deeper understanding of their performance. More data could also help in designing interventions, such as extensions of our proposed recommender system, that lead to even higher levels of performance.

# References

Albert R., Barabasi A.-L. 2002. Statistical mechanics of complex networks. Rev. Mod. Phys. 74: 47–97.

Aral, S., Brynjolfsson, E. and Van Alstyne, M., 2012. Information, technology and information Developer productivity. *Information Systems Research*, 23(3), 849-867.

Ball, B., Karrer B., and Newman M. E. J. 2011. An efficient and principled method for detecting communities in networks. Phys. Rev. E 84, 036103.

Blondel Vincent D., Guillaume Jean-Loup, Lambiotte R., Lefebvre E. 2008. Fast unfolding of communities in large networks. [arXiv:0803.0476](https://arxiv.org/abs/0803.0476) [physics.soc-ph].

Breese J.S., Heckerman D., and Kadie C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth Conference on Uncertainty in Artifical Intelligence.

Carley K. M. 2014. ORA: A Toolkit for Dynamic Network Analysis and Visualization. Encyclopedia of Social Network Analysis and Mining, Springer.

Chakraborty T., Kumar S., Ganguly N., Mukherjee A., Bhowmick S. 2016, GenPerm: A Unified Method for Detecting Non-overlapping and Overlapping Communities. preprint arXiv:1604.03512.

Clauset, A., Shalizi, C. R., Newman, M. E. J. 2009. Power-Law Distributions in Empirical Data. SIAM Review 51(4):661-703, 2009.

Clauset A., Moore C., Newman M. E. J. 2008. Hierarchical structure and the prediction of missing links in networks. Nature 453, 98-101 (2008), DOI: 10.1038/nature06830

Csárdi G. et al 2017. Network Analysis and Visualization. http://igraph.org

Das A., Datar M., Garg A., and Rajaram S. 2007. Google News Personalization: Scalable Online Collaborative Filtering. International World Wide Web Conference, Proceedings of the 16th international conference on World Wide Web

Doreian, P., and Stokman, F.N. (Eds.) 1997. Evolution of Social Networks. Amsterdam: Gordon and Breach.Fan Y., Li M., Zhang P., Wu J., Di Z. 2007. The effect of weight on community structure of networks. PHYSICA A 378 (2007) 583–590

Fortunato, S., Barthelemy, M. 2007. Resolution Limit in Community Detection. PNAS 104, 36- 41.

Fortunato S. 2010. Community detection in graphs. Phys. Rep. 486 (3–5): 75–174.

[Fortunato](https://arxiv.org/find/physics/1/au:+Fortunato_S/0/1/0/all/0/1) S., [Hric](https://arxiv.org/find/physics/1/au:+Hric_D/0/1/0/all/0/1) D. 2016. Community detection in networks: A user guide. arXiv:1608.00163 [physics.soc-ph]

Freeman, Linton 1977. A set of measures of centrality based on betweenness. Sociometry. 40: 35–41.

Girvan M., Newman M. E. J. 2002. Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA. 99 (12): 7821–7826.

Guillaume, J., Latapy, M. 2004. Bipartite structure of all complex networks. Information Processing Letters, 90(5):215-221.

Guimera, R., and Amaral L. A. N. 2005. Missing and spurious interactions and the reconstruction of complex networks. Nature 433, 895.

Hanneke S., Xing E. 2009. Network completion and survey sampling. In AISTATS ’09.

Hastings, M. B. 2006. Community Detection as an Inference Problem. Phys. Rev. E 74(3), 035102.

Hazewinkel, Michiel, ed. 2001, Kolmogorov–Smirnov test,  [Encyclopedia of Mathematics](https://en.wikipedia.org/wiki/Encyclopedia_of_Mathematics), Springer, [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-1-55608-010-4](https://en.wikipedia.org/wiki/Special:BookSources/978-1-55608-010-4).

Holland Paul W., Laskey Kathryn B., Leinhardt S. 1983. Stochastic blockmodels: First steps, Carnegie-Mellon University, Pittsburgh, PA 15213, U.S.A.

Kaiser M. 2008. Mean clustering coefficients: the role of isolated nodes and leafs on clustering measures for small-world networks. New Journal of Physics. 10 (8): 083042 arXiv:0802.2512Freely accessible.

Karrer B., and Newman M. E. J. 2011. Stochastic Blockmodels and Community Structure in Networks. Physical Review E 83, no. 1 (2011): 016107

Malliaros F. D., Vazirgiannis M. 2013. Clustering and community detection in directed networks: A surve. Phys. Rep. 533 (4): 95–142.

Marsaglia G, Tsang WW, Wang J. 2003. Evaluating Kolmogorov's Distribution. Journal of Statistical Software. 8 (18): 1–4.

McCulloh Ian, Carley K. 2009. Longitudinal Dynamic Network Analysis. CMU-ISR-09-118

McGarry F., Card D. et al 2001. Practical Software Measurement, p19. Addison-Wesley.

Neo Technology. 2015. Neo4j Graph Database. Retrieved from https://neo4j.com/product/

[Newman M. E. J. 2001. Scientiﬁc collaboration networks. II. Shortest paths, weighted networks, and centrality. PHYSICAL REVIEW E, vol. 64, 016132.](http://www-personal.umich.edu/~mejn/papers/016132.pdf)

Newman, M.E.J. 2010. Networks: An Introduction. Oxford, UK: Oxford University Press.

Newswire 2013. BlueOptima *Coding Effort Analytics™* Analytics Announces Support for Git. Retrieved from <https://www.newswire.com/blueoptima-coding-effort-analytics/252989>.

Opsahl T., Panzarasa P. 2009. Clustering in Weighted Networks. Social. Networks. 31 (2): 155–163.

Peixoto Tiago P. 2014. The graph-tool python library. doi:10.6084/m9.figshare.1164194 Key: citeulike:13701202.

Peixoto Tiago P. 2014. Hierarchical block structures and high-resolution model selection in large networks. Phys. Rev. X 4, 011047.

Pons, P., Latapy M. 2006, Computing Communities in Large Networks Using Random Walks. International Symposium on Computer and Information Sciences (Springer), pp. 284– 293.

Porter M. A., Onnela J.-P., Mucha P. J. 2009. Communities in Networks, Math. Soc. 56: 1082–1097, 1164–1166.

Ricci F., Rokach L., Shapira b. 2011. [Introduction to Recommender Systems Handbook](http://www.inf.unibz.it/~ricci/papers/intro-rec-sys-handbook.pdf), Recommender Systems Handbook, Springer, 2011, pp. 1-35

Rissanen, J. 1978. Modeling by shortest data description. Automatica. 14 (5): 465–658.

Rodriguez M. Gomez, Leskovec J., Krause A. 2010. Inferring networks of diffusion and influence. KDD ’10.

Satuluri V., Parthasarathy S. and Ruan, Y. 2011 Local Graph Sparsification for Scalable Clustering. In: Proceedings of the 2011 International Conference on Management of Data, ACM Press, New York, 721-732.

Statista. 2014. Worldwide mobile app revenues in 2015, 2016 and 2020 (in billion U.S. dollars). Retrieved from <https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/>

Xie J., Kelley S., and Szymanski B. K., 2013, Overlapping community detection in networks: The state-of-the-art and comparative study. ACM Comput. Surv. 45(4), 43:1.

Zhan, Benjamin F., Noon, Charles E. 1998. Shortest Path Algorithms: An Evaluation Using Real Road Networks. Transportation Science. 32 (1): 65–73. doi:10.1287/trsc.32.1.65.

Zhou T., Ren J., Matus M., Zhang Yi-Cheng. 2007. Bipartite network projection and personal recommendation. PHYSICAL REVIEW E 76(4): 046115.