Workload Assignment 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**DO NOT CITE/DISTRIBUTE**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>

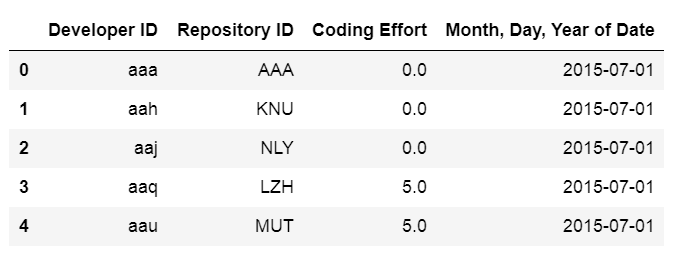
# Introduction

………

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.

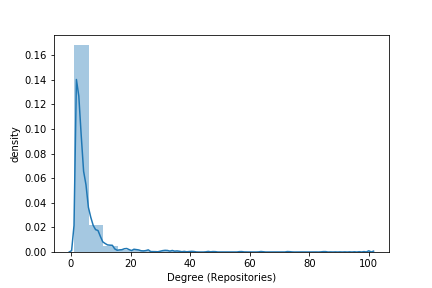
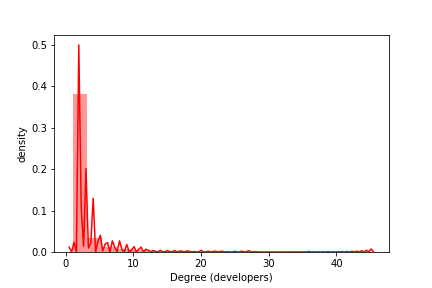


**Table 1.** **Sample of original dataset. *Coding Effort Analytics™* is a proprietary software measurement and is calculated through evaluating every change that software developers contribute to repositories in terms of a series of metrics, such as volume, complexity and interrelatedness of codes (Newswire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

There are many records with zero *Coding Effort Analytics™* in the data. This is due to the fact that as long as a developer is involved in a repository, *Coding Effort Analytics™* is going to be 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 challenge for collaborative filtering.



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

One more thing worth mentioning, regarding the sparseness of the data, 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. Naturally, our recommender system, which is neighborhood-based, will not make recommendations for those 61 developers.

Table X is the summary of the properties of the developer-repository 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 extensive 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 (……). Thus, topics in graph analysis such as graph projection, graph properties and community detection haven’t been as exhaustively applied in recommender systems as traditional ways such as Matrix Factorization (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 this paper, we explore neighborhood based collaborative filtering, to which we apply techniques such as graph projection and community detection. We calculate similarity in a manner of weight allocation in graph projection and detect neighbors by node attributes and underlying structure.

* 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 of paramount importance. 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 workload assignment, we apply a customized weighting method as the way of calculating similarity between two developers and finding neighbors.

* 1. **Graph Properties**

Statistical metrics to describe graph properties include degree centrality, closeness centrality, betweenness centrality, eigenvector centrality and PageRank. 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 utilize those metrics to group developers together according to their functionality and thus find their “neighbors”.

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. In other words, Eigenvector Centrality measures the influence of a [node](https://en.wikipedia.org/wiki/Node_(networking)) in a [network](https://en.wikipedia.org/wiki/Network_(mathematics)). PageRank, the famous Google search algorithm, is one of the variants of the eigenvector centrality. (Page, Larry, "PageRank: Bringing Order to the Web" at the Wayback Machine (archived May 6, 2002), Stanford Digital Library Project, talk. August 18, 1997).

* 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. Additionally, clustering is one of the techniques proved to be able to improve the results of collaborative filtering especially when data is sparse.

In order to capture the implications of both developers’ relationships (graph structure) and roles (graph properties), we employ a combination of community detection and clustering as mechanisms to identify “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, which is 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 algorithms are all aimed at optimizing modularity.

* + 1. **Algorithms for Clustering**

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One of the usual ways to learn distribution parameters is Expectation Maximization (EM). It works by repeating the following three steps: firstly, assign data points to random components, then compute the probability of data points being generated by other components, and finally adjust distribution parameters to maximize the likelihood of observing data points given assigned components. The algorithm repeats those three steps until converging to a local optimum.

In unsupervised learning problems, cluster numbers can in some ways be implied by domain knowledge. For example, in our software development case, we can get hints about the developer cluster from the number of job types and title levels.

In this paper alternatively, we turn to Bayesian information-theoretic criteria (BIC) to determine cluster number. BIC is a common statistics measurement which balances variance (overfitting) and bias (underfitting) in the process of model selection. The model with lower BIC is preferred. In addition to component number, covariance structure of components is another important parameter for building Gaussian mixture models. Component number combined with covariance matrix determines the orientation, shape and size of clusters. Covariance matrix can either be shared or differentiated among components.

We cluster developers together based on their attributes using four Gaussian mixture models with varied covariance structures.

# Methodology

* 1. **Training and Testing Data**

As a common step in machine learning problem, we split the data into training set and testing set, with the former to structure bipartite graph, provide property metrics, do projections (allocate weights/calculate similarities), generate communities and form groups, and the latter to validate model performance.

As mentioned before, the data is panel data, covering 92 days’ activities from July 1 to September 30. Given that recommender system need to observe and learn the past before making predictions for the future, training and testing set for the model are therefore generated in the order of time. Common ratios for training and testing split are like 80% versus 20% or 75% versus 25%, with training set always containing more records than testing set. In our repository assignment problem however, training set only covers records of first 7 days and testing set have all the rest.

There are two main reasons for this splitting way. One is that a “transferring effect” is expected in developers’ behaviors. To be more specific, it takes time before developers start working on new repositories. Therefore, the performance of a recommender system would be fairly evaluated if only based on short-term developers’ behaviors. To take care of this “transferring effect” in estimating our model, we thus need to make sure that testing set contains enough time points,

Another reason is that, in our case, the function of training set is to construct a bipartite graph which somehow implies developers’ mutual interactions, roles, functionality and working preference. Therefore, unlike other typical machine learning problems where training set is usually aimed at tuning model parameters, our neighborhood-based recommender system doesn’t necessarily require massive training data, especially under the circumstance when data is limited and long-term testing data is needed.

Bearing that in mind, combined with the assumption that most individual or corporate behavioral routines can be implied within one business-cycle time (one week), we decide to make training data set include the first 7 days’ records. In other words, by supposing that observing developers’ behaviors in one week gives us enough clues about their attributes (preference, roles, functionality, etc.), we divide the dataset in a way which results in a training set of 7 days’ incidents and testing set of 85 days.

Accordingly, there are 13,155 records about 2,321 developers’ contribution in the training set and 159,199 records regarding 2,601 developers’ in the testing set. There are 280 developers appearing in the testing set don’t have any records in training. Being a neighborhood based model by nature, our model will not be able to make recommendations for those 280 developers.

* 1. **Baseline Model**

One commonly used baseline model for recommender systems is global ranking method (GRM), which recommends items purely based on their global popularity. This method involves no personalization and therefore achieves deficient performance on individual level. However, it is scalable and feasible on extremely sparse data. In our case, it is intuitively infeasible to apply GRM on a workload assignment problem because of distinct specialization of labor division.

Alternatively, we will use a common collaborative filtering way with similarity metric being Pearson correlation coefficient as our baseline model.

As our baseline model, we firstly project the developer-repository graph in a binary mode, leaving edge weights unconsidered and then apply a combination of community detection and clustering methods on the resulted one-mode graph. As discussed above, community detection and clustering group nodes together respectively from the perspective of graph structure and nodes attributes, which, in our case, are translated as developers’ interaction activities and attributes.

In the category of community detection, five algorithms are employed: greedy optimization of modularity, label propagation, multi-level optimization of modularity, optimal community structure and walk trap, while in the clustering category, four Gaussian Mixture models with different covariance parameters are applied. The optimized component number is determined by using information-theoretic criteria (BIC). Metrics of describing graph properties are utilized to generate clustering features, namely degree centrality, betweenness centrality, closeness centrality, eigenvector centrality and PageRank.

* 1. **Similarity by Graph Projection**

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

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)

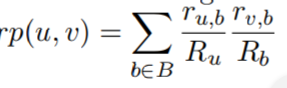
**Rb** (Total *Coding Effort* Repository b receives)

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.

Intuitively, 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)**

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 indicate the similarity between every two developers.

* 1. **Identify Neighbors**

In our paper, we identify neighbors from three ways: directly connected nodes on the projected graph, nodes from the same community and nodes from the same cluster.

With the projected developer-developer graph, we can locate all the nodes (developers) directly connected to a certain node (target developer) of ui. We define themas ui’s neighbors, and the similarity between ui and neighbors is the weight assigned to an edge in between.

Neighbors, moreover, can be from the same community, or the same cluster. The similarity between node ui and neighbors particularly from communities and clusters is assigned to a uniform distribution. In other words, we assume that to node ui., neighbors from communities and clusters have the same importance, or “recommendation power”.

# Model Evaluation

Mean average precision (MAP) is applied as our model evaluation metric. The advantage of mean average precision, compared to other metrics such as precision and recall, is that it takes recommendation order into consideration and penalizes bad guess. If every user is only recommended with 10 items, then the metric will be referred as MAP@10.

In our case, 945 developers start three new repositories in the following 85 days, with most developers having only one new repository. Accordingly, we use MAP@10 as our evaluation metric, which means that we give each out of 945 developers 10 suggested repositories (out of 921 repositories) to work on, and based on the fact of whether they truly work on any of them in the following 85 days, we determine the performance of our model.

We also use accuracy as our metric. For every recommendation attempt:

1; 10 suggested Repositories contain at least “True repository”

0; All 10 suggested Repositories miss the targets

Accuracy = {

# Conclusion

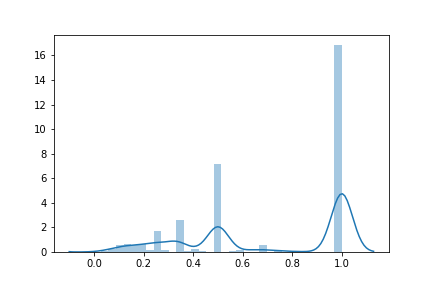
In our developer-repository case, one interpretation of similarity is the collaboration frequency. For example, if a large ratio of one developer’s repositories overlaps another’s, it is reasonable to infer that they use the same programming language or they share skillsets and thus they are similar. 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)

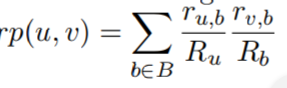
**Rb** (Total *Coding Effort* Repository b receives)

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.

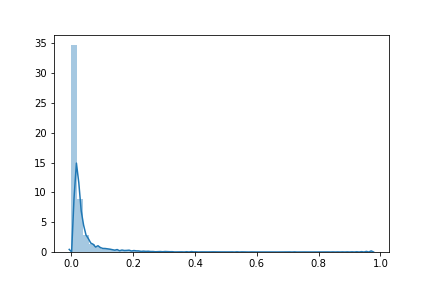


**TP (u, v)**

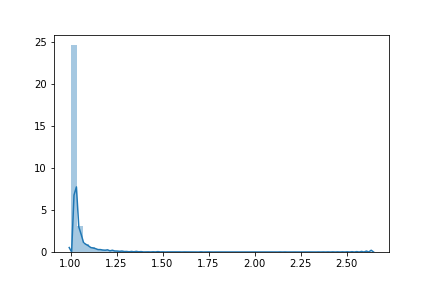
The resulted transition probability can therefore be seen as the recommendation power which Developer u gives to Developer v. As discussed above that recommendation power can be considered as similarity, we conclude with the formula below:

**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.

.

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.

# 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.