Clustering scientific articles

Jonas Arnfred, *School of Computer and Communication Sciences, EPFL*
Email: jonas.arnfred@epfl.ch

I. Introduction

When we try to make sense of a big amount of information, a natural way of beginning is to divide the information into smaller groups, and investigate these groups seperately. Often we assign these groups manually, but when reviewing a lot of information this can easily become unfeasible. In order to aid a lot of techniques for finding clusters in data using different heuristics and techniques have been developed. However the automatic methods don't always result in meaningful clusters. They often discard information we would implicitely include were we grouping the information manually and the groups that are calculated might be across features and of sizes that render them less useful. In this work I will address the choice of automated measures when clustering scientific articles.

A. Background

This project aims to explore how a group of conference articles are best clustered. It is part of a project to provide a tool for conference attendees to better get an overview over the hundreds of presentations that usually take place at a scientific conference. In order to facilitate exploring the presentations, we display the articles in a similarity graph where two similar articles are connected more tightly than articles describing different topics. The similarity is currently based on a naïve bayesian comparison of word frequencies, but could in theory be substituted with any given similarity measure.

Based on the similarity graph the aim is to create a grouping which satisfies a few practical as well as theoretical criteria. These criteria are selected with respect to real world usage where running times and implementation complexity are important. The criteria are:

- *Speed*. Since this is a tool made for real world usage it must be feasible to calculate a grouping of up to a thousand articles within a few hours on normal hardware
- *Quality*. A given solution should be as 'good' as possible. Later in this paper we will explore more in depth how we can possibly measure what a 'good' clustering implies
- *Simplicity*. Given two similar solutions, the simpler one is better for two reasons. First, a simple solution is easier to implement and readapt to changing requirements. Second, a simple solution is less prone to contain bugs
- *No Parameters*. Ideally an algorithm for finding groups should not need to be provided with anything else than the similarity graph

B. Structure of the paper

In this paper I will outline a selection of different methods used to cluster similarity graphs and evaluate each one based on the above criteria. Based on this selection I will propose two algorithims and outline first how they have been implemented and then how they compare. To begin with I will start with a quick primer on terms and notation used in the rest of this paper.

C. Terms and Notation

For the reminder of this paper I will rely on concepts and notation from Graph theory. To facilitate the discussion of different graph clustering algorithms I will introduce a few simple concepts here that will appear several times later on. A *weighted graph* G is a tripple of two sets and a function: $G = (V, E, \omega)$. V and E are sets of *vertices* and *edges* respectively, with $n = |V|$ being the *order* of the graph and $m = |E|$ being the number of edges. Each edge e_{ij} in an *undirected graph* is an unordered pair $\{v_i, v_j\}$ and has a weight w_{ij} assigned by the function $\omega : E \rightarrow \mathbb{R}$. The degree of a vertix $v_i \in V$ is defined as $d_i = \sum_{j=1}^{n} w_{ij}$. The adjecency matrix $\mathbf{A}$ is an $n \times n$ matrix $\mathbf{A} = (A_{i,j})$ where $A_{i,j} = \omega(\{v_i, v_j\})$ if $\{v_i, v_j\} \in E$ and 0 otherwise. The degree matrix $\mathbf{D}$ in turn is the diagonal matrix of size $n \times n$ such that $D_{ii} = d_i$ [1].

Additionally it is worth mentioning that the terms *cluster* and *community* are both used in the academic litterature usually covering over the same concept. Where algorithms on graphs are involved the word *clustering* has traditionally been used more in relation with graphs, communities are often mentioned in terms of *networks* that are then later formalized as graphs. In this paper I have chosen to use the words *cluster* and *clustering*, but this choice is not intended to reflect a stronger connotation with one group of methods than another.

II. Related work

Over the last 40 years a large variety of algorithms have been invented to cluster graphs. Most of these methods were based on different measures of an optimal clustering all of which were usually np-hard to derive optimal solutions for. This led to various algorithms attacking relaxed versions of these problems with different amount of success. The field has seen a boost in recent years due to the increased demands for handling large data that are usually neatly abstracted as graphs [Fortunato, 2010, p. 2]. Examples include social networks and the world wide web.

In this paper I don't intend to do a thorough review of the field, something which many better abled people have already done before me[2]. Instead I have tried to pick out a few distinct algorithms that are good enough that each could have been

[1] Adapted from Schaeffer [2007] and Von Luxburg [2007]
[2] Examples include Newman [2004], Schaeffer [2007] and Fortunato [2010]

a candidate for a clustering algorithm used for grouping the scientific articles.

A. How to measure clustering

The central difficulty when proposing a new clustering algorithm is to measure how it compares with other algorithms. Since there is no agreed upon definition on what makes a good cluster, there is naturally no definitive candidate for measuring the "goodness" of a cluster. Historically there has been several different approaches. Zachary [1977] did a study on the relationships between members of a karate club which eventually split into two factions which has since then been a benchmark for clustering algorithms. If an algorithm on the basis of the social graph can accurately predict the two factions in the split, then it has been judged as useful[3].

Fortunately better metrics have been proposed since. One of the most used has been the measure of *modularity* proposed by Girvan and Newman [2002] and defined as:

$$\begin{aligned} Q &= \text{(fraction of edges within communities)} \quad (1) \\ &\quad -\text{(expected fraction of such edges)} \quad (2) \\ &= \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j) \quad (3) \end{aligned}$$

Where $m = \frac{1}{2}\sum_{ij} A_{ij}$, $k_i = \sum_j A_{ij}$ and A_{ij} is the weight between the edge connecting i and j. $\delta(c_i, c_j)$ is the Kroneker Delta which is 1 only if the community assignment c_i is equal to the community assignment c_j. The community measure has been the de-facto standard but it is not without shortcomings. It has been shown by Brandes et al. [2007] that the metric doesn't work well with small communities.

An alternative measure called Surprise was introduced by Arnau et al. [2005] and has later been shown to perform better than the community measure in a series of cases by Aldecoa and Marín [2011], although it has yet to be independently adopted verified and adopted by other researchers. The idea behind Surprise is to model the distributions of intra- and inter-community links with a cumulative hypergeometric distribution. It is defined as follows:

$$S = -log \sum_{j=p}^{Min(M,n)} \frac{\binom{M}{j}\binom{F-M}{m-j}}{\binom{F}{n}} \quad (4)$$

Here F is the maximum possible number of edges in the graph, that is $F = (n^2 - n)/2$, while M is the maximum number of edges within a cluster A of size k, that is $M = (k^2 - k)/2$. p is the actual number of links found in the community. The authors have not proposed a similar formula for weighted graphs to the author's knowledge.

An alternative method of testing graph clustering algorithms is to design a series of graphs with known partitioning and see how well the algorithm in question predicts these graphs. The most widely used of such graphs was introduced by Girvan and Newman [2002]. They consisted of 128 vertices with four clusters of even size. The average degree of each vertex was 16. A parameter k_{out} would designate how many edges a vertex would have leading to other clusters. With $k_{out} < 8$ a good clustering algorithm should be expected to find the original partitioning. However these graphs are a very limited subset of the real graphs that clustering algorithms are supposed to work with. In particular they are small, the clusters are of an even size and the vertices have very similar degrees[4]. These shortcomings lead to misleading benchmark results for algorithms that are not suited to work on for example graphs with clusters of very different sizes.

To improve on these shortcomings, another group of random graphs was introduced in 2009 by Lancichinetti et al. [2008]. With the implicit assumption that degree and community size in real graphs are guided by power laws, they construct a set of graphs usually denominated as *LFR* graphs where each vertex is given a degree taken from a power law distribution with exponent γ such that the average degree is k. Each vertex keeps a fraction $1-\mu$ of its edges within its own cluster and the remaining fraction μ with vertices outside its cluster. The sizes of clusters are also taken from a power law distribution. The random graphs are then constructed by assigning the vertices randomly to clusters as long as each vertex has less inter-cluster links than the size of the cluster. The graph is then adjusted to make sure the amount of inter-cluster versus intra-cluster links is given according to μ.

The *LFR* graphs are like their predecessors the *GN* graphs a limited benchmark in the sense that they will reward algorithms that are particularly good at clustering graphs that correspond to the particular structure determined by the power distribution of of edges and cluster sizes. These graphs has later been used by Lancichinetti and Fortunato [2009] in a benchmark of 12 clustering algorithms, many of which are mentioned in the following section of this paper. The *LFR* are also used by Aldecoa and Marín [2010] to show that *Surprise* is better correlated with the true clustering than *Modularity* in a series of tests with various algorithms.

B. Girvan Newman clustering

If we look at the problem of clustering a graph as a problem of removing edges until we arrive at a set amount of communities, we need a good measure for what edges should be removed. The Girvan-Newman algorithm (in short *GN*) proposes that we remove edges based on a measure of "betweenness", that is how much a particular edge e_{ij} is situated between clusters. The betweenness of e_{ij} is calculated by counting how many shortest paths between pairs of vertices that run through e_{ij}. In case there is ($k > 1$) shortest paths, each path traversing e_{ij} contribues $\frac{1}{k}$ to the count. If we define σ_{st} as the number of shortest paths between vertex s and t, and $\sigma_{st}(e_{ij})$ as the number of shortest paths between s and t traversing e_{ij} we can write betweenness as

$$C_B(e_{ij}) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(e_{ij})}{\sigma_{st}} \quad (5)$$

[3] See for example Girvan and Newman [2002] and Lancichinetti and Fortunato [2009]

[4] As pointed out by Lancichinetti et al. [2008]

Using this measure we can iteratively construct a dendrogram by calculating all the shortest paths of the graph, then finding the edge with the highest betweenness and finally remove this edge and repeat. A common way to find the best amount of clusters is to stop removing edges when the highest value measure of *Modularity* is obtained. The complexity of this algorithm is $O(n^3)$ in the general case, but it can be calculated slightly faster when the graph is sparse. The algorithm does not perform particularly well on random graphs[5], but it is very simple to implement and requires no additional parameters.

C. Spectral clustering

From spectral graph theory we get the spectral clustering algorithm which in reality is an ensemble of different algorithms all solving the problem of clustering a similarity graph in similar ways. The general idea is to use the eigenspectrum of the graph Laplacian to derive a set of k othonormal vectors that are trivial to cluster using a k-means algorithm [Von Luxburg, 2007, pp. 3]. The parameter k needs to be decided in advance. The differences across spectral clustering algorithms usually lie in how the graph Laplacian is defined. One approach is to define the Laplacian L for a graph G as $L = D - A$. However it has been shown that using the normalized Laplacian yields better results. One way to define the normalized Laplacian as used by Ng et al. [2002] is $L = I - D^{-1/2} A_G D^{-1/2}$.

Spectral clustering suffers from the fact that we need to decide the amount of clusters to output before running the algorithm. This can be avoided by calculating clusterings for 2 up to k groupings and then find the number that optimizes a given measure of quality, for example the measure of *Modularity*. The complexity of calculating the eigenvectors is $O(n^3)$ in the worst case. This can be sped up using the Lanczos method[6], but the worst case complexity is still the same. As for the quality of a solution derived with spectral clustering a lot of research has been made on the subject, but there has not been any definite results. In Lancichinetti and Fortunato [2009] a spectral method was compared to several other algorithms and not found particularly well performing. A big advantage of spectral clustering is that it is simple to implement.

D. An approach from Information Theory

Instead of looking at a graph theoretic measures of clusterness, an alternative idea proposed by Rosvall and Bergstrom [2008] is to turn to Information Theory. The approach they propose is to look at clustering as a problem of how best to encode the path of a random walk. A random walk in a weighted graph is a traversal from vertex to vertex where a step from a vertex v_i to another vertex v_j connected to v_i by the edge e_{ij} happens with a probability p_{ij}. We can calculate p_{ij} as $\frac{w_{ij}}{d_i}$. This means that edges with a high weight are traveled often and vertices with a high degree are visited often. If we choose to characterize the random walk as a walk that most of the time remains within clusters and occasionally goes between clusters we can encode this walk by attributing each cluster a codeword, and then in each cluster reuse the same group of codewords. This way we can derive the average description length of a single step as follows:

$$L(\mathbf{M}) = qH(\mathcal{Q}) + \sum_{i=1}^{m} p^i \ H(\mathcal{P}^i) \tag{6}$$

In this equation $H(\mathcal{Q})$ is the entropy of codewords assigned to the clusters and $H(\mathcal{P}^i)$ is the entropy of the within-cluster movements seen as a string of codewords describing the path. This includes an extra word to designate that we are leaving the cluster. q is the probability that we switch between clusters at any given step, while p is the fraction of within-cluster movements that occur in cluster i plus the probability of exiting cluster i. Optimizing this measure leads us to an optimal clustering. In practice this is done by exploring the space of possible clusters using a breadth first search and refining these results using simulated annealing.

This approach has several positive properties. It doesn't need parameters and is relatively fast with a complexity of $O(m)$ and in Lancichinetti and Fortunato [2009] it is rated as one of the best performing algorithms. However the implementation is relatively involved using several different concepts stitched together. This algorithm is labeled *Infomod* when referenced later.

E. The Louvain Method

The *Louvain* Method was introduced in 2008 by Blondel et al. [2008] and made it possible to feasibly cluster more than 100 million vertices as compared to the few million vertices which had formerly been possible with the fastest methods. The method is based on the maximizing the *Modularity* of a graph iteratively. This is done efficiently because the gain in modularity from moving a vertex v_i to a community C can be calculated easily:

$$\Delta Q = \left[\frac{\sum_{in} + 2k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right] \tag{7}$$

Here as in the definition of *Modularity*, $m = \frac{1}{2} \sum_{ij} A_{ij}$, $k_i = \sum_j A_{ij}$ and $k_{i,in}$ is the sum of weights of the edges from v_i to vertices in C. $\sum_{in}$ is the sum of the weights of the edges of C while $\sum_{tot}$ is the sum of all edges with at least one vertex in C. The algorithm works by assigning each vertex to its proper cluster and then for each iteration reassign vertices to neighboring communities if it increases the *Modularity*. These iterations are carried out until there are no reassignments in which case an initial clustering has been reached. For graphs consisting of a large number of vertices, the algorithm can now be repeated, treating each of the resulting clusters as a vertex and repeating the same procedure.

In terms of speed and simplicity the *Louvain* method is one of the best methods available today. It has a computational

[5]According to Lancichinetti and Fortunato [2009]

[6]See for example Golub and Van Loan [1996]

complexity of $O(m)$ and is very simple to implement. What's more, there is no need to specify how many clusters we would expect and the individual cluster sizes have shown to be sensible in real world trials. As for the quality of the solution it compares favorably to other clustering methods in the comparison on *LFR* graphs made by Lancichinetti and Fortunato [2009].

F. Potts Model Approach

Inspired by the *Louvain* Method an algorithm based on Potts model was introduced by Ronhovde and Nussinov [2009]. Instead of working with the *Modularity* it is based on Potts Model Hamiltonian which is defined as follows:

$$\mathcal{H}(\{c\}) = \frac{1}{2}\sum_{i \neq j}(A_{ij} - \ \gamma \mathbf{1}_{\{A_{ij}=0\}}\delta(c_i, c_j) \tag{8}$$

$\delta(c_i, c_j)$ is the Kroneker Delta and is 1 only if the community assignment of v_i, c_i is equal to the community assignment of v_j, c_j. γ is a parameter specified before the algorithm is run. As with the *Louvain* method each vertex is assigned to its proper cluster and then iteratively reassigned to neighboring clusters based on the above measure. Different from the *Louvain* method, this assignment is done several times and the best cluster is picked out. Given the partitions A and B this is done with measures on the entropy ($H(A)$ and $H(B)$), the mutual information ($I(A, B)$), the normalized mutual information ($I_N(A, B)$) and the variation of information ($V(A, B)$). For the graph they are defined as follows:

$$H(A) = -\sum_{k=1}^{q_A}\frac{n_k}{n}log\frac{n_k}{N} \tag{9}$$

$$I(A, B) = \sum_{i=1}^{q_A}\sum_{j=1}^{q_B}\frac{n_{ij}}{N}\ log\frac{n_{ij}N}{v_i v_j} \tag{10}$$

$$I_N(A, B) = \frac{2I(A, B)}{H(A) + H(B)} \tag{11}$$

$$V(A, B) = H(A) + H(B) - 2I(A, B) \tag{12}$$

Here q_A is the amount of clusters in A where as n is the total amount of vertices and n_k the total amount of vertices in cluster k of partition A. n_{ij} is the amount of vertices shared by cluster i of partition A and cluster j of partition B. For each run, the gamma parameter is adjusted and another clustering is calculated. Then the best clustering is picked by looking at the regions of the graph that are strongly correlated measured by high I_N and low V. The algorithm performs well according to Lancichinetti and Fortunato [2009] and is very fast with a complexity of $O(m^\beta logn)$, $\beta \approx 1.3$. It is however complex to implement and requires that several model parameters are supplied. In the following this algorithm is referred to as *Potts*.

G. An overview

As seen in table I, the differences between the fastest and slowest algorithms are vast. With more than a few thousand vertices, calculating the spectral clustering or using the Girvan-Newman method will be unfeasible on most hardware. The quality of each algorithm as measured by benchmarking them on the *LFR* graphs show that the three best algorithms, *Louvain*, *Potts* and *Infomap* perform well on this type of random graphs, with *Infomap* performing better on larger graphs[7].

Author(s)	Year	Label	Order
Girvan and Newman	2002	*GN*	$O(n^3)$
Ng & al.	2002	*Spectral*	$O(n^3)$
Rosvall and Bergstrom	2008	*Infomap*	$O(m)$
Blondel & al.	2008	*Louvain*	$O(m)$
Ronhovede and Nussinov	2009	*Potts*	$O(m^\beta logn)$

TABLE I: Complexity Overview

III. Implementation

To get an understanding of the advantages and limitations of these algorithms, I have undertaken to implement two of them in order to cluster the conference articles. The choice of algorithms have been based on the four criteria mentioned in the introduction (*speed, quality, simplicity, no parameters*) with a special emphasis on simplicity. Simplicity is important because moving from a simple solution to a more advanced algorithm often is relatively straight forward if the two algorithms are related. With this in mind it is better to start out with a sub-optimal algorithm that works, than a better algorithm containing parts that aren't well understood and tested. Additionally a simple baseline solution makes it easy to test more advanced algorithms to check if they get similar results and if they might improve the *Modularity* or *Surprise*. Another factor that was important in the decision of what algorithm to implement was the actual real world usage of the algorithm. A strong preference is given to algorithms that have proven themselves and are being used in real world solutions all ready.

With this in mind I have implemented spectral clustering and the Louvain algorithm. They are both algorithms with wide usage outside of academia, and while spectral clustering has been surpassed in terms of speed and quality, it still stands as a nice baseline for measuring the success of later algorithms.

A. Implementing spectral clustering

Spectral Clustering is implemented based on the algorithm for normalized spectral clustering according to Ng et al. [2002] as shown and explained in [Von Luxburg, 2007, p. 7]. The algorithm is implemented using the *Breeze*[8] linear algebra library for *Scala*[9]. The code can be found on Github[10] in the branch *Cluster*[11]. It is adapted to work with the scala based project for parsing and measuring conference articles which can also be found on github[12]. Based on the conference

[7]A much more detailed analysis of these results can be found in Lancichinetti and Fortunato [2009]

[8]http://www.scalanlp.org/

[9]http://www.scala-lang.org

[10]https://github.com/Traill/papers-web/tree/cluster/src/paper

[11]Look at Cluster.scala and spectral.scala

[12]https://github.com/Traill/papers-web/tree/master/

Algorithm	Approx. Running Time [15]	Modularity[16]
Spectral Clustering	1-2 hours	0.43
Louvain	20-30 seconds	0.36

TABLE II: Comparison of Implemented algorithms

Partition Size	2	10	18	26	34	38
Modularity	0.43	0.32	0.31	0.30	0.31	0.31

TABLE III: Modularity for spectral clustering depending on partition size

articles I obtain a dense similarity graph on which the spectral clustering is calculated for $k-1$ different amount of clusters, going from 2 to k, $k < n$, usually $n/10$.

Since part of the project is an interactive website with a browsable graph, a simple tool was designed to dynamically load a partition of the graph from a server. Upon loading this partition the graph would be partitioned accordingly and graphically animated to reorganize into the given clusters.

B. Implementing Louvain clustering

The Louvain method is implemented by creating a tree-structure with leafs containing articles and communities containing either a leaf or another community. This structure ensures that we can recursively assign communities to other communities to facilitate running the Louvain algorithm on the found clusters, something that was proposed in Blondel et al. [2008] when dealing with large graphs. The similarity graph obtained from the conference papers is pruned for edges so each vertex retains only the four highest weighted links before calculating the clustering. While this doesn't make a difference for the running time of the spectral clustering, it drastically reduces the running time of the Louvain method.

The same modifications to the web application mentioned above were modified to accommodate Louvain clustering as well. The code can be found on Github[13] in the branch *Cluster*[14].

IV. Results and discussion

In table II the typical running time and *Modularity* of *Spectral* clustering and the *Louvain* method can be seen as measured on the 640 articles from the 2012 ISIT, a conference dealing with with topics in information theory. In terms of *Modularity* The table doesn't show the whole truth as can be seen in table III. As the partition size for spectral clustering is increased, the *Modularity* goes down at first and then stabilizes. The author speculates that spectral clustering's tendency to partition a graph evenly comes at a disadvantage when the graph contains clusters of varying sizes. It is possible that the known bias of the *Modularity* measure towards bigger clusters plays a role in this behaviour as well.

When evaluating these results it is important to keep in mind that both the *Modularity* and the results of Lancichinetti and Fortunato [2009] are based on implicit assumptions about the structure of the clusters in the graph used. In order to best apply their results it makes sense to have an idea of the statistical properties of the clusterings of the graph. This in turn is difficult to obtain without using an algorithm to obtain a clustering, something that implicitly introduces bias. This chicken and the egg problematic is a hard to escape from as long as we don't construct a model for how we expect the graph to behave and evaluate the clustering algorithms based on these terms. This however is beyond the scope of this paper.

Modularity aside the decrease running time of the *Louvain* method and the lack of partition size parameter makes it a lot more fitting choice than *spectral* clustering. As for the other candidates that shows promise, in particular *Potts* and *Infomap* they seem at the current stage not to provide enough of an advantage in terms of a higher quality clustering to justify the increased implementation difficulties.

[13]https://github.com/Traill/papers-web/tree/cluster/src/paper

[14]Look at Louvain.scala

[15]Approx. Running Time is on a standard laptop with 8gb ram and 2.2Ghz dual core processor for 640 articles

[16]For Spectral clustering the maximum modularity is given for all partition sizes

References

S.E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

M.E.J. Newman. Detecting community structure in networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):321–330, 2004.

W.W. Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, pages 452–473, 1977.

M. Girvan and M.E.J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

A. Lancichinetti and S. Fortunato. Community detection algorithms: a comparative analysis. *Physical Review E*, 80 (5):056117, 2009.

U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On finding graph clusterings with maximum modularity. In *Graph-Theoretic Concepts in Computer Science*, pages 121–132. Springer, 2007.

V. Arnau, S. Mars, and I. Marín. Iterative cluster analysis of protein interaction data. *Bioinformatics*, 21(3):364–378, 2005.

R. Aldecoa and I. Marín. Deciphering network community structure by surprise. *PloS one*, 6(9):e24195, 2011.

A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, 2008.

R. Aldecoa and I. Marín. Jerarca: Efficient analysis of complex networks using hierarchical clustering. *PloS one*, 5(7): e11585, 2010.

A.Y. Ng, M.I. Jordan, Y. Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

G.H. Golub and C.F. Van Loan. *Matrix computations*, volume 3. Johns Hopkins University Press, 1996.

M. Rosvall and C.T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10): P10008, 2008.

P. Ronhovde and Z. Nussinov. Multiresolution community detection for megascale networks by information-based replica correlations. *Physical Review E*, 80(1):016109, 2009.