An Efficient Spectral Algorithm for Network Community Discovery and Its Applications to Biological and Social Networks

Jianhua Ruan and Weixiong Zhang
Department of Computer Science and Engineering
Washington University in St Loius
One Brookings Dr, St Louis, MO 63130
{jruan,zhang}@cse.wustl.edu

Abstract

Automatic discovery of community structures in complex networks is a fundamental task in many disciplines, including social science, engineering, and biology. Recently, a quantitative measure called modularity (Q) has been proposed to effectively assess the quality of community structures. Several community discovery algorithms have since been developed based on the optimization of Q. However, this optimization problem is NP-hard, and the existing algorithms have a low accuracy or are computationally expensive. In this paper, we present an efficient spectral algorithm for modularity optimization. When tested on a large number of synthetic or real-world networks, and compared to the existing algorithms, our method is efficient and and has a high accuracy. In addition, we have successfully applied our algorithm to detect interesting and meaningful community structures from real-world networks in different domains, including biology, medicine and social science. Due to space limitation, results of these applications are presented in a complete version of the paper available on our website (`http://cse.wustl.edu/~jruan/`).

1 Introduction

The study of complex networks has become a fast growing subject in many disciplines, including physics, biology, and social science. At least part of the reason can be attributed to the discovery that real-world networks from totally different sources can share surprisingly high similarities in their topological properties, such as the power-law degree distributions and high clustering coefficients. (See [1, 14] for reviews.)

One of the key properties in complex networks that have attracted a great deal of interest recently is the so-called community structures, i.e. relatively densely connected sub-networks [15]. Community structures have been found in social and biological networks, as well as technological networks such as the Internet and power grid. Automatically discovering such structures is fundamentally important for understanding the relationships between network structures and functions, and has many practical applications. For example, identifying communities from a collaboration network may reveal scientific activities as well as evolution and development of research areas [12], while detecting hidden communities on the World Wide Web may help prevent crime and terrorism [2].

To design effective community discovery algorithms, Newman and Girvan [16] proposed a quantitative measure, called modularity (Q), to assess the quality of community structures, and formulated community discovery as a optimization problem. Since optimizing Q is a NP-hard problem, several heuristic methods have been developed, as surveyed in [4]. The fastest algorithm available uses a greedy strategy and suffers from poor quality [3]. A more accurate method is based on simulated annealing, which requires a prohibitively long running time on large networks [11]. Several spectral algorithms have been developed, which have relatively good performance, but still inefficient for large networks [21, 15].

In this paper, we propose a spectral algorithm that is effective in finding high quality communities as well as efficient on large networks. The algorithm adopts a recursive strategy to partition networks while optimizing Q. Unlike the existing algorithms, our method is a hybrid of direct k-way partitioning and recursive 2-way partitioning strategies [21, 15]. We evaluate our algorithm on a large number of synthetic and real-world networks. The results show that the algorithm is more efficient and more accurate than a recursive 2-way partitioning method. Compared to a direct k-way partitioning method, our algorithm is much more efficient, while having a comparable accuracy.

The paper is organized as follows. In Section 2, we introduce some basic concepts, notations, and previous works.

In Section 3, we describe our algorithm and its complexity, and discuss several related methods. We present experimental results in Section 4, and conclude in Section 5.

2 Preliminaries

2.1 Spectral graph partitioning

Let $G = (V, E)$ be a network of n vertices in V and m edges in E. Let $A = (A_{ij})$ be the adjacency matrix of G. A graph partitioning problem is to find two or more vertex subsets of nearly equal sizes, while minimizing the number of edges cut by the partitioning [8]. Known to be NP-hard, the problem exists in many real applications, such as circuit design and load balancing in distributed computing. Many heuristic methods have been developed for the problem, among which spectral methods have received much attention and are the most popular.

Spectral graph partitioning is in fact a family of methods. These methods depend on the eigenvectors of the Laplacian matrix or its relatives of a graph. Depending on the way they partition a graph, spectral methods can be classified into two classes. The first class uses the leading eigenvector of a graph Laplacian to bi-partition the graph. The second class of approaches computes a k-way partitioning of a graph using multiple eigenvectors. We briefly review some representatives of these two classes of algorithms below.

Let D be the diagonal degree matrix of A, i.e. $D_{ii} = \sum_j A_{ij}$. $L = D - A$ is called the Laplacian matrix of G. Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the eigenvalues and $\mu_1, \mu_2, \cdots, \mu_n$ be the corresponding eigenvectors for the generalized eigenvalue problem $L\mu = \lambda D\mu$. It can be shown that $\lambda_1 = 0$, and $\mu_1 = \mathbf{1}$, a vector with all ones.

Given the above notation, a representative of bi-partitioning, the *SM* algorithm [19], works as follows. (1) Compute μ_2, the second smallest generalized eigenvector of L. (2) Conduct a linear search on μ_2 to find a partition of the graph to minimize a normalized cut criterion [19]. It has been shown that when certain constraints are satisfied, the *SM* algorithm can reach the optima of normalized cuts [19]. To find more than two clusters, the *SM* algorithm can be applied recursively.

The most popular algorithm in the second class, the *NJW* algorithm [17], finds a k-way partition of a network directly as follows, where k is given by the user. (1) Compute the k smallest generalized eigenvectors of L and stack them in columns to form a matrix $Y = [\mu_1, \mu_2, \cdots, \mu_k]$. (2) Normalize each row of Y to have unit length. (3) Treat each row as a point in R^k, and apply standard k-means algorithm (or any other geometric clustering algorithm) to group them into k clusters.

2.2 Modularity and community structures

Given a partition of a network, Γ^k, which divides its vertices into k communities, the modularity is defined as $Q(\Gamma^k) = \sum_{i=1}^{k}(e_{ii}/c - (a_i/c)^2)$, where e_{ii} is the number of edges with both vertices within community i, a_i is the number of edges with one or both vertices in community i, and c is the total number of edges [16]. Therefore, the Q function measures the fraction of edges falling within communities, subtracted by what one would expect if the edges were randomly placed. A larger Q value means stronger community structures. If a partition gives no more within-community edges than would be expected by chance, the modularity $Q \leq 0$. For a trivial partitioning with a single community, $Q = 0$. It has been observed that most real-world networks have $Q > 0.3$ [16].

The Q function provides a good quality measure to compare different community structures. Several algorithms have been developed to search for community structures by looking for the division of a network that optimizes Q (see [4] for a survey). White and Smyth proposed a spectral algorithm (*WS*), which is effective on small networks [21]. They show that, when the number of communities k is given, the optimization of Q is equivalent to an eigen decomposition problem, if relaxing the discrete membership constraint [21]. Therefore, they directly applied a k-way spectral graph partitioning algorithm for this purpose. To automatically determine the number of communities, the spectral algorithm is executed multiple times, with k ranging from the user defined minimum K_{min} to maximum K_{max} number of communities. The k that gives the highest Q value is deemed the most appropriate number of communities. A slightly modified version of the *WS* algorithm is as follows. (1) For each k, $K_{min} \leq k \leq K_{max}$, apply *NJW* to find a k-way partition, denoted as Γ_k. (2) $k^* = \arg\max_k Q(\Gamma_k)$ is the number of communities, and $\Gamma^* = \Gamma_{k^*}$ is the best community structure.

While the *WS* algorithm is effective in finding good community structures, it scales poorly to large networks, because it needs to execute k-means up to K_{max} times. Without any prior knowledge of a network, one may overestimate K_{max} in order to reach the optimal Q. For sparse networks, K_{max} can be linear in the number of vertices in the worst case, making it impractical to iterate over all possible k's for large networks.

3 The *Kcut* algorithm

In order to develop a method that scales well to large networks while retaining effectiveness in finding good communities, we may take the strategy used in the *SM* algorithm, i.e., to recursively divide a network into smaller ones. However, two issues remain. First, when should the algorithm halt, or in other words, how do we decide whether

a (sub)network should be partitioned? Since our goal is to find a partition with a high modularity, we can test whether the Q value increases after the partition. If no partition can improve the modularity, the (sub)network should not be divided. Second, it has been empirically observed that if there are multiple communities, using multiple eigenvectors to directly compute a k-way partition is better than recursive bi-partitioning methods [17]. Here, we propose an algorithm that is a unique combination of recursive partitioning and direct k-way methods, which will achieve the efficiency of a recursive approach, while also having the same accuracy as a direct k-way method.

We follow a greedy strategy to recursively partition a network to optimize Q. Unlike the existing algorithms that always seek a bi-partition, we adopt a direct k-way partition strategy as in the *WS* algorithm. Briefly, we compute the best k-way partition with $k = 2, 3, \cdots, l$ using the NJW algorithm, and select the k that gives the highest Q value. Then for each subnetwork the algorithm is recursively applied. To reduce the computation cost, we restrict l to small integers. As we will shown in experiments, the algorithm with l as small as 3 or 4 can significantly improve modularity over the standard bi-partitioning strategy. Furthermore, the computation cost is also reduced with a slightly increased value of l compared to bi-partitioning.

Given a network G and a small integer l that is the maximal number of partitions to be considered for each subnetwork, our algorithm *Kcut* executes the following steps.

1. Initialize Γ to be a single cluster with all vertices, and set $Q = 0$.
2. For each cluster P in Γ,
   (a) Let g be the subnetwork of G containing the vertices in P.
   (b) For each integer k from 2 to l,
      i. Apply *NJW* to find a k-way partitioning of g, denoted by Γ_k^g,
      ii. Compute new Q value of the network as $Q'_k = Q(\Gamma \bigcup \Gamma_k^g \setminus P)$.
   (c) Find the k that gives the best Q value, i.e., $k^* = \arg\max_k Q'_k$.
   (d) If $Q'_{k^*} > Q$, accept the partition by replacing P with $\Gamma_{k^*}^g$, i.e., $\Gamma = \Gamma \bigcup \Gamma_{k^*}^g \setminus P$, and set $Q = Q'_{k^*}$.
   (e) Advance to the next cluster in Γ, if there is any.

The inner loop, step 2(b), is similar to the first step of the *WS* algorithm, except that in 2(b)(ii) we compute the modularity of the whole network G, which is different from the modularity $Q(\Gamma_k^g)$. On the other hand, we do not need to iterate over all communities in the network to re-compute Q. From the definition of Q in Section 2.2, the contribution of each community towards Q is independent of the other communities. Therefore, after g is partitioned, Q can be efficiently updated with the communities that have just been created in g. At step 2(c), we decide the best way to partition g that can improve Q the most. This step turns out to be crucial in identifying globally good community structures with high Q values. At step 2(d), we test if partitioning g can contribute positively towards Q, and the partition is accepted only if Q increases. When the algorithm terminates, no communities can be further created to improve Q, thus Γ contains the best community structure.

3.1 Computational complexity

We first review the computational complexity of the *WS* algorithm, since the inner loop of *Kcut* is simply the *WS* algorithm, except that the computation of Q is slightly different. The *WS* algorithm contains two major components: computing eigenvectors and executing k-means to partition the network. Note that although *WS* calls *NJW* multiple times, the eigen problem needs to be solved only once to obtain all K_{max} eigenvectors. To compute eigenvectors, we used the *eigs* function in MATLAB, which has a time complexity in $O(mKh + nK^2h + K^3h)$, where m and n are respectively the numbers of edges and vertices of the graph, $K = K_{max}$ is the number of eigenvectors to be computed, and h is the number of iterations for *eigs* to converge [21]. Since $K < n$, the running time of $eigs$ can be simplified to $O(mKh + nK^2h)$. Second, we adopted a fast k-means algorithm [6] in our implementation, which takes approximately $O(nKe)$ time, where e is the number of iterations for k-means to converge. Since k-means is called K times, the total running time is $O(mKh + nK^2h + nK^2e)$, where the first two terms are for $eigs$ and the last term is for k-means. Assuming e and h constants, the overall time complexity of *WS* is $O(mK + nK^2)$, which can be close to $O(n^3)$, since the maximal number of communities for a sparse network may be linear in n.

The running time of *Kcut* depends on the depth of the recursive calls. In the worst case, the partitions can be highly imbalanced, and the depth of the recursion is merely the number of partitions produced, K. A more practical estimate, however, is the average depth, which is close to $\log_l K$, where l is the maximal number of partitions considered by *NJW*. Therefore, the running time taken by *eigs* can be estimated to be $O((mlh + nl^2h)\log_l K)$, which can be further simplified to $O(mlh \log_l K)$, since l is small and therefore in general $m > nl$. Similarly, the average-case running time taken by k-means is $O(nl^2e \log_l K)$, and the total complexity is given by $O((mlh + nl^2e)\log_l K)$.

Our experimental results show that for large networks and small values of l, the time taken by $eigs$ dominates, giving an overall time complexity in $O(mlh \log_l K) = O(mh \ln K \frac{l}{\ln l})$ for *Kcut*. Therefore, assuming h is a con-

stant, also given that l is small and $K = O(n)$, the total complexity is $O(m \log n)$, which is much smaller than the $O(n^3)$ running time of the *WS* algorithm. An important observation from the analysis is that the total running time of *Kcut* is not a monotonically increasing function of l. Analytically, the minimum value of $l/\ln l$ is achieved at $l = 3$. Empirically, we observed that *Kcut* is most efficient with $l = 3$ to 5 (see Section 4.2).

The memory complexity of both algorithms is $O(m)$, linear to the number of edges.

3.2 Related methods

Besides our algorithm and the *WS* method, several other algorithms have also been developed for identifying communities by modularity optimization. Newman proposed an algorithm that is based on recursive spectral bi-partitioning [15]. The algorithm computes the leading eigenvector of a so-called modularity matrix, and divides the vertices into two groups according to the signs of the elements in the eigenvector. The algorithm runs recursively on each subnetwork, until no improvement to Q is possible. Compared to our method, this algorithm is faster for small networks, since no k-means is performed. On the other hand, the modularity matrix is very dense, with almost no zero entries. Therefore, the algorithm takes $O(n^2)$ memory even for sparse networks, in contrast to $O(m)$ for our method. Furthermore, the algorithm takes $O(n^2 \log n)$ running time, therefore, it does not scale well to large networks. Importantly, we will show that by combining k-way partitioning with a recursive method, *Kcut* usually achieves higher modularity than the Newman method.

There are also several methods that are not spectral-based. The edge betweenness algorithm [9] and the extremal optimization algorithm [5] are known to be very slow, with $O(n^3)$ and $O(n^2 \log^2 n)$ running time, respectively. Another greedy approach, the *C*NM algorithm [3], has approximately the same time complexity ($O(m \log^2 n)$) as our method, but the communities returned often have poor quality [15].

4 Evaluation

We now evaluate our algorithm on a variety of networks and compare it with three existing algorithms that were mentioned in Section 3.2: the *WS* algorithm, the *CNM* algorithm, and the Newman's algorithm (*NM*). In what follows, the results of our algorithm are denoted by *K-2*, *K-3*, $\cdots$, for $l = 2, 3, \cdots$. Note that Newman suggested in [15] a refining step to improve Q after the initial partitioning. To make a fair comparison, this refining step was omitted in our study, since in theory the same strategy can be applied to any other algorithm as well.

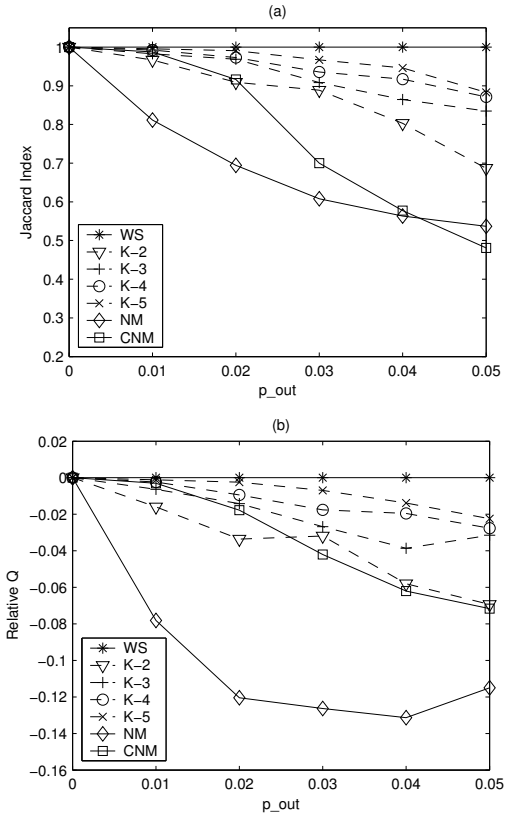

Figure 1. Results on computer-generated networks. $Q_{relative} = Q_{discovered} - Q_{true}$.

4.1 Computer-generated networks

To evaluate *Kcut*, we first tested it on computer-generated networks with artificially embedded community structures. Each network had 256 vertices forming 8 communities of equal sizes. Edges were randomly placed with probability p_{in} between vertices within the same community and with probability p_{out} between vertices in different communities. We varied p_{in} from 0.8 to 0.3, representing networks with dense to sparse communities. For each p_{in}, we varied p_{out} from 0 to $\frac{p_{in}}{10}$ with an increment of $\frac{p_{in}}{50}$. For each pair of (p_{in}, p_{out}), we generated 100 networks and clustered them with *WS* ($K_{min} = 2$, $K_{max} = 15$), *Kcut* ($l = 2, 3, 4$ and 5), and *NM* algorithms. To measure the accuracy of the results, we computed the Jaccard Index [20], which is roughly the percentage of within-community edges that were predicted correctly. The Jaccard Index between the true community structure (Γ) and predicted community structure (Γ') is defined as

$$J(\Gamma, \Gamma') = \frac{|S(\Gamma) \cap S(\Gamma')|}{|S(\Gamma) \cup S(\Gamma')|}, \quad (1)$$

where $S(\Gamma)$ and $S(\Gamma')$ are the sets of within-community vertex pairs in Γ and Γ', respectively.

Table 1. Q values for real-world networks.

Network	n	m	K^*	K_{max}	Q							
					WS	*K-2*	*K-3*	*K-4*	*K-5*	*NM*	*CNM*	Best
Karate	34	78	4	8	0.420	0.390	0.420	0.420	0.420	0.393	0.383	0.420 [21]
Football	115	613	10	20	0.602	0.524	0.600	0.596	0.590	0.493	0.577	
Jazz	198	5484	4	8	0.439	0.444	0.444	0.439	0.439	0.394	0.439	0.445 [5]
PPI	1440	6223	133	200	0.362	0.332	0.344	0.348	0.364	0.341	0.337	
Internet	3015	5156	52	100	0.604	0.594	0.600	0.601	0.601	0.524	0.620	
Physicists	27519	60793	-	600	-	0.734	0.738	0.739	0.743	-	0.659	0.723 [15]

K_{max}: maximal number of communities for *WS*. K^*: number of communities returned by *WS*. The last column are the best Q values achieved by existing methods in the literature, and references to the methods.

Table 2. Total CPU time (seconds).

Network	*WS*	*K-2*	*K-3*	*K-4*	*K-5*	*NM*	*CNM**
Karate	0.3	0.3	0.3	0.3	0.4	0.1	0.02
Football	1.1	0.7	0.6	0.7	1.1	0.3	0.04
Jazz	0.5	0.6	0.7	0.7	0.9	0.3	0.06
PPI	8k	40	26	31	23	58	0.8
Internet	3k	37	27	22	23	172	63
Physicists	-	6k	3k	2k	2k	-	283

*A significant difference between *CNM* and the other algorithms here is that *CNM* was implemented in C, while all the other algorithms compared here were implemented in MATLAB m-files.

Fig. 1(a) shows the Jaccard Index as a function of p_{out} for $p_{in} = 0.5$. Results for other values of p_{in} or using other types of accuracy measurement are similar (data not shown). The *WS* algorithm, which explicitly searches over all k's, has the best accuracy. On the other hand, *Kcut* with large l values can better approximate *WS* than with small l values. Moreover, as shown in Fig. 1(b), the Q values achieved by the algorithms match their accuracies: *WS* has the highest modularity, followed by *K-5*, *K-4*, ..., and the Newman algorithm at last. A third measure, the number of times an algorithm predicted k correctly, also shows that *WS* > *K-5* > $\cdots$ > *K-2* > *NM* (data not shown). The *CNM* algorithm has an accuracy similar to *K-2* for smaller p_{out}, but its accuracy drops significantly when p_{out} increases.

4.2 Real-world networks

We further tested our method on several real-world networks. These include an acquaintance network in a Karate club [22], the opponent network of American NCAA Division I college football teams in the year 2000 [9], a coperforming network of Jazz Bands [10], a protein-protein interaction network of *E. coli* [18], the Autonomous Systems topology of the Internet [7], and a collaboration network of physicists [13]. As shown in Table 1, the *WS* algorithm usually returns community structures with the highest Q value. Although *Kcut* with $l = 2$ often performed poorly, *Kcut* with $l \geq 3$ can usually achieve Q values as good as that by *WS*, whereas with a much reduced running time. Moreover, for the three networks (Karate, Jazz, Physicists) that have been analyzed by others, *Kcut* can find modularity values that are comparable to or better than the best known ones. The *NM* algorithm (without the refining step) and the *CNM* algorithm usually have much worse accuracy comparing to *WS* and *Kcut*. The *WS* and Newman algorithms failed to finish on the physicist network, due to their excessive running time or memory usage.

In addition, the communities returned by *Kcut* are often very close to the known communities if they are available. For example, for the Karate club network, *Kcut* precisely predicted the actual separation of the club caused by a dispute among its members [9]. For the football network, *Kcut* correctly revealed the official NCAA conference structure of the football teams [9], except for a few teams that do not belong to any conference. Because of space limit, we omit the detailed results here.

4.3 Running time

Table 2 shows the running time of the four algorithms on the six real-world networks. Table 3 shows the time spent on *eigs* and k-means by *WS*, *Kcut* and *NM*. *CNM* is based on a different rationale and does not have these two components. As shown in Table 2, although *WS* is efficient for small networks of up to a few hundred of vertices, it is very inefficient on large networks. The *Kcut* algorithm, on the other hand, can handle networks of several thousand of vertices in less than half minute. It appears in Table 2 that *CNM* is the most efficient, especially for small networks. At least part of the reason is that *CNM* was implemented in the C language, while the other three algorithms were all implemented in MATLAB M-files. M-files are interpreted at run time, and therefore have higher overhead.

Also observe that *Kcut* is often faster with $l = 3, 4, 5$ than with $l = 2$. Based on the analysis in Section 3.1, the time *Kcut* spent in *eigs* is approximately linear to $l/\ln l$, which reaches its minimum at $l = 3$. In contrast, the time *Kcut* spent on k-means is proportional to $l^2/\ln l$, which is monotonically increasing for $l \geq 2$. The experimental results in Table 3 partially support the theoretical analysis. For large networks, the total running time of *Kcut* is dominated by *eigs*. Therefore, *Kcut* can take advantage of a slightly increased l to reduce its running time. When l be-

Table 3. CPU time (seconds) for program components.

Network	*WS*		*K-2*		*K-3*		*K-4*		*K-5*		*NM*
Karate	0.08	0.16	0.16	0.11	0.1	0.11	0.1	0.2	0.08	0.22	0.11
Football	0.1	0.81	0.33	0.22	0.21	0.23	0.29	0.36	0.23	0.7	0.28
Jazz	0.11	0.29	0.26	0.06	0.31	0.23	0.25	0.3	0.25	0.5	0.25
PPI	14	7857	34	3	20	4	18	7	11	8	53
Internet	12	2892	31	3	19	5	14	6	11	9	150
Physicists	-	-	5353	79	2451	109	1473	152	1473	170	-

For *WS* and *Kcut*, the first and second columns are the time taken by *eigs* and k-means, respectively. The last column is the time spent on *eigs* in the Newman algorithm.

comes too large, however, the running time of both components increases, and the efficiency of *Kcut* may degrade.

5 Conclusions

We have developed a fast algorithm, *Kcut*, for identifying community structures in large networks. Our approach is based on a greedy optimization of a modularity function Q. Unlike previous methods, *Kcut* is not restricted to bi-partitions, but considers all k-way partitions for a small range of k. We have found that this relaxation not only improves the quality of the identified communities, but also increases the efficiency of the algorithm. We have demonstrated the performance of our method on a variety of random and real-world networks. Compared to the existing approaches, *Kcut* can find better Q values than other greedy approaches, and has an accuracy comparable to that of a much slower exhaustive search method.

In addition, we have applied our method to several real problems in several different fields: biology, medicine and social science. In all cases, our algorithm is able to detect significant and meaningful community structures that provide important information about the systems of interest. A longer version of this paper including details of these results is available on our website (`http://www.cse.wustl.edu/~jruan`).

Acknowledgments

This research was supported in part by NSF grants ITR/EIA-0113618 and IIS-0535257 and a grant from Monsanto Company to W.Z.

References

[1] R. Albert and A. Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47, 2002.

[2] J. Baumes, M. Goldberg, M. Magdon-Ismail, and W. Wallace. Discovering hidden groups in communication networks. 2nd NSF/NIJ Symposium on Intelligence and Security Informatics., 2004.

[3] A. Clauset and et. al. Finding community structure in very large networks. *Physical Review E*, 70:066111, 2004.

[4] L. Danon, J. Duch, A. Diaz-Guilera, and A. Arenas. Comparing community structure identification. *J. Stat. Mech.*, page P09008, 2005.

[5] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72:027104, 2005.

[6] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, pages 147–153, 2003.

[7] M. Faloutsos and et. al. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[8] P. Fjallstrom. Algorithms for graph partitioning: A survey. Linkoping Electron. Atricles in Comput. and Inform. Sci., 1998.

[9] M. Girvan and M. Newman. Community structure in social and biological networks. *Proc Natl Acad Sci U S A*, 99:7821–6, 2002.

[10] P. Gleiser and L. Danon. Community structure in jazz. *Advances in Complex Systems*, 6:565, 2003.

[11] R. Guimera and L. N. Amaral. Functional cartography of complex metabolic networks. *Nature*, 433:895–900, 2005.

[12] J. Hopcroft, O. Khan, B. Kulis, and B. Selman. Tracking evolving communities in large linked networks. *Proc Natl Acad Sci U S A*, 101 Suppl 1:5249–53, 2004.

[13] M. Newman. The structure of scientific collaboration networks. *Proc Natl Acad Sci USA*, 98:404–409, 2001.

[14] M. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[15] M. Newman. Modularity and community structure in networks. *Proc Natl Acad Sci USA*, 103:8577–82, 2006.

[16] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys Rev E*, 69:026113, 2004.

[17] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856, 2001.

[18] L. Salwinski, C. Miller, A. Smith, F. Pettit, J. Bowie, and D. Eisenberg. The database of interacting proteins: 2004 update. *Nucleic Acids Res*, 32:D449–51, 2004.

[19] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22:888–905, 2000.

[20] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2005.

[21] S. White and P. Smyth. A spectral clustering approach to finding communities in graph. In *SIAM Data Mining*, 2005.

[22] W. Zachary. An information flow model of conflict and fission in small groups. *J. Anthropol. Res.*, 33:452–473, 1993.