AAAI Technical Report FS-12-08
Social Networks and Social Contagion

Block Modeling in Large Social Networks with Many Clusters

Adam Anthony and **Shawn Biesan**
Baldwin Wallace University,
Berea, OH, 44017, USA

Marie desJardins
University of Maryland, Baltimore County,
Baltimore, MD, 21250, USA

Abstract

In this paper, we present an optimized version of the previously developed Block Modularity algorithm (Anthony 2009). The original algorithm was a fast, greedy method that effectively discovered a structured clustering in linked data and scaled very well with the number of nodes and edges. The optimized version is scalable in terms of the model complexity; the technique can now be used effectively to discover thousands of clusters in data sets with hundreds of thousands (and possibly more) nodes and edges. The optimization leads to an improvement of the runtime per iteration from cubic to quadratic with a small increase in the constant factor. The algorithm compares favorably with Karrer and Newman's Degree-Corrected Block Model (DCBM) in both runtime and quality of results.

Introduction

A block model is a representation for clustering objects into groups based upon patterns that occur in the relations between these objects. The structure of a block model is a matrix in which the (i, j)th entry is the number of directed edges from vertices in cluster i to vertices in cluster j. This structure is divided into squares or blocks that represent a cluster's edges. Unlike traditional community detection algorithms, in which objects are separated so that more edges exist within a community than between communities, block models can represent any pattern that arises in the relations between objects, such as bipartite relations, hierarchies, rings, bridges, and other unique aggregate connectivity patterns between groups of vertices. In principle, block modeling could be considered a superior method for clustering in social networks because it can detect any type of pattern, whereas community detection algorithms can only detect one type of pattern.

Community detection algorithms have generally been used extensively on large social networks for a variety of reasons, although there are a number of recurring themes that we have observed anecdotally. First, many researchers interpret an edge as implying "closeness" and a cluster to be a group of tightly packed objects. In this sense, a community detection algorithm is very intuitive. However, in social networks, edges do not necessarily imply any sort of distance and many other patterns arise. For example, in a dating network, one would expect to observe a more-or-less bipartite structure, and in a co-worker network, a hierarchical structure would dominate. While communities of highly connected clusters are generally the dominant pattern in small friendship networks, we believe that in large social networks such as Facebook, what actually exists is a much more complex type of mixed relation that may include friendship, family, co-worker, business, entertainment, and other links all of which are essentially unlabeled and collected into a single set. Each of these different types may have different connectivity characteristics, but overall their combined connectivity pattern may be very different from a basic community structure.

Another limitation of many block modeling algorithms is their overall design and runtime complexity. Many community detection algorithms are very simple to implement and analyze, and the output result is easy to interpret. This simplicity also leads to many fast techniques for optimization that have been widely disseminated (Clauset, Newman, and Moore 2004). In contrast, block modeling algorithms are more likely to be stochastic in nature, and have long execution times (Kemp et al. 2006a; Nowicki and B. 2001; Newman and Leicht 2007; Handcock, Rafferty, and Tantrum 2007).

The block modularity algorithm is both fast and easily optimized using a simple but effective randomized greedy approach and has already been shown (Anthony 2009) to scale well with the number of vertices and edges. Here we introduce a variation that also scales very well to domains that contain a very large number of clusters. This method can be used to find large, complex patterns in even larger data sets, and may reveal new patterns in large social networks, the detection of which has previously been computationally intensive and sometimes intractable.

In this paper, we detail the optimized version of block modularity, *BM-OPT*, which is both fast to compute and reasonably accurate in a variety of circumstances. In particular, we focus on the scalability of BM-OPT on larger datasets when the expected number of clusters is very high. With the large size of many real-world social networks "in the wild" such as Facebook and Twitter, this scalability has become increasingly important. We compare the BM-OPT algorithm to the original block modularity algorithm (BMOD)

(Anthony 2009), as well as to the Degree Corrected Block Model (DCBM) approach by Karrer and Newman (2011).

Background and Related Work

Block models have a long history of application in social network analysis, the extent of which is outside the scope of this paper. Our focus is on the *a priori* discovery of block model structures. The first such approach was *stochastic block modeling* (Nowicki and B. 2001), in which every object has a probability θ_k of being in a cluster k, where θ is a vector of size c, the total number of clusters. Every pair of clusters has an entry η_{kh} in a $c \times c$ matrix, η, where the entry corresponds to the probability that there is an edge going from cluster k to cluster h. Nowicki and Snijders (2001) define the objective function of this method as:

$$\begin{aligned} P(S, C|\theta, \eta) &= P(C|\theta)P(S|C, \eta) \\ &= \theta_1^{m_1} ... \theta_c^{m_c} \Big(\prod_{1 \le k,h \le c} \eta_{kh}^{e(k,h)} (1 - \eta_{kh})^{\bar{e}(k,h)} \Big), \end{aligned} \quad (1)$$

where m_c is the number of objects assigned to cluster c, $e(k, h)$ is the actual number of edges between clusters k and h, and $\bar{e}(k, h)$ is the number of pairs of objects in clusters k and h, respectively, with no edge between them. The $\theta_1 \dots \theta_k$ are the estimated fraction of objects that are in each of the clusters. η_{kh} is estimated as the number of edges in a certain block b_{kh} divided by the size of the block, yielding a Bernoulli probability.

There are a number of ways to implement a stochastic block modeling algorithm using the objective function in Equation 1. The first is Gibbs sampling, which is a prohibitively expensive but very accurate method to maximize the objective function (Kemp et al. 2006b). Simulated annealing is another option that uses the objective function directly in a hill-climbing approach that tends to avoid local maxima. The latter method is much faster, allowing stochastic block modeling to be used on larger datasets.

Karrer and Newman's approach for improving block model algorithms was to extend the stochastic block model to correct for heterogeneity in the degrees of vertices (2011). Let D_i^{out} and D_j^{in} be the total out-degree of cluster i and in-degree of cluster j, respectively. In DCBM, a measure of degree heterogeneity takes the place of the η parameter. Degree heterogeneity, referred to as β, is a $k \times n$ matrix. β_{ki} is defined as the probability that there is an edge from cluster k to object i specifically. Instead of being a Bernoulli probability, the matrix β_{ki} is defined to be

$$\beta_{ki} = \frac{Z_{ki}}{D_k^{out}}, \quad (2)$$

where D_k^{out} is defined as before (as the out-degree of cluster k) and Z_{ki} is the number of edges from cluster k to object i.

Block Modularity

Previously, we developed a simple, greedy approach to block modeling, block modularity (Anthony 2009). Block modularity scales better than the previously mentioned algorithms as the number of vertices and edges increase, both by reducing the number of calculations at each iteration and by reducing the overall number of iterations needed to find a good solution. The approach itself is similar in design to the modularity approach for community detection in directed networks (Leicht and Newman 2008), but extends the method to allow for any type of connectivity pattern.

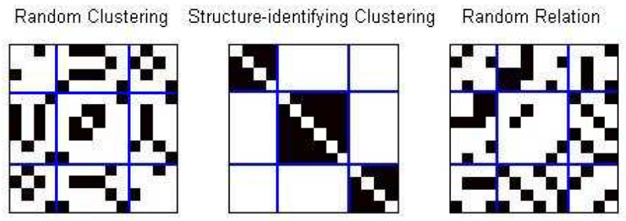

Figure 1: Left: a structured relation clustered randomly. Middle: The same relation clustered to identify a clear community structure. Right: a random relation; any clustering of this relation will have a similar block model.

The core concept of block modularity is that a structured clustering of a group of objects and its relations should be measurably different both from a random clustering and from any clustering of the vertices in a random graph. Figure 1 demonstrates this concept. The first two block model diagrams depict the same vertices and relational links, but one model uses a random clustering, while the second clustering identifies a clear structure. Our measurement is based on the observation that if the clustering was done in a random manner, or if the graph was random, then the edges would be equally distributed among the blocks. The basic approach of our method is to assume that an input relation is highly structured. Under this assumption, a structure-identifying clustering must exist and it can be measurably observed by counting the absolute difference in the number of edges in each block, compared to what we would expect in the random relation. This can be succinctly expressed by the formula:

$$BM(C) = \frac{1}{2|S|} \sum_{i,j} \left| |b_{ij}| - \frac{D_i^{out} D_j^{in}}{|S|} \right|. \quad (3)$$

where b_{ij} is the number of edges between cluster i and cluster j, and D_i^{out} and D_j^{in} are again the total out-degree of cluster i and in-degree of cluster j, respectively. The second term inside the summation is based on the random relation model described by Leicht and Newman (2008) to determine the distribution of edges in a random graph. $\frac{1}{2|S|}$ normalizes the values to the interval [0,1].

If a structure-identifying clustering is known, the above formula can confirm its validity and measure its difference from a random relation. If such a clustering is not known, it can be found using a simple greedy approach. The basic idea is that in each iteration, the vertices are enumerated in a random order, and each vertex in turn is moved to the cluster that maximizes BM(C), holding all other vertices temporarily constant. The key to this algorithm's effectiveness is that the order of the vertices' reassignment is different each time. Since moving one vertex necessarily affects each of its incident neighbors' contribution to the objective score, using the same order at each iteration would mean that items earlier in the list would dominate the eventual clustering.

Each algorithm has its own merits. However, none of them scale particularly well when the number of clusters increases because of the increase in model complexity. Even the original implementation of BMOD falls short because it naively recalculates the full block score on each possible cluster assignment of a vertex. This results in a cubic runtime in terms of the number of clusters, which restricts the number of clusters that BMOD can tractably discover in large data sets. We found that simple calculations can compute the change in score that results from moving a vertex from one cluster to another, without recomputing significant portions of the entire objective function.

Approach

Our focus is on further optimizing block modularity to scale efficiently in the number of clusters by incrementally computing the clustering's score. In order to find the best cluster in which to place a vertex, this local search scoring step must be done for every possible cluster assignment. In this section, we present a more efficient way to calculate this change that reduces the runtime from cubic to quadratic in terms of the number of clusters per iteration.

The summation in Equation 3 can be viewed as a summation over the rows and columns of a matrix, B. Changing the clustering of vertices would result in a change to this matrix, and hence a change to the final $BM(C)$ score. Our focus here is on quantifying the change in the score that results from moving a vertex from one cluster to another. In our previous work, the entire matrix B was essentially recomputed. After many steps of simplification, the partial quantity of $BM(C)$ that is potentially affected by a vertex's reassignment is:

$$\begin{aligned} BMP(C) = & B^{row}_{old} + B^{col}_{old} \\ & + B^{row}_{new} + B^{col}_{new} \\ & - B[old, new] - B[new, old] \\ & - B[old, old] - B[new, new], \end{aligned} \tag{4}$$

where B^{row}_{old} is the sum of row terms in B for the vertex's old cluster assignment, and B^{row}_{new} is the sum of row terms for the new clustering assignment. Similar terms are present for the column totals. The individual matrix positions that are then subtracted are to account for double counting that takes place from the overlapping of rows and columns.

The formula logically follows because of how a block model is defined; row i and column i correspond to a vertex i's edges. When shifting vertex v to a new cluster, we are moving v's row and column(edges) into a new cluster block. The sums will only change in cluster blocks that have gained or lost a row/column. The other blocks' total BMOD score have already been computed, so we do not need to recompute them. Visually speaking, the change in score only affects a single row and column of BM, leaving four quadrants of values that are unaffected by moving the vertex. As the number of clusters increases, the size of these quadrants becomes quite substantial, so avoiding recomputing their values presents a significant cost savings.

Finally, to actually compute the change in score resulting from the transfer of a vertex from one cluster to another, we update the edge counts for the matrix B to reflect the move, recompute $BMP(C)$ and compare it to the previous value of $BMP(C)$. If BMP(C) has increased, the change is retained. If it is not, then B is reverted to the previous version. While simple in nature, this optimization does result in a significant improvement in scalability to the number of clusters. In the next section, we detail our experimental methodology.

Methodology

The two algorithms to be compared with BM-OPT will be Karrer and Newman's DCBM and the original BMOD algorithm. The primary reasons we chose DCBM to compare in this paper are that the underlying heterogeneous random graph model used for DCBM is very similar to the model used for BMOD, and it is one of the most recently published block modeling algorithms. Karrer and Newman also offered a method of network generation, which we used extensively, so it is natural to include their pattern recognition algorithm in the analysis. We show that BM-OPT outperforms DCBM to the point that DCBM could not feasibly finish on the largest data sets, so we also developed a stress test for BM-OPT on a network generated from a realistic degree sequence taken from a Facebook data source (Gjoka et al. 2010).

We compare the algorithms based on quality of results, scalability as the number of nodes increases, and scalability as the number of clusters increases. It is expected that while DCBM will produce higher-quality results, due to its thorough statistical approach, the BM-OPT results will at least be comparable in each case, despite the greedy method that we apply. The original BMOD is not compared in terms of accuracy, since it is equivalent in output to BM-OPT, but slower. Three different connectivity patterns are used so as to show that the algorithms do well in a variety of graph types.

One particularly useful contribution of Karrer and Newman's work was a method for generating synthetic graphs using the Degree Corrected Block Model, which we use in our work to generate synthetic, but naturally realistic, graphs. In order to generate these graphs, the user needs to choose group assignments (g), the expected number of edges between groups (ω), the expected degree of all vertices, and θ (which is calculated once ω is chosen). Their method also introduces a mixing parameter λ, which varies the balance of structure versus randomness in the generated data. λ is on the interval [0,1] where 0 corresponds to a completely random graph and 1 corresponds perfectly to a pre-determined, or planted, structure. To start, they define the expected number of edges from cluster r to cluster s as:

$$\omega_{rs} = \lambda\omega^{planted}_{rs} + (1-\lambda)\omega^{random}_{rs}, \tag{5}$$

where ω^{random} is the expected edge matrix of a random graph and $\omega^{planted}$ is a *planted structure*, which may be defined in a number of ways. The first planted structure we use is the simple community structure defined by

$$\omega^{community} = \begin{pmatrix} k_1 & 0 & 0 & 0 \\ 0 & k_2 & 0 & 0 \\ 0 & 0 & k_3 & 0 \\ 0 & 0 & 0 & k_4 \end{pmatrix}, \qquad (6)$$

where k_r is the sum of all the degrees of vertices in cluster r. Another structure we use is a hierarchical network defined by

$$\omega^{hierarchical} = \begin{pmatrix} k_1 - A - C & A & C \\ 0 & k_2 - A - B & B \\ 0 & 0 & k_3 - B - C \end{pmatrix}. \qquad (7)$$

In this case, the values of k_r are the same as above, but A, B, and C are parameters that can be adjusted to require that edges are generated in between particular clusters, rather than just within communities. For the hierarchical example, we set $A = \frac{1}{4}min(k_1, k_2)$, $B = \frac{1}{4}min(k_2, k_3)$, $C = \frac{1}{10}k_2$. The A, B, and C values may be chosen arbitrarily, provided that $k_1 - A - C, k_2 - A - B, k_3 - B - C$ are non-negative.

We also introduce a *Bridge Network* pattern, which we use to stress test BM-OPT. It is defined as:

$$\omega^{bridge} = \begin{pmatrix} k_1 - A & A & 0 \\ 0 & k_2 - (2 * A) & A \\ 0 & 0 & k_3 - A \end{pmatrix}, \qquad (8)$$

where $A = \frac{1}{2}min(k_1, k_2, k_3)$. A bridge network is a naturally occurring type of connectivity pattern in social networks in which two communities are connected indirectly by a third party, commonly referred to as a bridge.

Once the input values are chosen, we follow Karrer and Newman's suggestion to use a Poisson distribution on the number of edges for each cluster pair r and s, with mean ω_{rs} or $\frac{1}{2}\omega rs$ when r=s. Finally, each side of the edge is connected to a vertex with probability θ_i

We implemented a synthetic network generator based on Karrer and Newman's algorithm. It was coded in C# using the Quickgraph library.[1] We found that the generation algorithm itself can have issues with scalability, so to simplify the computation, in contrast to Karrer and Newman's approach, our generation samples edges with replacement, so that the degree distribution of the vertices may not precisely match the defined values. However, as the number of vertices increases, the expected distribution will be very close to the original. Parallel edges are allowed, but in a large generated network, their occurrence would not be frequent. We omit self-edges because most real networks do not have such edges. Finally, while Karrer and Newman focused on undirected network generation, the model can also be used to generate directed networks; all of the data we present is for directed networks.

[1]QuickGraph version 3.6.61114.0 released November 19 2011: http://quickgraph.codeplex.com/

We created the community and hierarchical networks using similar parameters to those presented by Karrer and Newman. For a more realistic bridge network, which we use as a stress test, we sampled a degree distribution from a subset of a friendship network gathered from Facebook by Gjoka et. al, (2010). The nodes in our stress test had a degree sequence matching that of 100,000 of the nodes in the data. To build the full graph, we connected the edges in a Bridge Structure with 1000 clusters as defined by Equation 8, but repeating the structure in this equation until there are 1000 clusters. The generated graph has 8,391,333 directed edges.

Our experiments focus on both speed and quality of results. The measure of quality is the Normalized Mutual Information (NMI) of the proposed clustering and the actual clustering. NMI measures the information that two random variables share. Here, we use NMI to measure how much information is shared, that is, the similarity between our proposed clustering and the known synthetic clustering.

For the quality tests, each graph and λ combination is tested 10 times and the best result is used. We do this instead of simply averaging the result because the average considers both consistency and goodness, which is at best unnecessary and at worst misleading. Since BM-OPT is a greedy algorithm, it has the drawback of sometimes finding itself stuck in local maxima and achieving a very low quality score. However, since we expect BM-OPT to be significantly faster than DCBM, we are able to run the algorithm several times in the same amount of time it takes for a single run of DCBM. In practice, we would simply use the best result. Averaging the results unnecessarily penalizes the greedy algorithm and ignores BMOD's purpose.

The quality test varies the amount of randomness defined in Equation 5 in a fixed graph type to see how much noise the two algorithms can handle while still finding a majority of the planted structure of the graph. In this test, the number of clusters and number of nodes are fixed. Two graph types will be used: hierarchical and regular community structure (defined by Equations 7 and 6, respectively). There are 1000 nodes and three clusters.

The first speed test measures how well the algorithms scale as the number of nodes increase while leaving the number of clusters (3) and the graph type (Community) fixed. Also, the graph is fully planted, with mixture parameter of 1, so that varying random effects will not affect the timing. The second speed test measures how well the algorithms scale the number of clusters increases while all other properties remain fixed.

Results

As seen in Figure 2, the speed improvement of using BM-OPT is readily apparent. As the number of nodes increases, using BM-OPT results in up to a 1000× speed increase over DCBM with no significant difference between BM-OPT and BMOD. When the number of clusters increases, the speed increase of using BM-OPT is more modest, with a roughly 30× speed increase over DCBM and 10× speed increase

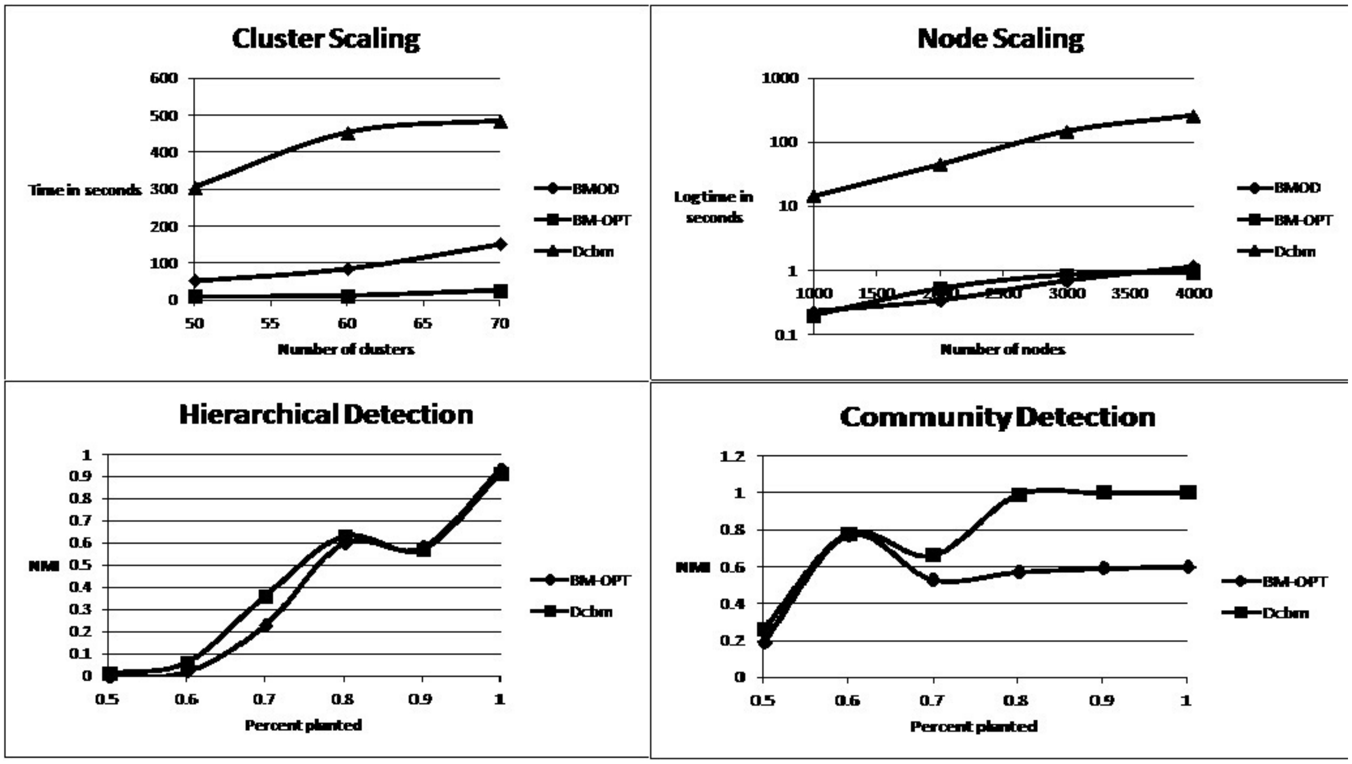

Figure 2: Time and quality comparisons for the optimized and unoptimized BMOD algorithms and DCBM. Logarithmic time (base 10) is used on the y axis for node scaling. The sample points of the time comparisons are the result of running the algorithms three times and averaging the result. The sample points of the quality comparisons are the best scores out of ten runs. The quality results use normalized mutual information as a method of determining how close the result is to the perfect clustering. Percent planted refers to λ as described by Equation 5.

over BMOD. This is a significant improvement in runtime.

It is interesting to see that the original version BMOD actually outperforms BM-OPT in terms of run time when the number of clusters is low. This is because our modifications do result in a larger constant factor when computing the change in score. As the number of clusters increases, this constant factor becomes less significant.

The quality of clustering results is mixed: for hierarchical networks, BM-OPT performs as well as DCBM. In the case of community detection, quality advantage of DCBM over BM-OPT is greater. This is due to the fact that BMOD (and therefore BM-OPT) scores a bipartite division of the data equally as high as communities. However, this result should not necessarily be seen as a weakness of BM-OPT, for several reasons. First, there is no objective reason to favor finding community structure over bipartite structure in the general case. (If clusters are *only* expected to take the form of traditional "communities," then there are specialized, fast algorithms for finding such structures (Newman 2006; Shi and Malik 1997). However, these methods cannot be used to find other types of structures, so they are less general than BM and BM-OPT.) Second, the case of fully disjoint communities is not realistic; therefore, we expect BM-OPT to perform better in scenarios where there are secondary patterns to guide the search. This hypothesis is confirmed by observing that BM-OPT performs very well in both the hierarchical case, as well as the bridge network case (below).

The stress test of BM-OPT was positive, as shown in Table 1. Not only was BM-OPT able to complete a run on such a large graph (100,000 vertices and 8,000,000 edges), it was able to identify nearly all of the structure within the graph:

Time(hrs)	epsilon	NMI	Score	Perfect Score
45.22	.001	.95	.99733	.99821

Table 1: Results of the Facebook data stress test.

While 45 hours seems to be rather slow, the reader should bear in mind that this is the result of 10 individual executions of the algorithm. As a further frame of reference, we were unable to complete a single execution of the reference implementation for the DCBM (2011) in the same time frame. If time were more critical, the data could have been processed in 4.5 hours, and if finding a near-optimal solution is not a necessity, then the epsilon value could be adjusted further to reduce the number of iterations dramatically. Also, to emphasize the significance of our optimization, we would expect the un-optimized version of block modularity to be 1000 times slower.

Conclusions and Future Work

We presented a simple yet effective optimization to the Block Modularity Algorithm. This optimization led to a significant speed increase over the previous version as the number of clusters increases (other than a small constant increase in runtime on small graphs). There is no change in the computed results in the algorithm, so this change is a net improvement. BM-OPT is a fast algorithm for clustering objects using block modeling. Its results are similar in quality to DCBM in all cases except the community structure, although we do not see this to be a serious fault. The benefit of such a high speed increase is that we can now run BM-OPT to find interesting block model structures on increasingly large graphs.

Finding a large number (thousands) of clusters using the optimized block modularity algorithm may yield results that could be nearly as difficult to interpret as the original data set. Therefore, we are currently developing methods of post-processing that can quickly characterize different types of block-structure patterns that are useful for specific practical applications such as information distribution, epidemiology, and link prediction. These methods will permit a user to explore a very large data set to identify patterns that are of particular relevance in a given domain.

References

Anthony, A. P. 2009. *Stochastic and iterative techniques for relational data clustering*. Ph.D. Dissertation, University of Maryland at Baltimore County.

Clauset, A.; Newman, M. E. J.; and Moore, C. 2004. Finding community structure in very large networks. *Phys. Rev. E* 70:066111.

Gjoka, M.; Kurant, M.; Butts, C. T.; and Markopoulou, A. 2010. Walking in Facebook: A Case Study of Unbiased Sampling of OSNs. In *Proceedings of IEEE INFOCOM '10*.

Handcock, M. S.; Rafferty, A. E.; and Tantrum, J. M. 2007. Model-based clustering for social networks. *Journal of the Royal Statistical Society* 170(2).

Karrer, B., and Newman, M. E. J. 2011. Stochastic blockmodels and community structure in networks. *Phys. Rev.* 83(1).

Kemp, C.; Tenenbaum, J. B.; Griffiths, T. L.; Yamada, T.; and Ueda, N. 2006a. Learning Systems of Concepts with an Infinite Relational Model. In *AAAI 2006*.

Kemp, C.; Tenenbaum, J. B.; Griffiths, T. L.; Yamada, T.; and Ueda, N. 2006b. Learning systems of concepts with an infinite relational model. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, 381–388. AAAI Press.

Leicht, E. A., and Newman, M. E. J. 2008. Community structure in directed networks. *Phys. Rev. Lett.* 100(11).

Newman, M. E. J., and Leicht, E. A. 2007. Mixture models and exploratory analysis in networks. *Proc. Natl. Acad. Sci.* 104.

Newman, M. E. J. 2006. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of the United States of America* 103(23):8577–8582.

Nowicki, K., and B., S. T. A. 2001. Estimation and prediction for stochastic blockstructures. *Journal of the American Statistical Association* 96:1077–1087.

Shi, J., and Malik, J. 1997. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22:888–905.