

# NISS

## Finding Clusters in Network Link Strength Data

Todd L. Graves

Technical Report Number 94  
March, 1999

National Institute of Statistical Sciences  
19 T. W. Alexander Drive  
PO Box 14006  
Research Triangle Park, NC 27709-4006  
[www.niss.org](http://www.niss.org)

# Finding clusters in network link strength data

Todd L. Graves\*

December 8, 1998

## Abstract

In this paper we introduce BEACON, a tool for finding clusters of objects using data which are measurements of strengths of links between those objects. This technique is useful for measuring modularity in software systems as well as in analysis of social network data. We apply Bayesian tools such as Markov chain Monte Carlo to estimate clusters. The paper discusses simulation experiments which demonstrate the power of the methodology to find true clusters, and applies the technique to the analysis of a subsystem of a large telecommunications software product.

## 1 Introduction and Problem Statement

In most statistical problems, each data point is a set of measurements of different variables, where the measurements pertain to a single object or point in time. A number of interesting applications include data which measure strengths of connections between pairs (or larger sets) of objects. One question of interest in such applications is whether it is possible to decompose the “network,” or set of objects, into clusters of nodes. Clusters should have the property that connections between pairs of nodes in the same cluster should be stochastically larger than connections between pairs of nodes in different clusters.

A motivating example comes from software engineering. A large software system has an official hierarchical decomposition into source files, modules which as used here are collections of files, and subsystems, which are collections of modules. As the code evolves, subsets of files are modified together as part of larger code changes. It is of interest to measure the success of modularity in the code, for instance how well the official decomposition matches the de facto composition into sets of files which are modified in tandem. Hutchens and Basili (1985) use “data bindings” obtained from analysis of the code to represent software as a dendrogram, thereby constructing clusterings of code units at several levels.

---

\*Todd L. Graves is Postdoctoral Fellow, National Institute of Statistical Sciences, and at Bell Laboratories, 263 Shuman Blvd, Naperville, IL 60566. E-Mail: graves@niss.org; Web: <http://www.niss.org/~graves>. This research was supported in part by NSF grants SBR-9529926 and DMS-9208758 to the National Institute of Statistical Sciences. The author is grateful to Audris Mockus, Alan Karr, Steve Marron, and Graham Wills for helpful suggestions.

Another field which has seen a great deal of work in statistical inference for network-type data is the field of social networks, for which see Wasserman and Faust (1994; a comprehensive introduction to the field), Wasserman and Galaskiewicz (1994; a collection of case studies demonstrating the application of social network analysis in several different domains), and Wasserman and Pattison (1996; a demonstration of a particularly rich class of network models). In addition, Wasserman and Anderson (1987) point out the need for techniques for estimating network structure from link data, suggests some techniques based on graphics or correspondence analysis, and recommends Bayesian analysis to future researchers. Nodes in social network analysis typically represent people, and an example of a type of linkage is if two people communicate frequently. Often links in social networks are directional, i.e. the link from node  $i$  to node  $j$  is distinct from the link from node  $j$  to node  $i$ . In much social network data, links between nodes are either present or absent, and many social network methods are derived from graph theory to deal with this special case. Emphasis in finding “clusters” in social networks has been mainly on finding sets of nodes which relate similarly to other nodes (two nodes are “structurally equivalent” if they are linked to exactly the same set of nodes, and there is also much interest in weakened versions of this concept), as opposed to finding sets of nodes with strong links among themselves. As will be seen in Section 5, the methodology described here can be applied to this sort of problem as well. In particular, a modification of the model can be used to estimate and evaluate blockmodels, which divide the network up into groups of nodes which have similar relationships to other nodes.

Another arena in which statistical inference for network data is necessary is in complementing network visualization algorithms, such as Wills (1998), multidimensional scaling as in Kruskal and Wish (1978), or Freeman (1998). These algorithms represent nodes as points in two- (or three-) space, and place nodes close together if the links connecting them are strong. Since these algorithms aim to find clusters, there is a danger of overinterpreting spurious clusters. Consequently, a statistical model which permits the analyst to ask if a cluster is real or perhaps only an artifact of variability in the measurements of link strengths would greatly strengthen these visualization techniques.

An example which arose in analysis of software data is depicted in Figure 1. The data here are measurements on “modules” (collections of files) in the software from a large telecommunications product. If two modules were frequently changed together as part of the same change to the code, the weight between them is large and NicheWorks (Wills, 1998) tries to place them close together in the visualization. (NicheWorks, intended for use with networks several orders of magnitude larger than this one, uses simulated annealing as well as a steepest descent algorithm to generate node locations.) Each dot represents a module, the three plots display all the data up through different years, and the ends of the tails display the data through the previous year, so that one can observe a year’s changes by looking from tail to head of the arrow. The first plot indicates that in 1988, the modules clustered into two major groups. This grouping began to break down in 1989, and by 1996 there was little if any hint of any way of breaking up the modules. We will return to the data used to construct this figure in Section 6.

While graphs such as Figure 1 help motivate the problem, drawing such a graph is not a complete solution. A graph is a two-dimensional representation of the complex clustering structure of the nodes. In particular, there is a lot of error in translating from weights to distances between nodes.

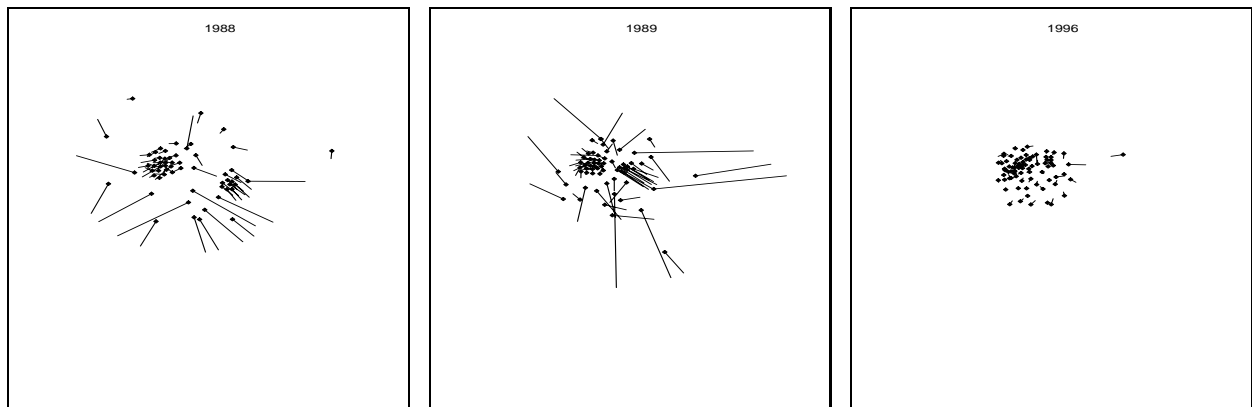


Figure 1: Left: NicheWorks view of the modules in one subsystem, using data through 1988 to place modules which have been changed at the same times close to one another. (The heads of the symbols show data through the year indicated, while the tails show the analogous view from the previous year, so the lines can be interpreted as displaying what happened in the last year.) Two clusters of modules are evident; a module within one of these clusters is often changed together with other modules in the cluster but not with other modules. Center: NicheWorks view of the modules in the left, this time incorporating the history through 1989. The clusters that appear in the left view are converging in on each other. This suggests that the architecture that was previously successful in separating the functionality of the two clusters of modules is breaking down. Right: the breakdown continued, and at the end of 1996, there was no suggestion of clusters of modules.

In this paper we fill these needs by devising a statistical model which explicitly allows identification of clusters together with measurement of their strengths. We make use of Bayesian techniques such as Markov chain Monte Carlo to estimate clusters and other parameters of interest. The implementation of this technology is called BEACON (Bayesian EstimAtion of Clusters Of Nodes). Wong (1987) applied Bayesian methods to the social network problem, specifically the  $p_1$  model of Holland and Leinhardt (1981), for which see also Wasserman and Pattison (1996).

The structure of the paper is as follows. In Section 2 we describe the statistical model. In Section 3 we describe the fitting of this model using Markov chain Monte Carlo. Section 4 gives the algorithm’s performance on simulated network data, including indications that noisy data are generally more to blame than the high dimensionality of the problem for failure to estimate the model correctly. Section 5 discusses the wealth of extensions the modeling framework admits, including multidimensional and nested clusterings, node-specific data, and many of the traditional network measures discussed by Wasserman and Pattison (1996). We return to the software engineering example in Section 6. Conclusions are in Section 7.

## 2 Simplest Model

Here we delineate the most simple incarnation of the cluster estimation model. (More complicated models will appear in Section 5.) The system under study has  $N$  nodes, which will be labeled  $1, 2, \dots, N$ . The data peculiar to this problem are the link strength measurements, also called *weights*. Let  $w_{ij}$  be the measured strength of the (nondirectional) link between nodes  $i$  and  $j$ .  $w_{ii}$  is undefined, and  $w_{ij} = w_{ji}$  for each  $(i, j)$ . We model clustering by defining latent cluster labels  $X_i, 1 \leq i \leq N$ .  $X_i = k$  means that node  $i$  is in the  $k$ th cluster. Cluster labels have no absolute meanings, and we circumvent the problem of specifying the number of clusters *a priori* by taking the possible values of the  $X_i$ ’s to be  $1, 2, \dots, N$ . In principle each node could belong to its own cluster, but in practice most of the “clusters” are empty and many a “cluster” will contain a single node (*i.e.* the number of  $X_i$ ’s equalling  $k$  will often be either one or zero).

We model the distribution of the weights conditional on the cluster labels as follows. If nodes  $i$  and  $j$  are in different clusters ( $X_i \neq X_j$ ), the link connecting them has a small mean  $E(w_{ij}) = \theta_0^{-1}$ . If on the other hand nodes  $i$  and  $j$  are in the same cluster ( $X_i = X_j$ ), their link strength has a larger mean  $E(w_{ij}) = (\theta_0 \theta_1)^{-1}$ , where  $\theta_1 < 1$ . For simplicity we have modeled the  $w_{ij}$ ’s as having exponential distributions, which seems not to be quite correct for software data (if only because exponential random variables are never exactly zero), but as discussed in §4.6, this choice seems to have adequate robustness properties, and in any case it is possible to extend this formulation to change the weight distribution to allow  $P\{w_{ij} = 0\} > 0$ .

Finally, a Bayesian solution to this problem requires prior probabilities  $p_k = P\{X_i = k\}$ , the unconditional probability that node  $i$  belongs to cluster  $k$ . We take a Dirichlet prior distribution for the  $p_k$ ’s. It is possible to use the parameters of this prior to encourage the formation of small or large numbers of clusters (e.g. by taking  $E(p_{i+1})/E(p_i) = \lambda$ ; presumably one could try to elicit a value of  $\lambda$  from an expert), but we have found that data readily overwhelm the choice of the prior.

Denote by  $\nu_k$  the Dirichlet parameter corresponding to the  $k$ th cluster i.e.

$$f(p|\nu) \propto \prod_{k=1}^N p_k^{\nu_k}.$$

In this paper we will use the notational convention common in Bayesian statistics in which  $f$  represents a number of functions distinguished by their arguments.

We also require prior distributions for the mean link strength parameters. To attain conjugacy, let  $\theta_0$  have a prior Gamma distribution with parameters  $\alpha_0$  and (inverse-scale)  $\beta_0$ , while  $\theta_1$  has a prior Gamma distribution with parameters  $\alpha_1$  and  $\beta_1$ , but where  $\theta_1$  is conditioned to lie in  $(0, 1)$ .

The foremost inferential goal in this formulation is the recovery of the  $X_i$ 's. Also important is estimation of  $(\theta_0, \theta_1)$ , to enable statements about the tendency toward strong links in this network, and in particular to assess the importance of the estimated clustering. If  $\theta_1$  is close to one, the clustering is more likely spurious and certainly less important in describing the network.

### 3 Markov Chain Monte Carlo Analysis of the Model

Since the formulation above requires  $2N + 2$  parameters and latent variables, we need to use Gibbs sampling to obtain approximate draws from the posterior distribution of quantities of interest such as the  $X_i$ 's and the  $\theta$ 's. (See Gelman et al (1995) and Tanner (1993) for tutorials on the use of Gibbs sampling and related methods.) The likelihood  $f(X, \theta, p, w|\alpha, \beta, \nu)$  is proportional to

$$\left(\prod_{k=1}^N p_k^{N_k + \nu_k}\right) \left(\prod_{d=0}^1 \{\theta_d^{\alpha_d - 1} \exp(-\beta_d \theta_d)\}\right) \prod_{i=2}^N \prod_{j=1}^{i-1} \{\theta_0 \theta_1^{I\{X_i=X_j\}} \exp(-\theta_0 \theta_1^{I\{X_i=X_j\}} w_{ij})\}$$

where  $N_k = \sum_{i=1}^N I\{X_i = k\}$  and  $I(A)$  equals one if event  $A$  occurs, zero otherwise.

The first step in a Gibbs sampling iteration is drawing a random set of probabilities  $p$ . For initial values use the Dirichlet distribution with parameters  $\nu_k$ ; subsequently use the Dirichlet distribution with parameters  $N_k + \nu_k$ .

The guts of the algorithm is the drawing of new values of the cluster labels. The distribution of the  $X$ 's given the remaining parameters and data does not have a familiar form. Suppose we contemplate moving node  $i$  into cluster  $k$ . The ratio of the probability of node  $i$  belonging to cluster  $k$  to the probability of it belonging to its current cluster (namely  $X_i$ ) is

$$\rho_k = \theta_1 \sum_{j \neq i} (I\{X_j=k\} - I\{X_j=X_i\}) \frac{p_k}{p_{X_i}} \exp\{\theta_0(\theta_1 - 1) (\sum_{j \neq i} w_{ij} I\{X_j = X_i\} - \sum_{j \neq i} w_{ij} I\{X_j = k\})\}.$$

To update the value of  $X_i$ , then, one computes these ratios, and draws a new value according to the resulting probabilities  $\rho_k / \sum_{\ell=1}^N \rho_\ell$ . (An alternate approach, using the Metropolis algorithm, involves picking a node  $i$  and cluster  $k$  at random, and changing the value of  $X_i$  to  $k$  with probability  $\rho_k$ . Within each Gibbs step, use a number of Metropolis steps; I have used  $N$  so that each node gets an average of one chance to move to another cluster. The Metropolis version seems to converge

to the limiting distribution of  $X$  considerably more slowly, so we do not discuss it further here, but it requires fewer computations within each iteration.)

The final stage in the Gibbs sampling is the drawing of new sample values of  $\theta$  from (constrained) Gamma distributions. The conditional distribution of  $\theta_0$  is Gamma with shape parameter  $\alpha_0 + N(N - 1)/2$  and inverse scale parameter

$$\beta_0 + \sum_{i=2}^N \sum_{j=1}^{i-1} w_{ij} \theta_1^{I\{X_i=X_j\}}.$$

The conditional distribution of  $\theta_1$  is Gamma constrained to lie in  $(0, 1)$ , with shape parameter  $\alpha_1 + \sum_{i=2}^N \sum_{j=1}^{i-1} I\{X_i = X_j\}$  and inverse scale parameter

$$\beta_1 + \theta_0 \sum_{i=2}^N \sum_{j=1}^{i-1} w_{ij} I\{X_i = X_j\}.$$

Since in this work we were most interested in obtaining point estimates for the values of the clustering variables, we stopped the Gibbs sampler after relatively small numbers of iterations: 200 was typical in the simulation experiment. Were we after posterior probabilities of complex events such as

$$P\{X_1 = X_2 = X_3 = X_4 = X_5 = X_6 = X_7 = X_8 = X_9 = X_{10} | w, \nu, \alpha, \beta\},$$

a considerable number of additional iterations probably would be required. Such posterior probabilities are certainly of interest, as they would address the question “is that cluster really there?” which one might ask after viewing a visualization. The emphasis of the present paper is on estimation of the clusters and other goals which can be attained using a relatively small number of iterations, in part because we envision using this technique on large numbers of data sets for different software subsystems and at different points in time.

After running the desired number of Gibbs iterations, it is necessary to summarize the results. One may compute the median or mean values or the standard deviation of the  $\theta$ 's after omitting those from the iterations in a burn-in period for the chain. Summarizing the clustering information in the chain of  $X_i$ 's is less straightforward; certainly it is unsatisfactory to use the clustering corresponding to the last iteration in the chain. I recommend a voting procedure in which  $X_i$  is reported to be equal to  $k$  if  $k$  is the most common value of  $X_i$  in the MCMC iterations (after a burn-in period if desired), provided that  $X_i = k$  in at least 50% (for example) of those iterations. If there exists no  $k$  for which the fraction of iterations that  $X_i = k$  exceeds 0.5, then node  $i$  is reported to be in its own cluster. A potential problem with this is if a pair of nodes  $i_1$  and  $i_2$  have identical estimated values of the clustering variable in a large number of iterations, but this common value changes excessively from iteration to iteration. This phenomenon seems to be relatively rare.

## 4 Performance on Simulated Data

In this section we discuss a simulation experiment which determines how strong a clustering needs to be before we can expect to find it reliably with the algorithm, and how rapidly convergence

occurs. We also look into robustness, asking how important it is that the distribution of the link strengths be nearly exponential by experimenting with dichotomous responses.

The results are very encouraging for networks with small numbers of large clusters. The algorithm rapidly discovers the true clusters in such situations even when the differences in link strengths are relatively small. Increasing the number and decreasing the size of the true clusters makes it more difficult for the algorithm to find the truth. Smaller clusters are more difficult to find, but presumably they are also less interesting than larger clusters.

In the simulations, we varied several characteristics of the true network. In all simulations, we worked with networks of sixty nodes. We used three different true clusterings:

1. two clusters of thirty nodes each;
2. six clusters of ten nodes each;
3. twelve clusters of size four, and twelve “clusters” of size one.

We also varied  $\theta_1$ , the measure of how much stronger within-cluster links are than between-cluster links; we experimented with  $\theta_1^{-1} = 10, 5$ , and  $2$ . We ran ten simulation runs under each set of conditions. In §4.6, we discuss varying the link strength distribution from exponential (which the algorithm assumes) to Bernoulli to assess the robustness of the methodology.

We measure success of a simulation run in several ways. The first is a measure of how well nodes which belong together are clustered together and is based on the voting technique discussed in Section 3. Obtain cluster labels from the voting, and then look at the set of nodes in a true cluster. Pick the most common cluster label within this cluster, and consider the nodes with that label to have been classified correctly, the remainder to be incorrect. Add the numbers of correctly placed nodes across all true clusters. This measure has a minimum value of one correct classification for each true cluster, since the absolute value of a cluster label is meaningless. We will often look at the equivalent measure obtained by subtracting this measure from  $N$ .

A complementary statistic measures how much “pollution” appeared in the simulation and is also based on voting results. For each true cluster, determine the cluster label most often assigned to it as above, then count how many nodes not in that true cluster were also assigned that label. Then add up these counts over true clusters. This measure is often guilty of double-counting; suppose that  $n$  nodes are each isolated but are all clustered together; these  $n$  nodes would contribute  $n(n-1)$  to the pollution measure.

Table 1 contains the numbers of errors, calculated by subtracting the values of the first statistic from sixty, for each of the three values of  $\theta_1$  and the three clustering configurations; the corresponding values of the pollution statistic are in Table 2. The problem is easiest when there are few large clusters, and of course when links between nodes in the same cluster are much stronger than links joining nodes in different clusters. In the bottom row of Table 1, which corresponds to the configuration with clusters of four nodes and isolated nodes, the numbers of classification errors are computed only for the four-node clusters, since the voting-based success measure would always score an isolated node as if it had been clustered correctly.

We also report, in Table 3 and Table 4, the estimates of  $\theta$  that the algorithm generated from the simulated data. We take the last 100 out of 200 iterations in each simulation, compute the mean



	$\theta_1^{-1} = 10$	$\theta_1^{-1} = 5$	$\theta_1^{-1} = 2$	Worst possible
$2 \times 30$	0.0	0.0	1.8	58
$6 \times 10$	3.3	4.8	45.1	54
$12 \times 4 + 12 \times 1$	6.4	19.4	35.2	36

Table 1: Average (over ten simulation runs) numbers of nodes excluded from their correct clusters. Computed using last 100 iterations out of a total of 200. The last column shows the worst possible score attainable for that configuration; the algorithm is guaranteed to get at least one node per cluster correct. The scores in the bottom row are computed using only the clusters of size four, omitting the isolated nodes.

	$\theta_1^{-1} = 10$	$\theta_1^{-1} = 5$	$\theta_1^{-1} = 2$
$2 \times 30$	0.0	0.0	0.1
$6 \times 10$	0.4	0.4	4.0
$12 \times 4 + 12 \times 1$	10.4	13.0	2.2

Table 2: Measures of pollution also calculated from votes using the 101st through 200th iterations.

and standard deviation of each parameter over those iterations, and then report the minimum, median, and maximum over the ten simulation runs. Table 3 indicates that the estimates of  $\theta_0$  are quite good, and the quality of the estimates is independent of the value of  $\theta_1$  and of the cluster configuration, even in configurations and with parameter values which make it impossible for the algorithm to find the correct clusters. It appears that even if the clusters are not identified correctly, enough pairs of nodes are correctly judged to be in different clusters so that  $\theta_0$  can be estimated with very little error.

The quality of the estimates of  $\theta_1$  does depend on the difficulty of the cluster identification problem, as can be seen in Table 4. For more accurate comparison of standard deviations of different parameters, we have reported the standard deviations divided by the true value of the parameter being estimated to arrive at a coefficient of variation. There is a strong dependence of

	$\theta_1^{-1} = 10$	$\theta_1^{-1} = 5$	$\theta_1^{-1} = 2$
$2 \times 30$	0.95 1.00 1.07	0.99 1.04 1.08	0.87 0.96 1.02
std. deviations	.029 .033 .040	.031 .033 .036	.028 .034 .078
$6 \times 10$	0.84 0.95 1.02	0.88 0.97 1.02	0.92 0.94 0.98
std. deviations	.021 .026 .050	.022 .026 .028	.024 .030 .033
$12 \times 4 + 12 \times 1$	0.91 0.99 1.02	1.01 1.03 1.05	0.95 1.01 1.05
std. deviations	.019 .024 .029	.025 .027 .029	.029 .031 .041

Table 3: Posterior mean of  $\theta_0$ , taken from the median of the last 100 out of 200 iterations, together with estimated posterior standard deviations. The three numbers in each row within a cell are minimum, median, and maximum over the ten simulation runs.

	$\theta_1^{-1} = 10$	$\theta_1^{-1} = 5$	$\theta_1^{-1} = 2$
$2 \times 30$	.092 .100 .106	.183 .196 .201	.467 .489 .535
sd/true mean	.041 .046 .053	.039 .045 .050	.041 .046 .057
$6 \times 10$	.095 .101 .129	.172 .201 .218	.386 .453 .550
sd/true mean	.057 .070 .151	.060 .068 .088	.076 .118 .258
$12 \times 4 + 12 \times 1$	.099 .120 .153	.149 .246 .332	.413 .620 .825
sd/true mean	.114 .151 .191	.096 .151 .210	.130 .216 .288

Table 4: Posterior mean of  $\theta_1$ , taken from the median of the last 100 out of 200 iterations, together with estimated posterior standard deviations. The three numbers in each row within a cell are minimum, median, and maximum over the ten simulation runs. The standard deviations are divided by the true value of the parameter being estimated to assist comparisons between columns.

coefficient of variation on the cluster configuration. This stands to reason because the large cluster configurations have more pairs of nodes in the same cluster, so that  $\theta_1$  is reflected in more data points. The large cluster configuration generates 870 such pairs of nodes compared to 270 for the moderate clusters and 72 for the small clusters. If coefficients of variation were proportional to the  $-\frac{1}{2}$  power of these quantities (as they would be if variances were proportional to  $\theta_1^2$  divided by the “sample size” available for estimating  $\theta_1$ ), they would be in ratios of 0.56:1:1.9, which are not far from the values in the table. The dependence of the coefficients of variation on the true value of  $\theta_1$  is weaker and for some pairs of values seems to be nonexistent. Unlike in the case of estimating  $\theta_0$ , here we are sometimes unable to cluster enough pairs of nodes together successfully to make  $\theta_1$  easily estimable.

We also need a measure of the speed by which a simulation run attains convergence. We do this by giving each iteration a score in the same way that we score a voting result. Then take the mode of this score over iterations. The first iteration for which the score equals or exceeds the mode is then a measure of how rapidly the algorithm locates a configuration which is at least relatively close to its ultimate limiting distribution.

#### 4.1 Few large clusters

Estimating two clusters of size thirty is nearly trivial. When  $\theta_1^{-1} = 10$ , the algorithm first located the correct clustering no later than the tenth iteration in each of the ten simulation runs. Performance was also very strong for  $\theta_1^{-1} = 5$ , when nine of the simulation runs found the correct clustering at the eleventh iteration or earlier. The tenth run did not succeed in finding the correct clustering until the 54th iteration.

When  $\theta_1^{-1}$  is decreased to 2, the cluster signal is weak enough so that the algorithm rarely finds exactly the right clusters. After computing the voting results as described in Section 3, using the 101st through 200th iterations, one iteration’s voting made five errors, and one made none, while the remaining eight had one or two errors. Despite not converging to a perfectly correct clustering, the algorithm still finds its apparent limit fairly rapidly: six out of ten times the iteration on which the algorithm first hit its modal score was the 16th iteration or earlier. Once it took 148 iterations

to attain the modal score. When  $\theta_1^{-1} = 2$ , the chances are too high that at least one or two of the sixty nodes will have link strengths which do not look sufficiently like they should for a node in that cluster.

## 4.2 Several moderate sized clusters

The problem becomes difficult when we shift from two large clusters to six clusters of ten nodes each. Even  $\theta_1^{-1} = 10$  is insufficiently large to guarantee that the algorithm will find the correct limit. However, the convergence to a set of states not too far from the correct limit is fairly rapid. The clustering obtained from voting were quite close to the truth for large and moderate values of  $\theta_1^{-1}$ : for  $\theta_1^{-1} = 10$ , three simulation runs led to no errors in the voting, while the maximum number of errors was nine. For  $\theta_1^{-1} = 5$ , two runs had perfect votes, but twice the number of errors reached 14. The drop in performance for  $\theta_1^{-1} = 2$  is large: while one run achieved 32 correct in its vote, the remainder ranged from nine through 21 correct. (Note that the minimum possible score is six.)

## 4.3 Many small clusters

Our final cluster configuration that we tried in the simulation experiment had twelve clusters of four nodes each, and twelve nodes which did not belong to multiple-node clusters. This simulation is therefore designed both to demonstrate how easy it is to find relatively small clusters, and also how resistant the algorithm is to putting nodes which should be left alone into clusters. We require two types of measures of the success of the algorithm for this configuration: how many of the nodes in four-node clusters are classed together, and how many of the isolated nodes are kept alone.

For  $\theta_1^{-1} = 10$ , the algorithm's score in classifying four-node clusters together ranged from 36 to the maximum of 48 over the ten runs. (The possible values of this score range from 12 to 48, because here we restrict attention to the four-node clusters). Six of the votes placed all twelve isolates alone, in one run eleven were alone, and three times ten were alone.

For  $\theta_1^{-1} = 5$ , the algorithm was considerably worse in placing nodes in the same cluster together, with scores ranging from 20 to 36. The scores for keeping isolates alone were only slightly worse, with three tens, three elevens, and four twelves. For  $\theta_1^{-1} = 2$ , the voting results are probably better at keeping isolates apart (one eleven and nine twelves) but there is almost no chance of finding clusters of size four. The clustering scores included one 19 and one 13, with the rest 12's, the minimum score.

For all three values of  $\theta_1$ , the convergence to the limiting behavior was quite rapid. The longest it took for the classification score to attain its modal value was the fifteenth iteration.

## 4.4 A mixture of cluster sizes

We ran one simulation scenario which used a mixture of large and small clusters: one cluster of size twenty, two of size ten, three of size four, and eight isolated nodes. Of interest here was, for example, whether the presence of larger clusters made it easier or harder to detect small clusters. We used the intermediate value of  $\theta_1^{-1} = 5$ , and ran ten replications of this experiment. All ten of the size-twenty clusters were identified perfectly with no pollution. Only thirteen of the 200

nodes (6.5%) in clusters of size ten were placed into incorrect clusters; the comparable value for the configuration with only clusters of size ten is 4.8 per sixty nodes (8%) when  $\theta_1^{-1} = 5$ , which is of the same order. A total of two nodes polluted the twenty ten-node clusters (0.1 pollutions per cluster), compared to an average of 0.4 pollutions per six ten-node clusters (0.067 per cluster) when all clusters are of size ten. The failed classification measure for four-node clusters was 65 out of a total of 120 nodes (54%), compared to a somewhat better 194 out of 480 (40%) in the configuration with only clusters of sizes four and one. Seven nodes polluted the thirty four-node clusters, better than the 92 pollutions of 120 clusters in the earlier experiment. Seventeen nodes polluted the eighty isolated nodes, somewhat less than the 38 per 120 obtained from the fours-and-ones configuration.

#### 4.5 Initial conditions equal to truth

One thing another simulation can help us do is to assign blame for the algorithm’s failure to find the true clustering: the algorithm may get stuck in a suboptimal place, or the data may be such that the true clustering does not provide the best fit. To test the algorithm for effect of initial conditions, we ran it on the simulated data again, this time beginning the simulation from the true clustering. If the algorithm has a problem with failing to find the best model because of getting stuck in suboptimal places, this choice of initial conditions should lead to considerably better results than the randomly generated initial conditions used in real applications.

However, initial conditions equal to the true clustering did not increase the success of the algorithm in the simulations. First, on the configuration with six clusters of size ten and  $\theta_1^{-1} = 2$ , the correct initial conditions were if anything harmful: while beginning with random initial conditions led to an average score of 14.9 correct on the vote, the correct initial conditions decreased this number by an average of one node per simulation run, though this difference was not significant at the 5% level. There was a close relationship between the voting scores for the two sets of initial conditions for each simulated data set; the correlation observed was 0.98.

We repeated the experiment with the small cluster configuration and with  $\theta_1^{-1} = 5$ . Again, if anything the correct initial values tended to make the algorithm perform slightly worse: the mean voting scores for the four-node clusters were 28.6 nodes correct (i.e. 19.4 errors; see Table 1) for random initial values and 27.7 for correct initial values. Again the difference was not statistically significant according to a permutation test on the differences. There was also no change in the frequency with which true isolated nodes are estimated to be alone.

#### 4.6 Robustness studies

We also perform studies to see if the fact that we use the exponential distribution for link strengths in the probability model requires that the data distributions can in fact be well approximated by the exponential. For an alternate response distribution we took the Bernoulli, with

$$P\{w_{ij} = 1|X, \theta\} = 1 - P\{w_{ij} = 0|X, \theta\} = \theta_0^{-1}\theta_1^{-I\{X_i=X_j\}}.$$

We continued to use the exponential distribution in the algorithm, even though it is certainly possible to incorporate the Bernoulli distribution into the model. A Bernoulli random variable is

particularly vulnerable to false positives, since the probability it will attain its maximum value is positive, and in that case the pair of nodes will appear as much as possible like they are in the same cluster. It is therefore quite rare that a node will be estimated to be in its own cluster, but this is not a fault with the algorithm but with the data.

The conclusion is that using the exponential model for 0-1 data does not make the problem excessively difficult. Use of 0-1 data if anything made it easier for the algorithm to find clusters, although there was the unavoidable increase in the false positive rate as well. As a first attempt, we set  $\theta_0^{-1} = 0.15$  and  $\theta_1^{-1} = 5$ . (This corresponds to success probabilities of 0.15 for nodes in different clusters, 0.75 for nodes in the same cluster.) In the configuration with six clusters of ten nodes each, the algorithm had less difficulty with 0-1 link strengths than with exponential link strengths and  $\theta_1^{-1} = 5$ , as the ten simulation runs had an average of 1.2 errors in the final vote, compared to 4.8 errors for exponential data.

The second scenario also used Bernoulli data with success probabilities 0.15 and 0.75, in the small cluster configuration with twelve four-node clusters and twelve isolated nodes. With Bernoulli data, the algorithm correctly placed an average of 31.3 out of forty-eight nodes, while the comparable exponential score was 28.6. However, Bernoulli data made it much more difficult for isolated nodes to be identified as such: typically only 1.1 isolated node out of 12 was evaluated as isolated in the final vote, while exponential data were able to score 11.1. Again, it is too much to ask for the algorithm to be able to avoid false positives of this type.

## 5 Extensions

The framework outlined in earlier sections is not difficult to extend to more elaborate models. Instead of estimating clusters from data, one may use a clustering known from other sources and measure how well it matches the data. In past sections we have discussed models where pairs of nodes are either in the same cluster or not, but it is also possible to model situations in which nodes are together in one respect but different in another. A special case involves hierarchies of clusterings. This framework can also incorporate some parameters of interest to social network researchers.

### 5.1 Fixed clusters

In some applications, the analyst will have a clustering in mind and will want to treat the  $X_i$ 's as fixed instead of being quantities to impute using the algorithm. A simple example in the domain of social networks might take nodes to be children, the weights being a measure of how much time they spend playing together, and the  $X_i$ 's being 1 or 2 depending on the gender of the child.

One example of using fixed clusters in data analysis is fitting a clustering to data from one period in time (for example, the first such period), and seeing how relevant that clustering is to similar data from a different period of time. This idea is particularly appealing for software data: by fitting a clustering to early data one can obtain a measurement of the original intended modularity. By evaluating this clustering with respect to new data one can measure the extent to which later development has remained true to the original design of the code.

This idea can become more interesting if we allow multidimensional clusterings.

## 5.2 Multidimensional models

For example, in the simple model described in Section 2, a pair of nodes can either be in the same cluster or in different clusters. One can introduce a finer classification by using a multidimensional clustering model. In such a model, pairs of unrelated nodes have small mean link strengths, but a pair of nodes can attain a larger mean link strength if the pair of nodes shares one or more characteristics. For example, in a social network one might model children as being more likely to be friends if they live near each other, or if they are of the same gender. In this case one increases the dimension of  $\theta$  to  $D + 1$ , defines additional clustering variables  $\{X_{id} : 1 \leq i \leq N, 1 \leq d \leq D\}$ , and writes the mean link strength as

$$E(w_{ij}|X, \theta)^{-1} = \theta_0 \prod_{d=1}^D \theta_d^{I\{X_{id}=X_{jd}\}}.$$

In the example, the possible values of  $X_{i1}$  might be codes for different neighborhoods in which children live, while  $X_{i2} = 1$  or  $2$  according as whether child  $i$  is a boy or girl.

Note that a multidimensional model is strictly more powerful than a one-dimensional model. In an example where the first clustering divides women and men, and the second Democrats and Republicans, there are in a sense four clusters, but the multidimensional model allows, for instance, Republican women to be more distant from Democratic men than from Democratic women.

## 5.3 Nested models

A particular form of multidimensional model involves nesting: here, we begin with a division of the nodes into clusters, then subdivide these large clusters into smaller clusters. The model is then tree-structured. The formula for the means in a two-dimensional example is

$$E(w_{ij}|X, \theta)^{-1} = \theta_0 \theta_1^{I\{X_{i1}=X_{j1}\}} \theta_2^{I\{X_{i1}=X_{j1} \text{ and } X_{i2}=X_{j2}\}}.$$

This model might be particularly sensible in large software applications, when one looks for modularity at several different levels. This sort of model can be fit all at once, or alternatively one may fit the outer clustering first, then reduce the data set to the nodes in one of the clusters, and fit a new model within that cluster.

Hierarchical clustering is a technique for estimated nested levels of clusterings in data. The measures of strengths of clustering and their uncertainties that the method here provides may add an extra dimension to hierarchical clustering analyses. See Wasserman and Galaskiewicz (1994) for discussion of use of hierarchical clustering in social network applications, as well as their references.

## 5.4 Some parameters of interest in social network research

In social network applications, it is often of interest to define network- and node-level parameters. Many of these are applicable to dichotomous links, but some interesting node-level parameters can

easily be incorporated into the present framework. For example, from Table 2 in Wasserman and Pattison (1996), “differential choice”  $\gamma_i$  measures the overall level of connections including node  $i$ . An analogue of this parameter can be introduced into the model by writing

$$E(w_{ij}|X, \theta, \gamma)^{-1} = \theta_0 \gamma_i \gamma_j \theta_1^{I\{X_i=X_j\}}.$$

Much attention in social network research has been devoted to “blockmodels” in which subsets of nodes are grouped together if they have similar relationships to other nodes. These subsets might be known *a priori*, in which case the data can be used to measure the goodness of fit of the blockmodel, but constructing and evaluating a blockmodel using the same set of data can be problematic; see Wasserman and Anderson (1987). A modification of the analysis presented here could be very well suited to this problem. Let  $N_0 < N$  be the number of blocks that will be allowed in the blockmodel; presumably it will make sense to specify a prior distribution for  $N_0$ . The problem is then one of estimating block labels  $X_i$  so that

$$E(w_{ij}|X, \theta)^{-1} = \theta_{X_i X_j},$$

where  $\{\theta_{k\ell} : 1 \leq k \leq N_0, 1 \leq \ell \leq N_0\}$  is a set of link strength parameters satisfying  $\theta_{k\ell} = \theta_{\ell k}$ .

## 6 Software Example Revisited

We now return to the motivating example of software data as depicted in Figure 1. The software here is part of a commercially successful telecommunications product (a telephone switching system) that, over its twenty year history, employed more than five thousand developers and is millions of lines of code long. The switch software consists of roughly fifty “subsystems,” which concentrate parts of the software functionality and which coincide with organizational divisions of developers. Subsystems are further broken down into collections of files called modules, which correspond to directories in the source code repository. Much of the code is written in the C language.

The data that go into the analysis are a list of “modification requests,” which are collections of changes to the code that have a common purpose, and which touch multiple modules of the code. Each node here will represent a module. We restrict analysis to one subsystem at a time. One could also perform the following analysis at the file level by allowing each node to represent a file, but there are roughly 4000 files in the subsystem under consideration, and they tend not to be altered often enough in a year to result in interesting year-to-year patterns.

The weights that go into the algorithm are computed as follows. Let  $N_i$  be the number of changes touching module  $i$ , and  $N_{ij}$  be the number of changes touching both modules  $i$  and  $j$ . Then

$$w_{ij} = N_{ij}(N_i N_j)^{-1/4}.$$

We use the fourth root as a compromise between the square root, which guarantees weights between zero and one but which tends to lead to modules touched a small number of times in total having the largest weights, and the zero power, which contains no normalization for total numbers of changes at all. Note that the changes in fact may touch more (or less) than two modules. The reduction of these data to a binary form is potentially a flaw: a change touching  $n$  modules, for

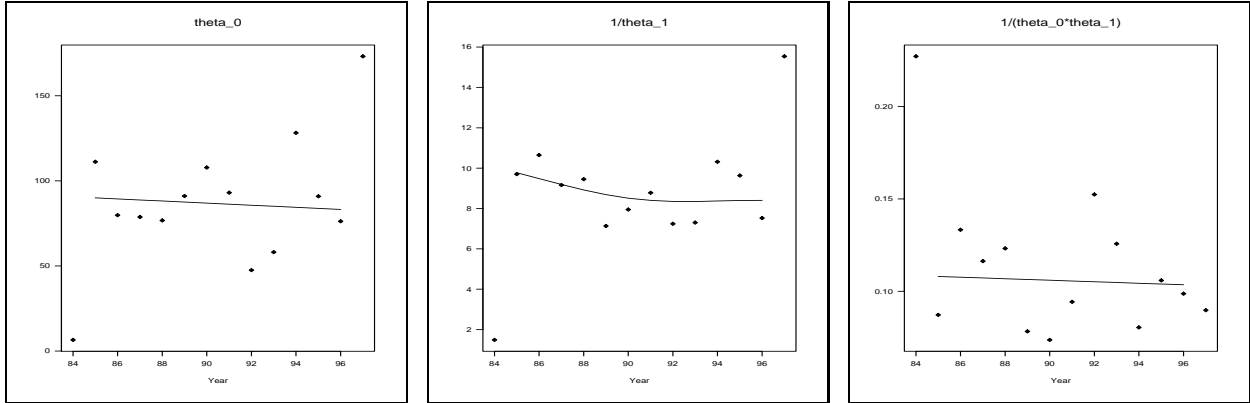


Figure 2: Estimated values of the link strength parameters  $\theta$  through time, with smoothing splines with cross-validated degrees of smoothness fit to the data from years 1985 through 1996. Left:  $\theta_0$  had only a negligible decrease through time after removing the extreme years. Center: The decrease in  $\theta_1^{-1}$  through time is also negligible. Right: nor is there an important trend in  $(\theta_0\theta_1)^{-1}$ , the mean weight between nodes in the same cluster.

instance, appears in the data  $n(n-1)/2$  times instead of once. To construct the plots in Figure 1, the data corresponding to a given year include all the changes up to and including that year. In the analysis that follows, the data for a given year are only the changes that actually took place in that year. The emphasis on data taken from the history of changes to the code helps distinguish this work from Hutchens and Basili (1985), who used analysis of the code itself.

We study the subsystem depicted in Figure 1 to see if we can verify the visualization’s suggested deterioration in the modularity of the software in this subsystem.

The subsystem had data over fourteen years, 1984-1997. For most purposes we will discard 1984 and 1997 because only five of the modules were modified in 1984, while the 1997 data are from only part of the year and are otherwise unusual. It contains one hundred modules, not all of which are under active development, and many of which were absent from the data of at least some of the years. For each year’s worth of data, we ran the Monte Carlo Markov chain algorithm for 100 iterations, obtained a clustering from the voting method described in Section 3, and computed the median values of  $\theta_0$  and  $\theta_1$ .

The changes in median values of  $\theta_0$  over time can indicate whether some sort of modularity is having success by preventing modules in different clusters from being changed together often. If  $\theta_0$  were to decrease over time, it would be cause for concern, and after removing the anomalous years in 1984 and 1997, the decrease is only an 0.6 per year, which seems small compared to variability; see also Figure 2 (left).  $\theta_1^{-1}$  is a unitless measure which compares within-cluster links to between-cluster links. As seen in Figure 2 (center), this coefficient also remains stable. Had it decreased, it would have been an indication that the clustering was growing less important.  $(\theta_0\theta_1)^{-1}$  measures the cohesion between nodes in the same cluster; see Figure 2 (right). There is also no trend in this measure.



Year	84	85	86	87	88	89	90	91	92	93	94	95	96	97
$\theta_0$	7	111	80	79	77	91	108	93	48	58	128	91	76	173
$sd(\theta_0)$	2.2	6.6	2.7	4.2	1.6	2.5	2.8	4.0	1.2	1.7	3.5	2.5	3.0	7.7
$\theta_1$	.67	.10	.094	.11	.11	.14	.13	.11	.14	.14	.097	.10	.13	.064
$sd(\theta_1)$	.227	.008	.009	.009	.004	.007	.007	.007	.013	.007	.004	.010	.011	.007
$ C_1 $	1	22	15	20	37	39	28	31	18	23	29	15	18	9
$ C_2 $	1	11	11	6	4	5	7	6	10	10	5	7	7	7
N	5	44	54	43	62	70	59	58	59	63	60	53	42	37

Table 5: Results from fitting for software data. Values of the  $\theta$ 's are medians over the 100 iterations.  $|C_1|$  is the size of the largest cluster,  $|C_2|$  the size of the second largest cluster.  $N$  is the total number of modules which were modified during that year.

Another potential indication of worsening modular structure would be if, as years passed, a larger proportion of the nodes were inside the largest cluster. Alternatively, if the size of the second largest cluster tended to decrease, it might indicate breakdown of the modularity from a situation with two separated clusters to one where all files need to communicate. For this data set, the two years with the largest primary clusters are 1988 and 1989, arguing that at that time there might have been beginnings of a problem but it has since gone away. Not surprisingly, the same period contains the smallest secondary clusters. See Table 5.

It is of great interest to measure the extent to which fitted clusterings from different years match. Methods for doing this besides the one presented here exist, such as in Fowlkes and Mallows (1983), which bases a measure on the conditional probability that two nodes are in the same cluster according to one clustering, given that they are in the same cluster according to the other clustering. Hubert and Arabie (1985) discuss measures attributed to Rand (1971) based on pairs of nodes placed either in the same cluster or in different clusters by both clusterings, as well as presenting their own measures based on concordant and discordant triples of nodes.

In the absence of an algorithm which models all years' data simultaneously and allows clusterings to evolve slowly over time, we can compare fitted values as follows. Pick a cluster  $C_1$  of nodes found in one year, and a cluster  $C_2$  from another year, and construct the two-by-two table (Table 6) of data from modules changed in both years. One can then compute the log-odds-ratio analogue  $\log(X_{11}X_{22}/X_{12}X_{21})$ , perhaps after first adding one to each cell to prevent infinities and lessen the impact of small clusters. It is also possible to standardize this quantity by dividing by  $(X_{11}^{-1} + X_{12}^{-1} + X_{21}^{-1} + X_{22}^{-1})^{1/2}$  (which can be computed using the delta method) to have a quantity asymptotically comparable to the standard normal distribution. One can then generate measures of the degree of agreement between two years' clusterings by choosing the  $C_2$  from the available clusters from the second year which maximizes the statistic above.

In the current work I have computed such indices for each pair of years, and kept  $C_1$  fixed as the largest cluster in the first year, optimizing over  $C_2$ .

These computations show, first of all, that the indices for a pair of years tend to be large if the years are close in time. See Figure 3. Another interesting feature is that pairs of years separated by

$X_{11} =  \{i : i \in C_1 \cap C_2\} $	$X_{12} =  \{i : i \in C_1 \setminus C_2\} $
$X_{21} =  \{i : i \in C_2 \setminus C_1\} $	$X_{22} =  \{i : i \notin C_1 \cup C_2\} $

Table 6: Two-by-two table useful for comparing clusterings from two different years.

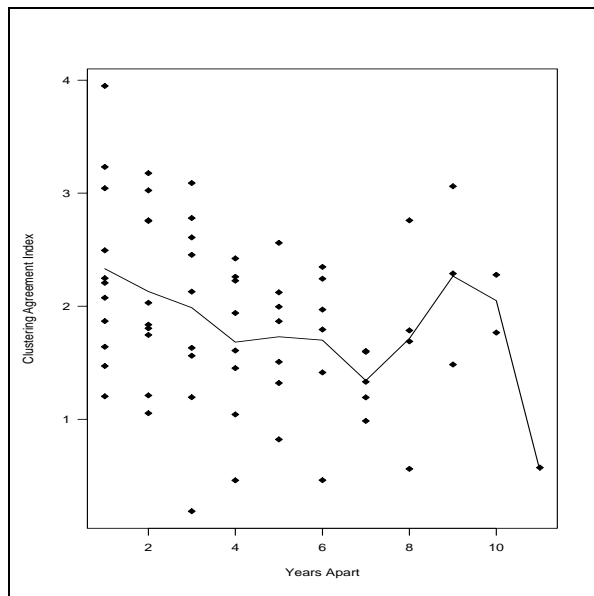


Figure 3: Strength of relationship between clusterings from pairs of years, as a function of the time between the years. The curve is a smoothing spline with cross-validated number of degrees of freedom. Clusterings are similar in consecutive years ( $x = 1$ ), slightly less so with one or two intervening years, but also when data are separated by eight to ten years, as the original modular structure made a bit of a comeback in recent years.

eight to ten years also tend to have similar clusters. Collections of files that were relevant between 1994 and 1995 were closely related to clusters in 1985 and 1986, although those clusters had been less relevant in intervening years.

A clearer depiction of the return in 1994-5 to the design principles of 1985 can be obtained by using the fixed clusters idea described in §5.1. Here we explore the hypothesis that the architecture has drifted away from its original intention. We do this by estimating a clustering using 1985 data, and fitting models to data from other years to see if this clustering is a good match to those data as well.

The results are presented in Table 7, which gives the estimated values of  $\theta_0$  and  $\theta_1$  for each year. The values of  $\theta_1$  can be interpreted as measures of how relevant the clustering is, with small values meaning that the cluster is very relevant since links within clusters are then much stronger than links across clusters. Not surprisingly,  $\theta_1$  is smallest in 1985, the year whose data generated

Year	85	86	87	88	89	90	91	92	93	94	95	96
$\theta_0$	116	42	22	12	22	27	18	20	22	32	40	29
$\theta_1$	.10	.35	.48	.86	.72	.62	.71	.66	.61	.42	.52	.67
$sd(\theta_1)$	.008	.033	.054	.068	.051	.060	.063	.067	.044	.040	.050	.053

Table 7: Results from fitting 1985 clusters to data from various years. Standard deviations computed from last 40 out of 50 iterations.

the clusters. The 1985 clustering is still relatively closely related to the 1986 data, and its relevance seems to disappear over the next two years, before eventually regaining some of its power in 1994-5.

In summary, we were not able to confirm the visualization’s suggestions that the quality of modularization of the code has deteriorated. Decreasing values of  $\theta_1^{-1}$  through time would have argued that modularity was worsening; if this is happening, it is not at all at an alarming rate. If the largest cluster found by the algorithm had tended to be larger in later years, this might also have indicated that the system was growing to resemble a collection of modules with no helpful substructure. This also did not happen. Fitting the clusters from 1986 to data from subsequent years suggested that the original design principles grew less important for a few years, but have since recovered some of their importance. The difference in results between the visualization and the modeling algorithm is partly due to the fact that the data that went into a year’s visualization included all changes up to and including that year, while the data that went into BEACON used one year’s data at a time.

## 7 Conclusions

We have introduced a model designed to identify clusters of objects in data which are strengths of links between those objects, and indicated how to estimate the parameters of that model using Markov chain Monte Carlo, a technique implemented in the tool BEACON. Simulation experiments indicated that the algorithm was quite successful at recovering true cluster structures, except when the importance of the clusters (as measured by the discrepancy between typical within-cluster links and typical across-cluster links) is small. Simulation demonstrated other encouraging qualities of the algorithm, including indications that local minima are not a particular problem, and that the algorithm is to some extent robust to the distribution of the link strengths (although some distributions make the problem unavoidably harder). Further strengths of this formulation are the many convenient extensions: measuring the importance of known clusters, and identifying multiple types of clustering mechanisms, particularly nested clusterings. This model is very well suited to software modularity data, as illustrated by an example data analysis given here. We studied a software system and showed that there is no reason to worry about the decline in its modularity, contrary to the indications of a visualization of the data, which was run on cumulative, instead of yearly, data. The model can also make contributions to the field of social networks, where it has the potential to address important issues of estimation of blockmodels with minor modifications.

## References

- [1] Fowlkes, E. B., and Mallows, C. L., (1983), "A method for comparing two hierarchical clusterings," *Journal of the American Statistical Association*, 78, 553-569.
- [2] Freeman, L., (1998), "Visualizing Social Networks," available on-line at [eclectic.ss.uci.edu/~lin/gallery.html](http://eclectic.ss.uci.edu/~lin/gallery.html).
- [3] Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B., (1995), *Bayesian Data Analysis*, London: Chapman and Hall.
- [4] Holland, P. W., and Leinhardt, S., (1981), "An exponential family of probability distributions for directed graphs," *Journal of the American Statistical Association*, 76, 33-65.
- [5] Hubert, L., and Arabie, P., (1985), "Comparing partitions," *Journal of Classification*, 2, 193-218.
- [6] Hutchens, D. H., and Basili, V. R., (1985), "System structure analysis: clustering with data bindings," *IEEE Transactions on Software Engineering*, SE-11(8), 749-757.
- [7] Kruskal, J. B., and Wish, M., *Multidimensional Scaling*, Newbury Park, CA: Sage.
- [8] Rand, W. M., (1971), "Objective criteria for evaluation of clustering methods," *Journal of the American Statistical Association*, 66, 846-850.
- [9] Tanner, M. A., (1993), *Tools for Statistical Inference*, New York: Springer-Verlag.
- [10] Wasserman, S., and Anderson, C., (1987), "Stochastic a posteriori blockmodels: construction and assessment," *Social Networks*, 9(1), 1-36.
- [11] Wasserman, S., and Faust, K., (1994), *Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences)*, Cambridge, UK: Cambridge University Press.
- [12] Wasserman, S., and Galaskiewicz, J., eds., (1994), *Advances in Social Network Analysis: Research in the Social and Behavioral Sciences (Sage Focus Editions 171)*, Thousand Oaks, CA: Sage.
- [13] Wasserman, S., and Pattison, P. (1996), "Logit models and logistic regressions for social networks: I. An introduction to Markov graphs and  $p^*$ ," *Psychometrika*, 60, 401-426.
- [14] Wills, G. J., (1998), "Nicheworks- Interactive visualization of very large graphs," *Journal of Computational and Graphical Statistics*, to appear.
- [15] Wong, G. Y., (1987), "Bayesian methods for directed graphs," *Journal of the American Statistical Association*, 82, 140-147.