Stochastic Equivalence Clustering Using Random Dot Product Graphs

David J. Marchette[*] Carey E. Priebe[*†]

Abstract

The random dot product graph provides a model for random graphs such that the probability of an edge between two vertices is a function of the dot product of vectors associated with the vertices. Given a graph, one can find vectors so that their dot products approximate the adjacency matrix, in a sense to be defined below. Using these vectors, we provide a clustering of the vertices, so that vertices with similar probabilistic structure (edge probabilities) are grouped together. This type of block modeling is referred to as stochastic equivalence, and we discuss variants of the method for use with large graphs.

1 Introduction

The random dot product graph provides a model for social networks that associates a vector to each vertex in the graph ($v_i \leftarrow x_i$), such that the probability of an edge is a function of the dot product:

$$P[v_i v_j \in E] = f(x_i \cdot x_j).$$

Usually f is taken to be a threshold, or some simple mapping $f : \mathbb{R} \to [0, 1]$. See Kraetzl et al. and references therein. In Marchette and Priebe [2007], a method is suggested for specifying that a fixed (small) number, K, of vectors are used, each vertex receiving one of these vectors. This provides an immediate clustering of the vertices into K clusters.

Other approaches are possible. The most obvious is to simply cluster the $\{x_i\}$ using whatever clustering method that one thinks appropriate. The only caveat is that one should use the dot product similarity in place of a Euclidean or other distance, if one wishes the clusters to be associated to the edge probabilities.

Given a graph on n vertices, one can use linear algebra to find vectors $\{x_i\}$ such that the Frobenius norm of the difference between the adjacency matrix

[*]Naval Surface Warfare Center, Code Q21, Dahlgren, VA 22448. dmarchette@gmail.com

[†]Department of Applied Mathematics and Statistics, Johns Hopkins University, Baltimore, MD 21218. cep@jhu.edu

and the dot product matrix of the vectors is minimized. A maximum likelihood approach is also possible, discussed in Marchette and Priebe [2007], but we will not consider this here. The linear algebra algorithm of Kraetzl et al. is shown as Algorithm 1.

```
input     : $A$, the $n \times n$ adjacency matrix of the graph.
initialize: Choose $p$, the dimension of the output vectors.
Set $D = 0_{n\times n}$.
while Not converged do
    $X = g_p(A + D)$
    $D = \text{diag}(X^T X)$
end
output    : $X$
```

Algorithm 1: Algorithm to fit vectors to an adjacency matrix

Here we define $g_p(M)$ to be the first p singular (column) vectors of the singular value decomposition of the matrix M, and *diag*(M) to be the diagonal matrix whose diagonal entries are those of M. The $\{x_i\}$ are the rows of X. Note that Algorithm 1 can be run on any square matrix, and we will use this fact below without comment. So $\text{kns}(M; p)$ will mean Algorithm 1 run on the matrix M. We will overload the kns function rather than giving different names to each variant. The arguments to the function will indicate which version is meant. We hope this does not cause too much confusion.

We define a clustering (C) into K clusters to be an assignment of an integer in $\{1, \dots, K\}$ to each vertex of the graph. Some of the algorithms discussed below return a hierarchy of clusterings, and for the purposes of comparison in the examples we select the clustering resulting in a fixed number of clusters from the hierarchy.

Given a graph G, we define $\text{kns}(G; p)$ to be the above algorithm applied to the adjacency matrix of G (p will always refer to the dimensionality of the returned $\{x_i\}$). Given a graph G and a clustering (C), we define $\text{kns}(G, \mathcal{C}; p)$ to be as shown in Algorithm 2. The idea is simple: run Algorithm 1 on the appropriate matrix, defined by the clustering.

```
initialize:  Given a graph $G$ with adjacency matrix $A$, and a
             clustering $\mathcal{C}$ on the $n$ vertices of $G$.
Set $M$ to be the $|\mathcal{C}| \times |\mathcal{C}|$ matrix defined by averaging the entries of $A$
according to their clustering in $\mathcal{C}$ (the rows and columns in a cluster
are replaced by their average)
output    : $\text{kns}(M; p)$.
```

Algorithm 2: Algorithm to fit vectors to an adjacency matrix.

Each element of a cluster then receives the appropriate returned vector.

Note that there is a subtle change to the algorithm that we have not made explicit here. The D matrix in the algorithm is needed because the diagonal of an adjacency matrix is 0, and we are not interested in finding vectors which fit this diagonal. Thus, D allows the algorithm to simultaneous fit the vectors to the off-diagonal terms and choose an appropriate diagonal. In the case of Algorithm 2, any cluster containing more than one vertex results in a diagonal element that we *do* want to fit. Thus, Algorithm 1 must be modified so that only the D entries corresponding to singleton clusters are non-zero. Note further that a consequence of this is that a clustering in which all clusters contain at least 2 elements requires a single call to **svd**; no iteration is necessary. We leave it to the reader to make the necessary modifications to the algorithms.

In this paper we discuss several variants of a clustering algorithm that can be used to group the vertices according to their stochastic structure. Issues related to the problems of the analysis of large graphs are discussed, and we provide simulation examples for the different algorithms.

2 Agglomerative Hierarchical Clustering

Agglomerative clustering is a common approach to hierarchical clustering. The basic algorithm is shown in Algorithm 3.

```
input     : $n$ $p$-dimensional vectors $\{x_i\}_1^n$, and a measure of
            dissimilarity of sets $d : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$. Here $\mathcal{P}$ corresponds to
            the power set of $\{x_i\}_1^n$.
initialize: Start with $k = n$ clusters, each $x_i$ in it's own cluster:
            $C_i = \{x_i\}$.
while $k > 1$ do
    Find $(i, j) = \arg\min_{(i,j)} d(C_i, C_j)$.
    Merge clusters $C_i$ and $C_j$: $C_i = C_i \cup C_j$, remove $C_j$.
    Set $k = k - 1$.
end
output    : The hierarchy of clusterings.
```

Algorithm 3: Agglomerative clustering algorithm.

The similarity version of this algorithm is obvious: simply replace the dissimilarly d with the similarity s and the $\arg\min$ with $\arg\max$.

We can now define a simple hierarchical clustering algorithm for graphs (Algorithm 4).

Note that this requires a choice of p, the dimensionality of the vectors, and that this choice is global throughout the clustering. In fact, the graph appears only initially, in the definition of the vectors to be clustered. Note however that at each stage of the algorithm there is a graph (actually a weighted graph)

```
initialize: Given a graph $G$ of order $n$ and an integer $p < n$.
Define a similarity or dissimilarity on sets of $p$-dimensional vectors.
Cluster $\mathrm{kns}(G; p)$, using the agglomerative clustering algorithm above.
output   : The hierarchy of clusters.
```

Algorithm 4: Hierarchical clustering for graphs.

defined by the M matrix in the $\mathrm{kns}(G, \mathcal{C}; p)$ function, although this (weighted) graph is not explicitly used (or computed) by the algorithm. It might seem reasonable that different values of p are appropriate for different of these (weighted) graphs. This is the basic idea that we wish to investigate in this paper.

Figure 1 shows the results of using different values of p in estimating the clusters. The simulation parameters are as follows:

- There are 6 clusters, defined according to their within and between cluster probabilities. These are drawn at random from $[0.5, 0.9]$ and $[0, 0.4]$ respectively. This results in the 6×6 matrix P.
- The singular value decomposition is used to define 6 6-dimensional vectors whose dot product is (essentially) equal to the probability matrix. The choice of dimensionality equaling the number of clusters is for convenience in running the simulations; it is necessary to pick the dimension at least that of the rank of the probability matrix, and rather than take the time to compute this, we chose to use the maximal dimension necessary.
- A random dot product graph is constructed using these vectors, where each cluster consists of 50 vertices.
- Using the graph, vectors are estimated using $\mathrm{kns}(G; p)$ for various values of p (see Figure 1). Cluster using one of the clustering algorithms discussed in this paper.
- Using the vectors and the clustering, a new probability matrix M is defined as the dot product of the vectors, and the error is reported as $\sum(M - P)^2$.
- This is repeated 100 times.

The simulation is thus: choose a probability matrix representing the within- and between-cluster probabilities; select vectors that produce the desired matrix; construct a random dot product graph with 50 vertices per cluster; use the resulting graph to estimate vectors and produce a clustering; use the estimated vectors and clustering to produce an estimate of the probability matrix; report the error between the true probability matrix and the estimate.

We can see the sampling effect in the difference between the baseline and 0. This is the effect of estimating the probabilities by 50 Bernoulli samples, and would decrease for larger graphs. The difference between the baseline and the estimates shows the estimation error resulting from the clustering: if we knew the clustering perfectly, and used this information to estimate a single

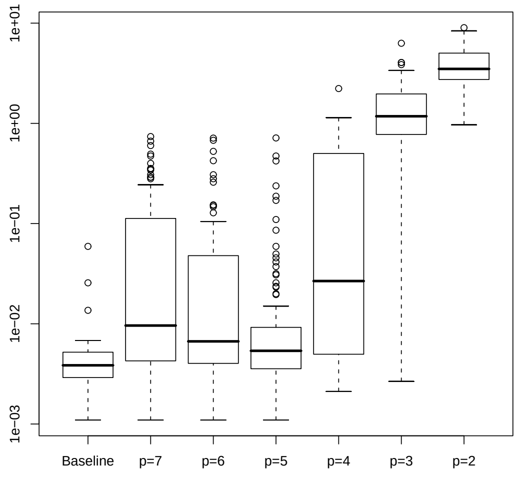

Figure 1: The error (on a log scale) for several choices of dimension p. The true dimensionality is 6. The baseline error is that between the probabilities defined by the adjacency matrix and the true probabilities. 100 simulations were run, with $K = 6$ clusters, and an agglomerative clustering cut at $K = 6$ is used. There are 50 vertices per cluster, so each graph is of order 300.

vector for each cluster, we would have the baseline error. The fact that we estimate a different vector per vertex adds error (see, for example, Trunk [1979] for a discussion of this phenomenon in a different context), and the clustering is imperfect, resulting in the error we see for the $p = 6, 7$ plots. For smaller p, the error in the choice of dimension clearly impacts the quality of the estimate.

Thus, for the best performance we need to know both the number of clusters K and the dimensionality p. If we are willing to estimate the number of clusters from the cluster tree, it seems critical that we have the correct dimension estimate, which in this simulation turns out to be equal to K (although the performance of the $p = 5$ runs indicates that perhaps the resultant matrices tend to be of rank 5 rather than 6, or rather that the gain incurred by the extra value is offset by the error associated with fitting that value (a la Trunk [1979]), and so $p = 5$ may be sufficient).

3 An Adaptive Clustering Algorithm

The basic idea is described in Algorithm 5. At each stage of the agglomerative clustering algorithm, the matrix is reformed, using the current clusters, and new vectors are fit to the resultant matrix.

```
Set a value $p_i$ for each $2 \leq i \leq n$.
Set $k = n$ and $\mathcal{C} = \{1, \ldots, n\}$.
Set $M = A$.   **while** $k > 2$ **do**
    Fit the vectors using $\text{kns}(M, ; p_k)$.
    Find $(i, j) = \arg\min_{(i,j)} d(C_i, C_j)$. Merge clusters $C_i$ and $C_j$:
    $C_i = C_i \cup C_j$, remove $C_j$.
    Set $k = k - 1$.
    Set $M$ to be $A$ averaged according to the clustering $\mathcal{C}$.
**end**
**output** : the hierarchical clustering.
```

Algorithm 5: Adaptive hierarchical clustering for graphs.

This requires $n - 2$ fits from the (weighted) graph to the vectors, that is more than $n - 2$ calls to **svd** (recall that each call to kns results in at least one call to **svd**, more if any diagonal elements are zero). Instead, one might choose to iterate the clustering step several times between fits. In this modification, one refits the vectors after moving a distance down the clustering tree.

We illustrate this algorithm in Figure 2, in which only one adaptation step is performed. In this simulation, the setup is the same as in the previous simulation of Figure 1, except that an initial clustering (using k-means) into 50 clusters is performed using vectors of dimension p. This can be viewed as a version of Algorithm 5 where the iterated agglomerative clustering is replaced by a single

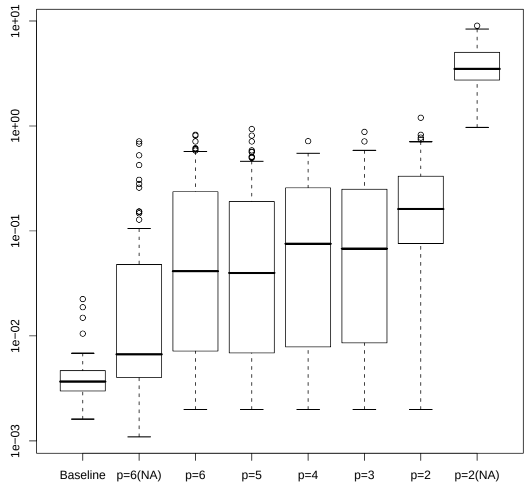

Figure 2: The error (on a log scale) for the adaptive algorithm for several choices of initial dimension p. The algorithm first clusters into 50 clusters using k-means on vectors of dimension p, then clusters into 6 clusters using dimensionality 6. The baseline error is that between the probabilities defined by the adjacency matrix and the true probabilities. The entries with the "(NA)" label refer to the case where no initial clustering is performed, instead 6 clusters are formed using the given value for p. 100 simulations were run, as in Figure 1.

k-means step. Subsequently, using the resulting reduced matrix, we cluster 6-dimensional vectors into $K = 6$ clusters. The "p=6(NA)" and "p=2(NA)" entries are the result of clustering at the given dimension on the full adjacency matrix, and are the same as the corresponding entries in Figure 1.

It is clear that we have gained significantly by performing the initial clustering followed by adaptation. While there is a small cost in performing the adaptation over performing the clustering at the correct dimension, there is little difference between the results at the different initial dimensions.

This has an important consequence for very large graphs. Instead of performing many SVD calculations and retaining many eigenvectors each time, we can initially reduce the number of eigenvectors retained, perform an initial clustering that both reduces the size of the matrix dramatically, and perform the final clustering on this smaller matrix. Note that by clustering so that each cluster contains at least two elements, the diagonal of the resultant probability matrix is filled in with valid entries, and so the iterative step of the fitting algorithm is no longer necessary: the optimal fit is obtained with a single SVD calculation.

We also illustrate the results of running the full algorithm, with an update

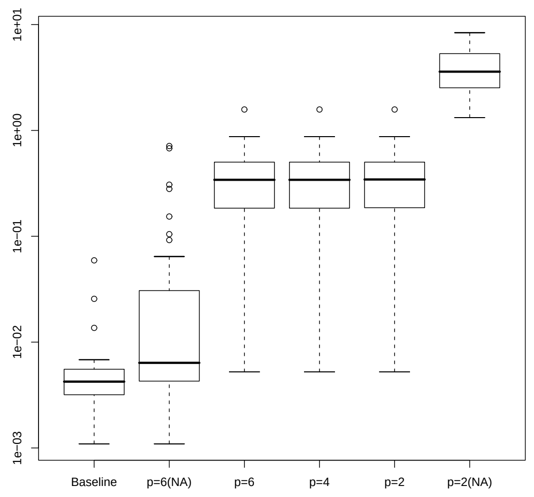

Figure 3: The error (on a log scale) for the adaptive algorithm for several choices of initial dimension p. The algorithm is agglomerative on vectors of dimension p, up until there are 20 clusters, after which the vectors are 6-dimensional. After each step of the agglomerative clustering, the vectors are recomputed using the new matrix. The algorithm stops at 6 clusters and the error is reported. The baseline error is that between the probabilities defined by the adjacency matrix and the true probabilities. The entries with the "(NA)" label refer to the case where no adaptation is performed, instead 6 clusters are formed using the given value for p. 50 simulations were run, with 50 vertices per cluster.

at each step, in Figure 3. Due to the high computational complexity of this algorithm, we only ran 50 iterations, with 50 vertices per cluster. The algorithm computes p-dimensional vectors at each stage, with an initial value of p as indicated in the figure. When the number of clusters reaches 20, the remaining vectors are of dimension 6.

4 A Recursive Hierarchical Clustering

The basic idea is to recursively split the clusters, recomputing the clustering at each stage. This is very much in the spirit of the work described in Priebe et al. [2004] and Giles et al. [2007]. The algorithm is given in Algorithm 6.

```
input   : A value $p$ and a number of clusters $K$.
while A "clusterability" criterion is satisfied do
    Fit the vectors using kns.
    Cluster, using the above agglomerative clustering algorithm (or
    any other clustering algorithm) into $K$ clusters.
    Repeat on each sub-cluster.
end
output  : The resulting hierarchical clustering
```
Algorithm 6: Algorithm to fit vectors to an adjacency matrix

At each stage in the algorithm, the matrix that is used for the clustering changes, as the adjacency matrix is averaged according to the clusters, to produce a matrix M, and new vectors are produced using $\text{kns}(M, \mathcal{C}; p)$. The "clusterability" criterion can be something as simple as ensuring that there are sufficient vertices in the current node to support a clustering into K clusters. Alternatively, a measure such as the ratio of within versus between covariances could be used to decide if the clustering has sufficient support in the data. This algorithm can be adjusted so that the values of K and p are different for different clusters. This also allows some level of user interaction in the algorithm. Users can investigate the (weighted) graphs at each stage, and select p and K for that stage according to intuition, analysis, prior knowledge, or caprice, as they wish.

5 Discussion

We have discussed several variations on the theme of clustering the vertices of a graph using a random dot product model to produce vectors which are then clustered using standard methods. This model is not a complete model for all possible random graphs, but it does seem to fit well to a large number of social networks (see Scheinerman [2005], Kraetzl et al., Marchette and Priebe [2007]). We have shown that it can be important to first perform preliminary clustering, then refit the vectors, followed by a final clustering (or more such steps).

This is particularly important for large graphs. When the adjacency matrix is very large, the singular value decomposition becomes a significant computational burden, and by reducing the number of singular values needed early on, we can reduce the overall complexity of the task without a large cost in quality of the clustering.

The issue of properly choosing the dimensions, especially in Algorithm refalg:agraph is an important and interesting one, which will require further research. In this paper, we suggest a small value for the p_i early on, followed by one which is appropriate to the number of clusters sought. This begs the question of how many clusters to seek, as well as what "early on" and "followed by" should mean in practice. Methods for selecting appropriate values of the p_i (perhaps combining computational constraints with model fitting and/or multi-resolution methodologies) are an area of important future research.

This approach works for those graphs for which the basic dot product model is appropriate. The decision of whether this is the case is one of model selection and goodness of fit, which is notoriously difficult. One rule-of-thumb is that if the dimension of the vectors needed to adequately represent the adjacency matrix is "too large", the model may not be well suited to the graph at hand. The definitions of "adequately represent" and "too large" require some thought, and may be problem-dependent.

Clustering, particularly the choice of the number of clusters, is also a model selection problem, and we have avoided discussion of these issues in this paper. Both the definition of what a "cluster" is and the choice of number of clusters have been pushed to the (traditional) clustering performed on the vectors. This has the advantage that we can simply point to the vast literature on these issues, but the disadvantage that we do not take into account how specifics about the problem defined by the graph might guide the choices. For extensive discussion of these issues see Doreian et al. [2005].

Acknowledgments

This work was funded in part by the Office of Naval Research under the In-House Laboratory Independent Research program.

References

Patric Doreian, Vladimir Batagelj, and Anusška Ferligoj. *Generalized Blockmodeling.* Cambridge University Press, Cambridge, 2005.

Kendall E. Giles, Michael W. Trosset, David J. Marchette, and Carey E. Priebe. Iterative denoising. *Computational Statistics, to appear*, 2007.

Miro Kraetzl, Christine Nickel, and Edward Scheinerman. Random dot product graphs: a model for social networks. submitted.

David J. Marchette and Carey E. Priebe. Modeling interstate alliances with constrained random dot product graphs. *Computational Statistics*, 2007.

Carey E. Priebe, David J. Marchette, and Dennis M. Healy Jr. Integrated sensing and processing decision trees. *IEEE PAMI*, 26, 2004.

Edward Scheinerman. Random dot product graphs, 2005. Talk given at IPAM, www.ipam.ucla.edu/schedule.aspx?pc=gss2005.

G. V. Trunk. A problem of dimensionality: A simple example. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1(3):306–307, 1979.