Computational Statistics and Data Analysis 60 (2013) 12–31

Contents lists available at SciVerse ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

Improved Bayesian inference for the stochastic block model with application to large networks

Aaron F. McDaid [*], Thomas Brendan Murphy, Nial Friel, Neil J. Hurley

Clique Research Cluster, University College Dublin, Ireland

ARTICLE INFO

Article history:
Received 2 April 2012
Received in revised form 22 September 2012
Accepted 27 October 2012
Available online 6 November 2012

Keywords:
Clustering
Social networks
Blockmodeling
Computational statistics
MCMC

ABSTRACT

An efficient MCMC algorithm is presented to cluster the nodes of a network such that nodes with similar role in the network are clustered together. This is known as *block-modeling* or *block-clustering*. The model is the stochastic blockmodel (SBM) with block parameters integrated out. The resulting marginal distribution defines a posterior over the number of clusters and cluster memberships. Sampling from this posterior is simpler than from the original SBM as transdimensional MCMC can be avoided. The algorithm is based on the *allocation sampler*. It requires a prior to be placed on the number of clusters, thereby allowing the number of clusters to be directly estimated by the algorithm, rather than being given as an input parameter. Synthetic and real data are used to test the speed and accuracy of the model and algorithm, including the ability to estimate the number of clusters. The algorithm can scale to networks with up to ten thousand nodes and tens of millions of edges.

1. Introduction

This paper is concerned with *block-modeling*—an approach to clustering the nodes in a network, based on the pattern of inter-connections between them. The starting point for the method presented here is the *stochastic block model* (SBM) Nowicki and Snijders (2001). The goal is to improve the speed and scalability, without compromising on accuracy. We use conjugate priors and integration in order to focus on the marginal distribution of interest, this marginalization is also referred to as the 'collapsing' of the nuisance parameters (Liu, 1994; Wyse and Friel, 2012). This allows us to implement an efficient algorithm based on the *allocation sampler* of Nobile and Fearnside (2007). We incorporate existing extensions, such as the weighted-edge model of Mariadassou et al. (2010), and show how this extension can be incorporated within our collapsing and within our algorithm. As required by the *allocation sampler*, we place a prior on the number of clusters, allowing the number of clusters to be directly estimated. Together, these techniques allow us to avoid the more complex forms of transdimensional MCMC and they also allow us to avoid the need for post-hoc model selection via criteria such as the ICL.

We show that our method can accurately and efficiently estimate the number of clusters—an improvement over many existing methods. Our algorithm, and the data we have used in Section 6.4 and our survey data used in Section 7, are available at http://sites.google.com/site/aaronmcdaid/sbm.

The concept of clustering is broad and originated outside of network analysis, where the input data is in the form of real-valued vectors describing the location of the data points in a Euclidean space. Network clustering takes a set of connected nodes as input and finds a partition of the nodes based on the network structure. This finds application in many different contexts. For instance, in bio-informatics, networks of protein–protein interactions are analyzed and clustering is applied to

[*] Correspondence to: CASL UCD, 8 Belfield Office Park, Clonskeagh, Dublin 4, Ireland. Tel.: +353 85775686.
E-mail address: aaronmcdaid@gmail.com (A.F. McDaid).

0167-9473/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2012.10.021

find functional groups of proteins. Interest in social network analysis has grown greatly in recent years, with the availability of many networks, such as Facebook datasets, of human interactions. Clustering of such social networks has been applied in order to find social communities. In the following, we will distinguish the community-finding problem from the more general setting of block-modeling.

In network analysis, the input data may be described mathematically as a graph, which is a set of nodes (where each node represents an entity, say, a person) and a set of edges linking pairs of nodes together. An edge might represent a friendship on Facebook or a phone call on a mobile phone network. In Section 7, we apply our method to the network of interactions between participants at a summer school.

Given a network, the goal in block-modeling is to cluster the nodes such that pairs of nodes are clustered together if their connectivity pattern to the clusters in the rest of the network is similar. A cluster might, for example, consist of a set of nodes which do not tend to have connections among themselves at all. Given two nodes in this cluster (node i and node j), the neighbors of i tend to be in the same clusters as are the neighbors of j. Community-finding has focussed on finding clusters of high internal edge density, where an edge between two nodes will tend to pull the two nodes into the same cluster, and a non-edge will tend to push them into separate clusters. This contrasts with block-modeling, which allows clusters to have *low* internal edge density. Block-modeling is able to find such community structure, but it is a more general method that is also able to find other types of structure.

A variety of other, non-probabilistic approaches have been used to tackle the broad problem of block-modeling (Everett, 1996; Chan et al., 2011). Outside of block-modeling, there are other solutions for community-finding in networks (Newman and Girvan, 2004; Girvan and Newman, 2002; Newman, 2004). Many probabilistic clustering models have also been applied (Handcock et al., 2007; Hoff et al., 2002; Airoldi et al., 2008).

There is a huge variety of methods, and we will not attempt to summarize them further; for the rest of this paper, we will focus on the SBM and on algorithms for the SBM. For more details, in particular about community finding, see the excellent review article of Fortunato (2010).

The remainder of the paper is structured as follows. In Section 2, we define the SBM and define the notation used in the paper. We then define, in Section 3, the conjugate priors and integration that we use in order to access the relevant marginal distribution. Section 4 discusses other closely-related models and algorithms and in particular gives consideration to the issue of how to select the number of clusters (model selection), comparing the approach we have used to other approaches and noting connections among the methods.

Section 5 describes the algorithm we use; without collapsing, it would have been necessary to use full Reversible Jump MCMC (Green, 1995) to search a sample space of varying dimension and this could be much slower.

In Section 6, we evaluate our method on synthetic networks, showing how the number of clusters can be estimated accurately and the nodes assigned to their correct cluster with high probability. We also test the scalability and efficiency of the algorithm by considering synthetic datasets with ten thousand nodes and ten million edges.

In Section 7, we evaluate our method on a dataset of interactions, gathered by a survey of participants at a doctoral summer school attended by one of the authors of this paper. The method is able to detect interesting structures, demonstrating the differences between *block-modeling* and *community-finding*. Section 8 draws some conclusions.

2. Stochastic block model (SBM)

As formulated in Nowicki and Snijders (2001), a network describes a relational structure on a set of nodes. Each edge in the network describes a relationship between the two nodes it links. A general case of a finite alphabet of states relating a pair of nodes is considered but in the simplest case, discussed by the same authors in Snijders and Nowicki (1997), relationships are binary—an edge joining a pair of nodes either exists or not. The network can be undirected, corresponding to symmetric relationships between the nodes, or may be *directed*, where a relationship from node i to node j does not necessarily imply the same relationship exists from node j to node i. Finally, a *self-loop* – a relationship from node to itself – may or may not be allowed.

Throughout the paper, we use $\mathrm{P}(\cdot)$ to refer to probability mass (i.e. of discrete quantities) and $\mathrm{p}(\cdot)$ to refer to probability density (i.e. of continuous quantities). N is the number of nodes in the network and K is the number of clusters. In the algorithm proposed in Nowicki and Snijders (2001), these are given input values, although in our approach, we treat K as a random variable with a given prior distribution. Given N and K, the SBM describes a random process for assigning the nodes to clusters and then generating a network. Specifically, the cluster memberships are represented by a random vector z of length N such that $z_i \in \{1, \dots, K\}$ records the cluster containing node i. z_i follows a multinomial distribution,

$$z_i \overset{i.i.d.}{\sim} \text{Multinomial}(1; \theta_1, \dots, \theta_K),$$

such that θ_i is the probability of a node being assigned to cluster i ($1 = \sum_{k=1}^{K} \theta_k$). The vector θ is itself a random variable drawn from a Dirichlet prior with dimension K. The parameter to the Dirichlet is a vector $(\alpha_1, \dots, \alpha_K)$ of length K. We follow Nowicki and Snijders (2001) by fixing the components of this vector to a single value α, and by default $\alpha = 1$,

$$\theta \sim \text{Dirichlet}(\alpha_1 = \alpha, \alpha_2 = \alpha, \dots, \alpha_K = \alpha).$$

This describes fully how the N nodes are assigned to the K clusters. Next we describe how, given this clustering z, the edges are added between the nodes.

A network can be represented as an $N \times N$ adjacency matrix, x, such that x_{ij} represents the relation between node i and node j (taking values 1 or 0 in the binary case). Denote by $x_{(kl)}$ the submatrix corresponding to the *block* of connections between nodes in cluster k and nodes in cluster l. If the network is undirected, there are $\frac{1}{2}K(K+1)$ blocks, corresponding to each pair of clusters; and if the network is directed, there are K^2 clusters, corresponding to each *ordered* pair of clusters.

It is generally simpler to discuss the directed model; unless otherwise stated, the formulae presented here apply only to the directed case. The definitions and derivations can easily be applied to the undirected case, provided that care is taken only to consider each pair of nodes exactly once.

If self-loops are not allowed, then the diagonal entries of x, x_{ii}, are excluded from the model. It is assumed that, given K and z, connections are formed independently within a block so that

$$\mathrm{P}(x|z, K, \pi) = \prod_{k.l} \mathrm{P}(x_{(kl)}|z, K, \pi_{kl}),$$

where

$$\mathrm{P}(x_{(kl)}|z, K, \pi_{kl}) = \prod_{\{i|z_i=k\}} \prod_{\{j|z_j=l\}} \mathrm{P}(x_{ij}|z, K, \pi_{kl}),$$

and the matrix $\pi = \{\pi_{kl}\}$ describes the cluster–cluster interactions. π is a $K \times K$ matrix, but for undirected networks only the diagonal and upper triangle are relevant. Specifically, for binary networks, π_{kl} represents the edge density within the block, and edges follow the Bernoulli distribution,

$$x_{ij}|z, K, \pi \sim \text{Bernoulli}(\pi_{z_i z_j}).$$

Each of the π_{kl} is drawn from the conjugate Beta(β_1, β_2) prior,

$$\pi_{kl} \overset{i.i.d.}{\sim} \text{Beta}(\beta_1, \beta_2).$$

Again we follow Nowicki and Snijders (2001) and choose $\beta_1 = \beta_2 = 1$, giving a Uniform prior.

This completes the description of the Bayesian presentation of the SBM. A different approach is taken in other work, such as that of Daudin et al. (2008), where, using essentially the same model, the goal is to take a point estimate of the parameters, (π, θ), for a given number of clusters K. Specifically, the aim is to find the MLE; the value of $(\hat{\pi}, \hat{\theta})$ which maximizes $\mathrm{P}(x|\pi, \theta, K)$. This is described as the frequentist approach, in contrast to the fully Bayesian approach where a distribution of parameter values is allowed instead of a point estimate. We will return to this issue in a little more detail in Section 4 in order to discuss the practical differences from an algorithmic point of view.

2.1. Data model variations

The model is naturally extended in Nowicki and Snijders (2001) to allow for a finite alphabet of two or more relational states, where instead of using a Bernoulli with a Beta prior for x and π, we can use a Multinomial and a Dirichlet to model this alphabet. The Bernoulli-and-Beta-prior model is just a special case of the Multinomial-and-Dirichlet-prior model. Alternatively, we can allow an infinite support and extend the model to allow for non-negative integer weights on the edges, by placing a Poisson distribution on $\mathrm{P}(x|\pi, z)$, as seen in Mariadassou et al. (2010). Now π_{kl} represents the edge rate and is drawn from a Gamma prior,

$$x_{ij}|z, K, \pi \sim \text{Poisson}(\pi_{z_i z_j})$$
$$\pi_{kl} \sim \text{Gamma}(s, \phi).$$

We do not suggest any default for the hyperparameters s and ϕ. A further extension to real-valued weights is also possible, by using a Gaussian for $p(x|\pi, z)$ and suitable prior on π, following Wyse and Friel (2012). These variations, and others, are described in Mariadassou et al. (2010), but they do not discuss conjugate priors.

The integration approach and algorithm described later in this paper can be applied to many variants of edge model, however we focus in the remainder of the paper on the Bernoulli and Poisson models that are supported in our software.

In summary, given N and K the random process generates θ, z, π and ultimately the network x. The two main variables of interest are the clustering z and the network x. In a typical application, we have observed a network x and perhaps we have an estimate of K, and our goal is to estimate z.

3. Collapsing the SBM

In this section, we show how *collapsing* can be used to give a more convenient and efficient expression for the model. This refers to the integration of nuisance parameters from the model, see Wyse and Friel (2012) for an application to a different, but related, bipartite model. The SBM has been partially collapsed by Kemp et al. (2004), but we will consider the full collapsing of both π and θ. As our primary interest is in the clustering z and the number of clusters K, we integrate out π and θ, yielding an explicit expression for the marginal $\mathrm{P}(x, z, K)$. We emphasize that integration does not change the model, it merely yields a more convenient representation of the relevant parts of the posterior. This integration is made possible by the choice of conjugate priors for π and θ. We treat K as a random variable and place a Poisson prior on K with rate $\lambda = 1$, conditioning on $K > 0$,

$$K \sim \text{Poisson}(1) \mid K > 0, \tag{1}$$

which gives us

$$\text{P}(K) = \frac{\frac{\lambda^K}{K!} e^{-\lambda}}{1 - \text{P}(K=0)} = \frac{1}{K!(e-1)}.$$

We are only interested in these expressions as functions of K and z up to proportionality, as this will be sufficient for our Markov Chain over $(K, z|x)$, and hence we can simply use $\text{P}(K) \propto \frac{1}{K!}$.

The Poisson prior is used in the *allocation sampler*, the algorithm upon which our method is based (Nobile and Fearnside, 2007). This allows the estimation of the number of clusters as an output of the model rather than requiring a user to specify K as an input or to to use a more complex form of model selection. Thus, we have a fully Bayesian approach where, other than N, which is taken as given, every other quantity is a random variable with specified priors where necessary,

$$\text{p}(x, \pi, z, \theta, K) = \text{P}(K) \times \text{p}(z, \theta|K) \times \text{p}(x, \pi|z). \tag{2}$$

With Eq. (2) we could create an algorithm which, given a network x, would allow us to sample the posterior $\pi, z, \theta, K|x$. However, we are only interested in estimates of $z, K|x$. We now show how to collapse π and θ.

Define $\mathbb{R}_+$ to be the set of non-negative real numbers, and write the set of real numbers between 0 and 1 as [0, 1]. Define Θ the *unit simplex* i.e. the subset of $\mathbb{R}_+^K$ where $1 = \sum_{k=1}^{K} \theta_k$. Define Π to be the domain of π. For the Poisson model this is $\mathbb{R}_+^B$ while for the Bernoulli model this is $[0, 1]^B$, where B is the number of blocks.

We can access the same posterior for z and K by *collapsing* two of the factors in Eq. (2),

$$\text{P}(x, z, K) = \text{P}(K) \times \int_\Theta \text{p}(z, \theta|K)\, \text{d}\theta \times \int_\Pi \text{p}(x, \pi|z)\, \text{d}\pi, \tag{3}$$

or, equivalently, using the block-by-block independence $x_{(kl)}|z, K$,

$$\text{P}(x, z, K) = \text{P}(K) \times \int_\Theta \text{p}(z, \theta|K)\, \text{d}\theta \times \prod_{k,l} \int_{\Pi_{kl}} \text{p}(x_{(kl)}, \pi_{kl}|z)\, \text{d}\pi_{kl}. \tag{4}$$

This allows the creation of an algorithm which searches only over K and z. The algorithm never needs to concern itself with θ or π.

Collapsing greatly simplifies the sample space over which the MCMC algorithm has to search. Without collapsing, the dimensionality of the sample space would change if our estimate of K changed; this would require a Reversible-Jump Markov Chain Monte Carlo (RJMCMC) algorithm (see Green, 1995). Finally, if estimates for the full posterior, including π and θ, are required, it should be noted that it is very easy to sample $\pi, \theta|x, z, K$, meaning that nothing is lost by the use of collapsing. Many of the other models described in Section 4 are collapsible, and this may be an avenue for future research.

The integration of Eq. (4) allows an expression for the full posterior distribution to be obtained. Details of the derivation of this expression are given in Appendix. Let n_k be the number of nodes in cluster k. n_k is a function of z. For the Bernoulli model, let y_{kl} be the number of edges that exist in block kl, i.e. the block between clusters k and l. For the Poisson model, y_{kl} is the total edge weight. y is a function of x and z. Let p_{kl} be the maximum number of edges that can be formed between clusters k and l. For off-diagonal blocks, $p_{kl} = n_k n_l$. For diagonal blocks, p_{kk} depends on the form of the network as follows,

$$p_{kk} = \begin{cases} \frac{1}{2} n_k(n_k - 1) & \text{undirected, no self-loops} \\ \frac{1}{2} n_k(n_k + 1) & \text{undirected, self-loops} \\ n_k(n_k - 1) & \text{directed, no self-loops} \\ n_k^2 & \text{directed, self-loops.} \end{cases} \tag{5}$$

The full posterior may be written as

$$\text{P}(x, z, K) \propto \frac{1}{K!} \times \frac{\Gamma(\alpha K) \prod_{k=1}^{K} \Gamma(n_k + \alpha)}{\Gamma(\alpha)^K \Gamma(N + \alpha K)} \times \prod f(x_{(kl)}|z), \tag{6}$$

where the final product is understood to take place over all blocks. The form of the function $f(x_{(kl)}|z)$ depends on the edge model. If Bernoulli, then

$$f(x_{(kl)}|z) = \frac{\text{B}(\beta_1 + y_{kl}, p_{kl} - y_{kl} + \beta_2)}{\text{B}(\beta_1, \beta_2)}, \tag{7}$$

where $B(.,.)$ is the Beta function. If Poisson, then

$$f(x_{(kl)}|z) = \frac{\Gamma(s+y_{kl})\left(\frac{1}{p_{kl}+\frac{1}{\phi}}\right)^{s+y_{kl}}}{\Gamma(s)\phi^s}. \tag{8}$$

4. Related estimation procedures for the SBM

Before defining our algorithm, we look at related work, particularly other methods that are based on the SBM. We will focus on models which are identical, or very similar to, the SBM. Therefore, we will not discuss other models which are loosely related, such as that of Newman and Leicht (2007), or the "degree-corrected" SBM of Karrer and Newman (2011).

All methods discussed here are aimed at estimating z, but they differ in the approach they take to the parameters π and θ and in whether they allow the number of clusters, K, to be estimated. We also discuss the issue of model selection, i.e. how the various methods estimate the number of clusters. This question was avoided in the original paper of Nowicki and Snijders (2001), where the number of clusters is fixed to $K = 2$ in the evaluation.

The method of Daudin et al. (2008) takes a network, x, and number of clusters K, and applies a variational algorithm. Point estimates are used for π and θ, but the clustering z is represented as a distribution of possible cluster assignments for each node. This makes the method analogous to the EM algorithm for the MLE—finding the pair (π, θ) which maximizes $\mathrm{P}(x|\pi, \theta, K)$.

The model used by Zanghi et al. (2008) is a subset of the model of Daudin et al. (2008). The cluster–cluster density matrix, π, is simplified such that it is represented by two parameters λ and ϵ, such that the on-diagonal blocks $\pi_{kk} = \lambda$ and the off-diagonal blocks $\pi_{kl} = \epsilon$ (for $k \neq l$). A Classification EM (CEM) algorithm to maximize $\arg\max_{z,\pi,\theta} \mathrm{P}(x, z|\pi, \theta, K)$ is briefly described in Zanghi et al. (2008) but not implemented. Instead, they implement an *online* algorithm. One node of the network is considered at a time and is assigned to the cluster which maximizes $\mathrm{P}(x, z|\pi, \theta, K)$, updating estimates of π and θ with each addition. Implicitly, their goal is to use point estimates both for the parameters *and* for the clustering, to find $(\hat{z}, \hat{\pi}, \hat{\theta})$ that would maximize $\mathrm{P}(x, z|\pi, \theta, K)$; as such, it is loosely related to the profile likelihood (Bickel and Chen, 2009).

The methods just discussed are based, directly or indirectly, on the frequentist approach of finding the maximum likelihood estimate of the parameters, (π, θ), i.e. the values $\hat{\pi}, \hat{\theta}$ that would maximize the likelihood of the observed network,

$$\mathrm{P}(x|\pi, \theta, K) = \sum_z \mathrm{P}(x, z|\pi, \theta, K).$$

The estimate of z that is used in this frequentist approach is the conditional distribution of z based on this point estimate of the parameters and on the observed network, $z|x, \hat{\pi}, \hat{\theta}, K$. In practice though, it is not tractable to calculate or maximize this likelihood exactly, and hence a variety of different approximations and heuristics have been used.

In a Bayesian method, such as ours, a distribution of estimates for (π, θ) is used instead of a point estimate. The goal is to directly sample from $z|x, K$. Another example of this Bayesian approach is the variational algorithm used in Hofman and Wiggins (2008), which is based on the simpler λ and ϵ parameterization of the π matrix used in Zanghi et al. (2008).

The modeling choices of Latouche et al. (2012), where a new model selection criterion called *ILvB* is introduced, are essentially identical to the standard SBM; each element of π_{kl} is independent, and conjugate priors are specified. A variety of other variational approximations are considered by Gazal et al. (2011), where there is more focus on parameter estimation and less focus on model selection.

A further specialization of this model is possible, by employing the λ, ϵ parameterization, but where $\lambda > \epsilon$, which explicitly constrains the expected edge density within clusters to be larger than the expected edge density between clusters. This can be considered to be *community-finding* as opposed to *block-modeling*. The authors of this paper considered this in McDaid et al. (2012).

4.1. Model selection

Later, in our experiments in Section 6, we will demonstrate the ability of the allocation sampler to accurately estimate the number of clusters. In this subsection, we will briefly discuss some of the theoretical issues around the estimation of the number of clusters.

The methods that involve the MLE for the parameters involve the risk of overfitting; for larger values of K, the parameter space of π and θ becomes much larger and therefore the estimates of $\mathrm{P}_{\theta=\theta_{\mathrm{mle}}}(x|K)$ will become over-optimistic, and will tend to overestimate K (Schwarz, 1978). Therefore, an alternative formulation such as the ICL is needed; see Zanghi et al. (2008) and Daudin et al. (2008) for derivations of the ICL in the context of models based on the SBM. Instead of using the MLE directly, those measures apply priors to the parameters and integrate over the priors, as described in Biernacki et al. (2000), such that the average likelihood is used instead of the maximum likelihood.

Typically, such integrations cannot be performed exactly and the ICL criterion consists of approximations that are based on first finding an estimate to the MLE, and then adding correction terms to this MLE. For the rest of this subsection, we will not consider those approximate methods and will instead consider the exact solutions to the integrations.

The *integrated classification likelihood*, which the ICL intends to approximate,

$$\mathrm{P}(x, z|K) = \iint \mathrm{P}(x, z, \pi, \theta|K)\,\mathrm{d}\pi\,\mathrm{d}\pi,$$

can be solved exactly in some models. The SBM is one of those models, and the posterior mass that our algorithm samples from is exactly equal to the integrated classification likelihood (if a uniform prior is used for K instead of the default Poisson). While it is easy to exactly calculate the integrated classification likelihood for a given (z, K), it would not be tractable to search across all possible (z, K) to find the state that maximizes the integrated classification likelihood, except for the smallest of networks.

The BIC is an attempt to approximate the *integrated likelihood*

$$\mathrm{P}(x|K) = \sum_z \iint \mathrm{P}(x, z, \pi, \theta|K)\,\mathrm{d}\pi\,\mathrm{d}\theta.$$

An exact solution to the BIC is not tractable for the SBM; the likelihood would require a summation over all possible clusterings z.

If we were to use a uniform prior for K, then $\mathrm{P}(x|K) = \mathrm{P}(K|x)$ and an irreducible ergodic Markov chain algorithm such as ours would visit each value of K in proportion to the integrated likelihood for that value of K. Of course, our algorithm only gives a *sample* from the true posterior, and there cannot be any guarantee that the distribution of the sample is representative of the true distribution.

The purpose of these last few paragraphs is to demonstrate that there are other (approximate) ways to calculate the *integrated likelihood* and the *integrated classification likelihood*. The Bayesian methods provide approximations that may, in practice, be at least as good as the approximations that would be provided by methods such as the ICL.

The model-selection criterion *ILvb* Latouche et al. (2012) is based on a variational approximation to a fully Bayesian model. As a result of its Bayesian model, it is an approximation of the integrated likelihood and no further adjustment is required for model selection. As with any variational Bayes method, we assume that the independence assumptions within the variational approximation are a good approximation of the true posterior. A second assumption made by those authors is that the Kullback–Leibler divergence, the difference between the true posterior and the variational approximation, is independent of K. If these two assumptions hold, then the measure they use, which they call the *ILvB*, is equivalent to $\mathrm{P}(x|K)$, the *integrated likelihood*. To select the number of clusters, they use that value of K which maximizes the *ILvB*.

5. Estimation

In this section, we describe our MCMC algorithm which samples, given a network x, from the posterior $K, z|x$. The moves are Metropolis–Hastings moves (Hastings, 1970). We define the moves and calculate the proposal probabilities and close the section with a discussion of the label-switching phenomenon, where we use the method proposed in Nobile and Fearnside (2007) to summarize the clusterings found by the sampler.

Our algorithm is closely based on the *allocation sampler* Nobile and Fearnside (2007), which was originally presented in the context of a mixture-of-Gaussians model. In fact, it can be applied to any model that can be collapsed to the form $\mathrm{P}(x, z, K)$ where x is some fixed (observed) data and the goal is to sample the clustering and the number of clusters (z, K).

In the Gibbs sampler used in Nowicki and Snijders (2001), the parameters are not collapsed, and sampling is from

$$z, \pi, \theta|x, K.$$

In their experiment on the Hansell dataset, K was fixed to 2. As a result of this value for K, θ reduced to a single real number specifying the relative expected size of the two clusters. Expressions were presented for $p(\theta|z, \pi, x, K)$, $P(z|\theta, \pi, x, K)$ and $p(\pi|z, \theta, x, K)$ such that the various elements z_i (or π_{kl}) are conditionally independent of each other, given (π, x, K) (or (z, x, K)), allowing for a straightforward Gibbs sampler.

In contrast, we develop an algorithm that searches across the full sample space of all possible clusterings, z, for all K, drawing from the posterior,

$$z, K|x,$$

using Eq. (6) as the desired stationary distribution of the Markov Chain.

We use four moves:

- *MK*: Metropolis move to increase or decrease K, adding or removing an empty cluster.
- *GS*: Gibbs sampling on a randomly-selected node. Fixing all but one node in z, select a new cluster assignment for that node.
- *M3*: Metropolis–Hastings on the labels in two clusters. This is the *M3* move proposed in Nobile and Fearnside (2007). Two clusters are selected at random and the nodes are reassigned to the two clusters using a novel scheme fully described in that paper. K is not affected by this move.
- *AE*: The *absorb–eject* move is a Metropolis–Hasting merge/split cluster move, as described in Nobile and Fearnside (2007). This move does affect K along with z.

At each iteration, one of these four moves is selected at random and attempted. All the moves are essentially Metropolis–Hastings moves; a move to modify z and/or K is generated randomly, proposing a new state (z', K'), and the ratio of the new density to the old density $\frac{\mathrm{P}(z',K|x)}{\mathrm{P}(z,K|x)} = \frac{\mathrm{P}(x,z',K)}{\mathrm{P}(x,z,K)}$ is calculated. This is often quite easy to calculate quickly as, for certain moves, only a small number of factors in Eq. (6) are affected by the proposed move. We must also calculate the probability of this particular move being proposed, and of the reverse move being proposed. The *proposal probability ratio* is combined with the *posterior mass ratio* to give us the move *acceptance probability*,

$$\min\left(1, \frac{\mathrm{P}(x,z',K')}{\mathrm{P}(x,z,K)} \times \frac{\mathrm{P}_{\mathrm{prop}}((K',z') \to (K,z))}{\mathrm{P}_{\mathrm{prop}}((K,z) \to (K',z'))}\right), \tag{9}$$

where $\mathrm{P}_{\mathrm{prop}}((K,z) \to (K',z'))$ is the probability that the algorithm, given current state (K,z), will propose a move to (K',z').

In the remainder of this section, we discuss the four moves in detail, derive the proposal probabilities and describe the computational complexity of the moves.

5.1. MK

The *MK* move increases or decreases the number of clusters by adding or removing an empty cluster. If *MK* is selected, then the algorithm selects with 50% probability whether to attempt to add an empty cluster, or to delete one. If it chooses to attempt a delete, then one cluster is selected at random; if that cluster is not empty, then the attempt is abandoned. If it chooses to attempt an insert, it selects a new cluster identifier randomly from $\{1, \dots, K+1\}$ for the new cluster and inserts a new empty cluster with that identifier, renaming any existing clusters as necessary.

The proposal probabilities are

$$\mathrm{P}_{\mathrm{prop}}((K,z) \to (K+1,z')) = \frac{0.5}{K+1}$$

$$\mathrm{P}_{\mathrm{prop}}((K',z') \to (K'-1,z)) = \begin{cases} \frac{0.5}{K'} & \text{if } K' > 1 \\ 0 & \text{otherwise.} \end{cases}$$

By adding an empty cluster, K increases to $K' = K+1$ and the posterior mass change is:

$$\frac{\mathrm{P}(x,z,K')}{\mathrm{P}(x,z,K)} = \frac{K!}{(K+1)!} \frac{\left(\frac{\Gamma(\alpha(K+1)) \prod_{k=1}^{K+1} \Gamma(n_k+\alpha)}{\Gamma(\alpha)^{K+1}\Gamma(N+\alpha(K+1))}\right)}{\left(\frac{\Gamma(\alpha K) \prod_{k=1}^{K} \Gamma(n_k+\alpha)}{\Gamma(\alpha)^{K}\Gamma(N+\alpha K)}\right)}$$
$$= \frac{\Gamma(\alpha(K+1))\Gamma(N+\alpha K)}{(K+1)\Gamma(\alpha K)\Gamma(N+\alpha(K+1))}.$$

The computational complexity of this move is constant.

5.2. GS

The Gibbs update move, *GS*, selects a node i at random to be assigned to a new cluster. All other nodes are kept fixed in their current cluster assignment i.e. a single element of the vector z is updated. Denote by $z' = z_{\{z_i \to k\}}$ the modified clustering resulting from a move of node i to cluster k. For each possible value of $z_i \in \{1, \dots, K\}$, z_i is chosen with probability proportional to $\mathrm{P}(x, z_{\{z_i \to k\}}, K)$. The proposal is then accepted. Bear in mind that this move often simply reassigns the node to the same cluster it was in before the *GS* move was attempted.

The calculations involved in *GS* are quite complex as many of the factors in Eq. (6) are affected. The sizes of the clusters are changed as the node is considered for inclusion in each cluster, and the number of edges and pairs of nodes are changed in many of the blocks. The computational complexity is $\mathcal{O}(K^2) + \mathcal{O}(N)$ as every block needs to be considered for each of the K possible moves and every node may be checked to see if it is connected or not to the current node.

The $\mathcal{O}(N)$ term is just a theoretical worst case over all possible networks. Our algorithm iterates over the neighbors of the current node and this is sufficient to perform all the necessary calculations. There is no need to iterate over the non-neighbors and therefore the average complexity is equal to the average degree, which will be much less than N in real-world sparse networks. For small Kk, and assuming a given average degree, the complexity of the *GS* move is independent of N.

5.3. M3

M3 is a more complex move and was introduced in Nobile and Fearnside (2007). Two distinct clusters are selected at random, j and k. All the nodes in these two clusters are removed from their current clusters and placed in a list which is then randomly reordered—call this ordered list $A = \{a_1, \dots, a_{n_j+n_k}\}$, of size equal to the total number of nodes in the two clusters. The software creates a temporary cluster to store these nodes until they are reassigned to the original two clusters.

One node at a time is selected from A and is assigned to one of the two clusters according to some assignment probability. As the nodes are assigned (or reassigned) the new cluster assignments are stored in a list $B_h = \{b_1, \ldots, b_{h-1}\}$, where b_i is the new cluster assignment of node a_i and B_h represents the assignments before the hth node in A is processed.

Iterating through the list A, a_h is assigned to either cluster j or cluster k with probability satisfying

$$p_{B_h}^{a_h \to j} + p_{B_h}^{a_h \to k} = 1,$$

conditional on the nodes B_h that have already been (re-) assigned. Conceptually, any arbitrary assignment distribution can be chosen, as long as the probabilities for each choice are non-zero and sum to one. Once all nodes in the list have been assigned to the two clusters, the proposal probability is given by

$$\mathrm{P}_{\mathrm{prop}}(z \to z') = \prod_{h=1}^{n_j+n_k} p_{B_h}^{a_h \to b_h}.$$

We remark that while the order in which the nodes are reinserted is random, it can be shown that this random ordering does not affect the acceptance probability.

In Nobile and Fearnside (2007), it is proposed to choose the ratio of the assignment probabilities as the ratio of the two posterior probabilities resulting from the assignments of the first h nodes. Specifically, denote by $z_{\{a_h \to l, B_h\}}$, the clustering that assigns the first $h-1$ nodes of A according to B_h and assigns a_h to cluster l. Let $\mathrm{P}(x', z_{\{a_h \to l, B_h\}}, K)$ be the posterior probability of this clustering on the network x' *where all unassigned nodes and edges involving these nodes are ignored*. Then

$$\frac{p_{B_h}^{a_h \to j}}{p_{B_h}^{a_h \to k}} = \frac{\mathrm{P}(x', z_{\{a_h \to j, B_h\}}, K)}{\mathrm{P}(x', z_{\{a_h \to j, B_h\}}, K)}.$$

This heuristic should guide the selection towards 'good' choices. To calculate the proposal probability of the reverse proposal, the list A is again traversed to calculate

$$\mathrm{P}_{\mathrm{prop}}(z' \to z) = \prod_{h=1}^{n_j+n_k} p_{B'_h}^{a_h \to z_{a_h}},$$

where $B'_h = \{z_{a_1}, \ldots, z_{a_{h-1}}\}$.

Our algorithm has been optimized for sparser networks. The complexity of *M3* is made up of three terms. First, it is possible that many or all nodes will be reassigned, causing a complexity of $\mathcal{O}(N)$ while updating the data structure that records the size of each cluster. Second, we keep a record of the number of edges within each block; the *M3* move will consider each edge in the network at most once, as it moves the edge from one block to another, leading to a complexity of $\mathcal{O}(M)$, where M is the number of edges in the network. Finally, once the data structures have been updated, a new posterior mass must be calculated by iterating over each cluster and over each block, querying the summary data structures, to sum the new terms in Eq. (6); this has a complexity of $\mathcal{O}(K^2)$.

Together, this gives a complexity of $\mathcal{O}(N) + \mathcal{O}(M) + \mathcal{O}(K^2)$. The first term may be ignored, since for most networks that are considered here and in the literature, $M > N$. As long as the number of clusters is small, $K^2 \ll M$, the $\mathcal{O}(M)$ term dominates. While in the worst case $M = N^2$, in practice, for the sparse networks we consider, $M \ll N^2$.

5.4. AE

In the *absorb–eject AE* move, a cluster is selected at random and split into two clusters, or else the reverse move can merge two clusters. This move therefore can both change the number of clusters K and change the clustering z. The move will first choose, with 50% probability, whether to attempt a merge or split.

In the case of the split move, one of the K clusters is selected at random. Also, the cluster identifier of the proposed new cluster is selected at random from $\{1, \ldots, K+1\}$. Finally, the nodes in the original cluster are assigned between the two clusters. This is similar to the *M3* move and a heuristic to guide the assignment, as in *M3*, could be considered. Instead, as in Nobile and Fearnside (2007), we use a *probability of ejection*, p_E, selected randomly from a Uniform (0, 1) distribution, such that each node is assigned to the new cluster with probability, p_E. In such as move, the proposal probability is dependent on p_E. Rather than specify an ejection probability, we integrate over the choice of p_E in much the same manner as collapsing.

Given (z, K) and a proposal to split into $(z', K' = K+1)$, where a cluster of size n_k is split into clusters of size n_{j_1} and n_{j_2}, the resulting proposal probability for an eject move is

$$\mathrm{P}_{\mathrm{prop}}((z, K) \to (z', K')) = \frac{\Gamma(n_{j_1}+1)\Gamma(n_{j_2}+1)}{K(K+1)\Gamma(n_k+2)}.$$

For a merge, the proposal probability is simply obtained as the probability of selecting the two clusters for merger from the $K' = K+1$ possible clusters. One cluster is selected which will retain its current nodes and which will expand to contain the nodes in another, randomly selected, cluster,

$$\mathrm{P}_{\mathrm{prop}}((z', K') \to (z, K)) = \frac{1}{K}\frac{1}{K+1}.$$

The complexity is similar to that of the *M3* move.

5.5. *Applying the moves*

In all simulations, discussed in Section 6, the algorithm is seeded by initializing $K = 2$ and assigning the nodes randomly to one of the two initial clusters. The first two moves, *MK* and *GS*, are sufficient to sample the space but have slow mixing. The *AE* move is sufficient on its own as it can add or remove clusters as well as move the nodes to reach any (z, K) state. In practice, we'll see in Section 6 that the combination of *AE* and *M3* is good in the initialization stages to burn-in to a good estimate of both z and K and lessen the dependence on the initialization. It is possible to envisage many possible extensions to these moves. For example, a form of *M3* could be made which selects three clusters to rearrange. The *AE* move could be extended to include the assignment heuristic of the *M3* move.

5.6. *Label switching*

For any given z, with K clusters (assuming they are all non-empty), there are $K!$ ways to relabel the clusters, resulting in $K!$ effectively equivalent clusterings. The posterior has this symmetry and as the MCMC algorithm proceeds it often swaps the labels on two clusters, in particular during the *M3* move. This is known as the *label switching phenomena*. The posterior distribution for any $z_i|x, K$ assigns node i to each of the K clusters with probability $\frac{1}{K}$, so in the long run every node is assigned with equal probability to every cluster. While each z_i is uniformly distributed between 1 and K, the components of z are dependent on each other and pairs of nodes that tend to share a cluster will tend to have the same values at their corresponding component of z. Depending on the context, this may not be an issue of concern. For example, if the aim is to estimate K or to estimate the probability of two nodes sharing the same cluster, see Fig. 3(b), or to estimate the size of the largest cluster, then label switching is not a problem.

However, it sometimes is desirable to undo this label switching by relabeling the clusters, such that nodes are typically assigned to a single cluster identifier along with those other nodes that they typically share a cluster with. Such a relabeling can, for example, make it easier to identify the nodes which are not strongly tied to any one cluster.

We use the algorithm in Nobile and Fearnside (2007) to undo the label switching by attempting to maximize the similarity between pairs of clusterings, after the burn-in clusterings have been discarded. Given two z vectors, at two different points in the Markov Chain, t and u, define the distance between them to be

$$D(z^{(t)}, z^{(u)}) = \sum_{i=1}^{N} I(z_i^{(t)}, z_i^{(u)}),$$

where I is an indicator function that returns 0 if node i is assigned to the same cluster at point t and point u; and returns 1 otherwise.

For each $z^{(t)}$, consider $z^{(*t)}$, one of the $K!$ possible relabeled versions of $z^{(t)}$. The Markov Chain is run for a iterations, discarding the first b iterations as burn-in.

Ideally, the goal is to find the relabeling that minimizes the sum over all pairs of u and t,

$$\sum_{t=b}^{a} \sum_{u=t+1}^{a} D(z^{(*t)}, z^{(*u)}),$$

but it is not computationally feasible to search across the full space of all relabelings. Each state can be relabeled in approximately $K!$ different ways, the precise number depends on the number of non-empty clusters. There are a states altogether, therefore the space of all possible relabelings of all states will have $(K!)^a$ elements; this will be untractable for non-negligible a. In our experiments, a tends to be of the order of one million.

Instead, we use the *online* algorithm proposed in Nobile and Fearnside (2007). It first orders the states from the Markov chain by the number of non-empty clusters. Then, it iterates through the states, comparing each state to all the preceding relabeled states and relabeling the current state such that the total distance to all the preceding relabeled states is minimized.

We will see in Section 7 how this algorithm helps to summarize the output of the Markov Chain. This algorithm is fast. On a 2.4 GHz Intel Zeon in a server with 128 GB RAM, it takes 43 s to process the output of 1 million iterations of that data. In comparison, it takes 610 s to run the SBM MCMC algorithm in order to get the states to feed into the label-unswitching algorithm. Note also that the algorithm doesn't take up much memory—even with a network with 10 million edges, the memory usage doesn't exceed 2 GB.

Once the label-switched set of states is obtained, a posterior distribution of the clustering for each node, $z_i|x$, can be calculated. There is a similarity here with variational methods Daudin et al. (2008); Latouche et al. (2012) as they model the posterior in this manner, where each node's variational posterior is independent of the other nodes' variational posterior. It may be interesting to compare these approximate posteriors to the approximate posterior found by our method.

In the experiments we perform later in Sections 6 and 7, the vast majority of nodes are strongly assigned by this label-switching algorithm to one of the clusters with at least 99% probability in the posterior. Therefore, the distance $D(., .)$ between each state and this 'summary' state is usually quite small. We take this as an indication that the online heuristic has done a reasonable job of minimizing the distance between the states, at least for those networks.

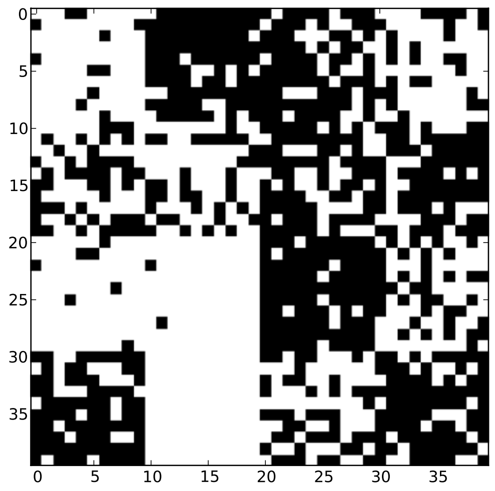


Fig. 1. The adjacency matrix (with $\delta = 0$) for the four-cluster synthetic network used in Section 6.1. Each of the four clusters has 10 nodes.

Table 1
The performance decreases as the noise level, δ, increases. The fifth column, $P(\hat{z} \equiv z|x)$, reports how often the sampler visits the 'correct' answer; i.e. where the visited state was equivalent, subject to relabeling, to the model from which the network was generated.

δ	$P(K = 4\|x)$	$\hat{K}_{\text{mode}}$	$P(K = \hat{K}_{\text{mode}}\|x)$	$P(\hat{z} \equiv z\|x)$	τ
0.0	0.8982	4	0.8982	0.974	50.12
0.1	0.8799	4	0.8799	0.952	63.99
0.2	0.8769	4	0.8769	0.124	80.18
0.3	0.0073	2	0.7865	0.000	371.96
0.4	0.0075	1	0.6293	0.000	1365.58

6. Evaluation

In this section we first look at experiments based on synthetic data and follow in the next section with an application of the collapsed SBM to a survey network gathered by one of the authors at a recent summer school. The synthetic analysis proceeds by generating networks of various sizes from the model and examining whether the algorithm can correctly estimate the number of clusters and the cluster assignments.

As mentioned in the previous section, all our experiments are done on a 2.4 GHz Intel Zeon in a server with 128 GB RAM, and the memory usage never exceeded 2 GB.

6.1. Estimating z

A 40-node directed, unweighted network is generated from the model, containing 4 clusters of 10 nodes each. The block densities π_{kl} are generated by drawing from a Uniform $(0, 1) \equiv \text{Beta}(1, 1)$ for each of the $4 \times 4 = 16$ blocks (see Fig. 1).

To challenge the algorithm further we add noise to the synthetic data, similar to simulation experiment described in Section 4 of Wyse and Friel (2012). The values in the matrix π are scaled linearly. For a given δ, define $\pi_{kl}^{(\delta)} = \delta + \pi_{kl}(1-2\delta)$. While the values in the original π are drawn from the full range, [0, 1], the elements in the matrix $\pi^{(\delta)}$ are in the range $(\delta, 1-\delta)$. Various networks for values of δ between 0 and 0.5 are generated. The original network model corresponds to $\delta = 0$. The network with $\delta = 0.5$ corresponds to an Erdos–Renyi model with $p = 0.5$—this is a random graph model with no block structure.

The algorithm is run for one million iterations, discarding the first 500,000 of these as burn-in.

Table 1 shows how the performance is affected as δ increases. The first column is the posterior probability for the "correct" answer for K, $P(K = 4|x)$. As the value of δ increases, the network approaches the Erdos Renyi model and therefore there is no longer any structure to detect; this explains why the accuracy decreases as δ increases. Next is the modal value of K which maximizes the posterior $P(K|x)$, followed by the posterior probability of the modal value, $P(K = \hat{K}_{\text{mode}}|x)$. The fifth column, $P(\hat{z} \equiv z|x)$ is the probability that the (non-empty) clusters are equivalent (allowing for relabeling) to the clustering used to generate the data. Note that sometimes there are empty clusters in the estimate and therefore $P(\hat{z} \equiv z|x)$ can be bigger than $P(K = 4|x)$.

The final column reports τ, the Integrated Autocorrelation Time (IAT) for the estimate of K, defined as $\tau = 1 + 2\sum_{t=1}^{\infty} \rho(t)$, where ρ_t is the autocorrelation at lag t. As the sampler visits the states, we consider how correlated the estimate of K is with the estimates for preceding states. A low autocorrelation, as summarized by the IAT, is an indicator of good mixing.

Table 2
The rows represent K_{true} and the columns are the estimates from the *ILvb* of Latouche et al. (2012) and from our algorithm.

	3	4	5	6	7
(a) ILvb					
3	100	0	0	0	0
4	0	99	1	0	0
5	0	4	96	0	0
6	0	0	24	76	0
7	0	5	29	41	25
(b) Our algorithm					
3	99	1	0	0	0
4	0	99	1	0	0
5	0	4	96	0	0
6	0	0	25	75	0
7	0	5	27	46	22

Table 3
The true number of clusters (rows) against the number estimated by *ILvb* (columns). The diagonal entries are underlined to aid readability, as these represent the correct answer. We see here a tendency to underestimate the number of clusters, especially for larger K_{True}.

	10	11	12	13	14	15	16	17	18	19	20
10	$\underline{72}$	15	0	1	0	0	0	0	0	0	0
11	15	$\underline{75}$	5	3	1	0	0	0	0	0	0
12	5	20	$\underline{64}$	6	5	0	0	0	0	0	0
13	2	3	21	$\underline{66}$	8	0	0	0	0	0	0
14	0	0	4	21	$\underline{61}$	10	4	0	0	0	0
15	0	0	2	8	28	$\underline{51}$	9	0	2	0	0
16	0	0	1	4	15	32	$\underline{33}$	11	4	0	0
17	0	0	0	2	4	11	30	$\underline{45}$	8	0	0
18	0	0	0	1	3	12	20	30	$\underline{23}$	10	1
19	0	0	0	0	0	1	12	24	38	$\underline{13}$	10
20	0	0	0	0	0	1	7	6	23	29	$\underline{23}$

6.2. Estimating K

We perform three different types of experiments to judge the ability of the algorithm to correctly estimate the number of clusters with networks of increasing size.

First, we repeat the experiments of Latouche et al. (2012). The true numbers of clusters, K_{true} is set to range from 3 to 7. For each K_{true}, 100 networks are randomly generated. The number of nodes in each network, N, is set to 50. The nodes are assigned to the clusters randomly, with $\theta_1 = \cdots = \theta_K = \frac{1}{K_{\text{true}}}$. Two parameters are used to control the density of the blocks. The first, λ, is the density within clusters i.e. $\pi_{kk} = \lambda$. Also, one of the clusters is selected to be a special cluster of 'hubs', well connected to the other nodes in the network, by setting $\pi_{1k} = \pi_{k1} = \lambda$. The second parameter, ϵ, represents the inter-block density of all the other blocks i.e. $\pi_{kl} = \epsilon$ for $k, l \neq 1$. As in the experiments of Latouche et al. (2012), the parameter values are $\lambda = 0.9$, and $\epsilon = 0.1$.

Each network is run through the variational method of Latouche et al. (2012). The estimated value of K which maximizes the *ILvB* measure is taken as the estimate of the number of clusters. A contingency table, showing the true number of clusters against the estimate from *ILvB*, is displayed in Table 2(a).For low K_{true} the algorithm is very accurate, and for larger values there is a tendency to underestimate the number of clusters. For example, when $K_{\text{true}} = 7$, the estimate was $\hat{K} = 6$ for 41 of the networks and $\hat{K} = 7$ for only 25 of the 100 networks.

The results from our algorithm, shown in Table 2(b) are similar to those obtained using the *ILvB*.

6.3. Synthetic SBM networks

The experiments of Section 6.2 involve synthetic data generated according to a model of *community structure*, where edges tend to form primarily within clusters. In order to explicitly test our algorithm in the more general setting of *block structure*, we generated another set of networks with data generated directly from the SBM.

Similarly to the previous experiment, for each of a range of values of K_{true}, 100 networks are generated. K_{True} is now set to range from 10 to 20 and the number of nodes is set to $N = 100$, in order that the size of each cluster not be very small. Each element of π_{kl} is chosen randomly from Uniform (0, 1) and for each of the 100 networks, a new π is created randomly. As these are undirected networks, only the upper triangular portion of π is used when generating the network. Again, we compared the estimates of K found by the *ILvB* to those found by our algorithm.

Table 4
The true number of clusters (rows) against the number estimated by our collapsed MCMC algorithm (columns). The diagonal entries are underlined to aid readability, as these represent the correct answer. The accuracy is better here than in Table 3; we can see that the numbers on the diagonal are larger.

	10	11	12	13	14	15	16	17	18	19	20
10	95	0	0	0	0	0	0	0	0	0	0
11	6	93	1	0	0	0	0	0	0	0	0
12	1	8	90	1	0	0	0	0	0	0	0
13	0	2	12	86	0	0	0	0	0	0	0
14	0	0	1	9	90	0	0	0	0	0	0
15	0	0	0	1	13	84	2	0	0	0	0
16	0	0	0	0	1	22	73	4	0	0	0
17	0	0	0	0	0	2	29	65	4	0	0
18	0	0	0	0	0	1	9	28	62	0	0
19	0	0	0	0	0	1	3	7	38	51	0
20	0	0	0	0	0	0	0	3	11	28	57

The results are shown in Tables 3 and 4. Each row of data represents the 100 networks generated for a given K_{True}. Each column represents the estimated $\hat{K}$. Ideally, the algorithm would correctly estimate the number of clusters in most cases, corresponding to large number on the diagonal. We have underlined the diagonal entries for clarity. Note that the sum of the entries in each row does not always sum exactly to 100, since there are cases where the algorithms underestimate or overestimate the number of clusters, beyond the shown range. For the *ILvb* algorithm, it is necessary to specify a range of K to be tested; we specified the range from 5 to 30. Our MCMC algorithm requires no such hint.

For networks with a small number of clusters, both algorithms perform well, with 72% accuracy for *ILvb* and 95% accuracy for our algorithm. As the true number of clusters increase, the performance decreases. Our algorithm maintains at least 50% accuracy in all cases, whereas the accuracy for *ILvb* falls to 23%. When they are incorrect, both algorithms have a tendency to underestimate the number of clusters.

In Section 6.4, a more thorough investigation of the speed and scalability of our algorithm with respect to larger networks is given but we close our comparison with *ILvb* with some remarks on speed. For the first set of small networks above, both methods are very fast; they complete within seconds. For example, the *ILvb* can be calculated for all values of K from 10 to 20 in a total under five seconds. We have not defined a convergence criterion for our MCMC algorithm, and therefore we make no attempt to halt the sampling early in order to define a 'runtime' for our algorithm. But in the occasions where both methods get the correct result, our algorithm typically reaches the correct result within nine seconds; and the sampler remains at, or very close to, the correct clustering for the remainder of the run.

Finally, to demonstrate the importance of the *AE* move, in Fig. 2, the time taken by our algorithm to reach the correct clustering for three synthetic networks is shown. The numbers of clusters in the networks is 5, 20, and 50 respectively, with π drawn from Uniform (0, 1). In each case, there were exactly 10 nodes in each cluster, giving $N = 10 \times K$ nodes in each network. The x-axis displays the number of iterations and the y-axis the number of clusters at that stage in the run of the sampler. The correct clustering is reached in approximately 10,000 iterations.

We found that the *AE* move is quite important, at least in the early stages. If *AE* is disabled, see Fig. 2(b), then it takes about 320,000 iterations for $K = 50$, instead of just 20,000 iterations when all moves are in effect. For fast burn-in, *M3* and *AE* are necessary. With similar experiments we noticed that, once the chain has burned in, the *M3* move is sufficient for good performance and the other simple moves, *GS* and *MK*, do not make major contributions.

6.4. Larger networks

Next, we investigate larger networks to demonstrate the scalability of the algorithm.

A number of synthetic networks are generated, each with approximately ten thousand nodes and ten million edges. The number of clusters ranges from 3 to 50, and the number of nodes in each clusters, O, is set such that the total number of nodes, $N = K \times O$, is close to 10,000. If we use the default SBM edge model, then the number of edges would be approximately 50 million. As this would take up a lot of computer memory, instead we modify the prior for the per-block densities to be Uniform (0, 0.2) in order to ensure that the expected number of edges is 10 million. Large real-world networks are typically quite sparse, even more sparse than this synthetic network. The details, including the speed and accuracy, are in Table 5.

The SBM algorithm is run for 100,000 iterations on each of these networks and the time to converge is recorded. In each case, when the algorithm first visits the 'correct' state, it remains in that state for practically all the remaining iterations. We record the number of iterations taken before the algorithm reaches the correct state, and the time that has elapsed at that point. It typically converges within one hour, but it takes nearly four hours for the 50-cluster network. Methods that scale to thousands of nodes have been presented in the literature, such as Daudin et al. (2008) and Latouche et al. (2012). To our knowledge, ours is the only method which has been demonstrated on networks with 10,000 or more nodes.

We have attempted to load these networks into the *R* software package in order to run them through *ILvb*. However, the memory requirements for such large adjacency matrices become prohibitive. For large networks, it may be necessary

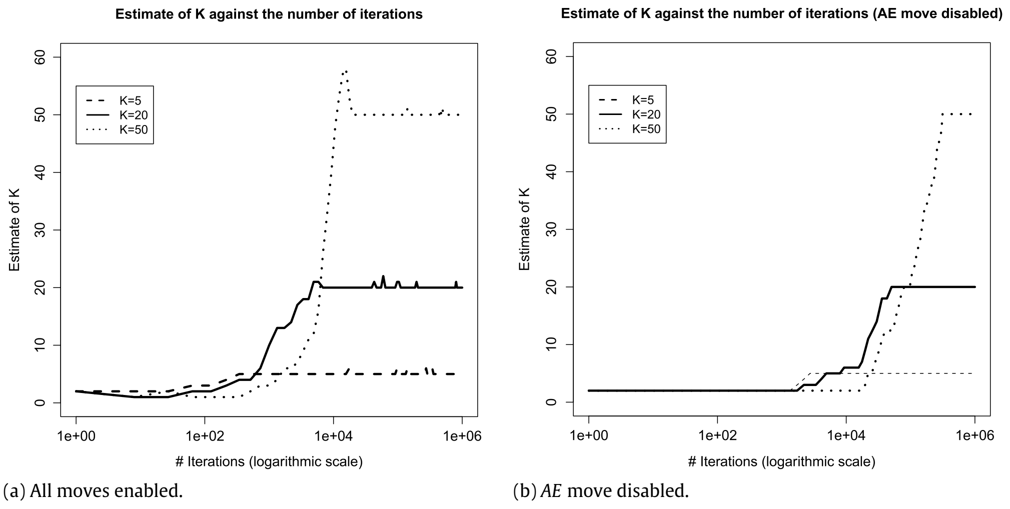


Fig. 2. The estimates of K in the synthetic networks, with $K = 5, 20, 50$. The x-axis (logarithmic scale) is the number of iterations; as the algorithm proceeds, in each case it converges on the correct estimate of K. The networks had $10 \times K$ nodes each. In the lower plot, we see the performance where the *AE* move has been disabled; demonstrating how it is important in burnin.

Table 5
The time-to-convergence for the larger synthetic networks. The networks have $N = K \times O$ nodes, made up of K clusters each with O nodes. After i iterations (t s), the algorithm reached the correct result and remained in, or close to, that state for the remainder of the 100,000 iterations. It should be noted that much of the runtime is simply taken up with loading the network into memory; the time spent in the MCMC algorithm itself is smaller than the t figure presented here.

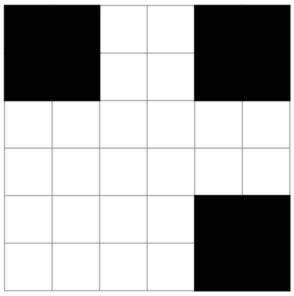
K	O	N	E	i	t
3	3333	9,999	9,722,580	41	3,317
4	2500	10,000	8,526,987	149	2,977
5	2000	10,000	8,627,869	190	2,460
6	1667	10,002	9,974,998	416	3,265
7	1429	10,003	9,316,651	749	3,449
8	1250	10,000	11,059,656	962	3,710
9	1111	9,999	9,581,440	1,383	4,052
10	1000	10,000	9,989,886	1,277	3,785
20	500	10,000	9,871,938	5,655	4,779
30	333	9,990	9,821,594	12,497	6,999
40	250	10,000	9,862,703	37,742	12,452
50	200	10,000	10,008,963	40,958	24,028

to consider a different implementation language and techniques in order to fully explore the scalability of a variational method such as *ILvb*. Instead, we generated five 500-node networks, with 20 clusters each, according to the SBM model and run *ILvB* on it, using only one value of K, namely $K = 20$. It takes between 38 and 56 s, depending on which of the five networks is used. In comparison, on the same data, our algorithm takes between 17 and 35 s, despite the fact that it is given no clue as to the correct value of K. With 1000-node networks, the runtimes for *ILvb* are between 636 and 814 s, whereas our algorithm takes between 55 and 78 s. This suggests our algorithm scales better than the *ILvb*—although perhaps this is an implementation issue rather than a limitation of the variational model.

In practice, it is necessary to run *ILvb* for every possible value of K, and this fact should be incorporated into any evaluation of its runtime. For larger networks, the range of possible values of K increases making this a significant issue. In contrast, an algorithm based on the allocation sampler, such as ours, does not suffer this limitation, suggesting that that our algorithm is well suited to large networks.

6.5. *Autocorrelation in K*

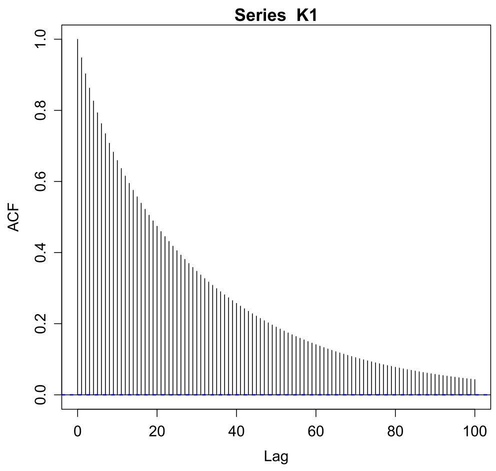
An autocorrelation analysis can reveal the mixing properties of the algorithm. However, in the above examples, and in the survey data discussed in Section 7, the estimates of K are very much peaked around a single value. Often the larger values of K are associated with empty clusters and the estimate of the number of non-empty clusters is even more peaked. This makes it difficult to use K as an interesting variable on which to perform autocorrelation analysis. To address this, we examine the 6-node network in Fig. 3(a), for which a greater variance in the values of K is observed. Define K_1 to be the number of non-empty clusters, $K_1 \leq K$. The posterior predictive probability for $K = 2$ is 57.0%, and for $K = 3$ it is 31.4%. For the non-empty clusters, it is 73.4% for $K_1 = 2$ and 24.4% for $K_1 = 3$. The autocorrelation in the estimates of K is shown in Fig. 3(c).



(a) Adjacency matrix.

	97	4	4	75	75
97		4	4	75	75
4	4	99	99	4	4
4	4	99	99	4	4
75	75	4	4		97
75	75	4	4	97	

(b) Percentage posterior probability of two nodes sharing a cluster.



(c) Autocorrelation on K.

Fig. 3. Adjacency matrix used in the analysis of varying K in Section 6.5. Fig. 3(b) estimates, for every pair of nodes, the predicted probability of them sharing a cluster. Fig. 3(c) shows the autocorrelation in the estimate of K.

The acceptance rates on this small 6-node network are relatively high: 8.1% for *MK*, 4.2% for *GS*, 20.5% for *AE*, 46.0% for *M3*. We'll see lower acceptance rates in the next section when the algorithm is applied to the survey network.

7. Survey of interaction data

A survey was performed by a team involving one of the authors of this paper at a summer school. We asked the 74 participants to fill in a survey and record which other participants they knew before the summer school and also which participants they met during the school. 40 of the participants responded and gave us permission to make their survey response available publicly in anonymized format. We created a directed, unweighted, network from the data by linking A to B if A recorded either type of relationship with B, resulting in 1138 edges. This network data is available at https://github.com/aaronmcdaid/Summer-school-survey-network.

Using the procedure described in Section 5.6, we are able to summarize the output of the Markov chain in Fig. 4. This is a matrix which records, for each (relabeled) cluster and node, the posterior probability of that participant being a member of that cluster. Each row represents one participant of the summer school, and the total weight in each row sums to 1.0. We have ordered the rows in this figure in order to bring similar rows together, helping to highlight the sets of nodes which tend to be clustered together in the Markov Chain. As may be observed, most of the participants are strongly assigned to one cluster. Every node is assigned to one of the clusters with at least 75% posterior probability, and the majority of nodes have at least 99% posterior probability.

The number of clusters selected is 7, with 90.7% posterior probability. We can summarize this into a single clustering by assigning each node to its 'best' cluster as found by the label-unswitching procedure. In Fig. 5, we see this clustering. This particular clustering (or label-switched equivalents) has posterior probability of 20.7%. (The order in which the clusters are presented is different in Fig. 5 than in Fig. 4.)

We then analyzed the clusters to see if they could be meaningfully interpreted. The first thing that stands out is that the final two rows of blocks are empty; these are simply the 33 people who did not respond to the survey. It is interesting to see that the non-respondents have been split into two clusters. Looking at the final two columns of blocks, the differences in how other clusters linked to the non-respondents can be seen.

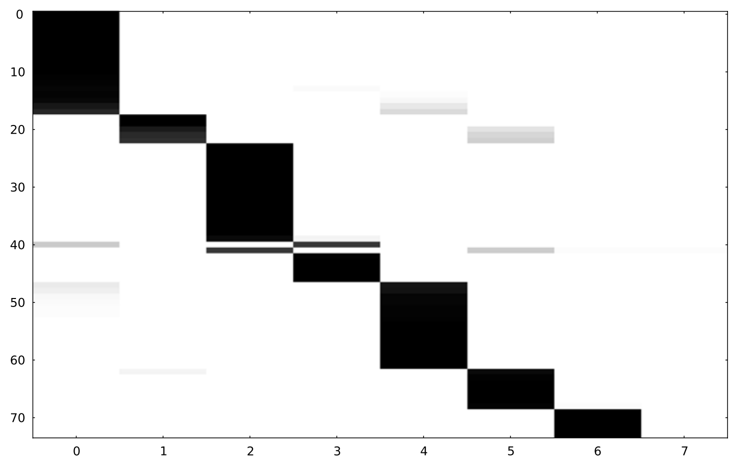


Fig. 4. The interaction survey network of Section 7. Node-to-cluster membership matrix. 74 rows, one for each participant. There are 8 columns, one for each of the main seven clusters, and an extra cluster which, with very small probability, is occupied by some nodes. Most nodes are strongly assigned to one cluster, but the grey areas off the diagonal show a small number of nodes that are partially assigned to multiple clusters.

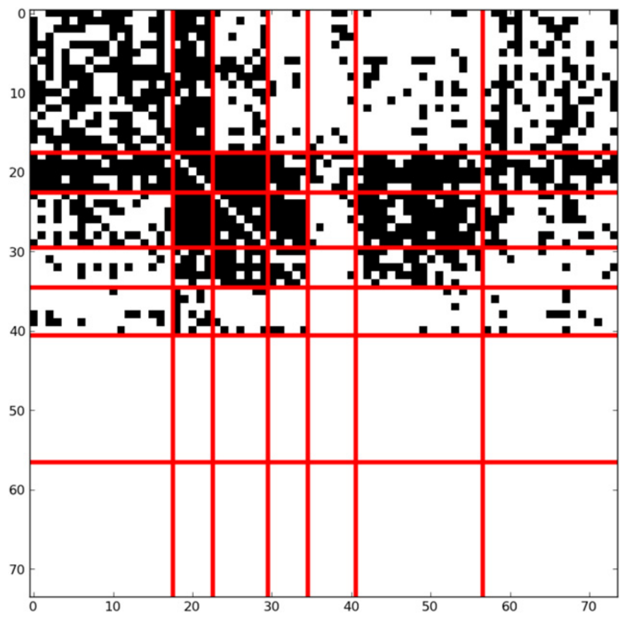


Fig. 5. The interaction survey network of Section 7 as a 74 × 74 adjacency matrix for the 74 participants in the summer school. 7 clusters were found by our method, and this matrix is ordered by the summary clustering found by the label-unswitching method of Section 5.6. In the text in Section 7, we interpret the clusters found and show how many of the clusters correspond to the different types of people that attended the event. There were 33 people who did not respond, these can be seen in the last two clusters.

With the help of one of the organizers, we verified that the second cluster (counting from the top, or from the left) is made up of the *Organizers* of the summer school, with one exception. These were people based in the research institute who were involved in organizing the summer school. Therefore, it is no surprise that the corresponding rows and columns of the adjacency matrix in Fig. 5 are quite dense. The *Organizers* interacted with almost everybody.

The third and fourth clusters are also made up of people who are based in the research institute where the summer school was hosted but who weren't on the programme committee. We call these *Locals*. The first cluster is made up of *Visitors*. These were people from further afield who attended the school and spoke at the summer school. Looking at the blocks at the top left of Fig. 5 you can see that the *Locals* know each other and the *Visitors* interacted with each other. But the two groups do not tend to interact strongly with each other. The *Organizers* are the glue that hold everybody together. The fifth cluster appears simply to be made up of participants who did not interact very much with anybody—in fact they did not even interact with each other.

We can now interpret the fact that there are two clusters of non-respondents. One of those clusters (the sixth cluster) is made of up of local people. Their names appeared in the surveys of the *Organizers* and *Locals*. The final cluster, the other non-respondent cluster, is made up of a broader range of people. It includes many non-responder *Visitors*, including many of the speakers at the summer school.

A community finding algorithm would not have been able to find these results, as it would expect to find dense clusters and is tied to the assumption that the probability of pairs of nodes being connected is, all other things being equal, greater if they share a cluster than if they do not share a cluster. This would manifest as dense blocks on the diagonal of this adjacency matrix. Clearly, a community-finding algorithm could not find the non-respondent clusters. Also, a community finding algorithm might have merged the *Organizers* and *Locals* clusters. This is because those two clusters are quite dense internally and also have many connections between them. The only difference between these two clusters is how they interact with the rest of the network; this demonstrates how the rich block structure of the Stochastic Block Model, including the various cluster–cluster interactions, can be helpful in clustering this data.

We ran the algorithm for 1 million iterations on this survey data, discarding the first 500,000 iterations as burn-in. The acceptance rates were as follows: 2.3% for *AE*, 64.6% for *M3*, 1.1% for *MK*. In the case of the Gibbs sampler, 2.5% of the time it assigned a node to a new cluster, otherwise the node was reassigned to its old cluster.

The *M3* and *AE* are both Metropolis–Hastings; a change to the clustering is proposed and then the change is accepted or rejected. Sometimes the accepted move actually places all the nodes back to the same position they were in, or sometimes it merely swaps the labels between the two clusters. If we consider these as 'rejections', then the rate and which new states are reached is just 1.0% for *M3*. So, *M3* is accepted a lot, but it usually only moves between label-switched equivalents; this tells us that the algorithm is able to move quickly between the various modes of the distribution, and also suggests that the posterior is quite peaked around the modes.

7.1. *Estimating the network probability,* $\mathrm{P}(x)$

In Section 4, we discussed how the fully Bayesian approach of the SBM presented here allows model selection criteria such as the ICL to be avoided to select between models with different input numbers of clusters K. It is also worth remarking that in certain circumstances, such as our survey data presented here, it is possible to compute an estimate of the network probability, $\mathrm{P}(x)$; that is, the probability, given just the total number of nodes N, that the network x is observed from an SBM. This provides an absolute measure of the fit of the SBM to the observed data and could be used to test the hypothesis that the data is drawn from an SBM against some alternative model.

In the survey data there is one clustering where it, along with its label-switched equivalents, take up 20.7% of the posterior probability. Call this $\hat{z}$. Thus we have a value $\hat{z}$ which is visited very often by the sampler and this allows an accurate estimate of $\mathrm{P}(K, \hat{z}|x)$ to be obtained using

$$7! \times \mathrm{P}(K = 7, z = \hat{z}|x) = 0.207.$$

Now inserting x, K and $\hat{z}$ into the expression for the joint distribution, an estimate of $\mathrm{P}(x)$ can be obtained using

$$\mathrm{P}(x)\mathrm{P}(K = 7, z = \hat{z}|x) = \mathrm{P}(x, z = \hat{z}, K = 7).$$

In the case of the survey data we obtain $\log_2 \mathrm{P}(x) \approx -2482$. To put some perspective on this value, we can compare with a model that selects x uniformly at random from all possible directed networks over $N = 74$ nodes. In this case, we obtain $\log_2 \mathrm{P}(x) = -N(N-1) = -5402$. As a second alternative, if x were generated from an Erdos–Renyi model, averaged over all possible edge probabilities drawn uniformly at random, then $\log_2 \mathrm{P}(x) \approx -4130$.

8. Conclusion

The original stochastic blockmodel was tested on a small network with two clusters. We have shown how Bayesian models, collapsing, and modern MCMC techniques can combine to create an algorithm which can accurately estimate the clusters, and the number of clusters, without compromising on speed.

It is sometimes stated that MCMC is necessarily slower than other methods, "effectively leading to severe size limitations (around 200 nodes)" (Gazal et al., 2011). The MCMC method we have presented scales to thousands of nodes, and is more scalable than a recent variational method. We do not claim that MCMC will always be necessarily faster than the alternatives, but we observe that comments on the scalability of Metropolis–Hastings MCMC depends on the particular model and on the particular proposal functions used. It may be an open question as to which methods will prove to be most scalable in the long term, as further improvements are made to all methods.

Our application to the survey data demonstrated that *block-modeling* can detect structure in networks that might be missed by *community-finding* algorithms. Sometimes the links between clusters are more interesting than the links within clusters.

Acknowledgment

This research was supported by Science Foundation Ireland (SFI) Grant No. 08/SRC/I1407—Clique Research Cluster.

Appendix

Here, we describe the integrations which show that Eq. (4) is equivalent to Eq. (6).

A.1. Collapsing θ

Here, we show how to calculate

$$\mathrm{P}(z|K) = \int_{\Theta} \mathrm{p}(z, \theta|K)\ \mathrm{d}\theta. \tag{10}$$

This corresponds to the first integration expression in Eq. (3). z is a vector which records, for each of the N nodes, which cluster it has been assigned to. The probability for each cluster is in a vector θ, where

$$1 = \sum_{k=1}^{K} \theta_k.$$

We integrate over the support of the Dirichlet distribution, which we have denoted with Θ in Eq. (10),

$$\theta \sim \mathrm{Dirichlet}(\alpha, \alpha, \ldots)$$

where we made the common simplification in our prior that all members of the vector α are identical; $\alpha_k = \alpha$.

θ is drawn from Dirichlet prior,

$$\mathrm{p}(\theta) = \frac{1}{\mathrm{B}(\alpha)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1},$$

where the normalizing constant $\mathrm{B}(\alpha)$ is

$$\mathrm{B}(\alpha) = \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma\left(\prod_{k=1}^{K} \alpha_k\right)}.$$

To collapse θ, the expression for $\mathrm{P}(z|K)$ becomes the Multivariate Pólya distribution. In the derivation, we have defined n_k to be the number of nodes in cluster k, i.e.

$$n_k = \sum_{i=1}^{N} \begin{cases} 1 & \text{if } z_i = k \\ 0 & \text{if } z_i \neq k. \end{cases}$$

In the following expression, we will also find it useful to define another vector of length K,

$$\alpha' = (\alpha_1 + n_1, \alpha_2 + n_2, \ldots, \alpha_K + n_K),$$

$$\begin{aligned}
\int_{\Theta} \mathrm{p}(z, \theta|K)\ \mathrm{d}\theta &= \int_{\Theta} \mathrm{p}(\theta|K)\mathrm{P}(z|\theta, K)\ \mathrm{d}\theta \\
&= \int_{\Theta} \mathrm{p}(\theta|K) \prod_{k=1}^{K} \theta_k^{n_k}\ \mathrm{d}\theta \\
&= \int_{\Theta} \frac{1}{\mathrm{B}(\alpha)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \prod_{k=1}^{K} \theta_k^{n_k}\ \mathrm{d}\theta \\
&= \int_{\Theta} \frac{1}{\mathrm{B}(\alpha)} \prod_{k=1}^{K} \theta_k^{\alpha_k + n_k - 1}\ \mathrm{d}\theta \\
&= \frac{\mathrm{B}(\alpha')}{\mathrm{B}(\alpha)} \int_{\Theta} \frac{1}{\mathrm{B}(\alpha')} \prod_{k=1}^{K} \theta_k^{\alpha_k + n_k - 1}\ \mathrm{d}\theta \\
&= \frac{\mathrm{B}(\alpha')}{\mathrm{B}(\alpha)} \\
&= \frac{\Gamma\left(\sum_{k=1}^{K} \alpha_k\right)}{\Gamma\left(N + \sum_{k=1}^{K} \alpha_k\right)} \prod_{k=1}^{K} \frac{\Gamma(n_k + \alpha_k)}{\Gamma(\alpha_k)} \\
&= \frac{\Gamma(K\alpha)}{\Gamma(N + K\alpha)} \prod_{k=1}^{K} \frac{\Gamma(n_k + \alpha)}{\Gamma(\alpha)}.
\end{aligned}$$

A.2. Collapsing π

Now we look at the second integration expression in Eq. (3). This describes how to calculate the probability of a network, x, given a clustering, z, and the number of clusters, K.

$$\mathrm{P}(x|z, K) = \int_{\Pi} \mathrm{P}(x, \pi|z, K)\ \mathrm{d}\pi.$$

This depends on whether we're using the unweighted (Bernoulli) or integer-weighted(Poisson) model for edges. It is also possible to allow real-valued weights with a Normal distribution and suitable priors, an example of such a model is solved in Appendix B.2 of Wyse and Friel (2012); that paper is relevant for all the derivations here as the collapsing approach is quite similar as in this paper.

The number of pairs of nodes in block between clusters k and l will be denoted p_{kl}—for blocks on the diagonal p_{kk} will depend on whether the edges are directed and on whether self loops are allowed; see Eq. (5) for details. The relevant probabilities for a given block will be shown to be a function only of p_{kl} and of the total weight (or total number of edges) in that block. We'll denote this total weight as

$$y_{kl} = \sum_{i,j|z_i=k,z_j=l} x_{ij}.$$

In an undirected graph, we should consider each pair of nodes only once,

$$y_{kl} = \sum_{i,j|i<j,z_i=k,z_j=l} x_{ij}.$$

We are to calculate the integral for a single block. $x_{(kl)}$ represents the submatrix of x corresponding to pairs of nodes in clusters k and l. Our goal is to simplify the expression such that there there is one factor for each block,

$$\begin{aligned}\mathrm{P}(x|z, K) &= \prod \mathrm{P}(x_{(kl)}|z, K)\\ &= \prod \int \mathrm{P}(x_{(kl)}, \pi_{kl}|z, K)\ \mathrm{d}\pi_{kl}.\end{aligned}$$

For directed graphs, the product is $\prod_{k,l}$, giving $K \times K$ blocks. But for undirected graphs, the product is $\prod_{k,l|k\le l}$, giving $\frac{1}{2}K(K+1)$ blocks. The domain of the integration will be either $\int_0^1$ or $\int_0^\infty$, depending on which of the two data models, unweighted or weighted, is in effect.

We'll first consider the unweighted (Bernoulli) model. The probability of a node in cluster k connecting to a node in cluster l is constrained by

$$0 < \pi_{kl} < 1,$$

and each element of $x_{(kl)}$ is drawn from a Bernoulli distribution with parameter π_{kl},

$$\mathrm{P}(x_{(kl)}|\pi_{kl}, z, K) = \pi_{kl}^{y_{kl}}(1-\pi_{kl})^{p_{kl}-y_{kl}}.$$

The prior for π_{kl} is a Beta(β_1, β_2) distribution.

$$\begin{aligned}\mathrm{P}(x_{(kl)}|z, K) &= \int_0^1 \mathrm{p}(x_{(kl)}, \pi_{kl}|z, K)\ \mathrm{d}\pi_{kl}\\ &= \int_0^1 \mathrm{p}(\pi_{kl})\ \mathrm{P}(x_{(kl)}|\pi_{kl}, z, K)\ \mathrm{d}\pi_{kl}\\ &= \int_0^1 \frac{\pi_{kl}^{\beta_1-1}(1-\pi_{kl})^{\beta_2-1}}{\mathrm{B}(\beta_1,\beta_2)}\pi_{kl}^{y_{kl}}(1-\pi_{kl})^{p_{kl}-y_{kl}}\ \mathrm{d}\pi_{kl}\\ &= \int_0^1 \frac{\pi_{kl}^{y_{kl}+\beta_1-1}(1-\pi_{kl})^{p_{kl}-y_{kl}+\beta_2-1}}{\mathrm{B}(\beta_1,\beta_2)}\ \mathrm{d}\pi_{kl}\\ &= \frac{\mathrm{B}(y_{kl}+\beta_1, p_{kl}-y_{kl}+\beta_2)}{\mathrm{B}(\beta_1,\beta_2)} \times \int_0^1 \frac{\pi_{kl}^{y_{kl}+\beta_1-1}(1-\pi_{kl})^{p_{kl}-y_{kl}+\beta_2-1}}{\mathrm{B}(y_{kl}+\beta_1, p_{kl}-y_{kl}+\beta_2)}\ \mathrm{d}\pi_{kl}\\ &= \frac{\mathrm{B}(y_{kl}+\beta_1, p_{kl}-y_{kl}+\beta_2)}{\mathrm{B}(\beta_1,\beta_2)},\end{aligned}$$

where $\mathrm{B}(\beta_1, \beta_2) = \frac{\Gamma(\beta_1)\Gamma(\beta_2)}{\Gamma(\beta_1+\beta_2)}$ is the Beta function. This result is closely related to the Beta-binomial distribution.

Next, we'll consider the Poisson model for edges in more detail. Again, we will see that p_b and y_b are sufficient for $\mathrm{P}(x_{(kl)}|K, z)$.

In this integer-weighted model, an edge (or non-edges) between a node in cluster k and a node in cluster l gets its weight from a Poisson distribution

$$x_i|\pi_{kl} \sim \text{Poisson}(\pi_{kl}),$$

and $\pi_{kl} > 0$.

This gives us, iterating over the pairs of nodes in the block,

$$\mathrm{P}(x_{(kl)}|\pi_{kl}, z, K) = \prod_{i,j\in k,l} \frac{\pi_{kl}^{x_{ij}}}{x_{ij}!} \exp(-\pi_{kl}).$$

We can combine this expression for every block,

$$\begin{aligned}\mathrm{P}(x|\pi, z, K) &= \prod_{kl} \mathrm{P}(x_{(kl)}|\pi_{kl}, z, K)\\ &= \prod_{kl}\prod_{i,j\in k,l} \frac{\pi_{kl}^{x_{ij}}}{x_{ij}!}\exp(-\pi_{kl})\\ &= \prod_{ij}\frac{1}{x_{ij}!}\prod_{kl}\prod_{i,j\in k,l}\pi_{kl}^{x_{ij}}\exp(-\pi_{kl}).\end{aligned}$$

We can ignore the $\prod_{ij}\frac{1}{x_{ij}!}$, as one of those will be included for every pair of nodes in the network. That will contribute a constant factor to Eq. (6); this factor will depend only on the network x, and it will not depend on K or z or any other variable of interest, and hence we can ignore it for the purposes of Eq. (6). Therefore, for our purposes we will be able to use the following approximation in the derivation

$$\mathrm{P}(x_{(kl)}|\pi_{kl}, z, K) = \prod_{i,j\in k,l} \pi_{kl}^{x_{ij}} \exp(-\pi_{kl}).$$

We'll place a Gamma prior on the rates,

$$\pi_b \sim \text{Gamma}(s, \phi).$$

$$\begin{aligned}\mathrm{P}(x_{(kl)}|z, K) &= \int_0^\infty \mathrm{p}(x_{(kl)}, \pi_{kl}|z, K)\mathrm{d}\pi_{kl}\\ &= \int_0^\infty \pi_{kl}^{s-1}\frac{e^{-\pi_{kl}/\phi}}{\Gamma(s)\phi^s}\prod_{i,j\in k,l}\frac{\pi_{kl}^{x_{ij}}}{x_{ij}!}e^{-\pi_{kl}}\mathrm{d}\pi_{kl}\\ &= \prod\frac{1}{x_{ij}!}\int_0^\infty \pi_{kl}^{s-1}\frac{e^{-\pi_{kl}/\phi}}{\Gamma(s)\phi^s}\prod_{i,j\in k,l}\pi_{kl}^{x_{ij}}e^{-\pi_{kl}}\mathrm{d}\pi_{kl}\\ &\propto \int_0^\infty \pi_{kl}^{s-1}\frac{e^{-\pi_{kl}/\phi}}{\Gamma(s)\phi^s}\prod_{i,j\in k,l}\pi_{kl}^{x_{ij}}e^{-\pi_{kl}}\mathrm{d}\pi_{kl}\\ &= \int_0^\infty \pi_{kl}^{s-1+\sum x_{ij}}\frac{\exp\left(-\pi_{kl}p_{kl} - \frac{\pi_{kl}}{\phi}\right)}{\Gamma(s)\phi^s}\mathrm{d}\pi_b.\end{aligned}$$

We said earlier that we'll define $y_{kl} = \sum_{i,j\in k,l} x_{ij}$. We'll now substitute that in and also use the following definitions:

$$s' = s + y_{kl}$$

$$\frac{1}{\phi'} = p_{kl} + \frac{1}{\phi}$$

where Gamma(s, ϕ) was the prior on π_b, Gamma(s', ϕ') is the posterior now that we have observed edges with total weight y_{kl} between p_{kl} pairs of nodes. Returning to f, and rearranging such that we can cancel out the integral (because it is the integral of a Gamma distribution and hence it will equal 1),

$$\begin{aligned}f(x_{(kl)}|z, K) &= \int_0^\infty \pi_{kl}^{s'-1}\frac{\exp\left(-\frac{\pi_{kl}}{\phi'}\right)}{\Gamma(s)\phi^s}\mathrm{d}\pi_{kl}\\ &= \frac{\Gamma(s')\phi'^{\,s'}}{\Gamma(s)\phi^s}\int_0^\infty \pi_{kl}^{s'-1}\frac{\exp\left(-\frac{\pi_{kl}}{\phi'}\right)}{\Gamma(s')\phi'^{\,s'}}\mathrm{d}\pi_{kl}\end{aligned}$$

$$= \frac{\Gamma(s')\phi'^{s'}}{\Gamma(s)\phi^s}$$

$$= \frac{\Gamma(s+y_{kl})\left(\frac{1}{p_{kl}+\frac{1}{\phi}}\right)^{s+y_{kl}}}{\Gamma(s)\phi^s}.$$

References

Airoldi, E.M., Blei, D.M., Fienberg, S.E., Xing, E.P., 2008. Mixed membership stochastic blockmodels. Journal of Machine Learning Research 9, 1981–2014.

Bickel, P.J., Chen, A., 2009. A nonparametric view of network models and Newman–Girvan and other modularities. Proceedings of the National Academy of Sciences 106, 21068–21073.

Biernacki, C., Celeux, G., Govaert, G., 2000. Assessing a mixture model for clustering with the integrated completed likelihood. IEEE Transactions on Pattern Analysis and Machine Intelligence 22, 719–725.

Chan, J., Lam, S., Hayes, C., 2011. Increasing the scalability of the fitting of generalised block models for social networks. In: International Joint Conference on Artificial Intelligence 2011, IJCAI.

Daudin, J., Picard, F., Robin, S., 2008. A mixture model for random graphs. Statistics and Computing 18, 173–183.

Everett, M., 1996. Exact colorations of graphs and digraphs. Social Networks 18, 319–331.

Fortunato, S., 2010. Community detection in graphs. Physics Reports 486, 75–174.

Gazal, S., Daudin, J.J., Robin, S., 2011. Accuracy of variational estimates for random graph mixture models. Journal of Statistical Computation and Simulation 82, 849–862.

Girvan, M., Newman, M., 2002. Community structure in social and biological networks. Proceedings of the National Academy of Sciences of the United States of America 99, 7821.

Green, P.J., 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. Biometrika 82, 711–732.

Handcock, M.S., Raftery, A.E., Tantrum, J.M., 2007. Model-based clustering for social networks. Journal of the Royal Statistical Society: Series A (Statistics in Society) 170, 301–354.

Hastings, W.K., 1970. Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57, 97–109.

Hoff, P.D., Raftery, A.E., Handcock, M.S., 2002. Latent space approaches to social network analysis. Journal of the American Statistical Association 97, 1090–1098.

Hofman, J.M., Wiggins, C.H., 2008. Bayesian approach to network modularity. Physical Review Letters 100, 258701+.

Karrer, B., Newman, M.E.J., 2011. Stochastic blockmodels and community structure in networks. Physical Review E 83, 016107+.

Kemp, C., Griffiths, T., Tenenbaum, J., 2004. Discovering latent classes in relational data. MIT Technical Report.

Latouche, P., Birmele, E., Ambroise, C., 2012. Variational Bayesian inference and complexity control for stochastic block models. Statistical Modelling 12, 93–115.

Liu, J.S., 1994. The collapsed Gibbs sampler in Bayesian computations with applications to a gene regulation problem. Journal of the American Statistical Association 89, 958–966.

Mariadassou, M., Robin, S., Vacher, C., 2010. Uncovering latent structure in valued graphs: a variational approach. The Annals of Applied Statistics 4, 715–742.

McDaid, A.F., Murphy, B.T., Friel, N., Hurley, N.J., 2012. Model-based clustering in networks with stochastic community finding. In: Compstat 2012.

Newman, M.E.J., 2004. Fast algorithm for detecting community structure in networks. Physical Review E 69, 066133+.

Newman, M.E.J., Girvan, M., 2004. Finding and evaluating community structure in networks. Physical Review E 69, 026113+.

Newman, M., Leicht, E., 2007. Mixture models and exploratory analysis in networks. Proceedings of the National Academy of Sciences of the United States of America 104, 9564.

Nobile, A., Fearnside, A., 2007. Bayesian finite mixtures with an unknown number of components: the allocation sampler. Statistics and Computing 17, 147–162.

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

Schwarz, G., 1978. Estimating the dimension of a model. The Annals of Statistics 6, 461–464.

Snijders, T.A.B., Nowicki, K., 1997. Estimation and prediction for stochastic blockmodels for graphs with latent block structure. Journal of Classification 14, 75–100.

Wyse, J., Friel, N., 2012. Block clustering with collapsed latent block models. Statistics and Computing 22, 415–428.

Zanghi, H., Ambroise, C., Miele, V., 2008. Fast online graph clustering via Erdős–Rényi mixture. Pattern Recognition 41, 3592–3599.