Overlapping Stochastic Community Finding

Aaron McDaid, Neil Hurley
Insight Centre for Data Analytics
School of Computer Science and Informatics
University College Dublin
Dublin, Ireland
Email: aaronmcdaid@gmail.com
Email: neil.hurley@insight-centre.org

Brendan Murphy
Insight Centre for Data Analytics
School of Mathematical Sciences
University College Dublin
Dublin, Ireland
Email: brendan.murphy@insight-centre.org

***Abstract*—Community finding in social network analysis is the task of identifying groups of people within a larger population who are more likely to connect to each other than connect to others in the population. Much existing research has focussed on *non-overlapping* clustering. However, communities in real-world social networks do overlap. This paper introduces a new community finding method based on *overlapping* clustering. A Bayesian statistical model is presented, and a Markov Chain Monte Carlo (MCMC) algorithm is presented and evaluated in comparison with two existing overlapping community finding methods that are applicable to large networks. We evaluate our algorithm on networks with thousands of nodes and tens of thousands of edges.**

Much research has been carried out on community finding in networks [1], but most of the existing research has focussed on *non-overlapping* clustering, where the nodes of the network are clustered such that each node is assigned to exactly one cluster. This paper introduces a new method for *overlapping* clustering[1], where each node may be assigned to more than one community or to zero communities. Overlapping community finding is the goal of research such as [2, 3] which has shown that a member of a Facebook network may be a member of six communities on average. Community-finding is a special case of *clustering*. The assumption in community-finding, as opposed to other clustering approaches such as *block-modelling* [4], is that if a pair of nodes are linked to each other in the network then it is more likely they will both be members of the same community than if they are not linked.

This paper introduces a new model, which we call Overlapping Stochastic Community Finding (OSCF) and which is related to the Overlapping Stochastic Block Model (OSBM) [5]. Non-statistical methods are usually described explicitly as a function which takes a network as input and computes a community labelling as a function of the network. In a Bayesian statistical approach, a stochastic generative model is described specifying a stochastic procedure to select a number of clusters and to assign the nodes randomly to those clusters. Then, given that assignment, we specify a stochastic procedure to generate a network based on the assignment. In a community-finding model such as ours, we specify a procedure which tends to place the edges within communities. This results in a stochastic procedure for which the network is a random *output* of the model. Given an *observed* network as *input*, we use a Markov Chain Monte Carlo algorithm (MCMC) to produce a sample from the *conditional* joint distribution, given that network, of the labelling and the number of clusters.

The variables of interest are the number of clusters, the labelling, and the network. Other variables in the model include the number of nodes assigned to each cluster and the density at which edges form within each cluster. A typical MCMC algorithm explicitly samples all variables. Then variables whose samples are not of interest to us can be numerically integrated in order to access the marginal posterior. However, some models allow such nuisance variables to be analytically integrated out, resulting in an algorithm whose state is simply the number of clusters and the labelling. This approach has been called *collapsing* and is used in this paper. The collapsed conditional ('posterior') distribution is a function of the network, of the proposed number of clusters, and of the proposed labelling. We use modern MCMC techniques, inspired by the *allocation sampler* [6], to improve the efficiency of the algorithm.

I. The OSCF model

Let n be the number of nodes in the network which is taken as given. The number of communities, q, is drawn from a Poisson prior, conditioned that the number of clusters is greater than zero. The n nodes are assigned to the q clusters such that s_k, the number of nodes in cluster k, an integer in the range 0 to n (inclusive), is drawn from a Geometric distribution.[2]. The clustering is recorded in an $n \times q$ matrix Z such that $z_{ik} = 1$ if node i is in cluster k and 0 otherwise. Given s_k, the elements z_{ik} are set to zero or one at random such that $s_k = \sum_{1<i<q} z_{ik}$. There is no other constraint on this matrix, in particular each node may be in any number of clusters (including zero).

Next, given (Z, s_k, q), we generate a random latent network for *each* of the q clusters, in a three-dimensional array $\boldsymbol{Y}$ of dimension $q \times n \times n$ such that the "slice" Y_{k**} records the random latent network corresponding to cluster k. These slices are generated independently given (Z, s_k, q) and the final observed network is the union of these networks. For any cluster k, we require that edges are possible only if the two endpoints are members of cluster k: $z_{ik}z_{jk}{=}0 \;\Rightarrow\; y_{kij}{=}0$. This places an upper limit of s_k^2 on the number of edges that may be present in the latent network of cluster k. Furthermore imposing that the network is symmetric with no self-loops, implies that $\sum_{i<j} y_{kij} \leq \frac{1}{2}s_k(s_k-1) \triangleq \frac{1}{2}a_k$ is the maximum number of edges possible in the latent network for cluster k.

[1] http://github.com/aaronmcdaid/OSCF-mcmc

[2] ignoring prob. that $s_k > n$, which is very low for large n

The random process next generates a vector $\boldsymbol{m}$ of length q, of integers between 0 and a_k (inclusive), $m_k \sim \text{Uniform}(\{0, 1, ..., a_k\})$, as the number of edges within the latent network for cluster k. Given m_k, the network Y_{k**} is chosen with uniform probability from all networks of m_k edges that can be generated from the a_k possible edges. Marginalising away m_k, we can write

$$\mathrm{P}(Y_{k**}|\boldsymbol{z}_{*k}, s_k, q) = \begin{cases} \frac{1}{(1+m_k^*)} \binom{a_k}{m_k}^{-1} & \text{if } \sum_i z_{ik} = s_k \text{ and sym, no self-loop} \\ 0 & \text{otherwise} \end{cases}$$

where m_k^* is defined as a function of Y_{k**}: $m_k^* = \sum_{i<j} y_{kij}$. Finally, we have the probability distribution of $\boldsymbol{Y}$: $\mathrm{P}(\boldsymbol{Y}|Z, \boldsymbol{s}, q) = \prod_{k=1}^{q} \mathrm{P}(Y_{k**}|\boldsymbol{z}_{*k}, s_k, q)$, and, given $\boldsymbol{Y}$, the observed network X is a (deterministic) function of $\boldsymbol{Y}$. An edge is observed in X if and only if there is at least one k for which y_{kij}=1 i.e. $x_{ij}|\boldsymbol{Y} = \min(1, \sum_{k=1}^{q} y_{kij})$.

A. Collapsing

Rather than sample from $\boldsymbol{Y}, \boldsymbol{m}, Z, \boldsymbol{s}, q|X$, we would like to integrate out $\boldsymbol{m}$, $\boldsymbol{s}$ and $\boldsymbol{Y}$ to sample directly the conditional distribution of the variables of interest, $Z, q|\cdot$, however the summation over $\boldsymbol{Y}$ appears impossible. Fortunately, we can take a different approach and derive a closed-form expression for $\mathrm{P}(X, \boldsymbol{Y}, q)$. Then, with this probability mass function, we can implement a Markov Chain which samples from the conditional distribution $\boldsymbol{Y}, q|X$. Now,

$$\mathrm{P}(X, \boldsymbol{Y}, q) = \mathrm{P}(X|\boldsymbol{Y}, q) \left(\sum_{Z} \mathrm{P}(\boldsymbol{Y}, Z|q) \right) \mathrm{P}(q) . \qquad (1)$$

We have that $\mathrm{P}(q) \propto \frac{1}{q!}$ and $\mathrm{P}(X|\boldsymbol{Y}, q)$ is the simple 0-1 function that tells us if $\boldsymbol{Y}$ is compatible with x. It may be shown that $\sum_Z \mathrm{P}(\boldsymbol{Y}, Z|q)$ can be computed as:

$$\prod_{k=1}^{q} \left(\sum_{s_k=0}^{n} \left(\sum_{\boldsymbol{z}_{*k}} \Big(\mathrm{P}(Y_{k**}|\boldsymbol{z}_{*k}, s_k) \mathrm{P}(\boldsymbol{z}_{*k}|s_k) \mathrm{P}(s_k) \Big) \right) \right) .$$

The inner summation can be simplified further. Each term is either zero, or it is equal to $\frac{1}{1+a_k} \frac{1}{\binom{a_k}{m_k}} \frac{1}{\binom{n}{s_k}}$. Therefore, we simply need to count how many terms will take the non-zero value. Again, some calculation leads to $\mathrm{P}(\boldsymbol{Y}|q) = \prod_{k=1}^{q} f(s_k^*, m_k^*, n)$, where

$$f(s_k^*, m_k^*, n) \triangleq \sum_{s_k=s_k^*}^{s_k=n} \frac{1}{2^{(1+s_k)}} \frac{1}{1+a_k} \frac{1}{\binom{a_k}{m_k^*}} \frac{1}{\binom{n}{s_k}} \binom{n-s_k^*}{s-s_k^*} ,$$

which depends only on three numbers (n, s_k^*, m_k^*). For large n, the number of terms in the summation in $f(\cdot)$ is large, but the value is very much dominated by the first few terms. Therefore, we have found it sufficient to approximate $f(\cdot)$ by the summation over the first 100 terms. Also, for a given triple (n, s_k^*, m_k^*), we calculate $f(\cdot)$ only once and the result is stored in a cache. As the algorithm makes changes to its estimates of $(\boldsymbol{Y}, q)$, we keep track of how this changes s_k^* and m_k^*, and then we can calculate the relevant probability mass, $\mathrm{P}(X, \boldsymbol{Y}, q)$, quickly by using these sufficient statistics and looking up the cache of $f(\cdot)$.

II. Contrast with MOSES and OSBM

The OSCF model is similar to the models used in MOSES [3] and in the OSBM [5]. In these models, each node may be be in any number of clusters. MOSES is a specialization of the OSCF, where each cluster is assumed to have the same density. In place of the m_k variable in MOSES, there is a single real-valued parameter which is $\mathrm{P}(y_{kij} = 1|z_{ik} = 1, z_{jk} = 1)$. The approach taken in OSCF, where each community effectively has its own density parameter, is perhaps more realistic and also has the advantage of allowing the collapsing mentioned above as it introduces more independence between the communities. In the OSCF and in MOSES, given Z, the probability of two nodes connecting depends only on the intersection of the sets of communities that the two nodes are in; the number of such communities is $\sum_{k=1}^{q} z_{ik} z_{jk}$. However, the OSBM considers every pair of communities in the product of those two sets and allows each to contribute to the log-odds of a connection. OSBM allows a more general type of structure, not simply assuming that edges are preferred within communities.

III. MCMC for OSCF

We present a Metropolis-Hastings algorithm to draw $\boldsymbol{Y}, q|X$. Note that the observed data X appears only in the 0-1 probability $\mathrm{P}(X|\boldsymbol{Y}, q)$. If $x_{ij} = 0$, then we know that $y_{kij}|X = 0$, for all k, and therefore these cells of y are fixed at zero and no proposal is ever made in our MCMC algorithm to change them. Similarly, if $x_{ij} = 1$ then we know this condition requires that at least one of the $y_{kij}|X$ be equal to 1. As long as these two conditions are preserved at all times, we need not concern ourselves with $\mathrm{P}(X|\boldsymbol{Y}, q)$ as it is constant under those conditions. For each edge $x_{ij} = 1$ we keep track of the total $\sum_k y_{kij}$. Every move in the algorithm can be decomposed into a series of moves that change q, or which flip individual cells y_{kij}. It is straight-forward to efficiently update the sufficient statistics as individual cells in y_{kij} are changed. The moves are as follows:

1) Move #1 - Metropolis on q: This move first chooses, with probability $1/2$, whether to attempt to *add* or to *remove* a community. Each existing community is identified with a number between 1 and q inclusive. If *add* is selected, then we propose to increase q via $q' = q + 1$ and the identity of the proposed new community, k, is selected randomly between 1 and $q + 1$ inclusive - if accepted and if $k \leq q$, the edges that were in the existing community k are moved to a new community with identifier $q + 1$, leaving k as the new empty community. If *remove* is selected, then one of the q existing communities is selected at random. If it is not empty then the proposal is immediately rejected.

2) Move #2 - Metropolis, one node one community: This is the simplest move. One edge is selected at random from the observed network X, a pair of nodes i and j such that $x_{ij} = 1$, and one community, k, is selected at random. The node is either in the community, $y_{kij} = 1$, or it is not, $y_{kij} = 0$. We propose to flip this, $y'_{kij} = 1 - y_{kij}$.

3) Move #3 - Gibbs update on 'Nearby' communities: A simple Gibbs update was implemented, but is not included by default as it is slow on networks with large numbers of communities. In the simple update, one edge is selected at random from the observed network X, which joins node i to node j.

This tells us that $x_{ij} = 1$ and therefore that $\sum_k y_{kij} \geq 1$. We consider the vector $\boldsymbol{y}_{*ij}$ which records which communities that edge has been assigned to. The existing value of this vector is discarded and a new value for this vector, $\boldsymbol{y}'_{*ij}$, is accepted, drawn from the conditional distribution where everything else in $\boldsymbol{Y}$ is fixed, $\boldsymbol{y}_{*ij}|\boldsymbol{Y}^{(-*ij)}, q, X$. A slightly more complicated variant of the Gibbs update identifies a subset of the cells within the $\boldsymbol{Y}$ object and updates all of them simultaneously, conditioning on all other entries remaining fixed. First, an edge is selected from the network, i.e. a pair (i, j) such that $x_{ij} = 1$. Then, a set of 'nearby' communities, $N(i, j)$, is identified for that edge, consisting of those communities containing edges that share one node with the current edge. This is the set of communities k such that there exists at least one edge in Y_{k**} which includes *exactly one* of $\{i, j\}$. When q is large, the set of nearby communities will typically be much smaller than the full set of communities.

4) Move #4 - M3: The M3 move is inspired by the *allocation sampler* of [6]. Two distinct communities k and l are selected at random. The M3 move proposes to rearrange the edges in *both* communities. The edges of interest fall into three categories, those that are in community k, those that are in community l, and those that are in both communities. The formal Metropolis proposal is built by iterating through the edges in a random order and assigning them to one of the three categories, keeping track of the acceptance probability. The probability of the reverse proposal is also calculated in the same way.

5) Move #5 - AnySM: Propose merge/split between any two communities: The split-merge move can take any two communities and propose to merge them, and it also can do the opposite move where a community is split in two. In order to create a split proposal, a new empty community with identifier $q+1$ is created, and then a community $1 \leq l \leq q+1$ is selected at random. If $l < q+1$, then the edges currently in community l are moved into the empty community, $q + 1$. Now, l is the empty community. Another community, $1 \leq k \neq l \leq q + 1$, is selected. The proposal is then to consider all edges that are currently in k and to assign them to one of three possible states correponding to placing the edge k, or in l, or in both.

Initially, a *launch state* [7] for the move is computed. With probability $1/2$, the launch state is formed by simply removing all edges from the community. Otherwise, a more complicated launch state policy first assigns the edges to one of the three possible states using a Dirichlet random variable to select the proportions to be used for this random assignment. After this initialization, five sweeps are performed where the edges are iterated over in a random order, using a simple Gibbs update like in the M3 move described above. Once the launch state is reached, the remainder of the move proceeds much like the M3 move; the proposal itself is a single sweep of the Gibbs update, again limited to the three allowable states, where the probability of each of the three states is proportional to the target distribution at those three states.

6) Move #6 - SharedSM: Propose merge/split based on a shared edge: The last move, SharedSM, is identical to AnySM except for one change. AnySM, when attempting a merge, will select two communities at random. Instead, SharedSM will select an edge at random, and if that edge is a member of two or more communities then it will propose that two of those communities be merged. Otherwise, the method is identical to AnySM.

7) Initialization: In the next section, we discuss experiments where the algorithm is initialized in two ways. A simple initialisation sets $q = 1$ and every edge is placed in that single community. A more complex initialization strategy is inspired by the seed expansion used in [3, 8]. An edge is selected at random and the two nodes of that edge are used to form the seed of a community. Nodes are added to the seed one at a time until a local maximum of eq. (1) is achieved. Then, those edges are ignored for the remainder of the initialization process, which proceeds to make new communities until every edge is in exactly one community.

IV. Evaluation on synthetic data

We performed an evaluation by creating a variety of synthetic networks with known "true" clusterings and tested the ability of our algorithm to find the true overlapping communities. Two other algorithms are included in this evaluation. The OSLOM method [9] also takes a statistical approach to finding overlapping communities, calculating the 'fitness' of a proposed community via the statistical significance of clusters with respect to random fluctuations. We also use MOSES [3] as it has a similar Bayesian model to that used in OSCF. We use implementations provided by the authors, with default parameters. We ran our OSCF algorithm for 1,000 iterations – where an iteration involves running the six MCMC moves once for each edge in the network. Also, we included the initialization strategy defined in section III-7. The networks each had 1,000 nodes and were generated by the LFR benchmark software [10]. The degree of the nodes is fixed so that each node connects to approximately seven members of each community that it is a member of. Therefore, when a node is in eight communities, its degree is $56 = 7 \times 8$.

In table I, the details of the parameters used to generate the files are presented and, for each of the algorithms, their accuracy at finding the true clustering. We use the overlappig NMI formula [11] to measure the accuracy between the true clustering and that found by the algorithm. We generate five networks of each type, and present the median, minimum, and maximum NMI score attained by each algorithm. We have also highlighted the best performing algorithm on each type of network by printing their median score in bold.

At the top of the table, we can see examples where all algorithms get high NMI scores. For example, when each node is a member of just two communities (O=2) the structure is relatively easy to detect and sometimes the algorithms can get a perfect score (NMI=1.0). However, as the number of overlapping communities increases, we see that the performance decreases. For example, where s=10, O=8 and μ=0.0, OSCF-SE (our OSCF method, with initialization by seed expansion) attains a median NMI of 0.999, but the other algorithms achieve only 0.259 at best. These types of networks are where OSCF performs best. An increase in the mixing parameter μ, which controls the proportion of edges that have been randomly rewired, also harms performance. If μ and O are both too high, then all algorithms perform badly. In the bottom portion of the table, we see results where the sizes of communities (s) are not fixed and are chosen uniformly

TABLE I: Overlapping NMI scores for a number of algorithms. Average NMI scores over 5 networks are presented. Best scores are printed in bold. s: the size of community. q: the number of communities. O: the number of communities that a typical node is in. μ: mixing parameter. Algorithms: OSLOM[9], MOSES[3], OSCF-SE (initialized by seed expansion), OSCF-10k (ran for 10,000 iterations, seeded with q=1.)

s	q	O	μ	Algorithm	NMI (median)	NMI(min/max)
10	200	2	0.0	OSLOM	***1.000***	(1.000 / 1.000)
10	200	2	0.0	MOSES	0.942	(0.898 / 0.967)
10	200	2	0.0	OSCF-SE	***1.000***	(1.000 / 1.000)
10	200	2	0.1	OSLOM	***0.994***	(0.988 / 0.999)
10	200	2	0.1	MOSES	0.937	(0.923 / 0.967)
10	200	2	0.1	OSCF-SE	0.983	(0.982 / 0.987)
10	200	2	0.2	OSLOM	0.970	(0.969 / 0.986)
10	200	2	0.2	MOSES	0.970	(0.947 / 0.982)
10	200	2	0.2	OSCF-SE	***0.986***	(0.981 / 0.996)
10	400	4	0.0	OSLOM	0.857	(0.843 / 0.881)
10	400	4	0.0	MOSES	0.569	(0.001 / 0.647)
10	400	4	0.0	OSCF-SE	***1.000***	(1.000 / 1.000)
10	400	4	0.1	OSLOM	0.688	(0.660 / 0.717)
10	400	4	0.1	MOSES	0.730	(0.675 / 0.835)
10	400	4	0.1	OSCF-SE	***0.994***	(0.991 / 0.996)
10	400	4	0.2	OSLOM	0.195	(0.168 / 0.272)
10	400	4	0.2	MOSES	0.895	(0.841 / 0.904)
10	400	4	0.2	OSCF-SE	***0.971***	(0.965 / 0.974)
10	800	8	0.0	OSLOM	0.259	(0.234 / 0.277)
10	800	8	0.0	MOSES	0.001	(0.001 / 0.001)
10	800	8	0.0	OSCF-SE	***0.999***	(0.999 / 1.000)
10	800	8	0.1	OSLOM	0.013	(0.011 / 0.021)
10	800	8	0.1	MOSES	0.001	(0.001 / 0.049)
10	800	8	0.1	OSCF-SE	***0.870***	(0.826 / 0.901)
10	800	8	0.2	OSLOM	0.006	(0.003 / 0.009)
10	800	8	0.2	MOSES	0.001	(0.001 / 0.001)
10	800	8	0.2	OSCF-SE	***0.240***	(0.230 / 0.253)
10-40	≈80	2	0.0	OSLOM	0.960	(0.912 / 0.987)
10-40	≈80	2	0.0	MOSES	0.779	(0.770 / 0.800)
10-40	≈80	2	0.0	OSCF-SE	***0.993***	(0.989 / 0.995)
10-40	≈80	2	0.1	OSLOM	0.924	(0.869 / 0.959)
10-40	≈80	2	0.1	MOSES	0.718	(0.610 / 0.795)
10-40	≈80	2	0.1	OSCF-SE	***0.970***	(0.961 / 0.977)
10-40	≈80	2	0.2	OSLOM	0.714	(0.643 / 0.873)
10-40	≈80	2	0.2	MOSES	0.501	(0.348 / 0.543)
10-40	≈80	2	0.2	OSCF-SE	***0.904***	(0.822 / 0.932)
10-40	≈160	4	0.0	OSLOM	0.258	(0.231 / 0.333)
10-40	≈160	4	0.0	MOSES	0.774	(0.763 / 0.791)
10-40	≈160	4	0.0	OSCF-SE	***0.993***	(0.993 / 0.994)
10-40	≈160	4	0.1	OSLOM	0.134	(0.111 / 0.153)
10-40	≈160	4	0.1	MOSES	0.552	(0.483 / 0.612)
10-40	≈160	4	0.1	OSCF-10k	***0.963***	(0.952 / 0.971)
10-40	≈160	4	0.1	OSCF-SE	0.239	(0.202 / 0.292)
10-40	≈160	4	0.2	OSLOM	0.056	(0.035 / 0.086)
10-40	≈160	4	0.2	MOSES	***0.235***	(0.169 / 0.244)
10-40	≈160	4	0.2	OSCF-10k	0.189	(0.157 / 0.217)
10-40	≈160	4	0.2	OSCF-SE	0.075	(0.058 / 0.088)

at random between 10 and 40. The NMI scores are generally smaller than the corresponding scores in the top portion of the table. In particular, the performance of MOSES decreases as the community sizes are allowed to vary. This may be because the model used by MOSES assumes that the edge density within each community is constant. However, the OSCF algorithm sometimes performs well when the size of the communities varies. The OSCF algorithm is the slowest. For example, when s=10 − 40, O=4 and μ=0.1, the average number of seconds for OSCF to complete 1,000 iterations is 2,436.68, compared to 35.84 for MOSES and 1,038.02 for OSLOM. For completeness we did allow OSCF to run to 10,000 iterations (without the seed expansion) for this particular set of parameters, now taking 26,607.55 seconds on average for a single run. These results are included in the table as "OSCF-10k" and we can see that the NMI score after 10,000 iterations is better than after 1,000 iterations. OSCF is slow, but it can be very accurate if allowed to run for a long time, attaining an NMI of 96% where no other algorithm achieves better than 56%.

V. Conclusion

We present a new model for overlapping clustering applied to networks, suitable for community finding. Despite the complexity of the model, we are able to derive a formula for the conditional probability which is a function only of the network, the proposed number of clusters and the proposed clustering. An MCMC algorithm is presented which can be applied to networks with up to 50,000 edges and 1,000 nodes. The most challenging networks are those were the number of communities is large and the size of the communities varies; OSCF performs best here, significantly outperforming the other methods, especially if allowed to run for a larger number of iterations. We can not generally assume that 'real-world' communities are all of the same size and density, and therefore researchers should consider using more challenging benchmarks such as those used here.

Acknowledgment

This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

References

[1] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75 – 174, 2010.

[2] M. Salter-Townshend and T. Murphy, "Variational bayesian inference for the latent position cluster model for network data," *Computational Statistics & Data Analysis*, vol. 57, pp. 661–671, 2012.

[3] A. McDaid and N. Hurley, "Detecting highly overlapping communities with model-based overlapping seed expansion," in *Proceedings of ASONAM*, 2010, pp. 112–119.

[4] K. Nowicki and T. A. B. Snijders, "Estimation and prediction for stochastic blockstructures," *Journal of the American Statistical Association*, vol. 96, no. 455, pp. 1077–1087, Sep. 2001.

[5] P. Latouche, E. Birmelé, and C. Ambroise, "Overlapping stochastic block models," *Annals of Applied Statistics*, Oct 2009.

[6] A. Nobile and A. Fearnside, "Bayesian finite mixtures with an unknown number of components: The allocation sampler," *Statistics and Computing*, vol. 17, no. 2, pp. 147–162, Jun. 2007.

[7] S. Jain and R. M. Neal, "A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model," Dept. of Statistics, University of Toronto, Technical Report 2003, July 27 2000.

[8] C. Lee, F. Reid, A. McDaid, and N. Hurley, "Detecting highly overlapping community structure by greedy clique expansion," *arXiv 1002.1827*, 2010.

[9] A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato, "Finding statistically significant communities in networks," *PLoS ONE*, vol. 6, no. 4, pp. e18 961+, Apr. 2011.

[10] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Physical Review E*, vol. 80, no. 1, p. 16118, 2009.

[11] A. McDaid, D. Greene, and N. Hurley, "Normalized mutual information to evaluate overlapping community finding algorithms," *arXiv 1110.2515*, 2010.