Community-Based Network Alignment for Large Attributed Network

Zheng Chen[1], Xinli Yu[2], Bo Song[1], Jianliang Gao[1], Xiaohua Hu[1], Wei-Shih Yang[2]
[1]College of Computing & Informatics, Drexel University, USA
[2]Department of Mathematics, Temple University, USA
zc86@drexel.edu, xinli.yu@temple.edu, bs484@drexel.edu, jg3258@drexel.edu, xh29@drexel.edu, yang@temple.edu

ABSTRACT

Network alignment is becoming an active topic in network data analysis. Despite extensive research, we realize that efficient use of topological and attribute information for large attributed network alignment has not been sufficiently addressed in previous studies. In this paper, based on Stochastic Block Model (SBM) and Dirichlet-multinomial, we propose "divide-and-conquer" models *CAlign* that jointly consider network alignment, community discovery and community alignment in one framework for large networks with node attributes, in an effort to reduce both the computation time and memory usage while achieving better or competitive performance. It is provable that the algorithms derived from our model have sub-quadratic time complexity and linear space complexity on a network with small densification power, which is true for most real-world networks. Experiments show CAlign is superior to two recent state-of-art models in terms of accuracy, time and memory on large networks, and CAlign is capable of handling millions of nodes on a modern desktop machine.

CCS Concepts: General and reference~Reference works; Information systems~Data mining; Information systems~Social networks; Networks~Online social networks; Human-centered computing~Social network analysis

Keywords: Attributed Network Alignment, Community Discovery, Large Network, Stochastic Block Model, Dirichilet-Mutinomial

1. INTRODUCTION

Network alignment can be roughly described as matching the same or similar nodes in different networks. It recently draws a lot of attention for a variety of purposes and applications: [1] [2] for social networks, [3] for academic co-authorship networks, [4-6] for protein networks, [7] for aligning knowledge bases, and [8] for chemical compound networks, etc. Previously, node attributes are typically first reduced to node similarities [9, 10]. A recent trend in methodology of network alignment is to directly include network attributes, and they demonstrate better performance. For example, [11] aligns vertices through constructing and then aligning bipartite node-feature networks. [12] computes node similarities by using Kronecker product to propagate attribute information. All these papers share the same fundamental assumptions,

- ***Topological consistency***: if two nodes are topologically close in one network, their matched nodes in other networks should be near. For example, two people are friends and have frequent interactions in Facebook, and they also appear in Twitter, then even if they have few direct interactions on Twitter, it is likely that they stay in the same neighborhood.
- ***Attribute consistency***: a node in one network and its matches in other networks should exhibit similar features. For example, the same author usually studies similar topics and uses similar terminology for both journal and conference publications.

Despite extensive research on network alignment, a crucial issue we recognize in previous models are the balance between computation complexity and performance. For example, FINAL-N in [12] makes use of both topology and node attributes, however, it scales at $O(|V|^2)$ for both time and space for every iteration where $|V|$ is the number of nodes, and runs out of 32GB memory on a network of about 50,000 nodes and 10 attributes. Consequently, its design is not able to handle high-dimensional attributes like text. For another example, UniAlign in [11] completes in one iteration, and supports high-dimensional attributes, however, the topological adjacency information is lost in the conversion of the original network to a node-feature bipartite network which results in low accuracy, especially on a large network. The algorithm also has $O(|V|^2)$ time complexity even though it is non-iterative. To our best knowledge, efficient use of full topology and attribute information for large attributed network alignment is challenging and has not been sufficiently addressed in the past.

Is it necessary and a good practice to consider entire network to determine node correspondence when matching two large networks consisting of millions of nodes? We believe the answer is no for real-world networks, and we take the considerations of the divide-and-conquer strategy and network communities. ***From complexity perspective***, real-world networks is organized into densely connected communities [13]: no matter a person in social network, an author in a co-authorship network, a protein in a PPI network, or a concept in a knowledge base, they mainly interact with part of the network. As a result, information outside that "part" has little impact on what happens inside, and intuitively

CIKM'17, November 6–10, 2017, Singapore, Singapore

ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3132904

need not be considered when performing alignment. ***From performance perspective***, many current network alignment models are formulated as non-convex optimization that is only able to find local optimum, therefore it is very important to guide the optimization search in the right places, especially for large networks that come with much similar local topology and immense solution universe. An example in Figure 1 illustrates similar local topology which can confuse the state-of-art models.

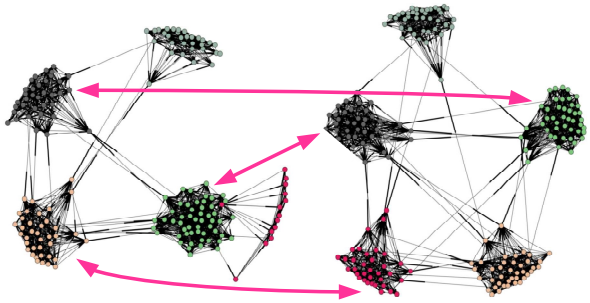


Figure 1 Example sub-networks from two Amazon co-purchase networks (see 3.1) that confuse global-alignment models [9, 11, 12] due to similar local topology. All three models have less than 60% accuracy on this example. Nodes from aligned communities computed by our model are pained the same color. Red arrows indicate communities with more than 50% error and the major source of error. Our model achieves 79% accuracy on this example.

Thus, it is appealing to us that if we can first discover communities and match those communities, then it is possible to perform local node alignments in some way with much less cost and potentially better performance.

Network community discovery problem is a classic and yet still popular research topic in network analysis, aiming to cluster nodes into "communities" [14]. A basic requirement behind most community discovery models is that topologically close nodes and more similar nodes should have higher probability to be grouped together, highly analogous to the above-mentioned alignment consistency assumptions, strengthening our motivation to handle the two types of problems together.

In particular, we propose a community-based network alignment model named *CAlign* that adopts stochastic block model (SBM) [15] as the start point. SBM is a widely-used probabilistic generative model for network community discovery with strong mathematical and statistical background [16] [17]. Probabilistic methods have an elegant theory in exploiting information of entire network [18, 19] and is easily extendable.

Meanwhile, we realize it is necessary to overcome a *community number dilemma*. In SBM, the number of communities in the network is pre-defined and denoted by letter k. As discussed in equation (10) in Section 2.3 that the standard solution depends quadratically on k. In contrast, the complexity of network alignment roughly decreases proportionally with k. For simplicity, if we only allow nodes being aligned within each pair of matched communities, then it is easy to see the alignment complexity is $O\left(\frac{|V|^2}{k}\right)$; the situation is the same for more advanced designs. Thus, we want a small k for community discovery, a large k for network alignment, and hence the dilemma. We will have to carefully modify the standard SBM so that it depends as less on k as possible. The complexity analysis and reduction relies on a fact that most real-world networks have low *densification power* $\log_{|V|}|E|$ [20], where $|E|$ denotes the number of edges in a network. The main contribution of this paper can be summarized as the following:

- *Novelty*: to our best knowledge, we are the first to jointly consider community discovery, community alignment and node alignment in one framework.
- *Efficiency*: time and space complexity is considered as a major concern in development of our approach, and we propose algorithms with provable sub-quadratic time complexity and linear space complexity; in addition, we experiment on a simple parallelization strategy that enables us to finish processing 1 million nodes in 1hr on an 8-core desktop machine.
- *Performance*: extensive experiments are conducted on large networks against two state-of-art models [11] and [12] that directly involve node attributes in computation, validating that our approach is superior in terms of both time and accuracy on large networks.

In the rest of this paper, section 2 will first define the problem, then formulate our model and develop the parameter inference. Section 3 presents experiment results on network alignment, model scalability, parallelization and communities. Related work and conclusion are given in section 4 and section 5 respectively.

2. MOEL & INFERENCE

Notation in this paper is summarized in Table 1. By convention, small bold letters represent vectors and big bold letters represent matrices. The operator $|\cdot|$ denotes the size of a set, and operator $\|\cdot\|$ denotes a norm. Other notation will be explained when they are first encountered through our discourse.

3.1 Problem Definition

We consider N directed network graphs $G_i = (V_i, E_i), i = 1, \dots, N$, where $V_i = \left\{v_1^{(i)}, \dots, v_{|V_i|}^{(i)}\right\}$ is the vertex set or node set of the ith network of size $|V_i|$, and $E_i \subseteq V_i^2$ is the edge set. Throughout this paper, script letters i, j are used to index networks. We develop the model theory for the general case of arbitrary finite N, but focus on the case of $N = 2$ for experiments and some special discussion in this paper. We also focus on directed networks for mathematical convenience (e.g. no need for special treatment of diagonal elements of the block matrix in SBM model), but it is readily applicable to undirected graphs.

Each network node inherently comes with some topological attributes, such as degree, centrality, h-index, adjacent triangles, average neighborhood degree, etc. For real-world networks, a node is usually also associated with certain non-topological network-specific attributes, like location, gender, user tags, and user vocabulary in social networks. We predefine a set of discrete node attributes used in the model, denoted by $\mathcal{A} = \{a_1, \dots, a_{|\mathcal{A}|}\}$ where $|\mathcal{A}|$ is the number of pre-defined attributes. In future discussion, we overload the symbol "a" to also represent the set of all possible values of a, or we say the range of a, and the size of a is denoted by $|a|$. The observed attribute values in network V_i of an attribute $a \in \mathcal{A}$ on some node v are denoted as $X_{a,v}^{(i)}$, and X_i is used to denote the collection of all observed attribute values in network G_i. $X_{a,v}^{(i)}$ is a singleton for most topological attributes, e.g. degree, while it might be non-singleton for some non-topological attributes, e.g. user vocabulary and user tags.

Network Data	
$\mathcal{A}, a, \lvert a\rvert, x \in a$	$\mathcal{A}$ is a set of predefined observable attributes in the network data, where each attribute in $\mathcal{A}$ is denoted by a. All possible values of a is named its range. We denote a value in a's range by $x \in a$ and the range size is denoted by $\lvert a\rvert$.
$G_i = (V_i, E_i)$	The ith network graph with its vertex set V_i and edge set E_i. Script letters i, j are used to index networks throughout the paper.
$X_i, X_{a,v}^{(i)}, \mathbf{n}_{a,v}^{(i)}(x)$	X_i is the collection of all observed attributes values in the ith network; $X_{a,v}^{(i)}$ denotes the observed attribute value of attribute $a \in \mathcal{A}$ and vertex $v \in V_i$. $\mathbf{n}_{a,v}^{(i)}$ is a counting vector of length $\lvert a\rvert$ s.t. $\mathbf{n}_{a,v}^{(i)}(x)$ is the number of occurrences of x in $X_{a,v}^{(i)}$.
Model	
$\mathcal{C} = \{C_1, \dots, C_k\}$	The set of k global latent communities.
$\mathcal{C}_i = \{C_1^{(i)}, \dots, C_{k_i}^{(i)}\}$	The set of $k^{(i)}$ latent communities of the ith network.
$\mathbf{z}_i(v)$	$\mathbf{z}_i$ is the membership vector of vertices V_i of the ith network, with $\mathbf{z}_i(v)$ denoting the membership of some vertex $v \in V_i$.
$\mathbf{P}_i, e_{i,j}^{(i)}, n_{i,j}^{(i)}$	In the i th network, $\mathbf{P}_i$ is the block matrix for communities, $e_{i,j}^{(i)}, n_{i,j}^{(i)}$ are respectively the number of actual and maximum possible edges between community $C_i^{(i)}$ and $C_j^{(i)}$.
$\mathbf{\Phi}_a = (\boldsymbol{\phi}_{a,i}, \dots, \boldsymbol{\phi}_{a,k})$	$\boldsymbol{\phi}_{a,i}$ is the Dirichlet parameter of attribute a for the ith meta community C_i; $\mathbf{\Phi}_a$ is a matrix of them.
$\mathbf{\Theta}_i = (\boldsymbol{\theta}_1^{(i)}, \dots, \boldsymbol{\theta}_{k_i}^{(i)})$ $\mathbf{y}_{a,i}^{(i)} = \mathbf{\Phi}_a \boldsymbol{\theta}_i^{(i)}$	$\boldsymbol{\theta}_i^{(i)}$ is the weight vector for the ith community $C_i^{(i)}$ in network G_i; $\mathbf{\Theta}_i$ is a matrix of them.
$\mathbf{C}_{i,j}$	The alignment matrix for communities in G_i and communities in G_j, $i \neq j$.
$\mathbf{V}_{i,j}$	The alignment matrix for vertices in G_i and vertices in G_j, $i \neq j$.

Table 1. Notations

- **Definition 1** $G_i = (V_i, E_i, X_i), i = 1, \dots, N$ is defined as the *attributed networks* in this paper, and they are the input data of our model.

As mentioned in section 1, aligning vertices of various networks is an important problem, many existing models have been trying to handle, and the ***main idea*** of our approach is "divide-and-conquer": first cluster nodes into communities, align communities, and then align nodes between aligned communities. The communities and their alignment serve as guidance for node alignment, reducing the solution space and computation cost; in return node alignment results have a way to feedback the clustering of community.

We assume each network G_i has k_i non-overlapping latent communities $\mathcal{C}_i = \{C_1^{(i)}, \dots, C_{k_i}^{(i)}\}$ where $\mathcal{C}_i$ is a partition of V_i. We also define a membership vector $\mathbf{z}_i$ indexed by vertices in V_i so that $\mathbf{z}_i(v)$ indicates which community v belongs to, and we have $v \in C_{\mathbf{z}_i(v)}^{(i)}$. The community alignment between $\mathcal{C}_i$ and $\mathcal{C}_j$ of every pair of networks G_i, G_j can be represented by a $\lvert\mathcal{C}_i\rvert \times \lvert\mathcal{C}_j\rvert$ matrix $\mathbf{C}_{i,j}$ s.t. $\mathbf{C}_{i,j}(i,j) \in [0,1]$ is a score indicating how strong $C_i^{(i)}$ aligns with $C_j^{(j)}$, which can be viewed as a probability. Likewise, alignment between V_i and V_j of every pair of networks G_i, G_j can be represented by a $\lvert V_i\rvert \times \lvert V_j\rvert$ matrix $\mathbf{V}_{i,j}$ s.t. $\mathbf{V}_{i,j}(u,v) \in [0,1]$ indicates how strong a vertex $u \in V_i$ aligns with a vertex $v \in V_j$.

- **Problem definition**. Altogether, a formal statement of our research question can be written as the following: ***given*** large attributed networks $G_i = (V_i, E_i, X_i), i = 1, \dots, N$, how to ***infer*** the following three sub-problems altogether,
  1) *community discovery*: the membership vectors $\mathbf{z}_i$ for every $i = 1, \dots, N$.
  2) *community alignment*: the community alignment matrices $\mathbf{C}_{i,j}$ for every $(i,j) \in \{1, \dots, N\}^2$.
  3) *network alignment*: the node alignment matrices $\mathbf{V}_{i,j}$ for every $(i,j) \in \{1, \dots, N\}^2$.

To start with, this paper mainly targets the third sub-problem, by designing an alignment model framework that uses community information as its guidance. We believe all three sub-problems can be and should be jointly solved, as argued earlier, they share similar assumptions: topological consistency and attribute consistency.

3.2 Model Formulation

This section describes our modeling of each sub problem and how various parameters are integrated into one framework. A plate diagram is shown in Figure 2 as an overview.

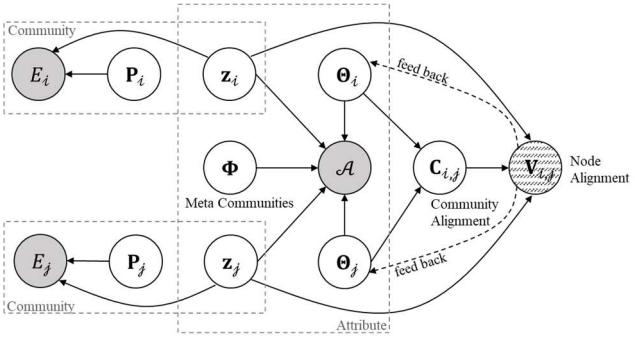


Figure 2 Plate diagram of our model CAlign. $E_i, E_j, \mathcal{A}$ are observed edges sets and attribute values. E_i, E_j are generated by SBM model defined by block matrices $\mathbf{P}_i, \mathbf{P}_j$ and membership vectors $\mathbf{z}_i, \mathbf{z}_j$. $\mathcal{A}$ is generated, by design of this paper, a Dirichlet-Multinomial defined by representation of meta communities $\mathbf{\Phi}$ and mix vectors $\mathbf{\Theta}_i, \mathbf{\Theta}_j$. Community alignment matrix $\mathbf{C}_{i,j} = \frac{1}{k}\mathbf{\Theta}_i^{\mathrm{T}}\mathbf{\Theta}_j$ as in equation (7). Node alignment $\mathbf{V}_{i,j}$ can be computed by any existing algorithms with a cost function, and can be treated as "partial observation" to feedback the mix vectors. $\mathbf{z}_i, \mathbf{z}_j$ plays the role of exchanging information between topology and node attributes within each network, and $\mathbf{\Phi}$ plays the role of exchanging information among networks.

For community discovery, we employ the Stochastic Block Model (SBM) [17] as the basic method. SBM has one additional parameter for each graph $G^{(i)}$: a $\lvert\mathcal{C}_i\rvert \times \lvert\mathcal{C}_i\rvert$ *block matrix* $\mathbf{P}_i$ s.t. $\mathbf{P}_i(i,j) \in [0,1]$ is the probability that a vertex of community $C_i^{(i)}$ has an edge to any vertex of community $C_j^{(i)}$.

Let $\delta_{u,v}^{(i)} = \begin{cases} 1 & (u,v) \in E_i \\ 0 & (u,v) \notin E_i \end{cases}$ be the indicator function for $(u,v) \in E_i$, then SBM assumes for each vertex pair $(u,v) \in V_i^2$, independently

$$\delta_{u,v}^{(i)} \sim \text{Bernoulli}\left(\mathbf{P}_i\left(\mathbf{z}_i(u), \mathbf{z}_i(v)\right)\right) \quad (1)$$

i.e. $\delta_{u,v}^{(i)} = 1$ with probability $\mathbf{P}_i\left(\mathbf{z}_i(u), \mathbf{z}_i(v)\right)$. Thus, given $\mathbf{z}_i, \mathbf{P}_i$, let $e_{i,j}^{(i)} = \sum_{u \in C_i^{(i)}, v \in C_j^{(i)}} \delta_{u,v}^{(i)}$ be the number of edges between community $C_i^{(i)}$ and $C_j^{(i)}$, and let $n_{i,j}^{(i)} = \left\lvert C_i^{(i)}\right\rvert \times \left\lvert C_j^{(i)}\right\rvert$ be the maximum possible number of edges between $C_i^{(i)}$ and $C_j^{(i)}$, then the log-likelihood of observing E_i is

$$\mathcal{L}_1^{(i)}(\mathbf{z}_i, \mathbf{P}_i | E_i) = \log \mathbb{P}(E_i | \mathbf{z}_i, \mathbf{P}_i) = \sum_{i,j \in \{1,\dots,k_i\}} \left(e_{i,j}^{(i)} \log \mathbf{P}_i(i,j) + \left(n_{i,j}^{(i)} - e_{i,j}^{(i)} \right) \log\left(1 - \mathbf{P}_i(i,j)\right) \right) \quad (2)$$

We will discuss inference of $\mathbf{z}_i, \mathbf{P}_i$ based on the observation of E_i in section 2.2. As mentioned earlier, the crucial issue here is the inference time complexity which should depend as less as possible on k_i, otherwise this part will itself introduce intolerable cost. We will later use a mathematical trick and a heuristic to reduce the complexity of the maximum likelihood estimator (MLE) derived from equation (2).

For community alignment, we follow the generative model scheme to indirectly model $\mathbf{C}_{i,j}$, assuming there exists some global latent meta-communities in this world which can be characterized by a set of Dirichlet distributions that generate vertex attributes in observable real-world networks. Moreover, we assume a community in $\mathcal{C}_i$ usually do not correspond to just one meta community, but rather a mix of them.

- **Definition 2**. Every *meta-community* $C_i \in \mathcal{C}, i = 1, \dots, k$ is a set of Dirichlet distributions $\{\text{Dir}(\boldsymbol{\phi}_{a,i}), a \in \mathcal{A}\}$, where $\boldsymbol{\phi}_{a,i}$ denote the Dirichlet parameters. Let $\boldsymbol{\Phi}_a = (\boldsymbol{\phi}_{a,i}, \dots, \boldsymbol{\phi}_{a,k})$, which is a $|a| \times k$ matrix with columns being the Dirichlet parameters of all communities for attribute a. Let $\boldsymbol{\Phi} = \left(\boldsymbol{\Phi}_{a_1}, \dots, \boldsymbol{\Phi}_{a_{|\mathcal{A}|}}\right)$ be the array of all such matrices.
- **Definition 3**. A *community* $C_i^{(i)}$ of network G_i is represented by a set of Dirichlet distributions $\left\{\text{Dir}\left(\mathbf{y}_{a,i}^{(i)}\right), \mathbf{y}_{a,i}^{(i)} = \boldsymbol{\Phi}_a \boldsymbol{\theta}_i^{(i)}, a \in \mathcal{A}\right\}$ whose parameters are a mix of $\boldsymbol{\Phi}$ specified by a *mix vector* $\boldsymbol{\theta}_i^{(i)}$ of length k s.t. every value of $\boldsymbol{\theta}_i^{(i)}$ is in range [0,1]. They will serve as prior distributions for generation of node attributes in each network community.

We assume for each vertex $v \in C_i^{(i)}$, each attribute $a \in \mathcal{A}$, a categorical distribution $\boldsymbol{\pi}_{a,v}$ is first drawn from $\text{Dir}\left(\boldsymbol{\Phi}_a \boldsymbol{\theta}_i^{(i)}\right)$, where $\boldsymbol{\pi}_{a,v}$ can be viewed as a random variant of $\boldsymbol{\Phi}_a \boldsymbol{\theta}_i^{(i)}$. Intuitively, each node is viewed as a random sample of its community s.t. the overall expectation of all members is the community "itelf" (a Dirichlet parameter is the expectation of its samples). Recall $X_{a,v}^{(i)}$ denotes the set of all attribute values of a for node $v \in V_i$, then each $x \in X_{a,v}^{(i)}$ is assumed independently drawn from $\boldsymbol{\pi}_{a,v}$. In summary,

$$\begin{aligned} \boldsymbol{\pi}_{a,v} &\sim \text{Dir}\left(\mathbf{y}_{a,\mathbf{z}_i(v)}^{(i)}\right), \forall v \in V^{(i)}, \forall a \in \mathcal{A} \\ x &\sim \text{Categorical}\left(\boldsymbol{\pi}_{a,v}\right), \forall x \in X_{a,v}^{(i)} \end{aligned} \quad (3)$$

Let $\boldsymbol{\Theta}_i = \left(\boldsymbol{\theta}_1^{(i)}, \dots, \boldsymbol{\theta}_{k_i}^{(i)}\right)$, $\boldsymbol{\theta}_v^{(i)} = \boldsymbol{\theta}_{\mathbf{z}_i(v)}^{(i)}$ for simplicity, and let $\mathbf{n}_{a,v}^{(i)}$ be a counting vector of length $|a|$ s.t. $\mathbf{n}_{a,v}^{(i)}(x)$ is the number of occurrences of x in $X_{a,v}^{(i)}$, then the log-likelihood of observing X_i given $\boldsymbol{\Phi}, \boldsymbol{\Theta}_i$ is

$$\mathcal{L}_2^{(i)}(\boldsymbol{\Phi}, \boldsymbol{\Theta}_i, \mathbf{z}_i | X_i) = \log \mathbb{P}(X_i | \boldsymbol{\Phi}, \boldsymbol{\Theta}_i, \mathbf{z}_i) = \sum_{a \in \mathcal{A}} \sum_{l \in \{1,\dots,k_i\}} \mathcal{L}_{2,a,l}^{(i)} \quad (4)$$

where

$$\begin{aligned} \mathcal{L}_{2,a,l}^{(i)}(\boldsymbol{\Phi}, \boldsymbol{\Theta}_i, \mathbf{z}_i | X_i) &= \log \mathbb{P}\left(X_{a,l}^{(i)} \middle| \boldsymbol{\Phi}, \boldsymbol{\Theta}_i, \mathbf{z}_i\right) \\ &= \log \frac{\Gamma\left(\left\|\mathbf{y}_{a,l}^{(i)}\right\|_1\right)}{\Gamma\left(\left\|\mathbf{n}_{a,l}^{(i)} + \mathbf{y}_{a,l}^{(i)}\right\|_1\right)} + \sum_{x \in a} \log \frac{\Gamma\left(\left(\mathbf{n}_{a,l}^{(i)} + \mathbf{y}_{a,l}^{(i)}\right)(x)\right)}{\Gamma\left(\mathbf{y}_{a,l}^{(i)}(x)\right)} \end{aligned} \quad (5)$$

derived from taking logarithm of compound Dirichlet-multinomial distributions and dropping constant terms (see [21] and [22]), with Γ denoting the Gamma function. We assume the generation of network data E_i, X_i of the i th network are conditional independent given $\mathbf{z}_i$, and the generation of network data of each network $(E_i, X_i), i = 1, \dots, N$ are independent given $\boldsymbol{\Phi}$, then the overall likelihood is simply sum of $\mathcal{L}_1^{(i)}$s and $\mathcal{L}_2^{(i)}$s.

$$\mathcal{L} = \sum_{i=1}^{N} \left(\mathcal{L}_1^{(i)}(\mathbf{z}_i, \mathbf{P}_i | E_i) + \mathcal{L}_2^{(i)}(\boldsymbol{\Phi}, \boldsymbol{\Theta}_i, \mathbf{z}_i | X_i) \right) \quad (6)$$

It is easy to see from above that $\mathbf{z}_i$ plays the role of exchanging information between topology and node attributes within network G_i, and $\boldsymbol{\Phi}$ plays the role of exchanging information among the networks. Finally, once all $\boldsymbol{\Theta}_i$ are inferred, we can compute the *community alignment matrix* $\mathbf{C}_{i,j}$ as

$$\mathbf{C}_{i,j} = \frac{1}{k} \boldsymbol{\Theta}_i^{\mathrm{T}} \boldsymbol{\Theta}_j \quad (7)$$

where coefficient $\frac{1}{k}$ normalizes all elements in $\mathbf{C}_{i,j}$ in range [0,1]. For a large network of at least tens of thousands of nodes, there will be hundreds of communities. It is realistic to assume one community $C_i^{(i)}$ in G_i is aligned with only a few communities in another graph G_j. Therefore, we impose a regularization terms on (6) as the following,

$$\mathcal{L} = \sum_{i=1}^{N} \left(\mathcal{L}_1^{(i)} + \mathcal{L}_2^{(i)} \right) - \sum_{1 \le i < j \le N} \lambda_{i,j} \left\| \mathbf{C}_{i,j} \right\|_0 \quad (8)$$

where $\|\cdot\|_0$ is the so-called "ℓ_0-norm", the number of non-zero elements of a matrix, and $\lambda_{i,j}$ is a parameter specifying the level of penalty. Due to the NP-hardness of optimizing zero-norm [11, 23], ℓ_0-norm is often relaxed as ℓ_1-norm, e.g. our later gradient (15) is derived using ℓ_1-norm. With such regularization, the resulting $\mathbf{C}_{i,j}$ will contain many small quantities. If we view communities as super-nodes, then the communities form a weighted multipartite network; in the case of $N = 2$, it is a bipartite network, and here is where weighted bipartite network clustering comes into play. We adopt a simple combination of [24] and [25] for our purpose, assuming edge weights obey Beta distributions of different parameters. Detailed discussion is omitted due to space limitation. The algorithm takes $O(k_i k_j)$ computation time, and yields a set of disconnected bipartite communities, where network alignment is performed within each component. We observe from our experiments that on average most components have 4 to 8 communities and do not depend on network size, and can be treated as a constant when analyzing complexity.

For network alignment, nodes of two networks G_i and G_j are aligned with respect to the bipartite community clusters derived from community alignment. In the process of parameter inference, community clusters may change, and network alignment is re-computed with respect to the new clusters, but with a restriction that a node in G_i switches its aligned node in G_j only if a better match is found, i.e. that switch decreases alignment cost. Thus, network alignment is ***NOT*** designed to be restricted by the community alignment, but community alignment functions more like a guide telling nodes where to find potentially better matches.

Technically, our model CAlign can combine any algorithm with a cost function to align nodes. In this paper, we experiment on [11] with $\|\mathbf{PA} - \mathbf{B}\|_{\mathrm{F}}$ as cost function where $\mathbf{P}$ is from their equation 5, and [12] with their equation 7 as cost function. In

addition, we will experiment on the following simple similarity-based approach that we call *Sim-Align*: given two networks G_i, G_j, every node is aligned with the most similar one in terms of first cosine-similarity and secondly Euclidean distance based on node attributes. We find such simple method already works well when rich attributes are available.

3.3 Parameter Estimation

We analyze that the model complexity as the number of vertices $|V_i|$ in each network grows. $|a|, |\mathcal{A}|, N$ are clearly fixed constants, but $k_i, k, |E_i|$ usually grows with $|V_i|$. We also assume $k < s_k = \sum_{i=1}^{N} k_i$.

Update membership vector z. We update $\mathbf{z}_i(v)$ for every node $v \in V_i$ as the community that maximizes the likelihood of the edge set E_i and the attributes X_i

$$\mathbf{z}_i^{(\text{new})}(v) = \arg\max_{\mathbf{z}_i(v)\in\{1,\dots,k_i\}} \mathcal{L}_1^{(i)} + \mathcal{L}_2^{(i)} \tag{9}$$

When optimizing (9), the likelihoods are computed based on the current $\mathbf{z}_i$ except for varying $\mathbf{z}_i(v) = 1, \dots, k_i$, and adjust the affected addends in $\mathcal{L}_1^{(i)}$ and $\mathcal{L}_2^{(i)}$ accordingly. Suppose we try switching $\mathbf{z}_i(v) = r$ to $\mathbf{z}_i(v) = r'$.

We first check the switch only affects those addends in $\mathcal{L}_2^{(i)}$ with counting vectors $\mathbf{n}_{a,r}^{(i)}, \mathbf{n}_{a,r'}^{(i)}$. The number of such addends is fixed, so re-computing $\mathcal{L}_2^{(i)}$ takes only constant complexity for each switch, and thus in total $O(k_i|V_i|)$ complexity for trying each $\mathbf{z}_i(v) = 1, \dots, k_i$ for every $v \in V_i$. We then discuss the more troublesome $\mathcal{L}_1^{(i)}$. From [17] we have

$$\mathcal{L}_1^{(i)} = \sum_{i,j\in\{1,\dots,k_i\}} \left(e_{i,j}^{(i)} \ln e_{i,j}^{(i)} - n_{i,j}^{(i)} \ln n_{i,j}^{(i)} + \left(n_{i,j}^{(i)} - e_{i,j}^{(i)}\right) \ln\left(n_{i,j}^{(i)} - e_{i,j}^{(i)}\right)\right) \tag{10}$$

- The switch only affects those $e_{i,j}^{(i)}$ s.t. one of $C_i^{(i)}$ and $C_j^{(i)}$ is the adjacent communities of v. Thus, the adjustment of $\sum_{i,j\in\{1,\dots,k_i\}} e_{i,j}^{(i)} \ln e_{i,j}^{(i)}$ takes at most $O(\deg v)$ time, where $\deg v$ is the sum of in-degree and out-degree of v.
- Unfortunately, all $n_{r,j}^{(i)}, n_{i,r}^{(i)}, n_{r',j}^{(i)}, n_{i,r'}^{(i)}$ need re-computation, and this takes $O(k_i)$ time.

Therefore, the time complexity of (10) is $O(k_i^2)$ in total for trying each $\mathbf{z}_i(v) = 1, \dots, k_i$, and $O(k_i^2|V_i|)$ for a whole sweep of updating $\mathbf{z}_i$. This complexity depends too much on k_i, causing the *community number dilemma* as discussed in section 1, where the "culprit" is $n_{i,j}^{(i)}$. We now try to remove "$n_{i,j}^{(i)}$" from (10) by first using the fact that

$$\ln(1-x) = -x\ln x + x - \sum_{n=1}^{\infty} \frac{x^{n+1}}{n(n+1)}, \forall |x| < 1$$

and $0 \le \frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}} \le 1$ to expand

$$\ln\left(n_{i,j}^{(i)} - e_{i,j}^{(i)}\right) = -\frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}} \ln \frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}} + \frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}} + \ln n_{i,j}^{(i)} + O\left(\left(\frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}}\right)^2\right)$$

Plug above back in (10) and after rearrangement we have

$$\mathcal{L}_1^{(i)} = \sum_{i,j\in\{1,\dots,k_i\}} e_{i,j}^{(i)} \ln \frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}} - |E_i| + O\left(\frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}}\right) \tag{11}$$

where the proof is in appendix A.1 at the end.

- **Definition 3**. A graph $G = (V, E)$ is *asymptotic sparse* if the ratio between the number of observed edges and the maximum possible number of edges in the network approaches zero as the graph becomes very large [26], i.e. $\lim_{|V|\to\infty} \frac{|E|}{|V|^2} = 0$. This is actually a weaker condition than the densification power law [20], thus many real-world networks that follow densification power law are immediately asymptotic sparse.

  For our purpose, we in addition assume the networks are also *locally asymptotic sparse*: the subgraphs of each community and the inter-community links are all spare, i.e. $\frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}} \to 0$ as $|V_i| \to \infty$ for any i, j, i.
- **Proposition 1**. Under the assumption of definition 3, we have

$$\mathcal{L}_1^{(i)} \approx \sum_{i,j\in\{1,\dots,k_i\}} e_{i,j}^{(i)} \ln \frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}} \tag{12}$$

  The proof is the derivation of (11) above.

At first glance $n_{i,j}^{(i)}$ is still there, but we note $n_{i,j}^{(i)} = \left|C_i^{(i)}\right|\left|C_j^{(i)}\right|$ and we can further decompose (12) as the following, proof in appendix A.2 at the end.

$$\sum_{i,j\in\{1,\dots,k\}} e_{i,j}^{(i)} \ln \frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}} = \sum_{i,j\in\{1,\dots,k\}} e_{i,j}^{(i)} \ln e_{i,j}^{(i)} - \sum_{i=1}^{k} e_{i,*}^{(i)} \ln\left|C_i^{(i)}\right| - \sum_{j=1}^{k} e_{*,j}^{(i)} \ln\left|C_j^{(i)}\right| \tag{13}$$

where $e_{i,*}^{(i)} = \sum_{j=1}^{k_i} e_{i,j}^{(i)}$ is the number of edges starting from a vertex of community $C_i^{(i)}$, and $e_{*,j}^{(i)} = \sum_{i=1}^{k_i} e_{i,j}^{(i)}$ is the number of edges ending at a vertex of community $C_i^{(i)}$.

- **Proposition 2**. On a large sparse graph (12) contributes $O(k_i|E_i|)$ time complexity to one complete sweep of updating $\mathbf{z}_i$.

  Proof: when $\mathbf{z}_i(v)$ varies, "$\ln\left|C_i^{(i)}\right|$" requires constant time to update, and as discussed earlier at most $\deg v$ of those $e_{i,j}^{(i)}$ might change, then adjustment of **Error! Reference source not found.**) takes $O(\deg v)$ time. Thus, an entire sweep of updating $\mathbf{z}_i$ takes $O(k_i|E_i|)$ time.

Iteration	1	2	3	4	5
Amazon 77222(1)	23677(2)/16100(3) (20.8%(4))	5389/2458 (3.2%)	1866/348 (0.45%)	710/122 (0.15%)	/(5)
PlosOne 53374	22096/19425 (36.4%)	9974/3642 (6.8%)	3161/1163 (2.2%)	1129/373 (0.7%)	512/143 (0.27%)
LinkedIn 34221	14020/8845 (25.8%)	3316/1963 (5.7%)	928/228 (0.7%)	297/115 (0.3%)	/
AMiner 30045	15903/5642 (18.8%)	3081/1274 (4.2%)	866/210 (0.7%)	282/108 (0.3%)	/
Fliker 21945	16422/10324 (47.0%)	7798/2666 (12.1%)	3596/503 (2.3%)	1322/369 (1.7%)	754/287 (1.3%)
Lastfm 26426	15650/11563 (43.8%)	7985/2994 (11.3%)	3297/620 (2.4%)	1617/294 (1.1%)	833/261 (1.0%)

Table 2 Membership switch statistics of various datasets, based on CAlign-Full-S. (1) total number of nodes; (2) number of nodes that switch membership; (3) number of nodes that switch to non-adjacent communities; (4) percentage of nodes switching to non-adjacent communities against total number of nodes; (5) algorithm converges and is terminated.

Here is where a heuristic kicks in to further reduce computation time: it quickly becomes less likely for a vertex to switch to a non-adjacent community after the initialization (see later) and the first iteration. Brief statistics for various datasets used in experiments of section 3 are shown in Table 2. This phenomenon is consistent with the dense intra-community connection and loose inter-community connection assumption mentioned in section 1. As the iterations go on, many nodes become "internal" inside a community and effectively stop switching membership.

Based on above observation, we propose the following strategy named *CAlign-Fast* to boost update of $\mathbf{z}$. First define a probability parameter p, and let $p_v = p$ for every node v in the network. Starting form the second iteration, a node can try non-adjacent communities if it switches to non-adjacent communities in the previous iteration. Otherwise, with probability p_v it can try non-adjacent communities; if the attempt fails, then $p_v = p_v^2$. We call our model without this strategy as *CAlign-Full*. as

Initialization of $\mathbf{z}$. We recommend an initialization of z based on the observation that high-degree node and its neighbors tend to be in the same community. So, we start with a high-degree node as a singleton community, expand the community by putting those of its neighbors in the same community whose number of edges to and from the community are higher than the average neighborhood degree by a threshold, see algorithm 1.

ALGORITHM 1: Initialization of membership $\mathbf{z}$ for a network G

input: a network G; number of communities k; *max_expan*: indicating the maximum number of expansion for each iteration; *t(avg_deg, i, j)*: a degree threshold function that grows with iteration index i and expansion index j.

```
for i ← 1 to k do
    Find the node of highest degree not assigned to any community, add it to C_i
    N(C_i) ← neighborhood of C_i
    avg_deg ← average degree of N(C_i)
    j ← 0
    while j < max_expan and there is new node in N(C_i) do
        for each node v in C_i do
            if the number of edges between v and C_i > t(avg_deg, i, j) then
                C_i ← v;
            end
            j ← j + 1
        end
    end
end
for each node v without community assignment do
    assign it to the community that has most links to v
end
```

output: initialized membership vector $\mathbf{z}$.

In our experiment of section 3, we use $t = (avg\_deg) \times 1.15^i \times \left(1.5 + \frac{0.5j}{k_i}\right)$, which grows with index i, j because we find the heuristic of algorithm 1 becomes less reliable as i, j increase. The initialization can be implemented in $O(|E|)$ time. It almost halves the iterations needed for convergence than a random initialization and always leads to a higher likelihood in our experiment.

Update community representation $\mathbf{Y}$ ***& mix vectors*** $\mathbf{\Theta}$ ***before node alignment***. We follow [27] and use gradients to update the parameter estimation of the Dirichlet-multinomial mixture in (4). As before, let $\mathbf{y}_{a,l}^{(i)} = \mathbf{\Phi}_a \boldsymbol{\theta}_l^{(i)}$, and we update $\mathbf{y}_{a,l}^{(i)}$s in order to update $\mathbf{\Phi}_a$. Let $\mathbf{n}_{a,l}^{(i)}$ be a counting vector of length $|a|$ s.t. $\mathbf{n}_{a,l}^{(i)}$ is the number of occurrences of attribute values in community $C_l^{(i)}$.

- **Proposition 3**. The gradient of (4) w.r.t. $\mathbf{y}_{a,l}^{(i)}$, denoted by $\nabla_{\mathbf{y}_{a,l}^{(i)}}$, is a vector of length $|a|$ given by

$$\nabla_{\mathbf{y}_{a,l}^{(i)}} = \Psi\left(\left\|\mathbf{y}_{a,l}^{(i)}\right\|_1\right) - \Psi\left(\left\|\mathbf{n}_{a,l}^{(i)} + \mathbf{y}_{a,l}^{(i)}\right\|_1\right) + \Psi\left(\mathbf{n}_{a,l}^{(i)} + \mathbf{y}_{a,l}^{(i)}\right) - \Psi\left(\mathbf{y}_{a,l}^{(i)}\right) \tag{14}$$

where we slightly abuse the notation for clarity: $\Psi(\cdot)$ applied on a vector means applying the digamma function on every component of that vector, and a number plus a vector means adding that constant to every component of that vector. Likewise, gradient of (4) w.r.t. $\boldsymbol{\theta}_l^{(i)}$, denoted by $\nabla_{\boldsymbol{\theta}_l^{(i)}}$, is a vector of size k determined by

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_l^{(i)}}(i) &= \sum_{a\in\mathcal{A}}\left(\left(\Psi\left(\left\|\mathbf{y}_{a,l}^{(i)}\right\|_1\right) - \Psi\left(\left\|\mathbf{n}_{a,l}^{(i)} + \mathbf{y}_{a,l}^{(i)}\right\|_1\right)\right)\|\mathbf{\Phi}_a(\cdot,i)\|_1 \right.\\ &\left. + \sum_{x\in a}\left(\Psi\left(\mathbf{n}_{a,l}^{(i)}(x) + \mathbf{y}_{a,l}^{(i)}(x)\right) - \Psi\left(\mathbf{y}_{a,l}^{(i)}(x)\right)\right)\mathbf{\Phi}_a(x,i)\right)\\ &- \sum_{j=1,\dots,N,j\neq i}\lambda_{i,j}\left\|\mathbf{\Theta}_j(i,\cdot)\right\|_1 \\ & i = 1,..,k\end{aligned} \tag{15}$$

The complexity of (14) and (15) are constant since the attribute set and attribute dimensions are fixed constants. Thus, computing all $\nabla_{\mathbf{y}_{a,l}^{(i)}}$ takes $O(k_i)$ time, and computing all $\nabla_{\boldsymbol{\theta}_l^{(i)}}$ takes $O(kk_i)$ time. We have a constant time Newton-Raphson update for $\mathbf{y}_{a,l}^{(i)}$ by

$$\mathbf{y}_{a,l}^{(i),\text{new}} = \mathbf{y}_{a,l}^{(i),\text{old}} - \mathbf{H}_{\mathbf{y}_{a,l}^{(i)}}^{-1}\nabla_{\mathbf{y}_{a,l}^{(i)}} \tag{16}$$

where $\mathbf{H}_{\mathbf{y}_{a,l}^{(i)}}$ is the Hessian matrix of (4) w.r.t. $\mathbf{y}_{a,l}^{(i)}$, due to the nice algebraic structure of $\mathbf{H}_{\mathbf{y}_{a,l}^{(i)}}$. Thus, updating all $\mathbf{y}_{a,l}^{(i)}$ can be completed in $O(k_i)$ time.

- **Proposition 4**. The Hessian of (4) w.r.t. $\mathbf{y}_{a,l}^{(i)}$, denoted by $\mathbf{H}_{\mathbf{y}_{a,l}^{(i)}}$, is a $|a| \times |a|$ matrix given by

$$\mathbf{H}_{\mathbf{y}_{a,l}^{(i)}} = \mathbf{\Lambda} + c\mathbf{1}\mathbf{1}^{\mathrm{T}} \tag{17}$$

where $\mathbf{\Lambda}$ is a diagonal matrix s.t.

$$\mathbf{\Lambda}(x,x) = \Psi'\left(\mathbf{n}_{a,l}^{(i)}(x) + \mathbf{y}_{a,l}^{(i)}(x)\right) - \Psi'\left(\mathbf{y}_{a,l}^{(i)}(x)\right), x \in a$$

and $c = \Psi'\left(\left\|\mathbf{y}_{a,l}^{(i)}\right\|_1\right) - \Psi'\left(\left\|\mathbf{n}_{a,l}^{(i)} + \mathbf{y}_{a,l}^{(i)}\right\|_1\right)$.

Unfortunately, Newton-Raphson update for all $\boldsymbol{\theta}_l^{(i)}$ will take unacceptable $O(k^4)$ time, so we instead use gradient ascent in (18) where η_l is step.

$$\boldsymbol{\theta}_l^{(i),\text{new}} = \boldsymbol{\theta}_l^{(i),\text{old}} + \eta_l\nabla_{\boldsymbol{\theta}_l^{(i)}} \tag{18}$$

Let $\mathbf{y}_{a,l}^{(i),*} = \mathbf{\Phi}_a\left(\boldsymbol{\theta}_l^{(i)} + \eta_l\nabla_{\boldsymbol{\theta}_l^{(i)}}\right)$, then the optimal step η_l is the root of the following complicated equation and hence hard to solve.

$$\begin{aligned}&\left(\Psi\left(\left\|\mathbf{y}_{a,l}^{(i),*}\right\|_1\right) - \Psi\left(\left\|\mathbf{n}_{a,l}^{(i)} + \mathbf{y}_{a,l}^{(i),*}\right\|_1\right)\right)\left\|\mathbf{\Phi}_a\nabla_{\boldsymbol{\theta}_l^{(i)}}\right\|_1 \\ &+ \sum_{x\in a}\left(\Psi\left(\left(\mathbf{n}_{a,l}^{(i)} + \mathbf{y}_{a,l}^{(i),*}\right)(x)\right) - \Psi\left(\mathbf{y}_{a,l}^{(i),*}(x)\right)\right)\nabla_{\boldsymbol{\theta}_l^{(i)}}(x) = 0\end{aligned} \tag{19}$$

We just start with a conservative value $\eta_l = 0.05$, check a fixed number of $2\eta_l, 3\eta_l$, etc. to see if they are better. The complexity of updating all $\mathbf{\Theta}_i$, including (18) and (19), is $O(kk_i)$.

Update $\mathbf{\Theta}$ ***& meta communities*** $\mathbf{\Phi}$ ***after node alignment***. Based on current $\mathbf{\Theta}$, we can perform node alignment with respect to the bipartite community clusters as discussed in section 2.2. After that, we first compute $\mathbf{C}_{i,j}$ by

$$\mathbf{C}_{i,j}(i,j) = \frac{\sum_{u\in C_i^{(i)},v\in C_i^{(i)}}\mathbf{V}_{i,j}(u,v)}{2\sum_{u\in C_i^{(i)}}\left\|\mathbf{V}_{i,j}(u,\cdot)\right\|_1} + \frac{\sum_{u\in C_i^{(i)},v\in C_i^{(i)}}\mathbf{V}_{i,j}(u,v)}{2\sum_{v\in C_j^{(j)}}\left\|\mathbf{V}_{i,j}(\cdot,v)\right\|_1} \tag{20}$$

$\mathbf{C}_{i,j}$ computed in this way satisfies $\mathbf{C}_{i,j} = \mathbf{C}_{j,i}^{\mathrm{T}}$. Then we perform SVD on $\mathbf{C}_{i,j}$ so that

$$\mathbf{C}_{i,j} = \boldsymbol{\mathcal{U}}_{i,j}\boldsymbol{\Lambda}_{i,j}\boldsymbol{\mathcal{V}}_{i,j}^{\mathrm{T}} \Rightarrow \begin{cases} \boldsymbol{\Theta}_i = \sqrt{k\boldsymbol{\Lambda}_{i,j}}\,\boldsymbol{\mathcal{U}}_{i,j}^{T} \\ \boldsymbol{\Theta}_j = \sqrt{k\boldsymbol{\Lambda}_{i,j}}\,\boldsymbol{\mathcal{V}}_{i,j}^{\mathrm{T}} \end{cases} \quad (21)$$

By property of SVD, the solution of $\boldsymbol{\Theta}_i, \boldsymbol{\Theta}_j$ are unique. For large network where each community has at least hundreds of nodes, this update of $\boldsymbol{\Theta}_i, \boldsymbol{\Theta}_j$ from will not change a lot and can be viewed as feedback adjustment from the node alignment. Let $\mathbf{Y}_a = \left(\mathbf{y}_{a,1}^{(1)}, \dots, \mathbf{y}_{a,k_1}^{(1)}, \dots, \mathbf{y}_{a,1}^{(N)}, \dots, \mathbf{y}_{a,k_N}^{(N)}\right)$, we now can update $\boldsymbol{\Phi}_a$ as

$$\mathbf{Y}_a = \boldsymbol{\Phi}_a\boldsymbol{\Theta} \Rightarrow \boldsymbol{\Phi}_a = \mathbf{Y}_a\boldsymbol{\Theta}^{+} \quad (22)$$

for every $a \in \mathcal{A}$ where $\boldsymbol{\Theta}^{+}$ is the Moore-Penrose inverse of $\boldsymbol{\Theta}$, which is also computed by SVD. Updates in (15) and (22) will converge because SVD is numerical stable. The feedback to $\boldsymbol{\Theta}_i, \boldsymbol{\Theta}_j$ will become less and less as fewer and fewer nodes switch alignment [28], and consequently change in $\boldsymbol{\Phi}_a$ will also decrease through the process. Suppose $k_i \le k_j, k < s_k = \sum_{i=1}^{N} k_i$, the time complexity of (15) is $O(k_i k_j^2)$ and of (22) is $O(ks_k^2)$.

Initialization of $\mathbf{Y}, \mathbf{C}, \boldsymbol{\Theta}, \boldsymbol{\Phi}$. Each $\mathbf{y}_{a,l}^{(i)}$ is the Dirichlet-multinomial parameter of a local community $C_l^{(i)}$, and we initialize it as the sample mean $\mathbf{y}_{a,l}^{(i)} = \frac{\mathbf{n}_{a,l}^{(i)}}{\left|C_l^{(i)}\right|}$. Community alignment matrix $\mathbf{C}_{i,j}$ is initialized as a cosine-similarity matrix of every pair in $\mathcal{C}_i \times \mathcal{C}_j$ based on the aggregated node attributes, and $\boldsymbol{\Theta}_i, \boldsymbol{\Theta}_j$ are initialized by SVD as in (21). $\boldsymbol{\Phi}_a$ is initialized according to (22).

Summary and choice of k_i, k. Putting everything together, we summarize the entire algorithm of our model in algorithm 2, for the case of $N = 2$. The design of our model is suitable for a general finite N, but we have to integrate alignment consistency [2] and find an efficient N-partite graph clustering. Therefore, this is left as future work.

ALGORITHM 2: CAlign-Full/Fast, $N = 2$

input: attributed networks $G_i, i = 1,2$; number of communities k_1, k_2; number of meta communities k; convergence threshold r (default 1%); necessary input for algorithm 1 and the network alignment model plugin.

```
initialize membership vectors as in algorithm 1, O(|E|)
initialize all necessary auxiliary variables like e_{i,j}, n_{i,j} in equation (13), O(|V|)
initialize Y, C, Θ, Φ in order, O(k_1(k_1 + k_2)^2)
while ratio of nodes that switch membership or alignment is higher than r do
    for each node v in each network do
        update its membership z(v) according to equation (9)
        Full: consider all communities for switch, O(k_i deg v)
        if not first iteration then
            Fast: consider adjacent communities for switch, almost O(deg v)
        end
    end
    update Y by equation (16), O(k_i)
    update Θ by equation (18), O(kk_i)
    update C by equation (7), cluster the weighted bipartite community graph, O(k_1 k_2)
    compute node alignment w.r.t. the clusters using a chosen model, scales at almost 1/k_i of
    the original complexity
    update Θ again by equation (21), O(k_1 k_2^2)
    update Φ by equation (22), O(k_1(k_1 + k_2)^2)
end
```

output: $\mathbf{z}_1, \mathbf{z}_2, \mathbf{Y}, \mathbf{C}, \boldsymbol{\Theta}, \boldsymbol{\Phi}$.

For choice of k, our approach is to sample sub-networks from G_1, G_2, consider their singular values $\sigma_1, \sigma_2, \dots$, and let k be the one such that $\sigma_1 + \cdots + \sigma_k$ account for 80% of the sum of all singular values. Repeat several times and take the average. If k is larger than $\min\{k_1, k_2\}$, let $k = \min\{k_1, k_2\}$.

For choice of k_1, k_2, [29] and [30] use the method to maximize the likelihood on held-out sub-network samples, but we have another more important concern. We let $k_i = |V_i|^{\alpha}, i = 1,2$ for some power α. Let d_i be the densification power of G_i, $|V| = \max\{|V_1|, |V_2|\}$, $d = \max\{d_1, d_2\}$, then under this setup, the overall ***time complexity of CAlign*** is $O(\max(|V|^{3\alpha}, |V|^{\alpha+d}))$, sub-quadratic if $\alpha < \min\left(\frac{2}{3}, 2 - d\right)$. The datasets in this paper have their $d < 1.3$ (social networks and co-author networks). As an overall consideration, we recommend use $\alpha = 0.4$ for our experiments.

CAlign-Fast saves time by a small power dependent on d, the network size, and how much of the algorithm is running under $O(|V|^{\alpha+d})$ complexity. For example, if $\alpha = 0.4, |V| = 10^6, d = 1.2$, and 20% of the algorithm runs under $O(|V|^{1.6})$, then the overall complexity is of power $\log_{|V|}(0.8 + 0.2|V|^{0.4})|V|^{1.2} \approx 1.6 - \log_{|V|}0.2 \approx 1.49$. This downgrade, seemingly small, but is actually remarkable; it means CAlign-Fast could save 75% time from CAlign-Full for 1 million nodes.

Finally, the ***space complexity of CAlign*** is determined by the maximum size of $\mathbf{z}_1, \mathbf{z}_2, \mathbf{Y}, \mathbf{C}, \boldsymbol{\Theta}, \boldsymbol{\Phi}$ and auxiliary variables including $e_{i,j}^{(i)}, n_{i,j}^{(i)}, \mathbf{n}_{a,l}^{(i)}$, which clearly scales at most $O(k_1 k_2)$, less than $O(|V|)$ when $\alpha = 0.4$.

3. EXPERIMENTS & EVALUATION

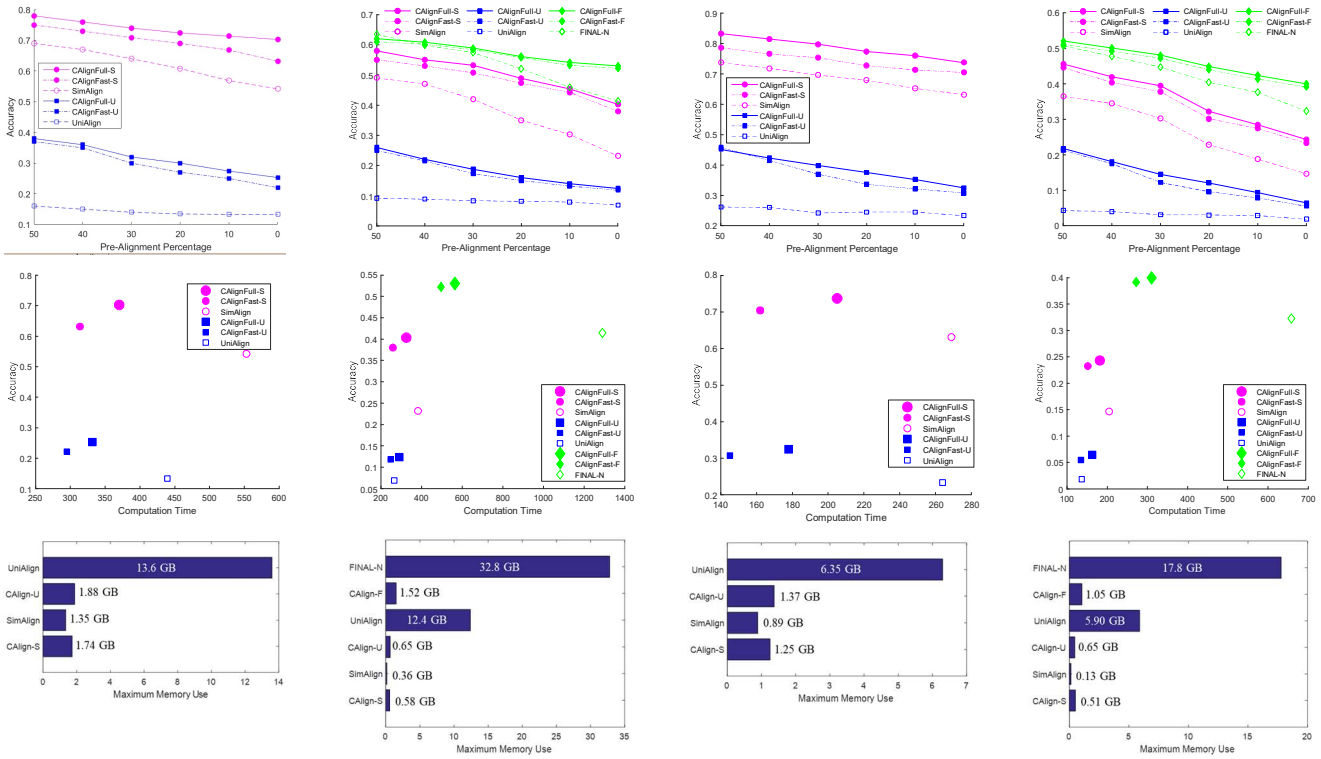
The experiments for our proposed C-Align model framework are done with Matlab on an 8-core Intel i7 3.00GHz machine with 32GB memory. We compare CAlign-Full/Fast with the Sim-Align described at the end of section 2.2, Uni-Align in [11] and FINAL-N in [12], using them as plugin models. We use the initial big letter to indicate which mode we use, e.g. CAlign-Fast-S means we use Sim-Align for CAlign-Fast. For all networks, we use attributes that have been experimented in pervious papers [11, 12, 29], etc., including degree, h-index (reflecting node significance), number of adjacent triangles, clustering coefficient and neighborhood average degree (reflecting local topology). All sub-networks are extracted by sampling edges, which better preserves topological information than sampling nodes.

3.1 Performance & Efficiency Analysis

Homogeneous networks. We experiment on two homogeneous datasets with rich text attributes.

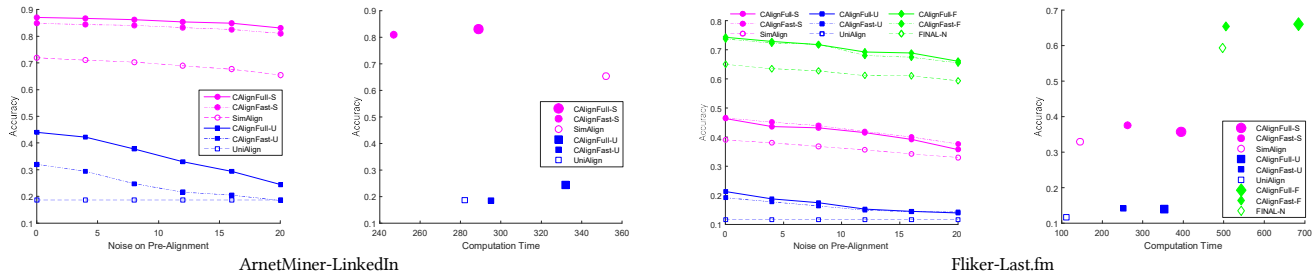
- Two Amazon co-purchase networks from [31]. In both networks, nodes are products, mostly book, DVD, music and video. If a product is frequently co-purchased with another product, then there is an edge between them. Two sub networks of about 40,000 nodes are extracted so that can be handled by all algorithms tested here. The first network contains 37,201 nodes and 104,105 edges. The second network contains 40,021 nodes and 144,284 edges. The two networks have 29,760 nodes in common as ground truth. We use top-2000 lemmatized title word counts as non-topological attributes of each product, and

  the count of other words is summed as one attribute "other". Densification power: 1.12; $k_1 = 69, k_2 = 69, k = 32$.
- Two PlosOne coauthor networks from http://journals.plos.org. In both networks, nodes are article authors, and there is an edge between two authors if they ever co-authored one article. The first network contains 21,211 distinct nodes and 116,032 edges. The second network contains 32,163 distinct nodes and 176,035 edges. The two networks have 6,507 overlapping nodes as ground truth. We extract top-2000 most frequent words in



Amazon Co-purchase network (with text attributes) | Amazon Co-purchase network (without text attributes) | PlosOne Coauthor network (with text attributes) | PlosOne Coauthor network (without text attributes)

Figure 3(a) Experiment results of CAlign with other models on two homogeneous networks. First row: alignment accuracy with varying percentage of pre-alignment. Second row: time-accuracy plot for the case of no pre-alignment. Third row, maximum memory use recorded when running the experiment; CAlign-Full/Fast have the same memory use, so we let CAlign represent both.



ArnetMiner-LinkedIn | Fliker-Last.fm

Figure 3(b) Experiment results of CAlign with other models on two heterogenous networks. Memory use is omitted for they are similar to (a). First chart: alignment accuracy with varying percentage noise on name-match based pre-alignment. Second chart: time-accuracy plot for the case of 20% noise.

- scientific terms recognized from the article texts by techniques developed in [32], and count their frequencies as non-topological node attributes. Densification power: 1.16; $k_1 = 54, k_2 = 63, k = 25$.

Note we are unable to pre-align nodes by name for above homogenous datasets because their ground truth is exactly established by name. As an alternative, we randomly sample a percent of the nodes and treat their alignment as known, from 50% to 0%. Accuracy are measured by how much of the remaining node can be correctly aligned, the same as [11] and [12]. Experiments at each percentage level are repeated 3 times, and the average results are shown in Figure 3(a).

Heterogeneous networks. We then experiment on two heterogeneous datasets from [2].

- *ArnetMiner-LinkedIn.* The ArnetMiner coauthor dataset comes with rich user profile like location, gender, sign, hobby, job. It in total has 1,053,188 nodes and 3,916,907 edges. In the LinkedIn "co-viewed" network, two users have an edge if their profile are frequently viewed by the same visitor. The dataset has everything you can see on LinkedIn, including skills, educating background and career history; it has 2,985,414 nodes and 25,965,384 edges. The rich profiles are broken down to lemmatized word counts as non-topological attributes. The entire data set is later used for parallelization and scalability tests. The ground truth contains 4,269 matches, and we extract two sub-networks containing it. The ArnetMiner network contains 30,045 nodes and 83,946 edges. The LinkedIn network contains 34,221 nodes and 112,852 edges. Densification power: 1.10 and 1.14; $k_1 = 62, k_2 = 65, k = 30$.
- *Fliker-Last.fm.* Edges in flicker dataset represent friendship, in Last.fm dataset represent following relationship. Only username and gender are available as non-topological attributes. We compute the Jaro-Winkler distance between names and split its range to three categories. The first network has 21,945 nodes and 544,217; the second network has 26,426 nodes and 226,506 edge. The ground truth has only 641 matches for this data set. Densification power: 1.30 and 1.21; $k_1 = 55, k_2 = 59, k = 31$.

For both networks, name-match has 70% and 37% accuracy on the ground truth. Following [12], we add 5% to 20% error to the pre-alignment. The results are shown in Figure 3(b). We omit demo of memory use for space, and they are similar to Figure 3(b).

Main result. By design in section 2.2, CAlign combines with other algorithms for alignment. Based on Figure 3, when combining with Sim-Align and Uni-Align, it achieves a +10% ~ +20% performance improvement, by introducing topological information originally not compatible with those algorithms. The improvement is especially obvious when rich attributes are not available. CAlign generally achieves better performance in comparison to FINAL-N, up to +15% when no pre-alignment is fed to FINAL-N.

On the scale of a network about 20,000 to 40,000 nodes, CAlign already runs faster on sufficiently sparse network. In Amazon dataset and PlosOne dataset, it uses only half of the computation time of FINAL-N. The exception is Fliker-Last.fm, CAlign runs much slower on this dataset because this network has a relatively higher densification power $d = 1.3$ that result in much more edges, and the complexity of CAlign grows with the number of edges. Despite this, since CAlign still scales sub-quadratically when $d = 1.3$, it will eventually outrun other models.

Moreover, CAlign makes good use of memory. Unlike UniAlign and FINAL-N, there is no large matrix operations since the matrices are "divided" by the "communities". Both UniAlign and FINAL-N are not able to run on larger datasets for our experiments due to the memory limit.

CAlign-Fast. Sometimes it is not performing as good as CAlign-Full, because of accumulation of errors through the iterations. This error might not be much in terms of community, but have certain impact on alignment. However, we still recommend it for large or "relatively denser" networks like in Fliker-Last.fm experiment for efficiency. When higher performance is desired, we can either postpone the partial membership switch to a later iteration, or increase the rate of full switch.

Rich attributes. They do help improve alignment performance: the performance is generally 20% higher for Amazon co-purchase network and 35% higher for PlosOne coauthor network when text attributes are considered. The results of ArnetMiner-LinkedIn experiment, which considers rich attributes, is also better than the Fliker-Last.fm experiment.

SimAlign, UniAlign and FINAL-N. When network is of moderate size and rich attributes are present, the simple SimAlign can already achieve good result. UniAligh might be of good use for a small network, but might not perform well on a large network, possibly because it is non-iterative and relaxing the permutation matrix **P** introduces too much error. FINAL-N has best overall performance among the three in case of no rich attributes, but it takes too much memory and we find its performance somewhat relies on the "preference matrix" **H** (the pre-alignment in our terminology). Altogether we choose the simple Sim-Align for our further experiments of scalability and parallelization.

3.2 Scalability & Parallelization

Scalability: We evaluated the scalability of CAlign, summarized in Figure 4. The first experiment is on entire Amazon data set, the second is on ArnetMine-LinkedIn dataset up to 1 million nodes. We can see the running time of both CAlign-Full and CAlign-Fast is sub-quadratic w.r.t. the number of network nodes. In particular, CAlign-Fast grows at a low power around 1.25 and looks almost linear. CAlign-Full grows at a power about 1.6. The accuracy of CAlign stays stable as network expands; we believe the communities that contain correctly aligned nodes "shield" them from the noise introduced by more nodes.

By Figure 5, the number of iterations to convergence grows with network size, but the growth is linear and slow:10 times node growth results in about 2 to 3 times number of iterations. Thus, its impact on time complexity is small, less than power of 0.1.

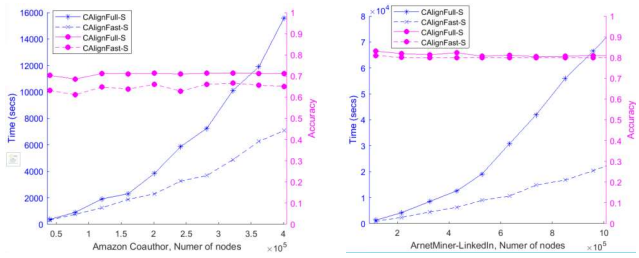


Figure 4 Scalability of CAlign on two datasets.

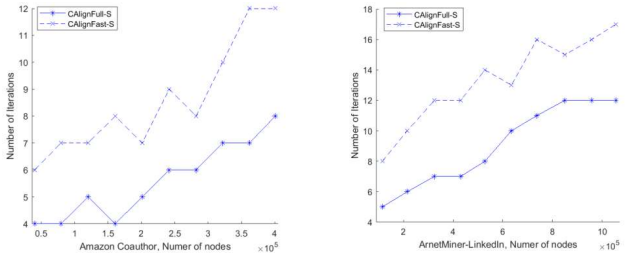


Figure 5 Iterations needed for convergence of CAlign on two datasets.

Parallelization. Our modeling approach easily lends itself to parallelization and a simple strategy as in Algorithm 3 can achieve 4 to 6 times acceleration on our 8-core virtual machine.

ALGORITHM 3: Parallelization of CAlign

1) After initialization in algorithm 1, for each network $G_i, i = 1,2$, starting with the community of smallest size, greedily group communities with more links into 4 partitions roughly of the same size $\frac{|V_i|}{4}$, based on adjacency information in $e_{i,j}^{(i)}$.
2) Launch 4 threads for each partition. Each thread is responsible for updating node membership and $\mathbf{y}_{a,l}^{(i)}, \boldsymbol{\theta}_l^{(i)}$ in its own partition. Only need to lock resources related to other partitions when updating "boarder" nodes.
3) After all threads report completion of one sweep, launch one thread to update $\boldsymbol{\Phi}$, and for each connected component of the community bipartite graph, launch a new thread to perform network alignment. Algorithm converges when less than 1% node switch membership or alignment.

Both the single-thread version and the parallelized version of CAlign-Full-S and CAlign-Fast-S are tested on the same data as in Figure 4. Time usage and memory usage are shown in Figure 6. Our simple parallelization achieves 3.2x to 3.8x acceleration. We achieve aligning 1 million nodes of ArnetMiner-LinkedIn dataset in about two hours' time.

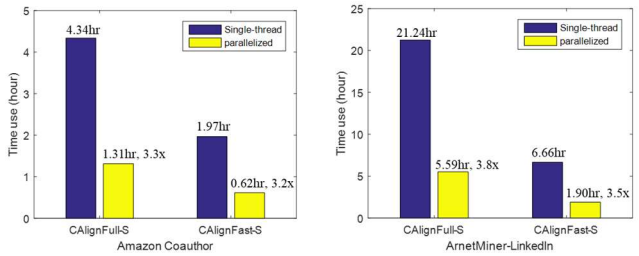


Figure 6 Parallelization acceleration of CAlign.

4. CONCLUSION

In this paper, we come up with the research question that if network community and network alignment can be integrated, based on that they share similar consistency assumptions. We target network alignment in this paper. We employ stochastic block model and Dirichlet multinomial and carefully construct sub-quadratic time model CAlign. Any existing alignment model with a cost function can plug in our framework. Communities and their alignment are intended to instruct alignment models to search for matches in right places. Experiments on homogeneous and heterogenous networks confirm our models are effective in terms of accuracy, time and memory use. At the end, we validate our model can handle millions of nodes. A simple parallelization strategy is designed and achieves more than 3 times acceleration on an 8-core desktop computer.

We have approached our research question from one perspective, but other aspects of the problem remain unsolved. This paper targets alignment, which tolerates error in community. We look forward to truly jointly solving three sub-problems defined in section 2.1 altogether in the future.

5. REFERENCES

[1] Zhang, J. and Yu, P. S. Pct: partial co-alignment of social networks. World Wide Web Conference, 2016.

[2] Zhang, Y., Tang, J., Yang, Z., Pei, J. and Yu, P. S. Cosnet: Connecting heterogeneous social networks with local and global consistency. 2015.

[3] Liu, R., Cheng, W., Tong, H., Wang, W. and Zhang, X. Robust Multi-Network Clustering via Joint Cross-Domain Cluster Alignment. 2015.

[4] Singh, R., Xu, J. and Berger, B. Global alignment of multiple protein interaction networks with application to functional orthology detection. Proceedings of the National Academy of Sciences, 105, 35 (2008), 12763-12768.

[5] Crawford, J., Sun, Y. and Milenković, T. Fair evaluation of global network aligners. Algorithms for Molecular Biology, 10, 1 (2015), 19.

[6] Song, B., Gao, J., Ke, W. and Hu, X. Achieving high k-coverage and k-consistency in global alignment of multiple PPI networks. 2016.

[7] Lacoste-Julien, S., Palla, K., Davies, A., Kasneci, G., Graepel, T. and Ghahramani, Z. Sigma: Simple greedy matching for aligning large knowledge bases. 2013.

[8] Smalter, A., Huan, J. and Lushington, G. Gpm: A graph pattern matching kernel with diffusion for chemical compound classification. 2008.

[9] Singh, R., Xu, J. and Berger, B. Pairwise global alignment of protein interaction networks by matching neighborhood topology. 2007.

[10] Bayati, M., Gerritsen, M., Gleich, D. F., Saberi, A. and Wang, Y. Algorithms for large, sparse network alignment problems. 2009.

[11] Koutra, D., Tong, H. and Lubensky, D. Big-align: Fast bipartite graph alignment. 2013.

[12] Zhang, S. and Tong, H. Final: Fast attributed network alignment. 2016.

[13] Yang, J. and Leskovec, J. Defining and evaluating network communities based on ground-truth. Knowledge and Information Systems, 42, 1 (2015), 181-213.

[14] Aggarwal, C. C. An introduction to social network data analytics. Social network data analytics (2011), 1-15.

[15] Holland, P. W., Laskey, K. B. and Leinhardt, S. Stochastic blockmodels: First steps. Social networks, 5, 2 (1983), 109-137.

[16] Abbe, E. Community detection and the stochastic block model (2016).

[17] Karrer, B. and Newman, M. E. Stochastic blockmodels and community structure in networks. Physical Review E, 83, 1 (2011), 016107.

[18] Frieze, A. and Karoński, M. Introduction to random graphs. Cambridge University Press, 2015.

[19] Durrett, R. Random graph dynamics. Cambridge university press Cambridge, 2007.

[20] Leskovec, J., Kleinberg, J. and Faloutsos, C. Graphs over time: densification laws, shrinking diameters and possible explanations. 2005.

[21] Ng, K. W., Tian, G.-L. and Tang, M.-L. Dirichlet and related distributions: Theory, methods and applications. John Wiley & Sons, 2011.

[22] Frigyik, B. A., Kapila, A. and Gupta, M. R. Introduction to the Dirichlet distribution and related processes. Department of Electrical Engineering, University of Washignton, UWEETR-2010-0006 (2010).

[23] Weston, J., Elisseeff, A., Schölkopf, B. and Tipping, M. Use of the zero-norm with linear models and kernel methods. Journal of machine learning research, 3, Mar (2003), 1439-1461.

[24] Aicher, C., Jacobs, A. Z. and Clauset, A. Adapting the stochastic block model to edge-weighted networks. arXiv preprint arXiv:1305.5782 (2013).

[25] Larremore, D. B., Clauset, A. and Jacobs, A. Z. Efficiently inferring community structure in bipartite networks. Physical Review E, 90, 1 (2014), 012805.

[26] Barabási, A.-L. Network science. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 371, 1987 (2013), 20120375.

[27] Minka, T. Estimating a Dirichlet distribution. Technical report, 2000.

[28] Stewart, G. W. Perturbation theory for the singular value decomposition. 1998.

[29] Yang, J., McAuley, J. and Leskovec, J. Community detection in networks with node attributes. 2013.

[30] Airoldi, E. M., Blei, D. M., Fienberg, S. E. and Xing, E. P. Mixed Membership Stochastic Blockmodels. Journal of Machine Learning Research, 9 (2008), 1981-2014.

[31] Leskovec, J., Adamic, L. A. and Huberman, B. A. The dynamics of viral marketing. ACM Transactions on the Web (TWEB), 1, 1 (2007), 5.

[32] Zheng Chen, E. Y. Domain-Independent Term Extraction & Term Network for Scientific Publications. In Proceedings of the iConference 2017 (2017)

A.1 $\mathcal{L}_1^{(i)} = \sum_{i,j\in\{1,\dots,k\}}\left(e_{i,j}\ln\frac{e_{i,j}}{n_{i,j}} - e_{i,j} + n_{i,j}\sum_{n=1}^{\infty}\frac{\left(\frac{e_{i,j}}{n_{i,j}}\right)^{n+1}}{n(n+1)}\right) = \sum_{i,j\in\{1,\dots,k\}} e_{i,j}\ln\frac{e_{i,j}}{n_{i,j}} - \sum_{i,j\in\{1,\dots,k\}} e_{i,j} + \sum_{i,j\in\{1,\dots,k\}}\left(\frac{n_{i,j}}{n(n+1)}\sum_{n=1}^{\infty}\left(\frac{e_{i,j}}{n_{i,j}}\right)^{n+1}\right) = \sum_{i,j\in\{1,\dots,k\}} e_{i,j}\ln\frac{e_{i,j}}{n_{i,j}} - |E| + \sum_{i,j\in\{1,\dots,k\}}\left(\frac{1}{n(n+1)}\sum_{n=1}^{\infty}\frac{e_{i,j}^{n+1}}{n_{i,j}^{n}}\right)$.

Observe the terms in the last sum in above expansion is of order $O\left(\frac{e_{i,j}^2}{n_{i,j}}\right), \forall i,j$, since the first term is $\frac{1}{n(n+1)}\frac{e_{i,j}^2}{n_{i,j}}$.

A.2 $\sum_{i,j\in\{1,\dots,k\}} e_{i,j}^{(i)}\ln\frac{e_{i,j}^{(i)}}{n_{i,j}^{(i)}} = \sum_{i,j\in\{1,\dots,k\}} e_{i,j}^{(i)}\ln e_{i,j}^{(i)} - \sum_{i,j\in\{1,\dots,k\}} e_{i,j}^{(i)}\ln|C_i^{(i)}| - \sum_{i,j\in\{1,\dots,k\}} e_{i,j}^{(i)}\ln|C_j^{(i)}| = \sum_{i,j\in\{1,\dots,k\}} e_{i,j}^{(i)}\ln e_{i,j}^{(i)} - \sum_{i=1}^{k}\left(\ln|C_i^{(i)}|\sum_{j=1}^{k} e_{i,j}^{(i)}\right) - \sum_{j=1}^{k}\left(\ln|C_j^{(i)}|\sum_{i=1}^{k} e_{i,j}^{(i)}\right) =$

$\sum_{i,j\in\{1,\dots,k\}} e_{i\,i}^{(i)}\ln e_{i\,i}^{(i)} - \sum_{i=1}^{k} e_{i\,*}^{(i)}\ln|C_i^{(i)}| - \sum_{j=1}^{k} e_{*\,j}^{(i)}\ln|C_i^{(i)}|$