Distributed Inference of Overlapping Communities

Xun Zheng
Machine Learning Department
Carnegie Mellon University
xzheng1@andrew.cmu.edu

Jingwei Zhuo
Computer Science Department
Carnegie Mellon University
jzhuo1@andrew.cmu.edu

Abstract

Overlapping community detection plays a key role in statistical network modeling. Despite the importance, popular models such as mixed membership stochastic blockmodels (MMSB) [2] are often not applicable to real world massive networks due to limited speed and memory of a single computing node. In this project, we develop distributed inference for models that can discover overlapping communities in real networks. Specifically, we address three key challenges in distributed network inference: 1) reduce $O(N^2)$ pairwise parameters to $O(N)$ by choosing constrained variational formulation described in [7]; 2) minimize communication cost between processes using a vertex-cut algorithm to partition the network; 3) synchronize global states between processes via a protocol designed for continuous aggregation and synchronization. Experimental results demonstrate our ability to tackle large scale network inference tasks.

1 Introduction

Statistical network modeling focuses on discovering hidden structures and properties from the network. A key problem among them is to identify communities from unlabeled data, where clusters are characterized by link patterns, i.e., dense internal connections and sparse external connections. The intuition behind is that the nodes in the same community have similar characteristics hence are more likely to connect to each other than the nodes in different communities. Classical methods in community detection such as [16, 12] assumes each node engages only in a single community. However it is evident that this assumption is rather unrealistic. Study on real data [19] provides evidence that overlapping communities exist in many social networks, making overlapping community detection preferred over single membership approach.

A number of methods [2, 3, 5, 17] have been proposed to address the overlapping community detection problem. In this project, we focus on mixed membership stochastic blockmodels (MMSB) [2], a probabilistic model that posits each node as a mixture of memberships. In particular, we consider a variant named assortative mixed membership stochastic blockmodel (a-MMSB) that captures assortativity by explicitly parameterizing intra-community and inter-community connectivity. Posterior inference in MMSB is intractable due to the normalization constant, hence approximate algorithms such as mean-field variational inference [11] comes into play. However, existing variational method [2] is hardly applicable to massive real-world networks because of the inefficiency: for N nodes and K communities, the time complexity per iteration is $O(N^2K^2)$ since it iterates over $O(N^2)$ node pairs and each update has quadratic complexity in K. Moreover, the memory complexity is $O(N^2K)$ since each pair (either link or non-link) is associated with a variational parameter of size K. For a toy network with 10000 nodes and 100 communities this already implies 40 GB of memory. Hence this method is clearly infeasible for real-world networks where N typically scales from tens to hundreds of millions.

To address this issue, stochastic variational inference (SVI) [9] has been applied to MMSB in [8], which achieved $O(NK)$ instead of $O(N^2K^2)$ per iteration complexity by 1) reducing the parameter space from $O(K^2)$ to $O(K)$ using explicit inter-community linking probability ϵ ; and 2) sampling random pairs of nodes to perform stochastic updates. However this method is inherently sequential, making it unable to utilize more computational resources. For instance it takes roughly 40 hours to infer Google webpage dataset with $N = 875,000$ and $K = 1,000$. Thus the biggest reported result on $N = 3.7$ M and $K = 1,000$ presumably takes weeks to converge, which is clearly undesirable for large scale network analysis. On the other hand, spectral methods have been successfully applied to overlapping community detection as well. In [10], a tensor decomposition approach with a provable consistency guarantee is introduced and efficiently parallelized. However the time complexity of $O(K^3)$ already precludes its application to large number of communities. Indeed, the biggest experiment reported in [10] is conducted with $K = 500$, whereas in real applications the ability to model tens of thousands of communities is always preferred.

In this project we address large scale overlapping community detection by parallelizing SVI algorithm for MMSB on multiple machines. Most existing distributed approach for latent variable model inference [1, 20, 4, 14] resort to data parallelism, i.e., partition the data into multiple machines and synchronize the shared parameters across different machines using a special component *parameter server*. However certain properties of MMSB and network data in general make it difficult to directly apply any of the existing approaches. In particular, we state three key challenges encountered in distributed network inference: 1) Excessive pairwise parameterization. In original mean-field variational approximation for MMSB [2, 8], both link and non-link pairs are associated with variational parameter of size K. This suggests that even if the input network is sparse, the resultant parameterized network is dense. It poses difficulty on distributed inference not only because there are too many parameters to estimate but also because different network partitions will share enormous amount of parameters, hence requiring heavy synchronization. 2) Efficient network partitioning. In distributed settings, one important factor of efficiency is minimizing the communication cost, or equivalently the amount of shared parameters between different machines. Existing approaches to distributed inference typically assumes data is trivially separable and distributes the workload by randomly partitioning the data. However for networks random partitioning will result in a great amount of shared nodes/edges, thus increased communication cost. 3) Parameter synchronization. It is now well accepted that the performance of traditional bulk-synchronous systems (e.g., [17]) is mostly determined by the slowest computing node, which is also known as the "curse of the last reducer". To avoid the inefficiency we need asynchronous communication between machines to keep shared parameters up-to-date. However implementing asynchronous parallel scheme is highly non-trivial, since it usually involves complicated resource management and access control.

In the following report, we introduce the MMSB model and its variational inference problem in Section 2. Section 3 describes the solution to efficient network partitioning that reduces the amount of communication between machines. We explain the overall architecture for distributed network inference and the asynchronous communication protocol in Section 4. Section 5 demonstrates some experimental results conducted on real networks and finally Section 6 concludes.

2 Stochastic VI for MMSB

In this section we introduce MMSB model, stochastic variational inference algorithm, and the link-based approach that leads to parsimonious pairwise parameterization.

2.1 Mixed membership stochastic blockmodel

Mixed membership stochastic blockmodel (MMSB) [2] is a popular model used in overlapping community detection. Unlike traditional community detection models where each node is assigned with one class label, MMSB posits each node to be a mixture of memberships, thus allowing the discovered communities to be overlapping. In particular, we consider assortative MMSB (a-MMSB), which is a variant of conventional MMSB.

Let K be the number of communities in the network. Let y_{ab} be the binary indicator of connection between node a and node b, in other words $y_{ab} = 1$ if a and b are linked, otherwise $y_{ab} = 0$. Let $\beta_k \in (0, 1)$ be the intra-community strength parameter that controls how dense the connections are within community k. Each node is associated with a mixture coefficient $\theta \in \Delta$, where Δ is a $(K-1)$-simplex; and each directed pair $(a \to b)$ is associated with a community indicator $z_{a\to b} \in \{1, 2, \ldots, K\}$.

The generative process for a-MMSB works as follows:

1. For each community k, draw an intra-community strength parameter $\beta_k \sim \text{Beta}(\eta)$.
2. For each node a, draw a mixture coefficient $\theta_a \sim \text{Dirichlet}(\alpha)$.
3. For each pair (a, b):
   (a) Draw a community assignment $z_{a\to b} \sim \text{Multinomial}(\theta_a)$.
   (b) Draw a community assignment $z_{a\leftarrow b} \sim \text{Multinomial}(\theta_b)$.
   (c) Draw connection $y_{ab} \sim \text{Bernoulli}(p)$, where $p = \begin{cases} \beta_k & \text{if } z_{a\to b} = z_{a\leftarrow b}, \\ \epsilon & \text{otherwise.} \end{cases}$

Given the generative procedure, our goal is to infer the posterior $p(\boldsymbol{\theta}, \boldsymbol{\beta}, \mathbf{z}|\alpha, \eta, \mathbf{y})$. Since exact inference is intractable, we resort to variational approximation.

2.2 Stochastic VI for MMSB

The classical mean-field approximation gives the following factorized distribution:

$$q(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{z}) = \prod_{n=1}^{N} q(\theta_n|\gamma_n) \prod_{a<b} q(z_{a\to b}|\phi_{a\to b}) q(z_{a\leftarrow b}|\phi_{a\leftarrow b}) \prod_{k=1}^{K} q(\beta_k|\lambda_k), \tag{1}$$

where the variational parameter γ is associated with nodes, ϕ is associated with pairs, and λ is the variational parameter for inter-community connectivity β. The problem with this formulation is there are $O(N^2)$ pairwise parameter ϕ, each with size K. Even if the original network is sparse (i.e. only a few links, which is also commonly observed in real networks), the number of parameter still depends quadratically on the number of nodes. For real networks quadratic complexity in data size is prohibitive since it can easily exceed any storage size. Not only so, dense parameterization poses difficulty on network partitioning and subsequent distributed inference, since it results in too many shared variables to synchronize.

Fortunately [7] introduced an approximation to avoid this issue by making slightly different variational assumptions compared to the classic mean-field assumptions. Since it can clamp the non-link pair parameters into nodes, we choose this "link-based" variational formulation instead of the classic mean-field approximation. Unlike (1), variational distribution in link-based approach is factorized as

$$\begin{aligned} q(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{z}) = \prod_{n=1}^{N} q(\theta_n|\gamma_n) \prod_{(a,b)\in links} q(z_{a\to b}, z_{a\leftarrow b}|\phi_{ab}) \\ \prod_{(a,b)\in nonlinks} q(z_{a\to b}|\phi_{a\to b}) q(z_{a\leftarrow b}|\phi_{a\leftarrow b}) \prod_{k=1}^{K} q(\beta_k|\lambda_k) \end{aligned} \tag{2}$$

with the constraints

$$\phi_{a\to b,k} = \frac{1}{d_a} \sum_{(a,b)\in link(a)} \sum_{l=1}^{K} \phi_{ab}^{kl}, \tag{3}$$

where d_a is the degree of a. The main difference lies in the indicator variables $\mathbf{z}$ and their variational parameters ϕ. The variational distribution of $\mathbf{z}$ is decomposed into two categories. For links ($y_{ab} =$

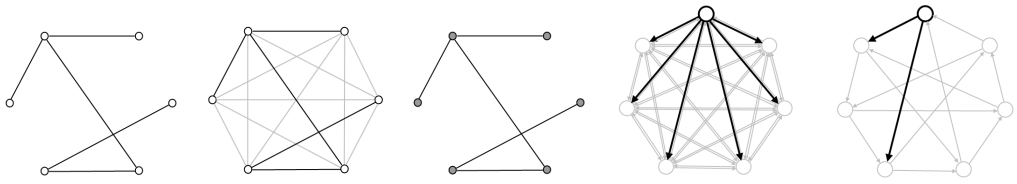

Figure 1: Original network (a), classic variational parameterization (b) and link-based variational parameterization (c). In (b) and (c) dark solid lines denote the link parameters $\phi_{a\to b}$ with $y_{ab} = 1$, light solid lines denote the non-link parameters $\phi_{a\to b}$ with $y_{ab} = 0$. Filled nodes denote the new variational parameters associated with the nodes. (d), (e): Refer relationship between shards, generated by random assignment and PDS algorithm. We focus on one partition and its refer set (dark). PDS only interacts with two partitions whereas random assignment requires all-to-all connection.

1), it is a joint distribution of $z_{a\to b}$ and $z_{a\leftarrow b}$ and controlled by a mixed parameter ϕ_{ab}. For non-links ($y_{ab} = 0$), it is a product of two independent distributions as the original mean-field assumption. The constraints of the non-link parameters $\phi_{a\to b}$ and $\phi_{a\leftarrow b}$ introduces dependency on interaction parameters, therefore reducing them into a node parameter $\phi_{a\to b,k}$. It not only reduces the number of variational parameters from $O(N^2)$ to $O(N)$, but also makes it easy to partition for distributed inference: the resultant parameterized network is as sparse as the original network. Figure 1 (a),(b), and (c) illustrates the different parameterization schemes.

Given the variational distribution, one can follow the standard procedure to obtain the evidence lower bound and coordinate ascent algorithm. We defer the detailed derivation to Appendix A.

3 Network partitioning

The first step in data-parallel distributed inference is to partition the data into smaller blocks. However, one of the major difference between network model and other latent variable models, e.g., topic models, is that the data is not clearly separable. In other words, it is impossible to partition the network data into non-overlapping shards. Thus our goal is to find a partition that minimizes the overlapping region between shards. Since the parameterized network has the same structure as the original network, also considering that the number of nodes are usually much smaller than number of edges in real networks, vertex-cut algorithms are preferred over edge-cuts since it keeps edges local while sharing the nodes between shards.

Assume we create L partitions, denoted by $G_1, \ldots, G_L$. For the shared node appeared in multiple partitions, we impose the relation that the node is "owned" by one partition and the node in other partitions is "borrowed" from the master partition. By doing so, we establish a connection between the partitions. More formally, for partition $G_i(V_i, E_i)$, we define $V_i = M_i \bigcup B_i$, where M_i denotes the nodes owned by this partition, and B_i denotes the nodes borrowed from other partitions. We use $master(u) = i$ to denote that node u is owned by partition i, then $M_i = \{u : u \in V_i, master(V_i) = i\}$ and $B_i = V_i - M_i$. We can further define the refer set $A_i = \{G_j : u \in M_i, u \in G_j, j \neq i, j = 1, ..., L\}$, which denotes the partitions that borrow nodes from G_i.

An efficient network partition should meet certain criteria: 1) The communication between different partitions should be minimized; 2) The scale of each partition should be balanced; 3) The communication pattern for different partition should be similar. Formally, 1) $|A_i|$ is as small as possible for any i; 2) G_i has almost the same size. 3) A_i has almost the same size; By constructing A_i as the perfect difference set [15, 6] and using a simple greedy algorithm for partition, we can meet all the criteria.

A perfect difference set (PDS) P of order L is $\forall 1 \leq i \leq L$, $\exists a, b \in P$, such that $a - b = i \mod L$. That is, every number form 1 to L can be expressed as the modulo of the difference of two numbers in P. With one PDS P, we can construct $A_i = \{G_j : j = (k + i) \mod L, k \in P\}$, which satisfies all the requests over the refer set. If we choose P such that $0 \in P$, the cardinality of the

Algorithm 1 PDS+greedy online algorithm

Input: Network $G = (V, E)$; Partition number: $L = p^2 + p + 1$, p is the prime number.
Output: K partitions: $G_1, ..., G_L$
  Construct basic PDS P of order L, such that $0 \in P$
  Construct the refer set A_i, $i = 1, ..., L$
  for each link $(u, v) \in G(V, E)$ **do**
    switch (link (u, v))
    case both u and v unobserved**:**
      assign (u, v) to random chosed partition G_i, and $master(u) = i$, $master(v) = i$;
    case u observed and v unobserved**:**
      assign (u, v) to partition $G_i \in G_{master(u)} \bigcup A_{master(u)}$ which minimizes the number of links associated with u and v in that partition, and $master(v) = i$;
    case both u and v observed**:**
      assign (u, v) to partition $G_i \in A_{master(u)} \bigcup A_{master(v)}$ which minimizes the number of links associated with u and v in that partition.
    end switch
  end for
  Output K partitions $G_1, ..., G_L$

resulting A_i will be the smallest. Figure 1 (d)&(e) compares the refer sets generated from PDS with those generated from naive random edge assignment. There are two advantages of PDS: 1) For any two partitions, there is only one direct connection while for random assignment there are two; 2) There are fewer total number of connections. This results in less communication between different machines when we perform inference in parallel. However a minor limitation of PDS is that it exists only for some specific order $L = p^2 + p + 1$ where p is the prime number.

With the nice property of PDS, we can use a simple greedy online algorithm to partition the network. The basic idea is that for every link (u, v), we assign it into the partition that has smallest degree of node u and v. This is not the optimal solution, and the result depends on the order of input links. However in practice it achieves reasonable results. The algorithm is described in Algorithm 1. Since each A_i is equivalent to the other A_j where $j \neq i$, and the greedy assignment strategy is independent from specific partition number i, this algorithm will generate the balanced partitions $G_1, ..., G_L$ where G_i has almost the same size.

Given the partition of data, the variational parameters can be partitioned as well: λ is defined as the global parameter which is shared by all the nodes and links. ϕ_{ab} for any $(a, b) \in E$ is the local link parameter. For each partition i, γ_a for any $a \in B_i$ are defined as the local "borrowed" node parameters and γ_b for any $b \in M_i$ are local "owned" node parameters.

4 Distributed SVI for MMSB

In this section we describe our methods on distributed inference for overlapping communities.

4.1 Parameter placement

Most of the *parameter server* approaches are based on key-value store, and hash functions are commonly used to index in to the keys. This results in uniform distribution of the keys, and hence nice load balancing of the servers. However, for distributed network inference, such uniform distribution of keys results in too many connections between machines, as shown in Figure 1 (d). Instead, we can place the parameters (i.e. the network) according to the network partition results. By doing so, we not only maximize the data locality, but also reduce the amount of communication as well as the number of machines need to connect to.

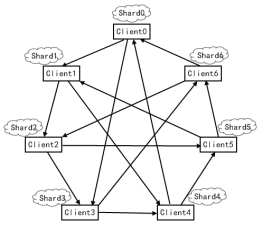

Figure 2: Parameter placement using PDS partitions.

Algorithm 2 Distributed SVI for MMSB

Server:
Partition G into $L = M$ blocks $G_i(V_i, E_i)$ and send partition information to workers.
repeat
  Receive $\partial\gamma$ or $\partial\lambda$, perform gradient descent using (8)
until receive terminating signal
Client-foreground:
Receive partition information, load data.
repeat
  Perform updates (4), (5), (7), (10)
  Update own γ using (6)
  Send borrowed $\partial\gamma$ to its master
  Send $\partial\lambda$ to server
until converged
Client-background:
repeat
  Fetch fresh values of borrowed γ and λ from servers
until receive terminating signal

4.2 Updates in a machine

Let $link_i(a) = \{(a,b) : b \in V_i, (a,b) \in E_i\}$ and $link(a) = \{(a,b) : (a,b) \in E\}$. For simplicity let $d_a^{(i)}$ be the cardinality of $link_i(a)$ and d_a be the cardinality of $link(a)$. As a result of the network partitioning, for each worker, only "own" parameters are updated directly, otherwise the gradients are send to the owners of the parameters, who performs gradient descent upon receiving the gradients.

For $(a,b) \in E_i$, we have a closed form update for local link parameters:

$$\phi_{ab,k}^t \propto \exp\left\{\mathbb{E}_{q_t}[\log\theta_{ak}] + \mathbb{E}_{q_t}[\log\theta_{bk}] + \mathbb{E}_{q_t}[\log\beta_k]\right\}. \tag{4}$$

Following the constraints, update for owned node parameters $a \in M_i$

$$\overline{\phi}_{a,k}^t = \frac{\sum_{(a,b)\in link_i(a)} \phi_{ab,k}}{d_a^{(i)}}, \tag{5}$$

$$\gamma_{a,k}^t = \gamma_{a,k}^{t-1} + \rho_t \partial\gamma_{a,k}^t, \tag{6}$$

where

$$\partial\gamma_{a,k}^t = \alpha_k + \sum_{(a,b)\in link_i(a)} \phi_{ab,k} + (n-1-d_a)\overline{\phi}_{a,k} - \gamma_{a,k}^{t-1}, \tag{7}$$

and learning rate $\rho_t = (\tau + t)^{\kappa}$, where κ is the decay parameter, τ is the rate, and t is the iteration number.

For community k, we have the following stochastic update for global parameters

$$\lambda_{k,0}^t = \lambda_{k,0}^{t-1} + \rho_t \lambda_{k,0}^t, \tag{8}$$

$$\lambda_{k,1}^t = \lambda_{k,1}^{t-1} + \rho_t \lambda_{k,1}^t, \tag{9}$$

where

$$\partial\lambda_{k,0}^t = \eta_0 + \sum_{(a,b)\in E_i} \phi_{ab,k} - \lambda_{k,0}^{t-1}, \tag{10}$$

$$\partial\lambda_{k,1}^t = \eta_0 + \left(\left(\sum_{a\in V_i} \overline{\phi}_{a,k}\right)^2 - \sum_{a\in V_i} \left(\overline{\phi}_{a,b}\right)^2\right)/2 - \sum_{(a,b)\in E_i} \overline{\phi}_{a,k}\overline{\phi}_{b,k}^{(i-1)} - \lambda_{k,1}^{t-1}. \tag{11}$$

Table 1: Datasets used in the experiments.

dataset	#node	#link	description
Youtube	1,134,890	2,987,624	Youtube online social network
web-Google	875,713	5,105,039	web graph from Google
LiveJournal	3,997,962	34,681,189	friendship network in LiveJournal

4.3 Asynchronous communication protocol

The server-client protocol is described in Algorithm 2. We simulate Hogwild! [18] in distributed environment by separating the foreground and background roles of client processes: the foreground role focuses on computing the gradients while the background role do best-effort asynchronous fetch to keep foreground computation consistent with other clients.

An advantage of this method is that the communication and computation is completely overlapped. In other words, no time is spent on waiting for remote responses. This is crucial in large scale inference since I/O waiting directly implies the waste of enormous amount of CPU cycles. Another advantage comes from the dedicated background synchronizer. In practice a dedicated synchronizer will synchronize the variables in a speed faster than the inference procedure happening in the foreground. This indicates that the parameters are mostly up-to-date. Also thanks to the efficient network partitioning, the synchronizer only need to establish connections with a few other machines. [1] reports the advantage of sparse connections over all-to-all connections in the cluster computing environment.

5 Experiments

In this section we demonstrate the performance of the distributed SVI. We first examine the benefit of the graph partitioning algorithm, and then the convergence speed and memory footprint of the distributed SVI on real networks. We use three real world network data[1] listed in Table 1.

Experiment settings: In the vertex-cut experiments we first convert the directed graph into undirected ones and removed all the duplicated edges, so as to accommodate subsequent processing for MMSB, which is designed to model undirected networks. For distributed SVI experiments, following the settings in [7], we set the hyperparameters to $\alpha = 1/K$, $\eta = 1$, and non-link probability $\epsilon =$ 1e-30. We fix the SVI forgetting rate $\kappa_\gamma = 0.5$ and $\kappa_\lambda = 0.9$, and tune the learning rate τ using cross validation.

5.1 Vertex-cut algorithm

We report the efficiency of the PDS vertex-cut algorithm. Since p is required to be 1 or a prime number, we chose $p = 1, 2, 3, 5$, that is, partition into $M = 3, 7, 13, 31$ shards respectively. We compare PDS algorithm with the baseline method that randomly assigns edges to different partitions. Since both of the algorithms can handle provided datasets within several seconds, we do not report the speed of the cut algorithm.

Figure 3 shows the replication factor, which is the total number of nodes generated by the algorithm divided by the network's original number of nodes. If the vertex-cut algorithm is inefficient, then it will generate more "ghost" nodes, i.e., higher replication factor, hence more synchronization burden to the subsequent distributed inference algorithm. As we can see from Figure 3, in all three datasets the PDS algorithm results in much fewer duplication compared to random edge assignment method. In addition, with increasing number of partitions the replication factor only increases slightly compared to the rapid increase of random edge assignment. This is a desired property since it allows the use of more computing nodes with higher marginal benefits, whereas inefficient cut algorithms will cancel out the benefit of additional computing resources by making the problem equally larger.

[1] http://snap.stanford.edu/data/

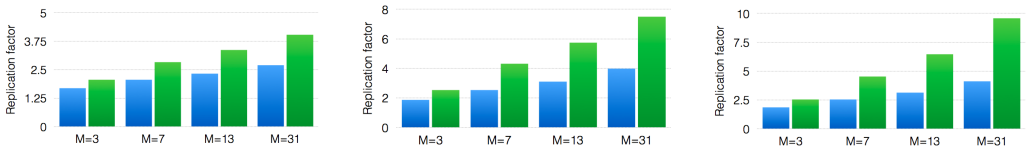

Figure 3: Duplication factor of PDS algorithm (blue) and random edge assignment (green) on three datasets: (a) Youtube; (b) web-Google; (c) LiveJournal. We can observe the advantage is salient on larger datasets and more partitions.

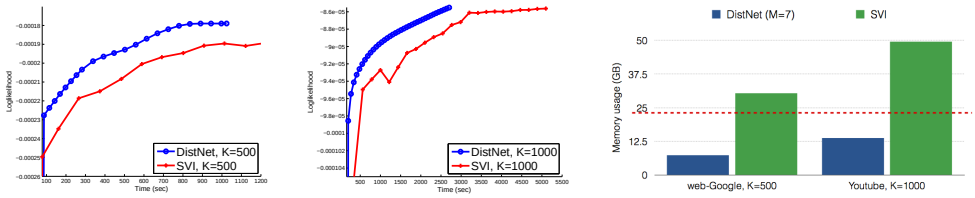

Figure 4: Convergence speed and memory footprint of distributed SVI and serial SVI on (a) web-Google and (b) Youtube dataset. We can observe faster convergence and lower memory consumption. Red dotted line in (c) indicates the typical RAM size in common computing clusters.

5.2 Distributed SVI

We perform distributed SVI using 7 processes and compare this with the original serial algorithm [7].

Figure 4 (a)&(b) shows the convergence speed in terms of log likelihood on web-Google and Youtube datasets. We can see that throughout the inference process distributed SVI exhibits faster convergence. This is expected since it can update more parameters than the serial version at a time. However also note that the speedup is not ideal. For instance in this experiment ideally distributed version should be nearly 7 times faster than the serial counterpart, however the actual speedup is around 3-5. Possible cause of the slow down include: 1) delayed synchronization of the parameters; or 2) conflict gradients computed in different machines.

Figure 4 (c) compares the memory footprint of two methods. It is obvious that the single machine SVI consumes a great amount of memory, even exceeding the typical RAM size of common computing clusters nowadays. By contrast, although there are some overheads in duplicating nodes, distributed SVI successfully reduces the memory consumption by efficient network partitioning. This suggests our ability to handle large scale real world networks.

6 Conclusion

In this project, we develop efficient distributed stochastic variational inference for MMSB, which enables large scale overlapping community detection. We tackle the problem by exploiting network structure and locality, specifically by employing parsimonious parameterization as well as efficient vertex-cut procedure. We implement a distributed asynchronous communication protocol for variable synchronization, and experimental results confirm the effectiveness our approach.

During the project we find that parallel update of the node parameters results in worse convergence, due to the conflicting information of the gradients computed in different machines. Recent work on variable scheduler [13] introduced a mechanism for selecting a set of non-conflict variables to update safely in parallel. This is an interesting future work that we expect can improve the convergence speed significantly.

References

[1] Amr Ahmed, Moahmed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J. Smola. Scalable Inference in Latent Variable Models. In *WSDM*, 2012.

[2] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed Membership Stochastic Blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, 2008.

[3] Brian Ball, Brian Karrer, and M. E. J. Newman. Efficient and principled method for detecting communities in networks. *Physical Review E*, 84(3):036103, 2011.

[4] Jianfei Chen, Jun Zhu, Zi Wang, Xun Zheng, and Bo Zhang. Scalable inference for logistic-normal topic models. In *NIPS*, 2013.

[5] Anna Goldenberg, Alice X. Zheng, Stephen E. Fienberg, and Edoardo M. Airoldi. A survey of statistical network models. *Foundations and Trends in Machine Learning*, 2(2):129–233, 2010.

[6] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Operating Systems Design and Implementation (OSDI)*, 2012.

[7] Prem K. Gopalan and David M. Blei. Efficient discovery of overlapping communities in massive networks. *Proceedings of the National Academy of Sciences*, 110(36):14534–14539, 2013.

[8] Prem K Gopalan, Sean Gerrish, Michael Freedman, David M. Blei, and David M. Mimno. Scalable inference of overlapping communities. In *NIPS*, 2012.

[9] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic Variational Inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[10] F. Huang, U. N. Niranjan, M. Umar Hakeem, and A. Anandkumar. Fast Detection of Overlapping Communities via Online Tensor Methods. *arXiv:1309.0787*, 2013.

[11] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[12] Can M. Le, Elizaveta Levina, and Roman Vershynin. Optimization via Low-rank Approximation, with Applications to Community Detection in Networks. *arXiv:1406.0067*, 2014.

[13] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A. Gibson, and Eric P. Xing. Primitives for Dynamic Big Model Parallelism. In *NIPS*, 2014.

[14] Mu Li, Dave Andersen, Alex Smola, Junwoo Park, Amr Ahmed, Vanja Josifovski, James Long, Eugene Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Operating Systems Design and Implementation (OSDI)*, 2014.

[15] Jo Agila Bitsch Link, Christoph Wollgarten, Stefan Schupp, and Klaus Wehrle. Perfect difference sets for neighbor discovery: energy efficient and fair. In *Proceedings of the 3rd Extreme Conference on Communication: The Amazon Expedition*, page 5. ACM, 2011.

[16] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(026113), 2004.

[17] M. E. J. Newman and E. A. Leicht. Mixture models and exploratory analysis in networks. *Proceedings of the National Academy of Sciences*, 104(23):9564–9569, 2007.

[18] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[19] Jaewon Yang and Jure Leskovec. Defining and Evaluating Network Communities Based on Ground-truth. In *ACM SIGKDD Workshop on Mining Data Semantics*, 2012.

[20] Jun Zhu, Xun Zheng, Li Zhou, and Bo Zhang. Scalable Inference in Max-margin Topic Models. In *Conference on Knowledge Discovery and Data Mining*, 2013.

A Variational Inference for link-based a-MMSB

A.1 Evidence lower bound (ELBO)

The evidence lower bound (ELBO) of link-based a-MMSB is

$$
\begin{aligned}
&\mathcal{L}(\lambda, \gamma, \phi, \overline{\phi}) \\
=&\mathbb{E}_q[\log p(\beta, \theta, Z, Y|\eta, \alpha)] - \mathbb{E}_q(\beta, \theta, Z|\lambda, \gamma, \phi, \overline{\phi}) \\
=&\sum_{k=1}^{K} \mathbb{E}_q[\log p(\beta_k|\eta)] - \sum_{k=1}^{K} \mathbb{E}_q[\log q(\beta_k|\lambda_k)] + \sum_{n=1}^{N} \mathbb{E}_q[\log p(\theta_n|\alpha)] - \sum_{n=1}^{N} \mathbb{E}_q[\log q(\theta_n|\gamma_n)] \\
&+ \sum_{a<b} \mathbb{E}_q[\log p(y_{ab}|z_{a\to b}, z_{a\leftarrow b}, \beta)] + \mathbb{E}_q[\log p(z_{a\to b}|\theta_a)] + \mathbb{E}_q[\log p(z_{a\leftarrow b}|\theta_b)] \\
&- \sum_{(a,b)\in \text{links}} \mathbb{E}_q[\log q(z_{a\to b}, z_{a\leftarrow b}|\phi_{ab})] \\
&- \sum_{(a,b)\in \text{nonlinks}} \{\mathbb{E}_q[\log q(z_{a\to b}|\phi_{a\to b})] + \mathbb{E}_q[\log q(z_{a\leftarrow b}|\phi_{a\leftarrow b})]\}
\end{aligned} \tag{12}
$$

where the factorized distributions are

$$q(\theta_n|\gamma_n) \sim \text{Dirichlet}(\gamma_n) \tag{13}$$

$$q(\beta_k|\lambda_k) \sim \text{Beta}(\lambda_k) \tag{14}$$

$$q(z_{a\to b}|\phi_{a\to b}) \sim \text{Multinomial}(\phi_{a\to b}) \tag{15}$$

$$q(z_{a\leftarrow b}|\phi_{a\leftarrow b}) \sim \text{Multinomial}(\phi_{a\leftarrow b}) \tag{16}$$

$$q(z_{a\to b}, z_{a\leftarrow b}|\phi_{ab}) \sim \text{Multivariate} - \text{Multinomial}(\phi_{ab}) \tag{17}$$

A.2 Coordinate ascent algorithm

The natural gradient for each variational parameter is as follows:

$$\partial\gamma_{a,k}^t = \alpha_k + \sum_{(a,b)\in \text{links(a)}} \phi_{ab}^{kk} + c_a\overline{\phi}_{a,k} - \gamma_{a,b}^{t-1} \tag{18}$$

$$
\begin{aligned}
\partial\lambda_{k,0}^t &= \eta_0 + \sum_{(a,b)\in \text{links}} \phi_{ab}^{kk} - \lambda_{k,0}^{t-1} \\
\partial\lambda_{k,1}^t &= \eta_0 + \sum_{(a,b)\in \text{nonlinks}} \overline{\phi}_{a,k}\overline{\phi}_{b,k} - \lambda_{k,1}^{t-1}
\end{aligned} \tag{19}
$$

And the update equation is

$$\gamma_{a,k}^t = \gamma_{a,k}^{t-1} + \rho_t \partial\gamma_{a,k}^{t-1} \tag{20}$$

and

$$
\begin{aligned}
\lambda_{k,0}^t &= \lambda_{k,0}^{t-1} + \rho_t \partial\lambda_{k,0}^t \\
\lambda_{k,1}^t &= \lambda_{k,1}^{t-1} + \rho_t \partial\lambda_{k,1}^t
\end{aligned} \tag{21}
$$

ρ_t is the learning rate: for stochastic variational inference, we can choose $\rho_t = (t+\tau)^{-\kappa}$; for batch method, we can just choose $\rho_t = 1$.
For $(a, b) \in$ links,

$$\phi_{ab,k}{}^t = \phi_{ab}^{kk\,t} \propto \exp\{\mathbb{E}_{q_t}[\log\theta_{ak}] + \mathbb{E}_{q_t}[\log\theta_{bk}] + \mathbb{E}_{q_t}[\log\beta_k]\} \tag{22}$$

Notice that here we ignore ϕ_{ab}^{kl} $(k \neq l)$ since we assume $\epsilon \to 0$. And for $(a, b) \in$ nonlinks,

$$
\begin{aligned}
\phi_{a\to b,k}^t &= \frac{\sum_{(a,b)\in\text{links}(a)} \sum_{l=1}^{K} \phi_{ab}^{kl\,t}}{d_a} = \frac{\sum_{(a,b)\in\text{links}(a)} \phi_{ab}^{kk\,t}}{d_a} = \overline{\phi}_{a,k}^t \\
\phi_{a\leftarrow b,k}^t &= \frac{\sum_{(a,b)\in\text{links}(b)} \sum_{l=1}^{K} \phi_{ab}^{lk\,t}}{d_b} = \frac{\sum_{(a,b)\in\text{links}(b)} \phi_{ab}^{kk\,t}}{d_b} = \overline{\phi}_{b,k}^t
\end{aligned} \tag{23}
$$