Overlapping Community Detection
with Graph Neural Networks

Oleksandr Shchur
Technical Univeristy of Munich
shchur@in.tum.de

Stephan Günnemann
Technical Univeristy of Munich
guennemann@in.tum.de

ABSTRACT

Community detection is a fundamental problem in machine learning. While deep learning has shown great promise in many graph-related tasks, developing neural models for community detection has received surprisingly little attention. The few existing approaches focus on detecting disjoint communities, even though communities in real graphs are well known to be overlapping. We address this shortcoming and propose a graph neural network (GNN) based model for overlapping community detection. Despite its simplicity, our model outperforms the existing baselines by a large margin in the task of community recovery. We establish through an extensive experimental evaluation that the proposed model is effective, scalable and robust to hyperparameter settings. We also perform an ablation study that confirms that GNN is the key ingredient to the power of the proposed model. A reference implementation as well as the new datasets are available under https://tinyurl.com/dlg-nocd.

1 INTRODUCTION

Graphs provide a natural way of representing complex real-world systems. Community detection methods are an essential tool for understanding the structure and behavior of these systems. Detecting communities allows us to analyze social networks [14], detect fraud [30], discover functional units of the brain [13], and predict functions of proteins [32]. The problem of community detection has attracted significant attention of the research community and numerous models and algorithms have been proposed [38].

In the recent years, the emerging field of deep learning for graphs has shown great promise in designing more accurate and more scalable algorithms. While deep learning approaches have achieved unprecedented results in graph-related tasks like link prediction and node classification [5], relatively little attention has been dedicated to their application for unsupervised community detection. Several methods have been proposed [7, 10, 43], but they all have a common drawback: they only focus on the special case of disjoint (non-overlapping) communities. However, it is well known that communities in real networks are overlapping [41]. Handling

overlapping communities is a requirement not yet met by existing deep learning approaches for community detection.

In this paper we address this research gap and propose an end-to-end deep learning model capable of detecting overlapping communities. To summarize, our main contributions are:

- Model: We introduce a graph neural network (GNN) based model for overlapping community detection.
- Data: We introduce 4 new datasets for overlapping community detection that can act as a benchmark and stimulate future research in this area.
- Experiments: We perform a thorough evaluation of our model and show its superior performance compared to established methods for overlapping community detection, both in terms of speed and accuracy. We highlight the importance of the GNN component of our model through an ablation study.

2 BACKGROUND

Assume that we are given an undirected unweighted graph G, represented as a binary adjacency matrix $A \in \{0, 1\}^{N \times N}$. We denote as N the number of nodes $V = \{1, ..., N\}$; and as M the number of edges $E = \{(u, v) \in V \times V : A_{uv} = 1\}$. Every node might be associated with a D-dimensional attribute vector, that can be represented as an attribute matrix $X \in \mathbb{R}^{N \times D}$. The goal of overlapping community detection is to assign nodes into C communities. Such assignment can be represented as a non-negative community affiliation matrix $F \in \mathbb{R}_{\geq 0}^{N \times C}$, where F_{uc} denotes the strength of node u's membership in community c (with the notable special case of binary assignment $F \in \{0, 1\}^{N \times C}$). Some nodes may be assigned to no communities, while others may belong to multiple.

Even though the notion of "community" seems rather intuitive, there is no universally agreed upon definition of it in the literature. However, most recent works tend to agree with the statement that a community is a group of nodes that have higher probability to form edges with each other than with other nodes in the graph [11]. This way, the problem of community detection can be considered in terms of the probabilistic inference framework. Once we posit a community-based generative model $p(G|F)$ for the graph, detecting communities boils down to inferring the unobserved affiliation matrix F given the observed graph G.

Besides the traditional probabilistic view, one can also view community detection through the lens of representation learning. The community affiliation matrix F can be considered as an embedding of nodes into $\mathbb{R}_{\geq 0}^{C}$, with the aim of preserving the graph structure. Given the recent success of representation learning for graphs [5], a question arises: "Can the advances in deep learning for graphs be used to design better community detection algorithms?". As we show in Section 4.1, simply combining existing node embedding approaches with overlapping K-means doesn't lead to satisfactory

results. Instead, we propose to combine the probabilistic and representation points of view, and learn the community affiliations in an end-to-end manner using a graph neural network.

3 THE NOCD MODEL

Here, we present the Neural Overlapping Community Detection (NOCD) model. The core idea of our approach is to combine the power of GNNs with the Bernoulli–Poisson probabilistic model.

3.1 Bernoulli–Poisson model

The Bernoulli–Poisson (BP) model [33, 40, 45] is a graph generative model that allows for overlapping communities. According to the BP model, the graph is generated as follows. Given the affiliations $F \in \mathbb{R}_{\geq 0}^{N \times C}$, adjacency matrix entries A_{uv} are sampled i.i.d. as

$$A_{uv} \sim \text{Bernoulli}(1 - \exp(-F_u F_v^T)) \quad (1)$$

where F_u is the row vector of community affiliations of node u (the u's row of the matrix F). Intuitively, the more communities nodes u and v have in common (i.e. the higher the dot product $F_u F_v^T$ is), the more likely they are to be connected by an edge.

This model has a number of desirable properties: It can produce various community topologies (e.g. nested, hierarchical), leads to dense overlaps between communities [41] and is computationally efficient (Section 3.3). Existing works propose to perform inference in the BP model using maximum likelihood estimation with coordinate ascent [40, 42] or Markov chain Monte Carlo [33, 45].

3.2 Model definition

Instead of treating the affiliation matrix F as a free variable over which optimization is performed, we generate F with a GNN:

$$F := \text{GNN}_\theta(A, X) \quad (2)$$

A ReLU nonlinearity is applied element-wise to the output layer to ensure non-negativity of F. See Section 4 and Appendix B for details about the GNN architecture.

The negative log-likelihood of the Bernoulli–Poisson model is

$$-\log p(A|F) = -\sum_{(u,v)\in E} \log(1 - \exp(-F_u F_v^T)) + \sum_{(u,v)\notin E} F_u F_v^T \quad (3)$$

Real-world graph are usually extremely sparse, which means that the second term in Equation 3 will provide a much larger contribution to the loss. We counteract this by balancing the two terms, which is a standard technique in imbalanced classification [18]

$$\mathcal{L}(F) = -\mathbb{E}_{(u,v)\sim P_E}\Big[\log(1-\exp(-F_u F_v^T))\Big] + \mathbb{E}_{(u,v)\sim P_N}\Big[F_u F_v^T\Big] \quad (4)$$

where P_E and P_N denote uniform distributions over edges and non-edges respectively.

Instead of directly optimizing the affiliation matrix F, as done by traditional approaches [40, 42], we search for neural network parameters $\theta^\star$ that minimize the (balanced) negative log-likelihood

$$\theta^\star = \underset{\theta}{\arg\min}\, \mathcal{L}(\text{GNN}_\theta(A, X)) \quad (5)$$

Using a GNN for community prediction has several advantages. First, due to an appropriate inductive bias, the GNN outputs similar community affiliation vectors for neighboring nodes, which

improves the quality of predictions compared to simpler models (Section 4.2). Also, such formulation allows us to seamlessly incorporate the node features into the model. If the node attributes X are not available, we can simply use A as node features [22]. Finally, with the formulation from Equation 2, it's even possible to predict communities inductively for nodes not seen at training time.

3.3 Scalability

One advantage of the BP model is that it allows to efficiently evaluate the loss $\mathcal{L}(F)$ and its gradients w.r.t. F. By using a caching trick [40], we can reduce the computational complexity of these operations from $O(N^2)$ to $O(N + M)$. While this already leads to large speed-ups due to sparsity of real-world networks (typically $M \ll N^2$), we can speed it up even further. Instead of using all entries of A when computing the loss (Equation 4), we sample a mini-batch of S edges and non-edges at each training epoch, thus approximately computing $\nabla\mathcal{L}$ in $O(S)$. In Appendix E we show that this stochastic optimization strategy converges to the same solution as the full-batch approach, while keeping the computational cost and memory footprint low.

While we subsample the graph to efficiently evaluate the training objective $\mathcal{L}(F)$, we use the full adjacency matrix inside the GNN. This doesn't limit the scalability of our model: NOCD is trained on a graph with 800K+ edges in 3 minutes on a single GPU (see Section 4.1). It is straightforward to make the GNN component even more scalable by applying the techniques such as [8, 44].

4 EVALUATION

Datasets. We use the following real-world graph datasets in our experiments. **Facebook** [27] is a collection of small (50-800 nodes) ego-networks from the Facebook graph. Larger graph datasets (10K+ nodes) with reliable ground-truth overlapping community information and node attributes are not openly available, which hampers the evaluation and development of new methods. For this reason we have collected and preprocessed 4 real-world datasets, that satisfy these criteria and can act as future benchmarks. **Chemistry**, **Computer Science**, **Medicine**, **Engineering** are co-authorship networks, constructed from the Microsoft Academic Graph [1]. Communities correspond to research areas in respective fields, and node attributes are based on keywords of the papers by each author. Statistics for all used datasets are provided in Appendix A.

Model architecture. For all experiments, we use a 2-layer Graph Convolutional Network (GCN) [22] as the basis for the NOCD model. The GCN is defined as

$$F := \text{GCN}_\theta(A, X) = \text{ReLU}(\hat{A}\,\text{ReLU}(\hat{A}X W^{(1)}) W^{(2)}) \quad (6)$$

where $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ is the normalized adjacency matrix, $\tilde{A} = A + I_N$ is the adjacency matrix with self loops, and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is the diagonal degree matrix of $\tilde{A}$. We considered other GNN architectures, as well as deeper models, but none of them led to any noticeable improvements. The two main differences of our model from standard GCN include (1) batch normalization after the first graph convolution layer and (2) L_2 regularization applied to all weight matrices. We found both of these modifications to lead to substantial gains in performance. We optimized the architecture and hyperparameters only using the Computer Science dataset —

Table 1: Recovery of ground-truth communities, measured by NMI (in %). Results for NOCD are averaged over 50 initializations (see Table 4 for error bars). Best result for each row in bold. DNF — did not finish in 12 hours or ran out of memory.

Dataset	BigCLAM	CESNA	EPM	SNetOC	CDE	SNMF	DW/NEO	G2G/NEO	NOCD-G	NOCD-X
Facebook 348	26.0	29.4	6.5	24.0	24.8	13.5	31.2	17.2	34.7	**36.4**
Facebook 414	48.3	50.3	17.5	52.0	28.7	32.5	40.9	32.3	56.3	**59.8**
Facebook 686	13.8	13.3	3.1	10.6	13.5	11.6	11.8	5.6	20.6	**21.0**
Facebook 698	45.6	39.4	9.2	44.9	31.6	28.0	40.1	2.6	**49.3**	41.7
Facebook 1684	32.7	28.0	6.8	26.1	28.8	13.0	**37.2**	9.9	34.7	26.1
Facebook 1912	21.4	21.2	9.8	21.4	15.5	23.4	20.8	16.0	**36.8**	35.6
Chemistry	0.0	23.3	DNF	DNF	DNF	2.6	1.7	22.8	22.6	**45.3**
Computer Science	0.0	33.8	DNF	DNF	DNF	9.4	3.2	31.2	34.2	**50.2**
Engineering	7.9	24.3	DNF	DNF	DNF	10.1	4.7	33.4	18.4	**39.1**
Medicine	0.0	14.4	DNF	DNF	DNF	4.9	5.5	28.8	27.4	**37.8**

no additional tuning was done for other datasets. More details about the model configuration and the training procedure are provided in Appendix B. We denote the model working on node attributes X as NOCD-X, and the model using the adjacency matrix A as input as NOCD-G. In both cases, the feature matrix is row-normalized.

Assigning nodes to communities. In order to compare the detected communities to the ground truth, we first need to convert the predicted continuous community affiliations F into binary community assignments. We assign node u to community c if its affiliation strength F_{uc} is above a fixed threshold ρ. We chose the threshold $\rho = 0.5$ like all other hyperparameters — by picking the value that achieved the best score on the Computer Science dataset, and then using it in further experiments without additional tuning.

Metrics. We found that popular metrics for quantifying agreement between true and detected communities, such as Jaccard and F_1 scores [26, 40, 42], can give arbitrarily high scores for completely uninformative community assignments. See Appendix F for an example and discussion. Instead we use overlapping normalized mutual information (NMI) [28], as it is more robust and meaningful.

4.1 Recovery of ground-truth communities

We evaluate the NOCD model by checking how well it recovers communities in graphs with known ground-truth communities.

Baselines. In our selection of baselines, we chose methods that are based on different paradigms for overlapping community detection: probabilistic inference, non-negative matrix factorization (NMF) and deep learning. Some methods incorporate the attributes, while other rely solely on the graph structure.

BigCLAM [40], EPM [45] and SNetOC [33] are based on the Bernoulli–Poisson model. BigCLAM learns F using coordinate ascent, while EPM and SNetOC perform inference with Markov chain Monte Carlo (MCMC). CESNA [42] is an extension of BigCLAM that additionally models node attributes. SNMF [36] and CDE [26] are NMF approaches for overlapping community detection.

We additionally implemented two methods based on neural graph embedding. First, we compute node embeddings for all the nodes in the given graph using two established approaches – DeepWalk [29] and Graph2Gauss [4]. Graph2Gauss takes into account both node features and the graph structure, while DeepWalk only uses the structure. Then, we cluster the nodes using Non-exhaustive Overlapping (NEO) K-Means [37] — which allows to assign them to overlapping communities. We denote the methods based on DeepWalk and Graph2Gauss as DW/NEO and G2G/NEO respectively.

To ensure a fair comparison, all methods were given the true number of communities C. Other hyperparameters were set to their recommended values. An overview of all baseline methods, as well as their configurations are provided in Appendix C.

Results: Recovery. Table 1 shows how well different methods recover the ground-truth communities. Either NOCD-X or NOCD-G achieve the highest score for 9 out of 10 datasets. We found that the NMI of both methods is strongly correlated with the reconstruction loss (Equation 4): NOCD-G outperforms NOCD-X in terms of NMI exactly in those cases, when NOCD-G achieves a lower reconstruction loss. This means that we can pick the better performing of two methods in a completely unsupervised fashion by only considering the loss values.

Results: Hyperparameter sensitivity. It's worth noting again that both NOCD models use the same hyperparameter configuration that was tuned only on the Computer Science dataset ($N = 22K, M = 96.8K, D = 7.8K$). Nevertheless, both models achieve excellent results on datasets with dramatically different characteristics (e.g. Facebook 414 with $N = 150, M = 1.7K, D = 16$).

Results: Scalability. In addition to displaying excellent recovery results, NOCD is highly scalable. NOCD is trained on the Medicine dataset (63K nodes, 810K edges) using a single GTX1080Ti GPU in 3 minutes, while only using 750MB of GPU RAM (out of 11GB available). See Appendix D for more details on hardware.

EPM, SNetOC and CDE don't scale to larger datasets, since they instantiate very large dense matrices during computations. SNMF and BigCLAM, while being the most scalable methods and having lower runtime than NOCD, achieved relatively low scores in recovery. Generating the embeddings with DeepWalk and Graph2Gauss can be done very efficiently. However, overlapping clustering of the embeddings with NEO-K-Means was the bottleneck, which led to runtimes exceeding several hours for the large datasets. As the authors of CESNA point out [42], the method scales to large graphs if the number of attributes D is low. However, as D increases, which is common for modern datasets, the method scales rather poorly. This is confirmed by our findings — on the Medicine dataset, CESNA (parallel version with 18 threads) took 2 hours to converge.

4.2 Do we really need a graph neural network?

Our GNN-based model achieved superior performance in community recovery. Intuitively, it makes sense to use a GNN for the reasons laid out in Section 3.2. Nevertheless, we should ask whether

Table 2: Comparison of the GNN-based model against simpler baselines. Multilayer perceptron (MLP) and Free Variable (FV) models are optimizing the same objective (Equation 4), but represent the community affiliations F differently.

| | Attributes | | Adjacency | | |
Dataset	GNN	MLP	GNN	MLP	Free variable
Facebook 348	**36.4 ± 2.0**	11.7 ± 2.7	34.7 ± 1.5	27.7 ± 1.6	25.7 ± 1.3
Facebook 414	**59.8 ± 1.8**	22.1 ± 3.1	56.3 ± 2.4	48.2 ± 1.7	49.2 ± 0.4
Facebook 686	**21.0 ± 0.9**	1.5 ± 0.7	20.6 ± 1.4	19.8 ± 1.1	13.5 ± 0.9
Facebook 698	41.7 ± 3.6	1.4 ± 1.3	**49.3 ± 3.4**	42.2 ± 2.7	41.5 ± 1.5
Facebook 1684	26.1 ± 1.3	17.1 ± 2.0	**34.7 ± 2.6**	31.9 ± 2.2	22.3 ± 1.4
Facebook 1912	35.6 ± 1.3	17.5 ± 1.9	**36.8 ± 1.6**	33.3 ± 1.4	18.3 ± 1.2
Chemistry	45.3 ± 2.3	**46.6 ± 2.9**	22.6 ± 3.0	12.1 ± 4.0	5.2 ± 2.3
Computer Science	**50.2 ± 2.0**	49.2 ± 2.0	34.2 ± 2.3	31.9 ± 3.8	15.1 ± 2.2
Engineering	39.1 ± 4.5	**44.5 ± 3.2**	18.4 ± 1.9	15.8 ± 2.1	7.6 ± 2.2
Medicine	**37.8 ± 2.8**	31.8 ± 2.1	27.4 ± 2.5	23.6 ± 2.1	9.4 ± 2.3

it's possible achieve comparable results with a simpler model. To answer this question, we consider the following two baselines.

Multilayer perceptron (MLP): Instead of a GCN (Equation 6), we use a simple fully-connected neural network to generate F.

$$F = \text{MLP}_{\theta}(X) = \text{ReLU}(\text{ReLU}(XW^{(1)})W^{(2)}) \qquad (7)$$

This is related to the model proposed by [19]. Same as for the GCN-based model, we optimize the weights of the MLP, $\theta = \{W^{(1)}, W^{(2)}\}$, to minimize the objective Equation 4.

$$\min_{\theta} \mathcal{L}(\text{MLP}_{\theta}(X)) \qquad (8)$$

Free variable (FV): As an even simpler baseline, we consider treating F as a free variable in optimization and solve

$$\min_{F \geq 0} \mathcal{L}(F) \qquad (9)$$

We optimize the objective using projected gradient descent with Adam [20], and update all the entries of F at each iteration. This can be seen as an improved version of the BigCLAM model. Original BigCLAM uses the imbalanced objective (Equation 3) and optimizes F using coordinate ascent with backtracking line search.

Setup. Both for the MLP and FV models, we tuned the hyperparameters on the Computer Science dataset (just as we did for the GNN model), and used the same configuration for all datasets. Details about the configuration for both models are provided in Appendix B. Like before, we consider the variants of the GNN-based and MLP-based models that use either X or A as input features. We compare the NMI scores obtained by the models on all 11 datasets.

Results. The results for all models are shown in Table 2. The two neural network based models consistently outperform the free variable model. When node attributes X are used, the MLP-based model outperforms the GNN version for Chemistry and Engineering datasets, where the node features alone provide a strong signal. However, MLP achieves extremely low scores for Facebook 686 and Facebook 698 datasets, where the attributes are not as reliable. On the other hand, when A is used as input, the GNN-based model always outperforms MLP. Combined, these findings confirm our hypothesis that a graph-based neural network architecture is indeed beneficial for the community detection task.

5 RELATED WORK

The problem of community detection in graphs is well-established in the research literature. However, most of the works study detection of non-overlapping communities [3, 35]. Algorithms for overlapping community detection can be broadly divided into methods based on non-negative matrix factorization [23, 26, 36], probabilistic inference [24, 33, 40, 45], and heuristics [12, 15, 25, 31].

Deep learning for graphs can be broadly divided into two categories: graph neural networks and node embeddings. GNNs [17, 22, 39] are specialized neural network architectures that can operate on graph-structured data. The goal of embedding approaches [4, 16, 21, 29] is to learn vector representations of nodes in a graph that can then be used for downstream tasks. While embedding approaches work well for detecting disjoint communities [7, 34], they are not well-suited for overlapping community detection, as we showed in our experiments. This is caused by lack of reliable and scalable approaches for overlapping clustering of vector data.

Several works have proposed deep learning methods for community detection. [43] and [6] use neural nets to factorize the modularity matrix, while [7] jointly learns embeddings for nodes and communities. However, neither of these methods can handle overlapping communities. Also related to our model is the approach by [19], where they use a deep belief network to learn community affiliations. However, their neural network architecture does not use the graph, which we have shown to be crucial in Section 4.2; and, just like EPM and SNetOC, relies on MCMC, which heavily limits the scalability of their approach. Lastly, [9] designed a GNN for *supervised* community detection, which is a very different setting.

6 DISCUSSION & FUTURE WORK

We proposed NOCD— a graph neural network model for overlapping community detection. The experimental evaluation confirms that the model is accurate, flexible and scalable.

Besides strong empirical results, our work opens interesting follow-up questions. We plan to investigate how the two versions of our model (NOCD-X and NOCD-G) can be used to quantify the relevance of attributes to the community structure. Moreover, we plan to assess the inductive performance of NOCD [17].

To summarize, the results obtained in this paper provide strong evidence that deep learning for graphs deserves more attention as a framework for overlapping community detection.

ACKNOWLEDGMENTS

REFERENCES

[1] [n. d.]. Microsoft Academic Graph. https://kddcup2016.azurewebsites.net/.
[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16).*
[3] Emmanuel Abbe. 2018. Community Detection and Stochastic Block Models: Recent Developments. *JMLR* 18 (2018).
[4] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *ICLR.*
[5] Hongyun Cai, Vincent W Zheng, and Kevin Chang. 2018. A comprehensive survey of graph embedding: problems, techniques and applications. *TKDD* (2018).
[6] Jinxin Cao, Di Jin, Liang Yang, and Jianwu Dang. 2018. Incorporating network structure with node contents for community detection on large networks using deep learning. *Neurocomputing* 297 (2018).
[7] Sandro Cavallari, Vincent W Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2017. Learning community embedding with community detection and node embedding on graphs. In *CIKM.*
[8] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction.. In *ICML.*
[9] Zhengdao Chen, Xiang Li, and Joan Bruna. 2019. Supervised Community Detection with Hierarchical Graph Neural Networks. In *ICLR.*
[10] Jun Jin Choong, Xin Liu, and Tsuyoshi Murata. 2018. Learning community structure with variational autoencoder. In *ICDM.*
[11] Santo Fortunato and Darko Hric. 2016. Community detection in networks: A user guide. *Physics Reports* 659 (2016).
[12] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. 2014. Overlapping community detection in labeled graphs. *Data Mining and Knowledge Discovery* 28 (2014).
[13] Javier O Garcia, Arian Ashourvan, Sarah Muldoon, Jean M Vettel, and Danielle S Bassett. 2018. Applications of community detection techniques to brain graphs: Algorithmic considerations and implications for neural function. *Proc. IEEE* 106 (2018).
[14] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *PNAS* 99 (2002).
[15] David F Gleich and C Seshadhri. 2012. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *KDD.*
[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD.*
[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS.*
[18] Haibo He and Edwardo A Garcia. 2008. Learning from imbalanced data. *TKDE* 9 (2008).
[19] Changwei Hu, Piyush Rai, and Lawrence Carin. 2017. Deep Generative Models for Relational Data with Side Information. *ICML.*
[20] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR* (2015).
[21] Thomas N Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *NIPS Workshop on Bayesian Deep Learning.*
[22] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *ICLR.*
[23] Da Kuang, Chris Ding, and Haesun Park. 2012. Symmetric nonnegative matrix factorization for graph clustering. In *SDM.*
[24] Pierre Latouche, Etienne Birmelé, Christophe Ambroise, et al. 2011. Overlapping stochastic block models with application to the French political blogosphere. *The Annals of Applied Statistics* 5 (2011).
[25] Yixuan Li, Kun He, David Bindel, and John E Hopcroft. 2015. Uncovering the small community structure in large networks: A local spectral approach. In *WWW.*
[26] Ye Li, Chaofeng Sha, Xin Huang, and Yanchun Zhang. 2018. Community Detection in Attributed Graphs: An Embedding Approach. In *AAAI.*
[27] Julian Mcauley and Jure Leskovec. 2014. Discovering social circles in ego networks. *TKDD* 8 (2014).
[28] Aaron F McDaid, Derek Greene, and Neil Hurley. 2011. Normalized mutual information to evaluate overlapping community finding algorithms. *arXiv:1110.2515* (2011).
[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD.*
[30] Carlos André Reis Pinheiro. 2012. Community detection to identify fraud events in telecommunications networks. *SAS SUGI proceedings: customer intelligence* (2012).
[31] Yiye Ruan, David Fuhry, and Srinivasan Parthasarathy. 2013. Efficient community detection in large networks using content and links. In *WWW.*
[32] Jimin Song and Mona Singh. 2009. How and when should interactome-derived clusters be used to predict functional modules and protein function? *Bioinformatics* 25 (2009).
[33] Adrien Todeschini, Xenia Miscouridou, and François Caron. 2016. Exchangeable random measures for sparse and modular graphs with overlapping communities. *arXiv:1602.02114* (2016).
[34] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *WWW.*
[35] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17 (2007).
[36] Fei Wang, Tao Li, Xin Wang, Shenghuo Zhu, and Chris Ding. 2011. Community discovery using nonnegative matrix factorization. *Data Mining and Knowledge Discovery* 22 (2011).
[37] Joyce Jiyoung Whang, Inderjit S Dhillon, and David F Gleich. 2015. Non-exhaustive, Overlapping k-means. In *SDM.*
[38] Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. 2013. Overlapping community detection in networks: The state-of-the-art and comparative study. *CSUR* 45 (2013).
[39] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. *ICML* (2018).
[40] Jaewon Yang and Jure Leskovec. 2013. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM.*
[41] Jaewon Yang and Jure Leskovec. 2014. Structure and Overlaps of Ground-Truth Communities in Networks. *ACM TIST* 5 (2014).
[42] Jaewon Yang, Julian McAuley, and Jure Leskovec. 2013. Community detection in networks with node attributes. In *ICDM.*
[43] Liang Yang, Xiaochun Cao, Dongxiao He, Chuan Wang, Xiao Wang, and Weixiong Zhang. 2016. Modularity Based Community Detection with Deep Learning.. In *IJCAI.*
[44] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems.
[45] Mingyuan Zhou. 2015. Infinite edge partition models for overlapping community detection and link prediction. In *AISTATS.*

A   DATASETS

Table 3: Dataset statistics. K stands for 1000.

Dataset	Network type	N	M	D	C
Facebook 348	Social	224	3.2K	21	14
Facebook 414	Social	150	1.7K	16	7
Facebook 686	Social	168	1.6K	9	14
Facebook 698	Social	61	270	6	13
Facebook 1684	Social	786	14.0K	15	17
Facebook 1912	Social	747	30.0K	29	46
Computer Science	Co-authorship	22.0K	96.8K	7.8K	18
Chemistry	Co-authorship	35.4K	157.4K	4.9K	14
Medicine	Co-authorship	63.3K	810.3K	5.5K	17
Engineering	Co-authorship	14.9K	49.3K	4.8K	16

B   MODEL CONFIGURATION

B.1   Architecture

We picked the hyperparameters and chose the model architecture for all 3 models by only considering their performance (NMI) on the Computer Science dataset. No additional tuning for other datasets has been done.

GNN-based model. (Equation 6) We use a 2-layer graph convolutional neural network, with hidden size of 128, and the output (second) layer of size C (number of communities to detect). We apply batch normalization after the first graph convolution layer. Dropout with 50% keep probability is applied before every layer. We add weight decay to both weight matrices with regularization strength $\lambda = 10^{-2}$. The feature matrix X (or A, in case we are working without attributes) is normalized such that every row has unit L_2-norm.

We also experimented with the Jumping Knowledge Network [39] and GraphSAGE [17] architectures, but they led to lower NMI scores on the Computer Science dataset.

MLP-based model. (Equation 7) We found the MLP model to perform best with the same configuration as described above for the GCN model (i.e. same regularization strength, hidden size, dropout and batch norm).

Free variable model (Equation 9) We considered two initialization strategies for the free variable model: (1) Locally minimal neighborhoods [15] — the strategy used by the BigCLAM and CESNA models and (2) initializing F to the output of an untrained GCN. We found strategy (1) to consistently provide better results.

B.2   Training

GNN- and MLP-based models. We train both models using Adam optimizer [20] with default parameters. The learning rate is set to 10^{-3}. We use the following early stopping strategy: Every 50 epochs we compute the full training loss (Equation 4). We stop optimization if there was no improvement in the loss for the last $10 \times 50 = 500$ iterations, or after 5000 epochs, whichever happens first.

Free variable model. We use Adam optimizer with learning rate $5 \cdot 10^{-2}$. After every gradient step, we project the F matrix to

ensure that it stays non-negative: $F_{uc} = \max\{0, F_{uc}\}$. We use the same early stopping strategy as for the GNN and MLP models.

C   BASELINES

Table 4: Overview of the baselines. See text for the discussion of scalability of CESNA.

Method	Model type	Attributed	Scalable
BigCLAM [40]	Probabilistic		✓
CESNA [42]	Probabilistic	✓	✓*
SNetOC [33]	Probabilistic		
EPM [45]	Probabilistic		
CDE [26]	NMF	✓	
SNMF [36]	NMF		✓
DW/NEO [29, 37]	Deep learning		
G2G/NEO [4, 37]	Deep learning	✓	
NOCD	Deep learning + probabilistic	✓	✓

- We used the reference C++ implementations of BigCLAM and CESNA, that were provided by the authors (https://github.com/snap-stanford/snap). Models were used with the default parameter settings for step size, backtracking line search constants, and balancing terms. Since CESNA can only handle binary attributes, we binarize the original attributes (set the nonzero entries to 1) if they have a different type.
- We implemented SNMF ourselves using Python. The F matrix is initialized by sampling from the Uniform[0, 1] distribution. We run optimization until the improvement in the reconstruction loss goes below 10^{-4} per iteration, or for 300 epochs, whichever happens first. The results for SNMF are averaged over 50 random initializations.
- We use the Matlab implementation of CDE provided by the authors. We set the hyperparameters to $\alpha = 1$, $\beta = 2$, $\kappa = 5$, as recommended in the paper, and run optimization for 20 iterations.
- For SNetOC and EPM we use the Matlab implementations provided by the authors with the default hyperparameter settings. The implementation of EPM provides to options: EPM and HEPM. We found EPM to produce better NMI scores, so we used it for all the experiments.
- We use the TensorFlow implementation of Graph2Gauss provided by the authors. We set the dimension of the embeddings to 128, and only use the μ matrix as embeddings.
- We implemented DeepWalk ourselves: We sample 10 random walks of length 80 from each node, and use the Word2Vec implementation from Gensim (https://radimrehurek.com/gensim/) to generate the embeddings.The dimension of embeddings is set to 128.
- For NEO-K-Means, we use the Matlab code provided by the authors. We let the parameters α and β be selected automatically using the built-in procedure.

D HARDWARE AND SOFTWARE

The experiments were performed on a computer running Ubuntu 16.04LTS with 2x Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz CPUs, 256GB of RAM and 4x GTX1080Ti GPUs. Note that training and inference were done using only a single GPU at a time for all models. The NOCD model was implemented using Tensorflow v1.1.2 [2]

E CONVERGENCE OF THE STOCHASTIC SAMPLING PROCEDURE

Instead of using all pairs $u, v \in V$ when computing the gradients $\nabla_\theta \mathcal{L}$ at every iteration, we sample S edges and S non-edges uniformly at random. We perform the following experiment to ensure that our training procedure converges to the same result, as when using the full objective.

Experimental setup. We train the model on the Computer Science dataset and compare the full-batch optimization procedure with stochastic gradient descent for different choices of the batch size S. Starting from the same initialization, we measure the full loss (Equation 4) over the iterations.

Results. Figure 1 shows training curves for different batch sizes $S \in \{1000, 2500, 5000, 10000, 20000\}$, as well as for full-batch training. The horizontal axis of the plot displays the number of entries of adjacency matrix accessed. One iteration of stochastic training accesses $2S$ entries A_{ij}, and one iteration of full-batch accesses $2N + 2M$ entries, since we are using the caching trick from [40]. As we see, the stochastic training procedure is stable: For batch sizes $S = 10K$ and $S = 20K$, the loss converges very closely to the value achieved by full-batch training.
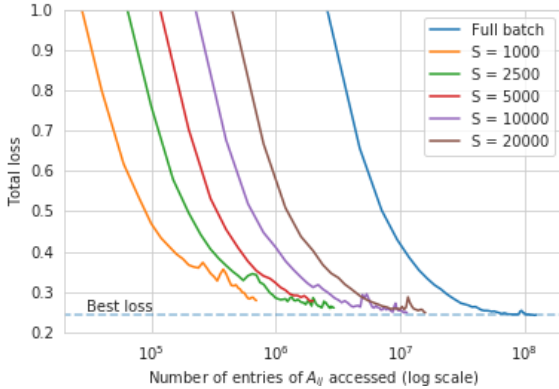
defined as

$$\frac{1}{2|\mathcal{S}^*|} \sum_{S_i^* \in \mathcal{S}^*} \max_{S_j \in \mathcal{S}} \delta(S_i^*, S_j) + \frac{1}{2|\mathcal{S}|} \sum_{S_j \in \mathcal{S}} \max_{S_i^* \in \mathcal{S}^*} \delta(S_i^*, S_j) \quad (10)$$

where $\delta(S_i^*, S_j)$ is a similarity measure between sets, such as F_1-score or Jaccard similarity.

We discovered that these frequently used measures can assign arbitrarily high scores to completely uninformative community assignments, as you can in see in the following simple example. Let the ground truth communities be $S_1^* = \{v_1, ..., v_K\}$ and $S_2^* = \{v_{N-K}, ..., v_N\}$ (K nodes in each community), and let the algorithm assign all the nodes to a single community $S_1 = V = \{v_1, ..., v_N\}$. While this predicted community assignment is completely uninformative, it will achieve symmetric F_1-score of $\frac{2K}{N+K}$ and symmetric Jaccard similarity of $\frac{K}{N}$ (e.g., if $K = 600$ and $N = 1000$, the scores will be 75% and 60% respectively). These high numbers might give a false impression that the algorithm has learned something useful, while that clearly isn't the case. As an alternative, we suggest using overlapping normalized mutual information (NMI), as defined in [28]. NMI correctly handles the degenerate cases, like the one above, and assigns them the score of 0.

Figure 1: Convergence of the stochastic sampling procedure.

F QUANTIFYING AGREEMENT BETWEEN OVERLAPPING COMMUNITIES

A popular choice for quantifying the agreement between true and predicted overlapping communities is the symmetric agreement score [26, 40, 42]. Given the ground-truth communities $\mathcal{S}^* = \{S_i^*\}_i$ and the predicted communities $\mathcal{S} = \{S_j\}_j$, the symmetric score is