Federated Graph Learning with Periodic Neighbour Sampling

Bingqian Du
Department of Computer Science
The University of Hong Kong
bqdu@connect.hku.hk

Chuan Wu
Department of Computer Science
The University of Hong Kong
cwu@cs.hku.hk

***Abstract*—Graph Convolutional Networks (GCN) proposed recently have achieved promising results on various graph learning tasks. Federated learning (FL) for GCN training is needed when learning from geo-distributed graph datasets. Existing FL paradigms are inefficient for geo-distributed GCN training since neighbour sampling across geo-locations will soon dominate the whole training process and consume large WAN bandwidth. We derive a practical federated graph learning algorithm, carefully striking the trade-off among GCN convergence error, wall-clock runtime, and neighbour sampling interval. Our analysis is divided into two cases according to the budget for neighbour sampling. In the unconstrained case, we obtain the optimal neighbour sampling interval, that achieves the best trade-off between convergence and runtime; in the constrained case, we show that determining the optimal sampling interval is actually an online problem and we propose a novel online algorithm with bounded competitive ratio to solve it. Combining the two cases, we propose a unified algorithm to decide the neighbour sampling interval in federated graph learning, and demonstrate its effectiveness with extensive simulation over graph datasets from real applications.**

***Index Terms*—Graph Neural Network, Federated Learning**

I. Introduction

Many data generated in various online services can be naturally expressed as graphs, such as social networks [1], knowledge graphs [2], networks of web pages [3], etc. Graph Convolutional Network (GCN) [4] has been proposed as a learning model to exploit structure information and node features of graphs for various tasks such as node classification [5], link prediction [6] and graph representation learning [7], achieving the state-of-the-art performance. Similar to the need for federated learning over traditional datasets (e.g., images) across multiple edge devices [8] [9], graph datasets may well be collected on multiple geo-distributed sites (or devices), while a global model is expected to be learned. For example, social networks stored at geo-distributed data centers can all contribute to a joint learning task.

In classical federated learning [10], for tasks such as image classification or next word prediction for keyboard typing, each device trains a local copy of the DNN model using the local dataset, and updates model parameters by periodically exchanging gradients with other devices. Only gradient communication is involved, but not training data exchange. In federated learning of a GCN on a geo-distributed graph, differently, training data communication across devices[1] is often unavoidable: there may well be edges between nodes stored on different devices, and embedding calculation of one node usually requires information of its recursive neighbours from several hops away, which may well be stored on other devices. Training by treating sub-graphs at different devices as independent would lead to bias and unacceptable performance degradation of the global model learned.

With federated learning, we can achieve similar benefits as classical federated learning [12] for graph datasets, e.g., in terms of data privacy preservation (with less data sharing across sites) and much reduced network bandwidth costs (by avoiding transmitting large volumes of training samples). Graph datasets produced in different geo-sites in practice can be very large and dynamic. Collecting the whole dataset centrally and constantly updating it would be too bandwidth-consuming and more easily privacy infringing than keeping the distribution of sub-graphs and just sampling neighbour features/embeddings across devices when needed.

The numbers of nodes and edges in graphs from real applications are on the order of millions, e.g., the citation network ogbn-paper100M contains more than 100 million nodes and edges [13]. The distribution of nodes/edges across devices can be quite arbitrary. Even with a small batch size for graph learning and sampling only part of the neighbourhood for output node embedding computation, the key overhead in federated graph training is still due to fetching neighbour information from other devices. Neighbour sampling incurs large network bandwidth, and leads to long wall-clock runtime for model training convergence.

Fig. 1 gives the time consumed for training a three-layer GCN over ogbn-products dataset (with average node degree 50) randomly distributed over 16 devices. The numbers of sampled nodes for 1-hop, 2-hop and 3-hop neighbours are 15, 10 and 5 respectively for each output node. Neighbour sampling is initiated locally by each device. When sampled neighbour nodes are not at the current device, the current device would send neighbour sampling requests and other necessary information to devices with these sampled nodes.

[1]Privacy-preserving methods can be applied to training samples before sample communication across devices [11].

978-1-6654-6824-4/22/$31.00

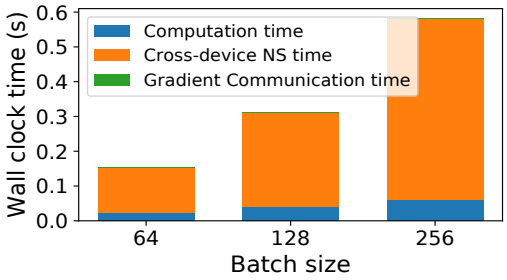

Fig. 1: Wall-clock time for computation, cross-device neighbour sampling and gradient communication

These devices would continue neighbour sampling and transmit needed information back to the requesting device. The computation time is derived by averaging model training time per training iteration on the devices; gradient communication time and cross-device neighbour sampling (NS) time are computed by averaging gradient transmission time per model aggregation and neighbour information transmission time per neighbour sampling over multiple training steps, respectively. Under different batch sizes, the neighbour sampling time always remains 6 to 8 times larger than training time, and the gradient communication time is negligible since the size of the GCN model is only about 1-2MB [14]. Therefore, it is critical to reduce the overhead caused by cross-device neighbour sampling in federated graph learning.

In this paper, we propose efficient federated graph learning algorithms by carefully analyzing the trade-off between convergence error and sampling interval. Specifically:

▷ We analyze the convergence of federated GCN training with distributed neighbour sampling, and formulate the connection between convergence error, wall-clock runtime and neighbour sampling interval. The biased gradients and the variance of node embedding caused by neighbour sampling make the convergence analysis of GCN training more complex than other neural networks with unbiased gradients. Taking cross-device neighbour sampling time into account, we derive a close-form expression of optimal sampling interval when there is no explicit budget for neighbour sampling, which achieves the best trade-off between convergence error and wall-clock time of neighbour sampling.

▷ We further extend our analysis to the constrained case, where an explicit budget/upper bound for total numbers of neighbour samplings that could be conducted is given. We show that derivation of optimal sampling interval in this case is actually an online problem. We propose a novel light-weight online algorithm achieving a bounded competitive ratio, based on the interpretation of optimal Lagrangian multiplier derived from KKT conditions, which measures the change of optimal objective given relaxed constraint. We also provide a numerical analysis of our competitive ratio, to facilitate better understanding of performance of the proposed online algorithm.

▷ Based on these analysis, we summarize a unified algorithm for deciding neighbour sampling interval in federated graph learning. Experiments on training federated GCN over real-world graphs with millions of nodes and edges for different tasks demonstrate the effectiveness of our algorithm. The adaptive sampling interval returned by our algorithm achieves a good trade-off between convergence and runtime, test accuracy and the ratio of sampling time over computation time.

II. Preliminaries

Graph Convolutional Networks are learning models designed for structured graph data, which generate embeddings for each node by recursively aggregating neighbour information. We introduce basics of GCN based on the example of a semi-supervised node classification problem. Given an undirected graph $\mathcal{G} = (V, E)$, where V is the node set and E is the edge set. Each node $v \in V$ has a feature vector x_v, and each node $v \in V_L$ has a corresponding label y_v, where $V_L \in V$ is the set of labelled nodes. Let X denote the feature matrix of all nodes. Matrix A and matrix D represent the adjacency matrix and degree matrix of $\mathcal{G}$, respectively. The embedding $H^{(l+1)}$ of layer $l+1$ produced by the GCN is in (1). Here $P = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is the propagation matrix, with $\tilde{A} = A + I$, $\tilde{D}_{vv} = \sum_u \tilde{A}_{vu}$, $\forall v \in V, \forall u \in V$. $H^{(0)} = X$, $w^{(l+1)}$ is the trainable parameter of layer $l+1$, and $\phi(\cdot)$ is the activation function to introduce non-linearity:

$$Z^{(l+1)} = PH^{(l)}w^{(l+1)}, H^{(l+1)} = \phi(Z^{(l+1)}) \tag{1}$$

Given the embedding computation (1) in GCN and the one-hop neighbour set $n(v)$ of node v, we have $z_v^{(l+1)} = (PH^{(l)})_v w^{(l+1)} = (\sum_u P_{vu}H_u^{(l)})w^{(l+1)} = (\sum_{u\in n(v)} P_{vu}H_u^{(l)})w^{(l+1)}$. That is, one GCN layer computes the embedding of one node by aggregating its one-hop neighbour embeddings from the previous layer.

The full-batch and mini-batch based training loss of an L-layer GCN is defined as follows, respectively, where B is a mini-batch of training samples uniformly sampled from V_L, $z_v^{(L)}$ is the embedding of node v from the L-layer GCN, which requires the information of L-hop neighbours of node v as in (1), and $f(\cdot)$ can be any feasible loss function (e.g., cross entropy [15], l_2-norm [16]):

$$\mathcal{L} = \frac{1}{|V_L|}\sum_{v\in V_L} f(y_v, z_v^{(L)}), \qquad \mathcal{L}_B = \frac{1}{|B|}\sum_{v\in B} f(y_v, z_v^{(L)}) \tag{2}$$

Even though mini-batch based training could reduce computation, $\mathcal{L}_B$ is still expensive to compute. Suppose the average degree in a graph is d. In order to compute the embedding for one node in an L-layer GCN, on average the number of neighbours involved would be d^L [17]. The exponentially growing number of sampled neighbours with L leads to substantial computation and memory resource demand for GCN training.

The exponential neighborhood expansion problem in GCN training has been mitigated by **neighbour sampling strategies**: a small number of neighbours are sampled from the complete neighbour set at each training step according to some probability distributions over neighbour set [5] [18]

[19], instead of using the entire neighbor set. The propagation matrix P_s, instead of P, is constructed according to sampled neighbours at each layer. A neighbour sampling strategy can commonly ensure unbiased embedding with $\mathbb{E}[P_s H^{(l)}] = PH^{(l)}$. However, due to non-linearity of activation function $\phi(\cdot)$, the unbiased gradient estimation $\mathbb{E}[\nabla \mathcal{L}_B] = \nabla \mathcal{L}$ does not hold for neighbour sampling-based GCN training.

III. Neighbour Sampling for Federated Graph Learning

In this section, we present our methods for reducing cross-device neighbour sampling overhead in federated graph learning. We assume each node has a unique ID and a feature vector. Also, each node has a list of node IDs (whose feature vectors may well reside in other devices), with whom it connects. This is consistent with real-world graph data distribution. Take a social network graph as an example: each user only maintains his/her friend list locally while the details/activities of his/her friends may well reside with other agents. When conducting neighbour sampling, each device first locally samples a mini-batch of output nodes whose embeddings would be computed as the output of the neural network. Features/embeddings from L-hop neighbours of output nodes are needed according to (1) for a L-layer GCN. Neighbor sampling is then conducted locally according to the preset sampling strategy. Once features of sampled neighbour nodes are not residing with the same device as the output node, the device would send a request (via a central server) to the corresponding devices to continue neighbour sampling, and cross-device feature/embedding transmission would occur afterward (privacy-preserving method can be applied [11]).

We take Fig. 2 as an example for better illustration of neighbour sampling in a federated learning scenario. The blue dotted lines indicate different devices. Suppose we use a three-layer GCN to compute the embedding of node A, which is denoted as $z_A^{(3)}$ according to (1). Given sampled 1-hop neighbour node B, the sampling procedure would continue sampling the neighbours of node B to compute $H_B^{(2)}$. If node C and node D are sampled afterwards, which are not residing on the same device as node A, the device of node A would send a request to device of node D/C, which would continue sampling 1-hop neighbours of nodes D and C according to the sampling strategy to compute the embeddings $H_D^{(1)}$ and $H_C^{(1)}$ for D and C based on (1). Then privacy-preserved embeddings $H_D^{(1)}$ and $H_C^{(1)}$ would be transmitted to device of node A to finish the computation of $H_B^{(2)}$ and $z_A^{(3)}$.

Our method is to advocate sparse cross-device neighbor sampling: each device conducts neighbour sampling from other devices once per τ training iterations (referred to as the *cross-device neighbour sampling interval*), and re-uses most recent cross-device samples in other iterations. Let $\mathcal{I}_\tau$ denote the set of training iterations when cross-device neighbour sampling is conducted, i.e., $\mathcal{I}_\tau = \{t | t \mod \tau = 0\}$. In an iteration $t \in \mathcal{I}_\tau$, sampled device participating training carries out neighbour sampling process described above and communicates with other devices for transmitting features and embeddings of sampled nodes; in all other iterations, each device re-uses information of most recently sampled nodes for last training step to conduct updates in the current iteration.

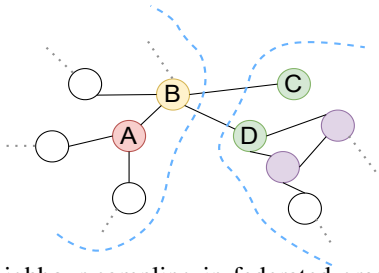

Fig. 2: Neighbour sampling in federated graph learning

We investigate the best τ that balances the trade-off between training convergence and actual runtime. We divide our results in two cases, without and with a limitation on the number of neighbour samplings allowed during GCN training. The scarce WAN bandwidth and privacy consideration make it desirable to consider the case with limited cross-device neighbour sampling budget.

A. Case without cross-device neighbour sampling constraint

We begin with the unconstrained case. To find an appropriate value of τ, we follow similar idea from [20] to establish the trade-off between convergence error and cross-device neighbour sampling interval τ. The work of [20] focuses on distributed DNN training with an unbiased gradient estimator and derives the optimal parameter aggregation interval according to the trade-off between convergence error and wall-clock time of parameter aggregation. As Fig. 1 shows, the key overhead in federated GCN training is due to neighbour sampling, instead of parameter communication. The main challenges in federated graph learning lie in biased gradients in sampling-based GCN training and the need to incorporate neighbour sampling time.

To analyze neighbour sampling-based training, taking mini-batch based loss for example, we rewrite the loss function in (2) to include propagation matrix explicitly:

$$F(w, \boldsymbol{P}(\boldsymbol{B})X) = \frac{1}{|B|} \sum_{v \in B} f_v(w, [P^{(L-1)}, P^{(L-2)}, \cdots, P^{(0)}, X]) \tag{3}$$

where w represents the set of parameters in GCN and B is the mini-batch of the output nodes, whose embeddings based on L-hop neighbours would be computed. $\boldsymbol{P}(\boldsymbol{B})X = [P^{(L-1)}, P^{(L-2)}, \cdots, P^{(0)}, X]$ is the concatenation of propagation matrix of different layers and the node feature matrix, which serves as the input to $f(\cdot)$ and is related to the output node set B and neighbour sampling strategy. Let n and $P_{(i,j)}$ denote the total number of nodes across devices and the entry (i, j) of matrix P, respectively. $P^{(l)} \in \mathbb{R}^{n \times n}$ is the propagation matrix for embedding computation of layer $l+1$ with $P^{(l)} = P$ in equation (1) when there is no neighbour sampling. We use $P_s^{(l)}$, instead of $P^{(l)}$, to better denote neighbor sampling-based propagation matrix when neighbour sampling is conducted. Normally $P_s^{(l)}$ would be re-scaled $P^{(l)}$

by multiplying $P_{(i,j)}$ by a coefficient related to the sampling distribution and the number of samples.

N is the number of geo-distributed devices. Based on the above formulation, the objective of federated graph leaning can be written as

$$\min_{w} F(w, \boldsymbol{P(V)}X) = \sum_{k=1}^{N} p_k F_k(w, \boldsymbol{P(V_k)}X_k) \tag{4}$$

where V_k is the node set stored on device k. $p_k = \frac{n_k}{n}, \forall k$ and $n_k = |V_k|$. We use $F(w)$ as a shorthand for $F(w, \boldsymbol{P(V)}X)$. We adopt stochastic gradient descent and sample mini-batch B_t^k from V_k uniformly randomly to compute the gradient of parameters w in iteration t, at each device k. Suppose all devices participate in each training iteration and synchronize their parameters once every E iterations, and $\mathcal{I}_E = \{t | t \mod E = 0\}$ is the set of iterations for parameter communication. We first state our assumptions for convergence analysis.

Assumption 1. *Each function $F_k(w, \boldsymbol{P(\cdot)}X), \forall k \in [N]$,[2] has an L-Lipschitz gradient with respect to w, and $\nabla_w F_k(w, \boldsymbol{P(\cdot)}X)$ is L_e-Lipschitz with respect to $\boldsymbol{P(\cdot)}X$.*

Assumption 2. *If* **no sampling** *is performed and all neighbors of a node are used for embedding generation, the gradient computed using the mini-batches w.r.t w is an unbiased estimator of the true gradient and has a bounded variance at each device k. That is, $\mathbb{E}[\nabla F_k(w, \boldsymbol{P(B_k)}X)] = \nabla F_k(w, \boldsymbol{P(V_k)}X)$ and $\mathbb{E}[||\nabla F_k(w, \boldsymbol{P(B_k)}X) - \nabla F_k(w, \boldsymbol{P(V_k)}X)||^2] \leq \sigma^2, \forall k$.*

The first two assumptions are commonly used in non-convex optimization for convergence analysis.

Assumption 3. *The parameter synchronization interval satisfies $E = K\tau$.*

For a fixed τ, we can always find a K making this assumption hold.

Assumption 4. *The expected gradient norm of full neighbour set training across all devices is upper bounded by the norm of expected gradient of neighbour sampling-based training: $\sum_{k=1}^{N} p_k \mathbb{E}[||\nabla F_k(w_t^k, \boldsymbol{P(B_t^k)}X)||^2] \leq \beta \mathbb{E}[||\sum_{k=1}^{N} p_k \nabla F_k(w_t^k, \boldsymbol{P_s(B_t^k)}X)||^2]$.*

This assumption can be easily satisfied with a large enough β. Parameter update at device k in neighbour sampling-based GCN training goes as follows:

$$w_{t+1}^k = \begin{cases} w_t^k - \eta \nabla F_k(w_t^k, \boldsymbol{P_s(B_t^k)}X), & t+1 \notin \mathcal{I}_E \\ \sum_{k=1}^{N} p_k (w_t^k - \eta \nabla F_k(w_t^k, \boldsymbol{P_s(B_t^k)}X)), & t+1 \in \mathcal{I}_E \end{cases} \tag{5}$$

To facilitate convergence analysis, we introduce an intermediate variable $\bar{w}_t = \sum_{k=1}^{N} p_k w_t^k$. We have $\bar{w}_{t+1} = \bar{w}_t - \eta \sum_{k=1}^{N} p_k \nabla F_k(w_t^k, \boldsymbol{P_s(B_t^k)}X), \forall t$. We follow the convention in non-convex optimization and use expected gradient norm as the convergence metric [21].

Theorem 1. *Let $\tilde{g}_t$ denote $\sum_{k=1}^{N} p_k \nabla F_k(w_t^k, \boldsymbol{P_s(B_t^k)}X)$ and $C = \sum_{k=1}^{N} p_k \mathbb{E}[||\boldsymbol{P_s(B_k)} - \boldsymbol{P(B_k)}||^2]$. Suppose the total training iteration number is T and the optimal value of $F(w)$ is F^*. When learning rate $\eta \leq \min\{\frac{1}{2L}, \tilde{\eta}\}$, where $\tilde{\eta}$ is the solution satisfying $6L^3\eta^3 K\tau\beta + 3L^2\eta^2 K\tau\beta - \frac{1}{2} \leq 0$ and Assumptions 1 to 4 hold, we have the following bound on convergence error*

$$\begin{aligned} \frac{\sum_{t=1}^{T} \mathbb{E}[||\nabla F(\bar{w}_t)||^2]}{T} &\leq \frac{F(\bar{w}_1) - F^*}{T(\frac{\eta}{2} - L\eta^2)} \\ &+ 3\frac{(L\eta^2 + \frac{\eta}{2})}{(\frac{\eta}{2} - L\eta^2)}[\sigma^2 + (L_e^2 + 8K^2\tau^2\eta^2 L^2 L_e^2)C] \end{aligned} \tag{6}$$

All missing proofs can be found in the Appendix.

Given a fixed η, the last term in (6) would not be 0 even when training iteration T is large enough. This matches previous observation and analysis that fixed learning rate would lead to a higher error floor [22]. A similar result also holds for mini-batch variance σ^2. Increasing the batch size to make σ^2 smaller in the training process would also lead to a small error floor [23]. From (6), we can also observe that, GCN training with neighbour sampling can also reach a lower error floor if we increase the sampled neighbour size during the training process to make the value of C smaller.

We consider wall-clock runtime instead of training iteration number, since cross-device neighbour sampling mainly causes more bandwidth consumption and larger transmission time. Suppose the average per-minibatch processing time among all devices is $\bar{C}$, average neighbour sampling time is S and parameter synchronization time is P.[3] The average runtime for a training iteration is $\bar{C} + \frac{S}{\tau} + \frac{P}{K\tau}$ and the total runtime for T iterations is $C_T = T(\bar{C} + \frac{S}{\tau} + \frac{P}{K\tau})$.

Replacing T in (6) by $T = \frac{C_T}{(\bar{C} + \frac{S}{\tau} + \frac{P}{K\tau})}$, we can get the convergence error bound in terms of the runtime:

$$\begin{aligned} \frac{\sum_{t=1}^{T} \mathbb{E}[||\nabla F(\bar{w}_t)||^2]}{T} &\leq \frac{F(\bar{w}_1) - F^*}{C_T(\frac{\eta}{2} - L\eta^2)}(\bar{C} + \frac{S}{\tau} + \frac{P}{K\tau}) \\ &+ 3\frac{(L\eta^2 + \frac{\eta}{2})}{(\frac{\eta}{2} - L\eta^2)}[\sigma^2 + (L_e^2 + 8K^2\tau^2\eta^2 L^2 L_e^2)C] \end{aligned} \tag{7}$$

There exists a trade-off between τ and the convergence error in (7). Larger τ would make sampling runtime per iteration, $\frac{S}{\tau}$, smaller, but will also introduce a larger coefficient for neighbour sampling related term C, which would cause a higher convergence error. So we need to carefully decide the value of τ. (7) itself is a convex function on τ. By setting its derivative to 0, we can obtain the optimal cross-device neighbor sampling interval:

$$\tau^* = \sqrt[3]{\frac{(F(\bar{w}_1) - F^*)(KS + P)}{C_T(L\eta^2 + \frac{\eta}{2})K(48L^2\eta^2 L_e^2 K^2 C)}} \tag{8}$$

From (8), we know that τ^* is a function of C_T, whose optimal value would vary with the change of current training iteration T and runtime C_T. To make calculation tractable, we can calculate the optimal τ every fixed $\hat{C}$ runtime and treat the training after each $\hat{C}$ runtime as a re-started training

[2] We use $[X]$ to denote the set of integers from 1 to X.

[3] We consider constant values of $\bar{C}$, S and P in this work. More complicated setting is left as future work.

with initial parameter $\bar{w}_{T_l}$ and initial loss value $F(\bar{w}_{T_l})$, where T_l denotes the training iteration at the start of the l-th $\hat{C}$ runtime. When $\hat{C}$ is small enough, the value of τ^* is accurate. We use A and B_l to denote $24(L\eta^2+\frac{\eta}{2})K^2\eta^2L^2L_e^2C$ and $\frac{F(\bar{w}_{T_l})-F^*}{\hat{C}}(\frac{KS+P}{K})$, respectively. The optimal τ_l^* for the l-th $\hat{C}$ runtime can be simplified as $\tau_l^* = \sqrt[3]{\frac{B_l}{2A}}$. We find the value of τ_1^* by grid search since we do not know the value of L, L_e and C in A. The value of B_1 can be easily computed by treating $F^*=0$. Based on τ_1^* and B_1, we can estimate the value of A, which will be used for calculating $\tau_l^*, l>1$, in both the current unconstrained case and the following constrained case.

B. Case with cross-device neighbour sampling constraint

We next study a more complicated setting where there is an explicit budget for the number of neighbour samplings that could be conducted throughout the GCN training process. Similarly, we use A and B_l to denote $24(L\eta^2+\frac{\eta}{2})K^2\eta^2L^2L_e^2C$ and $\frac{F(\bar{w}_{\Upsilon_l})-F^*}{C_{\Upsilon_l}}(\frac{KS+P}{K})$, the coefficient before τ_l^2 and $\frac{1}{\tau_l}$ in (7), respectively. We update τ every Υ training iterations (Υ training iterations are one decision slot, with C_{Υ_l} runtime for the l-th decision slot), not every $\hat{C}$ runtime to simplify the formulation in this setting. Suppose the number of decision slots for re-calculating τ is M. l denotes the index of a decision slot, $l\in[M]$, and $\bar{w}_{\Upsilon_l}$ denotes the parameter at the start of the l-th decision slot. The budget for neighbour sampling is denoted as Λ. We have

$$\max_{\tau_l}\sum_{l=1}^{M}-A\tau_l^2-\frac{B_l}{\tau_l}\quad \text{s.t}\sum_{l=1}^{M}\frac{\Upsilon}{\tau_l}\le\Lambda,\tau_l\ge 1,\forall l\in[M] \tag{9}$$

The goal is to minimize the convergence error across all decision slots while respecting the neighbour sampling budget. (9) is a convex problem with a concave objective over a convex set. We introduce Lagrangian multiplier λ for the budget constraint and μ_l for constraint $\tau_l\ge 1,\forall l\in[M]$. According to the KKT conditions [24], we have the following offline optimal solution:

- For $\tau_l>1$, $\mu_l=0, -2A\tau_l+B_l\tau_l^{-2}+\lambda\Upsilon\tau_l^{-2}=0$, which denotes $\tau_l^3=\frac{B_l+\lambda\Upsilon}{2A}$. Since $\lambda\ge 0$ and $\Upsilon>0$, we have $-2A+B_l>-2A\tau_l+B_l\tau_l^{-2}=-\lambda\Upsilon\tau_l^{-2}>-\lambda\Upsilon$.
- For $\tau_l=1, \mu_l\ge 0, -2A+B_l=-\lambda\Upsilon-\mu_l\le-\lambda\Upsilon$.

The value of λ in the above solution would not be known unless we have the complete sequence of $B_l,\forall l\in[M]$, which is related to the loss value $F(\bar{w}_{\Upsilon_l})$ experienced during training. Therefore, deciding optimal τ in this constrained case renders an online problem, as B_l would only be known when the training proceeds to the l-th decision slot.

The key idea in our online algorithm design is that if we can estimate the value of λ in the offline optimal solution, then we can solve the online problem (9) according to the offline solution above, based on the relationship between $-2A+B_l$ and $-\lambda\Upsilon$.

We make the assumption that B_l in the objective function is non-increasing. We wish to emphasize that $B_l=\frac{F(\bar{w}_{\Upsilon_l})-F^*}{C_{\Upsilon_l}}(\frac{KS+P}{K})$ includes the actual runtime $C_{\Upsilon_l}=\Upsilon(\bar{C}+\frac{S}{\tau_l}+\frac{P}{K\tau_l})$ of Υ iterations and the loss value $F(\bar{w}_{\Upsilon_l})$ at the start of decision slot l. The loss value should follow a decreasing trend when training progresses. C_{Υ_l} is related to the value of τ_l. In the training process, normally τ_l should follow a non-increasing trend to minimize the variance caused by neighbour sampling and ensure better convergence, which would make C_{Υ_l} non-decreasing. The increasing trend of C_{Υ_l} matches the assumption of decreasing B_l, which justifies our assumption on B_l.

Towards online algorithm design to solve for τ_l's, we first discuss the value of λ in the offline optimal solution based on the interpretation of Lagrangian multipliers [25], that an optimal Lagrangian multiplier evaluates the rate of change of the optimal objective value if the constraint corresponding to the multiplier is relaxed by an infinitesimal value.[4] In our case, the optimal value of λ should correspond to the greatest value increase in the optimal objective value when an infinitesimal amount of neighbour sampling budget (i.e., smaller τ) can be used, which corresponds to the additive inverse of the smallest gradient of the offline optimal solution among all decision slots due to the concave objective.

Under the non-increasing assumption of B_l, we can conclude from offline optimal solution that the gradient of objective in (9) with respect to decision variable τ_l for decision slot l is $-2A\tau_l+B_l\tau_l^{-2}$, which is monotonically decreasing with respect to l and implies that the smallest gradient is achieved at the last decision slot M. Hence, the value of λ in the offline optimal solution would resume the additive inverse of $-2A\tau_M+B_M\tau_M^{-2}$.

We next discuss possible values of λ under different offline optimal solutions of the last slot M.

Suppose $\tau_M>1$, $\lambda=2A\tau_M-B_M\tau_M^{-2}=\lambda\Upsilon\tau_M^{-2}=\lambda\Upsilon(\sqrt[3]{\frac{B_M+\lambda\Upsilon}{2A}})^{-2}$; after solving this equation, we have $\lambda_1=\Upsilon^{\frac{1}{2}}2A-\frac{B_M}{\Upsilon}$ or $\lambda_1=0$. Suppose $\tau_M=1$; we have $\lambda_2=2A-B_M$.

Training would achieve a small loss value at convergence (small $F(\bar{w}_{\Upsilon_M})-F^*$) and C_{Υ_M} would be larger than $S+P/K$ since we only calculate τ periodically, which would make $B_M=(F(\bar{w}_{\Upsilon_M})-F^*)\frac{KS+P}{KC_{\Upsilon_M}}$ close to 0, so that we can estimate the value of λ by treating $B_M=0$ and derive the online solution as in the offline optimum. Our proposed method is to first use a heuristic (introduced in Sec. III-C) to classify problem (9) into three cases:

(i) $\lambda=0$. $\lambda=0$ indicates that the sampling times budget is abundant, so that each τ_l resumes the optimal solution of (8).

(ii) $\tau_M>1$ and $\lambda>0$. We use estimated $\tilde{\lambda}_1=\Upsilon^{\frac{1}{2}}2A$.

(iii) $\tau_M=1$ and $\lambda>0$. We use estimated $\tilde{\lambda}_2=2A$.

$\tilde{\lambda}_1$ and $\tilde{\lambda}_2$ are upper bound of λ_1 and λ_2, respectively, so that the sampling budget constraint can always be satisfied. Let π be the ratio of the objective value of the offline optimal solution over the objective value achieved by our online algorithm. We have $\pi\in(0,1]$ since the objective is a negative value. Larger π indicates a better performance with

[4]We provide the proof in the Appendix for completeness.

our algorithm. We use π_l to denote the performance ratio for decision slot l; τ_l and $\tilde{\tau}_l$ represent the offline optimal solution and the solution of our online algorithm, respectively. We have $\pi \geq \min_l \pi_l$.

Lemma 2. *Given correct classification from the heuristics, we use $\tilde{\lambda}_1 = 2A\Upsilon^{\frac{1}{2}}$ and $\tilde{\lambda}_2 = 2A$ to solve for $\tilde{\tau}_l$ in each decision slot l as follows:*

$$\tilde{\tau}_l = \sqrt[3]{\frac{B_l + \tilde{\lambda}_1 \Upsilon}{2A}}, \text{if } \tau_M > 1 \text{ and } \lambda > 0 \tag{10}$$

$$\tilde{\tau}_l = \sqrt[3]{\frac{B_l + \tilde{\lambda}_2 \Upsilon}{2A}}, \text{if } \tau_M = 1 \text{ and } \lambda > 0 \tag{11}$$

Since $\tilde{\lambda}_2 = 2A, B_l \geq 0, \Upsilon \geq 1$ and $B_l + \tilde{\lambda}_2 \Upsilon > 2A$ always hold, we would not have online solution of $\tilde{\tau}_l = 1$. Suppose B_M of the last decision slot is no larger than ϵ and B_l in the objective is non-increasing. The algorithm achieves a bounded competitive ratio as follows:

$$\pi \geq \begin{cases} 1 - \frac{\epsilon}{3\epsilon + 2A\Upsilon^{\frac{3}{2}}}, & \text{if } \tau_M > 1 \text{ and } \lambda > 0 \\ 1 - \frac{\epsilon\Upsilon}{3\epsilon + 2A\Upsilon}, & \text{if } \tau_M = 1 \text{ and } \lambda > 0 \end{cases} \tag{12}$$

Furthermore, assuming $A \geq 2\epsilon$ and $\Upsilon \geq 1$, we have a lower bound to the value of the competitive ratio:

$$\pi \geq \begin{cases} 0.86, & \text{if } \tau_M > 1 \text{ and } \lambda > 0 \\ 0.75, & \text{if } \tau_M = 1 \text{ and } \lambda > 0 \end{cases} \tag{13}$$

We can see that our algorithm performs better and achieves close-to-1 performance ratio when the neighbour sampling budget is small, i.e., the communication resources are scarce ($\tau_M > 1$ and $\lambda > 0$). The ratio is worse when $\tau_M = 1$ and $\lambda > 0$. This is because $\tilde{\lambda}_1$ is a close estimation to λ_1 with $\frac{B_M}{\Upsilon}$ difference while the gap between $\tilde{\lambda}_2$ and λ_2 is B_M. However, since $\tilde{\lambda}_1 \geq \lambda_1$ and $\tilde{\lambda}_2 \geq \lambda_2$, there would be sampling budget left in both situations. We can improve the performance ratio in practice by setting smaller τ_l for later slots, to use up all the budget.

C. Unified solution

Next, we elaborate the heuristic used to classify problem (9) into the three different cases, given the neighbour sampling budget Λ, to provide a unified solution to federated graph learning. The key component of this heuristic is a loss prediction function

$$loss(t, \tau) = \frac{\beta_0}{\beta_1 t} + \beta_2 \tau^2 + \beta_3 \tag{14}$$

where β_0, β_1, β_2, β_3 are coefficients to be learned and t indicates the training iteration. The loss prediction function takes the above form based on the convergence rate in (6). To learn this function, we collect data points $(loss, t, \tau)$ [5] for different τ values, and then use the Lmfit package [26] for curve fitting. With the help of this loss prediction function, we can stimulate the training process beforehand so that the value of B_l and hence the value of τ_l returned from our method in different cases can be roughly estimated. With the value of τ_l, we can estimate the number of neighbour samplings conducted in different cases.

For $\lambda = 0$, we estimate the optimal value of τ_1^* by grid search and estimate each τ_l^* with (8) and (14). Then we can compute a lower bound $\Lambda_l^{(1)}$ of the neighbour sampling budget needed for case (i) by summing up the result of the number of iterations in the l-th run-time $\tilde{C}$ divided by τ_l^* for all $l \in [M]$, which would classify problems with $\Lambda \geq \Lambda_l^{(1)}$ into class of $\lambda = 0$. If estimated $\tau_M^* > 1$ in this case, we do need to consider the class with $\tau_M = 1$ and $\lambda > 0$, so we are left with only one class of $\tau_M > 1 \wedge \lambda > 0$ when $\Lambda < \Lambda_l^{(1)}$.

If estimated $\tau_M^* = 1$ for case $\lambda = 0$, we estimate an upper bound $\Lambda_l^{(2)}$ of the neighbour sampling budget needed for the case (ii) of $\tau_M > 1$ and $\lambda > 0$. Similarly, we take $\tilde{\lambda} = \Upsilon^{\frac{1}{2}} 2A$, and estimate each τ_l by $\tau_l = \sqrt[3]{\frac{B_l + \tilde{\lambda}\Upsilon}{2A}}$ and (14). With the estimated value of τ_l, we could calculate $\Lambda_l^{(2)}$ easily by summing up the values of $\frac{\Upsilon}{\tau_l}$ for all $l \in [M]$.

Then given a problem of (9) with sampling times budget Λ, its classification result is as follows:

$$\begin{cases} \text{case (i) with } \lambda = 0, & \text{if } \Lambda \geq \Lambda_l^{(1)} \\ \text{case (ii) } \tau_M > 1 \wedge \lambda > 0, & \text{if } \Lambda < \Lambda_l^{(2)} \\ \text{case (iii) } \tau_M = 1 \wedge \lambda > 0, & \text{if } \Lambda_l^{(2)} \leq \Lambda < \Lambda_l^{(1)} \end{cases}$$

We present the unified algorithm in Algorithm 2.

IV. Evaluation

A. Experimental Setup

Dataset We train GCNs on two large graphs: (1) ogbn-products, an undirected and unweighted graph from Amazon, whose nodes represent products while edges indicating a co-purchased relationship; the task is to predict the category of products. (2) ogbn-arxiv, a directed citation network among papers; the learning task is to predict missing citations. Details of the two datasets are in Table I.

Model and Hyper-parameters We implement the most widely used neighbour sampling-based GCN model, graph-SAGE [5]. The number of layers in the GCN is 3, each with 128 hidden neurons except the output layer. We choose a neighbour sampling strategy that leads to less number of neighbour nodes sampled, while the performance remains similar to that with more neighbour nodes being sampled. For the products dataset, its neighbour sampling strategy is to uniformly sample 15, 10 and 5 neighbors for output layer, hidden layer and input layer, respectively, for each node. For the arxiv dataset, the number of neighbour nodes being sampled at each layer for each node is fixed to 4. To simulate federated learning, we partition each dataset to 16 devices. At each round of federated training, we randomly sample 4 devices to conduct local model update. We consider two partition strategies: random (uniformly randomly partition nodes among devices) and label/time-based. We conduct label-based partition on products dataset, which partitions nodes

[5]To reduce overhead, data points can be collected during the grid search phase. The loss function fitting results of different tasks based on data points from the grid search are given in the Sec. IV-C

Algorithm 2: Federated graph learning

Input: $\mathrm{B}, \Lambda, \hat{C}, \Upsilon, K, S, P, T$

Initialize $t = 1, r = 1, s = 0, r' = 0$;
// t,r,s for tracking iterations, runtime and numbers of conducted update of τ ;

Grid search τ_1^*, calculate A, collect points $(loss, t, \tau)$, $loss(t, \tau)$ curve fitting by Lmfit;

Estimate $\Lambda_l^{(1)}$ by $loss(t, \tau)$ and (8). Estimate $\Lambda_l^{(2)}$ by $loss(t, \tau)$ and $\tau_l = \sqrt[3]{\frac{B_l + 2A\Upsilon^{\frac{3}{2}}}{2A}}$;

while $t \in [T]$ **do**

  if $\Lambda \geq \Lambda_l^{(1)}$ **then**

    if $r \% \hat{C} == 0$ **then**

      $B = \frac{F(\bar{w}_t) * (KS+P)}{\hat{C}K}$, $\tau = \sqrt[3]{\frac{B}{2A}}$;

    end

  else if $\Lambda \leq \Lambda_l^{(2)}$ **then**

    if $t \% \Upsilon == 0$ **then**

      $B = \frac{F(\bar{w}_t) * (KS+P)}{(r-r')K}$, $\tau = \sqrt[3]{\frac{B + 2A\Upsilon^{\frac{3}{2}}}{2A}}, r' = r$;

    end

  else if $t \% \Upsilon == 0$ **then**

    $B = \frac{F(\bar{w}_t) * (KS+P)}{(r-r')K}$, $\tau = \sqrt[3]{\frac{B + 2A\Upsilon}{2A}}, r' = r$;

  end

  Sample workers for current iteration;

  if $t \% \tau == 0$ **then**

    Conduct neighbour sampling, $r+ =$S, s += 1;

  Sampled workers update model locally, $r+ = \bar{C}$;

  if $t \% K\tau == 0$ **then**

    Conduct parameter averaging, $r+ =$P ;

  end

  if $s \geq \Lambda$ **then**

    break ;

  end

end

belonging to the first 20 classes to the first 8 devices with 0.8 probability, and nodes of other classes to other devices with 0.8 probability, to simulate a non-iid distribution. For the arxiv dataset, we partition papers according to the publication dates so that different devices hold papers with different publication time ranges. We set $\hat{C} = 10s$ in the unconstrained case and $\Upsilon = 100$ iterations for the constrained case. Learning rate and mini-batch size are fixed to 0.01, 256 for products and 0.01, 1024 for arxiv, respectively. K is set to 10.

We implement our algorithm on Pytorch [27] and DGL [28], and simulate the federated learning scenario on a machine with four NVIDIA Tesla V100 GPU. In the experiment, we run sampling and training for 100 iterations to calculate the average number of nodes sampled residing on other devices per neighbour sampling and average computation time $\bar{C}$ per iteration. The cross-device network bandwidth is set to 1Gbps. Given the average number of nodes sampled residing on other devices per neighbour sampling, the feature/embedding dimensions and bandwidth, we can calculate the value of S. The detailed statistics about sampling time and computation time are summarized in Table I.

TABLE I: Details of graph datasets

Dataset	# of nodes	# of edges	degree	S	$\bar{C}$
products	2,449,029	61,859,140	50.5	0.51s	0.06s
arxiv	169,343	1,166,243	13.7	0.27s	0.07s

B. Experimental Results

Fig. 3 and Fig. 4 present losses during training of the GCN on ogbn-products dataset with random graph partition among devices. $\tau = 1$ denotes the baseline solution which conducts neighbour sampling in each training iteration. $\tau = 40$ is the optimal neighbour sampling interval identified by grid search after running training for a few iterations and choosing the τ with the best loss reduction/runtime value, which is also the solution to τ_1^* for unconstrained case. In the unconstrained case in Fig. 3, training is run until model convergence. Our method ('Adaptive') can achieve about $2\times$ speed-up than $\tau = 1$ for reaching the same error floor. Besides, with $\tau = 40$, training loss decreases rapidly at the start of training, but the error it converges to is larger than the error floor found by adaptive τ and $\tau = 1$. This implies grid search is not sufficient for finding a good τ in federated graph learning. In the unconstrained case, $\Lambda_l^{(1)}$ is about 3000 for each device.The value of τ_M^* estimated for unconstrained case is about 20, which means when $\Lambda < \Lambda_l^{(1)}$, $\lambda > 0$ and $\tau_M > 1$. So in the constrained case, we give each device a neighbour sampling budget of 2000, and run training until the budget is used up. λ in the offline optimum is estimated by B_M and A from the unconstrained case according to KKT conditions. In Fig. 4, we observe that our online algorithm ('Online') can achieve comparable performance as the offline optimum, and outperforms the baseline and grid-searched τ in terms of convergence speed and error. Similar results can be observed under label-based partitioning of the graph across devices in Fig. 6 and Fig. 7. The test accuracy and ratio between neighbour sampling runtime and computation runtime of the training process are shown in Fig. 5 and Fig. 8, for random partition and label-based partition separately (Adaptive refers to the unconstrained case while other methods are compared under constrained case).

The results on the arxiv dataset of time-based partition are given in Fig. 9 to Fig. 11. The estimated $\Lambda_l^{(1)}$ is about 3000 for each device. For constrained case, we give each device a more stringent neighbour sampling budget of 1000. From the results we can see that our methods achieve the best trade-off between training error and actual runtime in both unconstrained and constrained cases. $\tau = 40$ leads to similar performance as our method since the number of cross-device nodes is small in this scenario. Test accuracy and the ratio between neighbour sampling runtime and computation runtime are in Fig. 11.

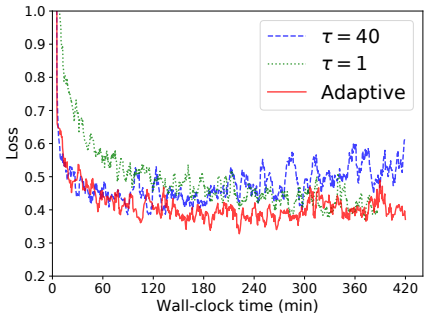

Fig. 3: Unconstrained random partition

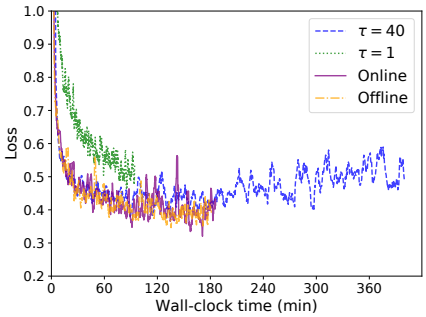

Fig. 4: Constrained random partition

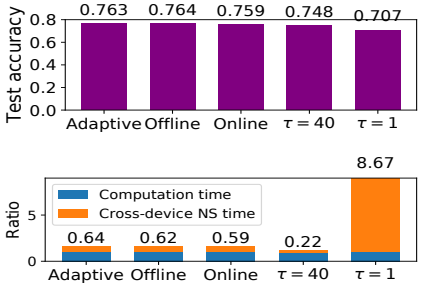

Fig. 5: Test accuracy and ratio

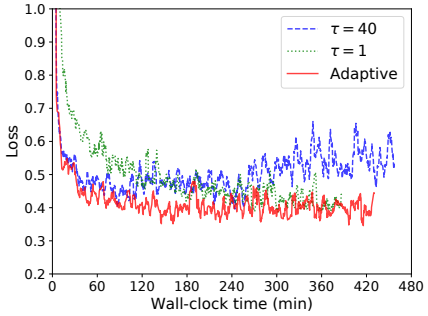

Fig. 6: Unconstrained label-based partition

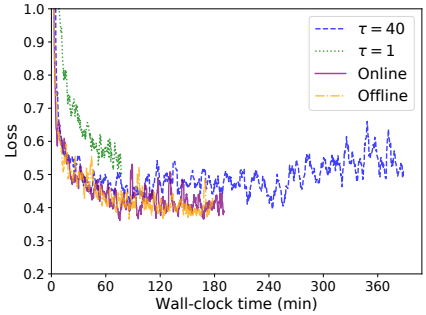

Fig. 7: Constrained label-based partition

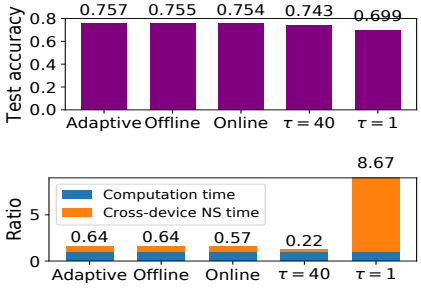

Fig. 8: Test accuracy and ratio

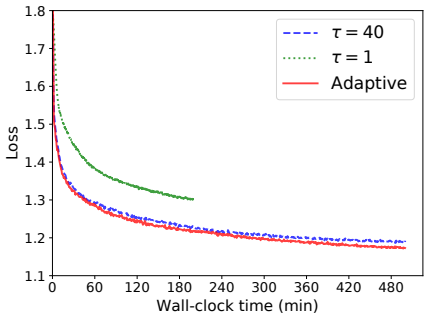

Fig. 9: Unconstrained time-based partition

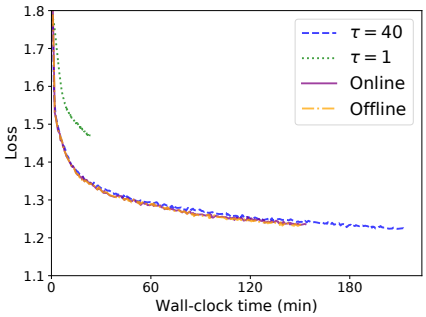

Fig. 10: Constrained time-based partition

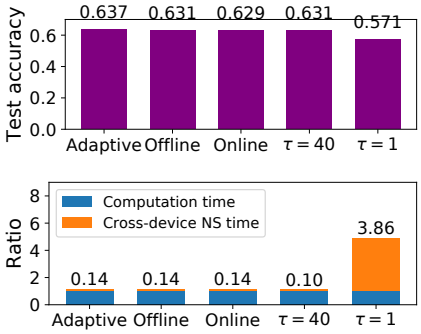

Fig. 11: Test accuracy and ratio

C. Impact of Heuristics and Assumptions

To unify the solutions to τ under unconstrained and constrained case, we propose a heuristic method for estimating the number of neighbour samplings conducted in different cases so that given sampling budget Λ, we could classify problems and solve the value of τ accordingly. Next, we explain all the experimental details about this heuristic.

Recall that the accuracy of this heuristic relies on accurate loss prediction. To avoid extra communication and computation resource consumption for leaning the parameters in function (14), we collect $(loss, t, \tau)$ data points during the grid search phase of τ_1^*, whose value set for conducting grid search is $[10, 20, 30, 40, 50]$. $[0, 10]$ turns out to be a good value range for β_0, β_1 and β_2 and $[0, l]$ would be a good value range for β_3 when curve fitting is conducted by Lmfit, here l is the smallest loss value from grid search. Initial value of these parameters are all set to $\min(1.0, l)$. Fig. 15 shows the comparison between estimated loss and loss returned by real training process when τ is updated by (8). Since our loss curve fitting is only based on data points from grid search phase, the estimated loss value would be larger than real loss value when training proceeds, which remains about 1.7 times higher than the real loss for products dataset and about 1.3 times higher than the real loss for arxiv datset. However, this larger estimated loss would only have very small effect to the value of estimated τ given the cubic root in the solutions to τ in both unconstrained and constrained case. For example, τ calculated by (8) ($\tau_l^* = \sqrt[3]{\frac{B_l}{2A}}, B_l = \frac{F(\bar{w}_{\Upsilon_l})}{\hat{C}}(\frac{KS+P}{K})$) using estimated loss would only be about 1.2 times of the τ calculated by real loss.

Given estimated value of τ, the number of neighbour samplings $\Lambda_l^{(1)}$ and $\Lambda_l^{(2)}$ conducted in different cases can be

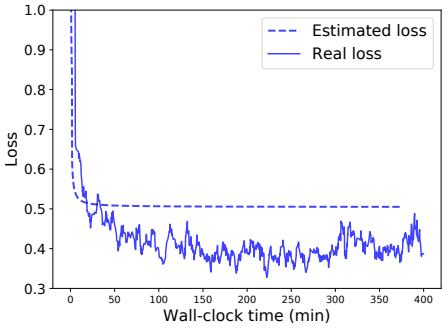

Fig. 12: Products dataset(random)

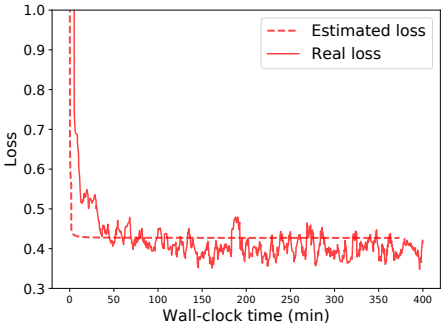

Fig. 13: Products dataset(label-based)

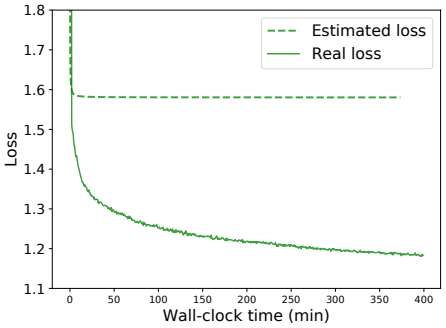

Fig. 14: Arxiv dataset(time-based)

Fig. 15: Loss estimation results

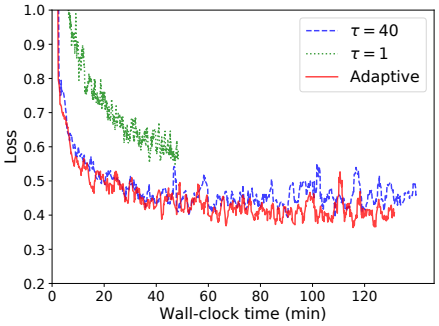

Fig. 16: $K = 1$

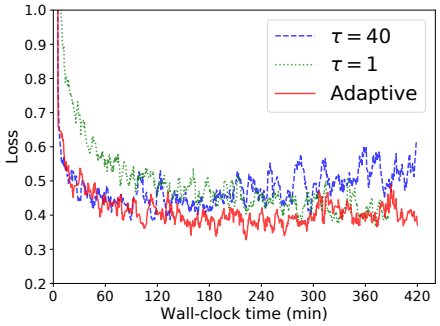

Fig. 17: $K = 10$

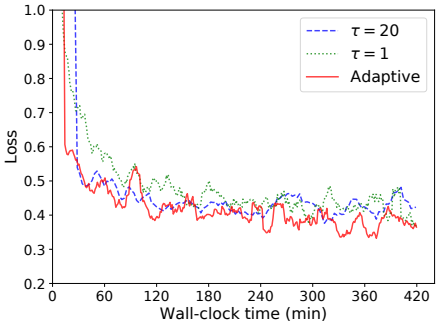

Fig. 18: $K = 100$

Fig. 19: Training result by varying K

estimated to classify problems.

Next, we discuss the value of $B_M = \frac{F(\bar{w}_{\Upsilon_M})}{C_{\Upsilon_M}}(\frac{KS+P}{K})$, which is assumed to be close to 0 when estimating λ for constrained case under the assumption of $F^* = 0$. The parameter averaging time P is negligible in GCN training process and the value of S is shown in Table I. In our experiment, we set $\Upsilon = 100$ because we do not need to update τ too frequently, then C_{Υ_M} would be larger than 7s in both products and arxiv dataset given the computation in each iteration takes about 0.06 to 0.07 seconds ($\bar{C}$ in Table I), which renders the value of B_M less than 0.05.

D. Impact of K

We test the performance of our method under different choices of K, which is used in Assumption 3 to define the relationship between neighbour sampling interval and parameter synchronization interval for federated learning. We test a small value of $K = 1$, a medium value of $K = 10$ and a large value of $K = 100$ to show that the effectiveness of our method does not rely on specific value of K. The experimental results on products dataset are shown in Fig. 19, where the optimal τ_1^* for $K = 100$ returned by grid search is 20. We can see that, under different choices of K, it always takes a longer time to converge with frequent neighbour sampling interval $\tau = 1$, and the grid search returned neighbour sampling interval always leads to higher loss. Our method decides neighbour sampling interval adaptively and finds a good trade-off between convergence error and actual runtime.

V. Related Work

Few studies focus on graph training across geo-distributed devices. Most existing works consider distributed graph training in one data center [29] [30]. Wu *et al.* [11] address privacy issues in federated graph learning by applying differential privacy to local gradients and proposing a privacy-preserving method for sampling neighbour nodes. Zheng *et al.* [31] address the non-IID data issue and jointly optimize hyper-parameters across devices in federated GCN training with Bayesian optimization. Wang *et al.* [32] propose a federated learning framework for semi-supervised node classification tasks on geo-distributed graphs. They utilise a model-agnostic meta-learning based method to deal with non-IID data, new label domain issue and unlabelled data. In the federated graph learning algorithm proposed by [33], predicted results and embeddings are sent to the server along with local gradients for global model learning. Sending these can consume large WAN bandwidth. There is no existing work explicitly considering communication overhead in federated graph learning.

For communication reduction in GCN training, the most common approach is to perform neighbour sampling [5] [18]. As we show in Fig. 1, even when neighbour sampling is

enabled, the communication overhead is still overwhelming in the federated learning setting. Ramezani *et al.* [34] propose periodic neighbour sampling when training GCN in one GPU-equipped machine to reduce the time consumed by transmitting neighbour nodes from CPU to GPU. They use a fixed sampling interval and do not take actual runtime into consideration. Parallel GCN training to reduce communication is considered by [35]. In this work, propagation and embedding matrices are partitioned to parallelize training and reduce unnecessary communication, which is prohibited in federated learning. Other works for reducing communication in distributed GCN training manipulate training data, such as collecting and re-partitioning the complete graph [29] [36]. None of the above approaches can be applied to federated learning directly, which motivates the study in this paper.

VI. Conclusion

This paper presents a federated graph learning framework to reduce the communication overhead incurred by cross-device neighbour sampling. We propose to conduct neighbour sampling periodically and derive the optimal neighbour sampling interval based on the trade-off between convergence error and actual runtime. We both theoretically and empirically show the effectiveness of our methods.

VII. Acknowledgements

This work was supported in part by grants from Hong Kong RGC under the contracts HKU 17204619, 17208920, 17207621, and C5026-18G (CRF).

References

[1] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 591–600.

[2] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, 2014.

[3] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv:1609.02907*, 2016.

[5] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[6] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin, "Revisiting graph neural networks for link prediction," *arXiv:2010.16103*, 2020.

[7] P. Li, Y. Wang, H. Wang, and J. Leskovec, "Distance encoding: Design provably more powerful neural networks for graph representation learning," *arXiv:2009.00142*, 2020.

[8] S. A. Rahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani, "A survey on federated learning: The journey from centralized to distributed on-site learning and beyond," *IEEE Internet of Things Journal*, 2020.

[9] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[11] C. Wu, F. Wu, Y. Cao, Y. Huang, and X. Xie, "Fedgnn: Federated graph neural network for privacy-preserving recommendation," *arXiv:2102.04925*, 2021.

[12] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[13] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv:2005.00687*, 2020.

[14] G. Li, M. Müller, B. Ghanem, and V. Koltun, "Training graph neural networks with 1000 layers," *arXiv:2106.07476*, 2021.

[15] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 7370–7377.

[16] G. Zhang, H. He, and D. Katabi, "Circuit-gnn: Graph neural networks for distributed circuit design," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7364–7373.

[17] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[18] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv:1801.10247*, 2018.

[19] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," *arXiv:1710.10568*, 2017.

[20] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd," 2019.

[21] H. Yu, S. Yang, and S. Zhu, "Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 5693–5700.

[22] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv:1212.5701*, 2012.

[23] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv:1706.02677*, 2017.

[24] S. Boyd and L. Vandenberghe, *Convex Optimization*. USA: Cambridge University Press, 2004.

[25] C. P. Simon, L. Blume *et al.*, *Mathematics for economists*. Norton New York, 1994, vol. 7.

[26] M. Newville, T. Stensitzki, D. B. Allen, M. Rawlik, A. Ingargiola, and A. Nelson, "Lmfit: Non-linear least-square minimization and curve-fitting for python," *Astrophysics Source Code Library*, pp. ascl–1606, 2016.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[28] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," 2020.

[29] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken, "Improving the accuracy, scalability, and performance of graph neural networks with roc," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 187–198, 2020.

[30] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai, "Neugraph: parallel deep neural network computation on large graphs," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019, pp. 443–458.

[31] L. Zheng, J. Zhou, C. Chen, B. Wu, L. Wang, and B. Zhang, "Asfgnn: Automated separated-federated graph neural network," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1692–1704, 2021.

[32] B. Wang, A. Li, H. Li, and Y. Chen, "Graphfl: A federated learning framework for semi-supervised node classification on graphs," *arXiv:2012.04187*, 2020.

[33] C. Chen, W. Hu, Z. Xu, and Z. Zheng, "Fedgl: Federated graph learning framework with global self-supervision," *arXiv:2105.03170*, 2021.

[34] M. Ramezani, W. Cong, M. Mahdavi, A. Sivasubramaniam, and M. Kandemir, "Gcn meets gpu: Decoupling "when to sample" from "how to sample"," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[35] A. Tripathy, K. Yelick, and A. Buluc, "Reducing communication in graph neural network training," *arXiv:2005.03300*, 2020.

[36] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 257–266.