FedWalk: Communication Efficient Federated Unsupervised Node Embedding with Differential Privacy

Qiying Pan
Shanghai Jiao Tong University
Shanghai, China
sim10_arity@sjtu.edu.cn

Yifei Zhu*
Shanghai Jiao Tong University
Shanghai, China
yifei.zhu@sjtu.edu.cn

ABSTRACT

Node embedding aims to map nodes in the complex graph into low-dimensional representations. The real-world large-scale graphs and difficulties of labeling motivate wide studies of unsupervised node embedding problems. Nevertheless, previous effort mostly operates in a centralized setting where a complete graph is given. With the growing awareness of data privacy, data holders who can be represented by one vertex in the graph and are only its neighbors demand greater privacy protection. In this paper, we introduce FedWalk, a random-walk-based unsupervised node embedding algorithm that operates in such a node-level visibility graph with raw graph information remaining locally. FedWalk is designed to offer centralized competitive graph representation capability with data privacy protection and great communication efficiency. FedWalk instantiates the prevalent federated paradigm and contains three modules. We first design a hierarchical clustering tree (HCT) constructor to extract the structural feature of each node. A dynamic time warping algorithm seamlessly handles the structural heterogeneity across different nodes. Based on the constructed HCT, we then design a random walk generator, wherein a sequence encoder is designed to preserve privacy and a two-hop neighbor predictor is designed to save communication cost. The generated random walks are then used to update node embedding based on a SkipGram model. Extensive experiments on two large graphs demonstrate that FedWalk achieves competitive representativeness as a centralized node embedding algorithm does with only up to 1.8% Micro-F1 score and 4.4% Marco-F1 score loss while reducing about 6.7 times of inter-device communication per walk.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Security and privacy** → **Privacy-preserving protocols**; • **Theory of computation** → **Unsupervised learning and clustering**; • **Computer systems organization** → **Distributed architectures**.

*Corresponding author

KEYWORDS

Graph representation learning; federated analytics; unsupervised node embedding; differential privacy

1 INTRODUCTION

With the rise of graph convolutional networks and the abundant graph-structured data, graph representation learning has drawn great attention as it effectively extracts the structural information from complex graphs. Node embedding is a representation learning problem that captures the features of nodes and topological information in a graph with vectors. The essence of node embedding is to represent each node such that the distance in the embedding space measures the dissimilarity among nodes in the original graph. If two nodes have similar structural features, the distance of their corresponding embedding vectors is small. Node embedding can be deployed in various downstream prediction tasks, such as link prediction [27], anomaly detection [34], and so on.

Typical node embedding approaches follow a similar encoder-decoder framework. First, an encoder is defined to map a node to a vector. Then a node proximity function is defined to examine whether the current encoding result approximates the topological information of the original graph. Last but not least, the encoder is optimized based on the previous examination result. The key differences among various node embedding approaches are the definitions of encoders and node proximity functions. For unsupervised node embedding, node proximity functions typically rely on random walk methods due to their expressiveness and efficiency. Deepwalk [20] pioneers the random-walk-based embedding by leveraging unbiased random walk to measure the similarity between each node. Node2vec [9] later replaces unbiased random walk with biased one to capture local microscopic and global macroscopic views of a graph. On the other hand, for supervised node embedding, node proximity functions are usually defined based on the labels in the downstream tasks. As neural networks have shown their great power in various fields, some frameworks also deploy neural networks to embed nodes in supervised and unsupervised settings [10–12].

All the representation learning methods discussed above operate in a centralized setting, where a server owns the complete graph information. However, the growing awareness of privacy and the establishment of regulations and laws, e.g., GDPR [7] and CCPA

[3], demand stricter data privacy protection. With the increasing computing power of mobile devices, the federated paradigm is proposed by Google and later being widely studied and deployed in real life[15]. In the federated paradigm, devices with private data work collaboratively with a central server to conduct machine learning and data analytics tasks without uploading the raw data. The core of the federated paradigm is to train local models on individual devices first. Then the trained local models are aggregated at the server side to generate a global model after multiple iterations between clients and the server. In this whole process, only the extracted insights are uploaded to the server without revealing clients' raw data. The applications of federated learning on regular data structures, e.g., images [14], text[31], have demonstrated tremendous success in real life.

With the wide prevalence of graph-structured data, the federation of graph embedding, especially graph neural network, has also attracted significant attention. Existing federated graph embedding studies can be categorized into three types depending on how much graph information each local client owns: node-level[17, 18], subgraph-level[30] and graph-level[19, 35, 37] federated graph embedding. Whereas graph-level and subgraph-level federated graph embedding still assume each client holds a correlated complete graph or a part of a global graph, node-level federated graph embedding emphasizes the privacy of individual nodes in one global graph. Each node in the graph is owned by one client and it is only aware of its neighboring situation. The server only maintains the minimal node membership information. Many real-world networks such as social networks [36] and sensor networks [32] naturally born with this setting. It is straightforward to notice that node-level graph embedding is the most challenging task since it contains the least structure information on each local client.

Previous works on node embedding in federated node-level scenarios mainly concentrate on supervised settings with labels from downstream tasks. SGNN[17] takes advantage of a degree list to compute node distance and manages to perform supervised node embedding under privacy constraints. GraphFL[26], motivated by meta learning, addresses semi-supervised node embedding in federated networks. Nevertheless, in reality, most data are unlabeled; manual labeling data is also expensive due to the large scale of real-world graphs. These all motivate the study of unsupervised node embedding in the federated environment, which has barely been studied yet. *Considering the huge demand in processing unlabeled graph data and the growing restrictions on node-level data access, we aim at filling this gap in this paper.*

The challenges for providing unsupervised node embedding in the node-level federated setting come from three perspectives. First, *learning*. Inaccessibility of graph information impedes deployment of conventional node proximity functions in federated graphs. With each client possessing just one-node information, achieving centralized competitive representation performance calls for new designs. Second, *privacy*. How to protect raw graph connection information from other clients and the central server, preferably theoretically guaranteed, is non-trivial. Naive deployment of the differential privacy measures may greatly affect the expressiveness of the learned results. Last but not least, *communication efficiency*. The node size of a practical network can scale to tens of thousands or even larger. Straightforward collaborations among all clients incur prohibitive

communication costs in practice and hinder its deployment. How to embed nodes communicationally efficiently should be seriously addressed.

In this paper, we propose a federated random-walk-based node embedding approach, named FedWalk, that successfully operates in decentralized graphs with node-level visibility. To do so, FedWalk first adopts a novel hierarchical clustering tree (HCT) constructor which hierarchically clusters vertices based on their linkage similarity in a federated graph. Second, it has a random walk sequence encoder to output a sequence with $\epsilon-$differential privacy. Last but not least, FedWalk includes a two-hop neighbor predictor to predict the possible two-hop nodes without actually visiting the direct ones. To the best of our knowledge, FedWalk is the first work for node-level federated unsupervised graph embedding, where each node is isolated and maintained by a data holder with only one-hop connection information available. In summary, our contributions are summarized as follows:

- We propose a novel federated unsupervised node embedding framework that robustly captures graph structures, guarantees differential privacy and operates with great communication efficiency.
- We propose a differentially private HCT constructor that extracts structural information from decentralized data holders. A dynamic time warping algorithm seamlessly handles the structural heterogeneity across different nodes.
- We propose a novel random walk generator with a differentially private sequence encoder and a neighbor predictor for communication efficiency.
- We prove that the dissimilarity value loss of node pairs in our framework is bounded. We also theoretically quantify the reduced communication cost benefited from our neighbor predictor.
- Extensive experiments on two datasets of different scales demonstrate that FedWalk can achieve centralized competitive results by losing up to 1.8% Micro-F1 score and 4.4% Marco-F1 score while decreasing about 6.7 times of inter-device communication per random walk.

2 RELATED WORK

In this section, we review the related works of this paper from three perspectives: unsupervised node embedding in the centralized setting; the emerging federated learning and analytics paradigm; the growing works on federated graph representation.

2.1 Unsupervised Node Embedding

Node embedding uses low-dimensional vectors to represent nodes in a graph. In practice, the scale of a graph is usually too big to have all nodes labeled. The inevitable absence of supervision information triggers the formation and development of unsupervised node embedding algorithms. We can classify these algorithms into two sorts: (1) shallow networks with one hidden layer; (2) deep networks with multiple graph-based layers.

Random-walk-based shallow embedding supports effective solutions for many graph-related tasks[8]. DeepWalk[20] pioneers this branch by treating the path of random walk similar to the word sequences processing in the NLP field and takes advantage of the

Skipgram model to accomplish embedding. Since DeepWalk, we have witnessed significant efforts in random-walk-based shallow embedding, including Node2Vec[9], HARP[4], etc.

Deep embedding leverages multi-layer convolutional networks to learn graph representations. Application of convolutional networks in node embedding is started with GCN[12] propagating neighboring embedding through layers. Since then, extensive efforts [10] [11] have been made to improve representation performance in practical large graphs. Albeit the above well-developed node embedding methods, all of them require upload of raw data to the central server, which gradually becomes hard due to laws and regulations.

2.2 Federated Learning and Analytics

Federated learning and analytics have attracted great interest from both academia and industry due to their capability in conducting learning and analytic tasks without centralizing the raw data[15]. Specifically, federated learning has been widely studied for collaborative neural network training. Tremendous efforts have been devoted to improving federated learning in terms of convergence speed, training accuracy, communication efficiency, fairness, etc. Several good surveys are presented here for interested readers[13]. In practice, federated learning has also been applied in natural language processing[31], computer vision[14], and many other areas.

Federated analytics, a recently proposed sibling problem, extends the applications of federated paradigm from neural network training to diverse data analytic tasks or unsupervised learning scenarios[21]. TriHH[38] discovers the heavy hitters in a population of data without uploading the data to the server. Federated location heatmap from Google applies distributed differential privacy to generate location heatmaps[1]. FedFPM [28] presents a unified framework for frequent pattern analysis. While most federated designs follow an iterative collaboration pattern, the exact extracted insights and aggregation methods vary greatly across different problems. This work pushes the boundary of the federated applications by studying a novel federated unsupervised node embedding problem. FedWalk also encompasses its design to provide communicating efficient, differentially private, centralized competitive node embedding.

2.3 Federated Learning on Graphs

Unlike classical federated learning problems, the explicit linkage between nodes acts as a barrier to adapting algorithms in centralized graphs to federated ones. Due to privacy concerns, graphs cannot be uploaded to the server and can only be stored locally on each data holders. Despite this, some studies still succeed in designing algorithms to compute node embedding in federated graphs, especially under the subgraph-level visibility[30] and graph-level visibility[37][35][19], where each client owns a part of the graph or one correlated graph. In the most strict node-level federated graph works, devices only store the feature and edge information of one node. Leakage of node feature and edge information is forbidden. Under some circumstances, even the sharing of node labels is limited. This setting is applicable in social networks, Internet of Things (IoT) networks, and so on. Some studies have addressed

the problem of inaccessibility of node data. SGNN[17] uses one-hot encoding to process local information and introduces a new node distance calculation method based on an ordered degree list to process structural information. A paradigm called CNFGNN[18] alternatively trains models on local devices and the server in a graph representing spatio-temporal data structure.

Although there are some novel methods to train federated GNN, their models can only be applied in supervised settings with nodes labeled. How to provide node embedding in unsupervised federated scenarios remains to be solved. Furthermore, the extensive communication cost in federated node-level graph embedding should be carefully addressed to be practical in real-world systems.

3 PRELIMINARIES

In this section, we first present a basic introduction to the classical node embedding process in the central situation. Then considering the special emphasis of privacy in a federated system, we introduce the preliminaries on a widely-used privacy protection measure, differential privacy.

3.1 Classical Node Embedding

Node embedding is one of the most fundamental problems in graph theory. For a graph $G = (V, E)$, node embedding represents the structural information of vertices. Vertices that are linked with shorter distances or share similar topological structures tend to be embedded with shorter distances.

The node embedding algorithm is generally composed of three parts: (1) an encoder; (2) a dissimilarity function; (3) an optimizer. The encoder (ENC) is a function that maps a vertex to a fixed-dimension vector defined as Eq.1.

$$ENC : V \rightarrow \mathbb{R}^d \tag{1}$$

One of the most basic encoders is shallow encoding which serves as a look-up table for all the vertices. A matrix stores all the embedding vectors and an indicator vector assigns different embedding vectors to different vertices.

The dissimilarity function (DISSIM) in the form of Eq. 2 defines to what extent two vertices are dissimilar to each other.

$$DISSIM : V \times V \rightarrow \mathbb{R} \tag{2}$$

A typical dissimilar function is the probability of two vertices that don't co-occur in a random walk.

The optimizer is a machine learning algorithm deployed to optimize the parameters in the encoder aiming to realize that the distance between two embedding vectors approximates the corresponding similarity value. The goal can be formulated as

$$\min_{ENC} \sum_{v_1, v_2 \in V} \text{dist}(ENC(v_1), ENC(v_2)) - DISSIM(v_1, v_2). \tag{3}$$

In unsupervised settings, random-walk-based graph embedding is widely used to learn vector representations for nodes. A random walk generator randomly samples the neighbors of the last vertex to form a sequence of vertices of a certain length. Using the embedded node proximity information in these sequences, models like the Skipgram model map the results of the random walks into real

values representing certain node proximity. The probability of co-occurrence of two vertices in a random walk sequence represents the similarity between two vertices.

3.2 Differential Privacy

Differential privacy (DP) [5] is a prevalent measure defining whether an information-sharing strategy preserves privacy or not. It is motivated by the idea that if a single element in the dataset is replaced, the result of published information differs very small from the ideal one. The formal definition is shown in Def.3.1.

Definition 3.1. Let $\epsilon \in \mathbb{R}^+$ and $\mathcal{A}$ be a randomized algorithm. The algorithm $\mathcal{A}$ is defined as $\epsilon-$ differential privacy if for all datasets D_1 and D_2 whose elements are the same except for one element and all sets $S \subseteq \text{Range}(\mathcal{A})$,

$$\Pr[\mathcal{A}(D_1) \in S] \le \exp(\epsilon)\Pr[\mathcal{A}(D_2) \in S] \qquad (4)$$

Many DP mechanisms have been proposed, such as Laplace mechanism [6], exponential mechanism[16] and randomized response[29].

4 DESIGN

In this section, we first briefly introduce the overall design of our framework, FedWalk. We then present two core components of FedWalk, hierarchical clustering tree constructor and random walk generator in detail.
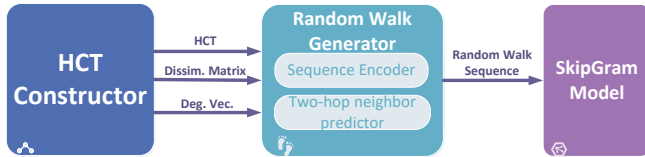
4.1 Design Overview

Figure 1: FedWalk framework

FedWalk is a random-walk-based graph embedding algorithm for unsupervised federated node embedding when each client [1] only has its neighboring information. The server is not allowed to access the raw local graph information and is only aware of which node is in the graph. FedWalk can not only embeds structure information without labeled data but also provides differential privacy to the clients' local connection information. FedWalk can be divided into three parts as shown in Fig. 1. To tackle the problem of inaccessibility of global graph, we introduce a differentially private *hierarchical clustering tree (HCT) constructor* based on a newly-proposed distance computing metric that clusters vertices according to their structural information. The tree is constructed gradually among clients under the coordination of a central server. A revised *random walk generator* is then developed to construct random walk sequences in federated settings based on the constructed hierarchical clustering tree. In the random walk generator, FedWalk contains a sequence encoder to encrypt the edge information to provide privacy and a two-hop neighbor predictor to save the communication cost among devices. The typical *Skip-Gram model* is then used

---

[1]Clients and data holders are used interchangeably in this paper.

to improve node embedding based on the random walk sequence generated in the second part.

4.2 Hierarchical Clustering Tree Constructor

The HCT constructor aims to build a hierarchical clustering tree in federated settings which will later be used for encoding and transmission of random walk sequences. It is designed to preserve structural information with privacy guaranteed.
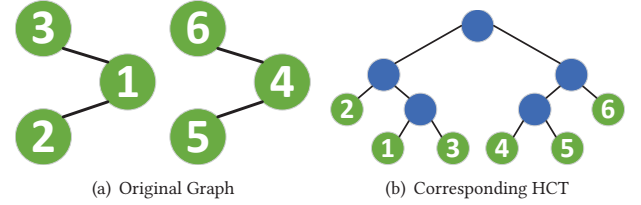

(a) Original Graph   (b) Corresponding HCT

Figure 2: An example of HCT

A hierarchical clustering tree (HCT) clusters vertices in the graph using edge information. It is widely used in NLP tasks[25], biological information processing[33] and so on. A hierarchical clustering structure separates vertices into two groups and then these groups further split themselves into two small groups. This splitting process continues until every group only contains one vertex. The leaf nodes in the tree correspond to the vertices in the graph. A none-leaf node, sometimes called an internal node, clusters a group of vertices by putting them to its left subtree or right subtree. As Figure 2 illustrates, the root node clusters the vertices into two groups $\{1, 2, 3\}$ and $\{4, 5, 6\}$. The internal nodes further split the groups into different clusters. It is straightforward to observe that every HCT has $|V| - 1$ internal nodes for a certain graph $G = (V, E)$.

In a centralized graph, a typical framework to construct a hierarchical clustering tree is first to compute the structural dissimilarity matrix (e.g., topological overlapping matrix [22]) between all vertices pairs of the graph. Then it utilizes a hierarchical clustering algorithm to build a tree based on the dissimilarity matrix. However, since the federated settings forbid direct update of edge information to the server, we need to design a new approach to compute the dissimilarity matrix.

In our design, the constructor first computes an ordered degree matrix for each vertex to represent its structural feature. Dynamic Time Warping (DTW) is introduced to compute the dissimilarity between each vertex pair due to the heterogeneous sizes of the matrices emerged in our setting. Then these dissimilarity values help to form a hierarchical clustering tree. The process is illustrated in Fig.3. The complete algorithm for HCT construction is displayed in Alg. 2 in Appendix B due to the space limit. We then present the design details as follows.

Ordered Degree matrix. We first define an ordered degree matrix for each vertex to compute the dissimilarity between two different nodes. The server randomly numbers the vertices in its system and virtually groups all the vertices into k bins. Each bin contains at least one vertex. Larger k preserves finer structural information while smaller k preserves more privacy. In our scenario, k is upper-bounded by $\ln |V|$ due to privacy concerns. The server then sends
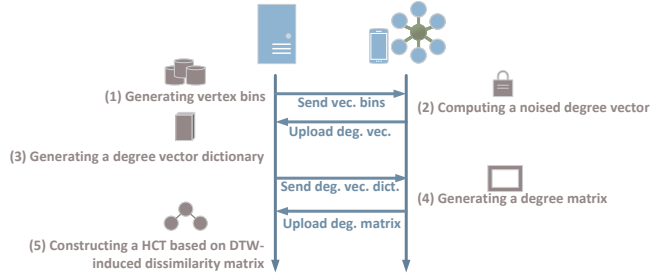
Figure 3: HCT constructor framework

the group plan to each device. Since every device stores a subgraph representing the link relationship between itself and neighboring devices, each device can count the number $c_i, i \in [1, 2, \cdots, k]$ of its neighbors in the i-th bin. c_i values are then noised with Laplacian noise $Lap(0, \frac{1}{\epsilon})$ to guarantee $\epsilon-$ differential privacy, forming a vector $\boldsymbol{c_v} = (c'_1, c'_2, \cdots, c'_k) \in \mathbb{R}^k$ for a vertex $v \in V$. It is worth mentioning that elements in $\boldsymbol{c_v}$ can be negative real numbers due to the Laplacian noise. Moreover, the sum of vector elements $\sum_{i=1}^{k} c'_i$ is the noised degree value of vertex v. 3 These vectors $\boldsymbol{c_{v_1}}, \boldsymbol{c_{v_2}}, \cdots, \boldsymbol{c_{v_n}}$ are sent to the server and the server advertises these vectors to all the devices. Each device corresponds to a vertex v with degree $deg(v)$ in the graph. Denote the neighbors of a node v as $u_1, u_2, \cdots, u_{deg(v)}$. Each device organize its neighboring vertices into an ordered list $N(v)$, where $N(v) = \left[u_{(1)}, u_{(2)}, \cdots, u_{(deg(v))}\right]$, in an ascending order based on the estimated degrees of its neighboring vertices. In this way, each device can form an ordered degree matrix $\boldsymbol{M_v}$ (Eq.5) of size $|N(v)| \times k$ where each row is the degree vector of the corresponding neighboring vertex.

$$M_v = \begin{bmatrix} c_{u_1} \\ c_{u_2} \\ \vdots \\ c_{u_{|N(v)|}} \end{bmatrix} \tag{5}$$

This ordered degree matrix represents 2-hop structural information for each vertex. Metrics from different devices have different numbers of rows because they have different numbers of neighbors. Each device uploads this ordered matrix to the server to calculate the dissimilarity matrix.

Dissimilarity matrix. Given the structure-embedded ordered degree matrix from the previous step, the server constructs a dissimilarity matrix for the whole graph. A dissimilarity matrix in a graph of size $|V| \times |V|$ quantifies dissimilarity for each node pair. Since the ordered degree matrix is expressive of the structural information of a vertex, we use the distance between two ordered degree matrices to determine the dissimilarity matrix. However, the metrics of each device can have unequal numbers of rows because they have different numbers of neighbors. Classical l1-norm of the subtraction of two metrics no longer applies. Therefore, we adopt Dynamic Time Warping (DTW)[2] which computes the distance between two sequences of unequal length by deploying the dynamic programming algorithm. The algorithm can be further extended to compute distances of arrays of different dimensions.

Let us suppose that two matrices $\boldsymbol{M_u}$ and $\boldsymbol{M_v}$ have x and y rows where $x \geq y$ correspondingly. As mentioned before, the first matrix represents neighboring vertices $N(u)$ while the second matrix represents neighboring vertices $N(v)$. Each row vector can be seen as a feature vector of a vertex. The server uses l1-norm to compute the dissimilarity between two row vectors, which aims to pair each vertex in $N(u) = [u_1, u_2, \cdots, u_x]$ with exactly one vertex in $N(v) = [v_1, v_2, \cdots, v_y]$ with smallest dissimilarity. This can be achieved using dynamic programming formulated in Eq. 6

$$\begin{aligned} cost(i, j) = \ &\min\{cost(i-1, j), cost(i, j-1), cost(i-1, j-1)\} \\ &+ ||c_{u_i} - c_{v_j}||_1, \forall i \in [1, x], j \in [1, y] \end{aligned} \tag{6}$$

where cost is a function measuring how well the former i vertices in $N(u)$ are paired with the former j vertices in $N(v)$.

$$\text{dissim}(u, v) = cost(x, y) \tag{7}$$

is the dissimilarity value for two vertices u and v. The dissimilarity value is noised due to privacy restrictions but it is proved to have upper-bounded information loss according to the following Theorem 4.1. Proofs of our theorems are presented in the appendix.

THEOREM 4.1. *The expectation* $\mathbb{E}[dissim(u, v) - dissim'(u, v)]$ *where* $dissim'(u, v)$ *is the original dissimilarity value of vertices* u *and* v *is bounded by*

$$\mathbb{E}[dissim(u, v) - dissim'(u, v)] \leq \frac{3k(\max_{v \in V} |N(v)|)^2}{2\epsilon}. \tag{8}$$

In practice, the left-hand side in Eq.8 is dependent upon the degrees of two vertices.

The server computes the dissimilarity for every vertex pair in this way and gets a dissimilarity matrix. Then it generates an HCT using a hierarchical algorithm based on this dissimilarity matrix. Considering the large scale of practical networks, we use hybrid hierarchical clustering [23] to construct the HCT. Other compatible clustering algorithms can also be used in our framework.

4.3 Random Walk Generator

Based on the constructed HCT, we design a random walk generator to generate random walk of limited length l starting with a specific vertex. The server sends each vertex a tuple (l, ϵ, p) to define and start a random walk. Value ϵ is the noise parameter to encode vertex with quantifiable privacy level. Value p is the probability of whether to use a 2-hop neighbor predictor, acting like a knob on controlling communication cost. As long as receiving the tuple, the corresponding device randomly selects a vertex from its neighbors. It utilizes the random walk sequence encoder to encode its vertex index and appends the encoder result to the tail of the walk sequence. Then it decides whether to trigger the 2-hop neighbor predictor based on the probability p. The length l is assumed to be greater than 2 to capture enough graph structures. If the predictor is triggered, it encodes the selected 2-hop neighbor vertex index and appends the encoded index to the output sequence as well. It sends the tuple $(l - 2, \epsilon, p)$ and the current walk sequence to the predicted 2-hop neighbor vertex to continue the walk. If the predictor is not triggered, it directly sends the tuple $(l - 1, \epsilon, p)$ and the current walk sequence ended to the selected neighbor vertex. The random

walk stops when the first number received in the tuple is 1. The corresponding device, also representing the ending vertex of the whole random walk, sends the encoded random walk sequence to the server. The complete algorithms are presented in Alg.1. The detailed procedures for random walk sequence encoder and two-hop neighbor predictor are presented as follows.

Random walk sequence encoder. The random walk sequence encoder takes advantage of exponential mechanism to noise the elements in the sequence to protect the privacy of edge information.

Exponential mechanism deals with contradiction between information loss and privacy protection through a score function $u : D \times \text{Range}(\mathcal{A}) \rightarrow \mathbb{R}$, which measures how informative the output of $\mathcal{A}$ is given $d \in D$. The mechanism assigns a sampling probability defined in Eq. 9 to the element in the range of $\mathcal{A}$ given different input $d \in D$ and samples an output based on the probability.

$$\Pr[r|d] = \frac{\exp(\epsilon u(d,r))\mu(r)}{\int \exp(\epsilon u(d,r))\mu(r)dr} \quad (9)$$

where $\mu(r)$ is the distribution of r in Range of $\mathcal{A}$.

The advantage of exponential mechanism is the ensuring of the utility of encoding result due to its dependence upon the score function. Another advantage is that it provides a way to encode discrete datasets. In our scenario, the random walk sequence should still imply the structural information of the graph despite alternation of the original sequence. In addition, the dataset V to encrypt is a discrete vertex set. Hence, we choose exponential mechanism to preserve $\epsilon-$ differential privacy.

We define the score function to be used in Eq. 9 as following:

$$u(v_1, v_2) = -\text{dissim}(v_1, v_2)|\text{leaves}(T[v_1 \vee v_2])| \quad (10)$$

where dissim function is defined in Eq.7, T is the subtree rooted by the least common ancestor of v_1 and v_2 nodes in HCT constructed before and leaves function returns all the leaves of a subtree. For two vertices with similar structural information, they tend to have a smaller dissimilarity value and are put more closely in the HCT. If two leaves are close to each other in a tree, $T[v_1 \vee v_2]$ have very few leaves. Hence, Eq.10 measures how similar two vertices are in terms of structure, which implies how informative the random walk sequence is if the original vertex is substituted with the other.

It is obvious that any vertex $v \in V$ follows a uniform distribution

$$Pr[v] = \frac{1}{|V|}. \quad (11)$$

Hence, after we plug Eq.10 and Eq.11 into Eq.9, the sampling probability becomes

$$\Pr[v_2|v_1] = \frac{\exp(-\epsilon \text{dissim}(v_1, v_2)|\text{leaves}(T[v_1 \vee v_2]))}{\sum_{v_2 \in V} \exp(-\epsilon \text{dissim}(v_1, v_2)|\text{leaves}(T[v_1 \vee v_2]))} \quad (12)$$

Using Eq.12, we encode the random walk sequence by sampling one vertex given the original index.

Two-hop neighbor predictor. To decrease inter-device communication cost, we design a two-hop neighbor predictor to directly build a bridge from the ith element in the sequence to the $(i + 2)$th element, saving the communication with the $(i + 1)$th element. Our predictor takes advantage of HCT and the ordered degree matrix

---

Algorithm 1 Random Walk Generator

Input: a HCT T; dissimilarity matrix; the length of the walk l; vertices V; noise parameter ϵ; triggering probability p; number of walks per vertex γ;

Server operation
1: Send T and dissimilarity matrix to all devices;
2: **for** $i = 1 \rightarrow \gamma$ **do**
3:     **for** each device **do**
4:         Server sends a tuple (l, ϵ, p) to the local device;
5:         Execute **Device operation**;
6:         Server receives the random walk sequence;
7:         Server inputs the random walk sequence to the SkipGram Model;
8:     **end for**
9: **end for**

Device operation
10: **if** the random walk sequence is empty **then**
11:     Initiate a new random walk sequence with an empty array
12: **end if**
13: Randomly select a vertex u from its neighbors;
14: Encode the vertex index according to Eq.12;
15: Append the encoded index to the end of the random walk sequence;
16: **if** $l = 1$ **then**
17:     Send the random walk sequence to the server;
18: **else if** $l > 2$ **then**
19:     Randomize a real number from $[0, 1)$;
20:     **if** the randomized number$< p$ **then**
21:         Choose the row c_u which corresponds to the degree vector of u from the degree vector dictionary;
22:         Initiate a vertex pool with an empty set;
23:         **for** bin i in bins **do**
24:             Add $c_u[i]$ vertices which are nearest to u in tree T to the vertex pool;
25:         **end for**
26:         Randomly pick a vertex u' from the pool;
27:         Encode the vertex index u according to Eq.12;
28:         Append the encoded index of u to the end of the random walk sequence;
29:         Device sends a tuple $(l - 2, \epsilon, p)$ and the current random walk sequence to device u';
30:         Device u' execute **Device operation**;
31:         **return**
32:     **end if**
33: **end if**
34: Device sends a tuple $(l - 1, \epsilon, p)$ and the current random walk sequence to device u;
35: Device u execute **Device operation**;

M_v obtained in the HCT constructor. The predictor is triggered given probability p customized by the server.

Each device is aware of the vectors $c_{u_1}, c_{u_2}, \cdots, c_{|N(v)|}$ with respect to its 1-hop neighbors. The device has randomly selected one neighbor denoted as u_i before. In HCT constructor, the degree

vector $c_{u_i} = (c'_1, c'_2, \cdots, c'_k)$ of u_i provides the number of its 2-hop neighbor connected to u_i in k different bins. Since HCT hierarchically clusters vertices according to their linkage information, for bin j, the predictor selects c'_j vertices that are closest to the u_i in HCT. These selected vertices form a pool. Then a vertex is uniformly drawn from this pool as the predicted 2-hop neighbor of v. In this way, the device can directly communicate with the device with the predicted 2-hop neighbor index. This reduces the times of communication by saving the contact involving the 1-hop neighbor u_i. We formally analyze the saved communication cost caused by our predictor and present the theorem as follows.

THEOREM 4.2. *For a $l-$long random walk sequence, it is expected that 2-hop neighbor predictor will reduce $|V|\gamma\mathcal{A}$ device to device communication given the triggering probability p and the number of walks per vertex γ where*

$$\mathcal{A} = l - 1 - \left[\frac{l-2}{1+p} + (2 - 2p - \frac{1}{1+p}) \frac{1 - (-p)^{l-2}}{1+p} \right] \quad (13)$$

As long as a random walk sequence is generated, the server computes the probability of co-occurrence for any vertex pair in this sequence, which provides a measurement of the similarity of two vertices in a graph. A Skipgram model is used to update node embedding based on the computed probability such that the distance between node embedding vectors approximates the corresponding vertex similarity.

5 EXPERIMENTS

In this section, we present the empirical results and analysis of our proposed algorithm. We first examine how FedWalk performs compared with the existing baseline in traditional node embedding tasks. Then we discuss the influence of key hyper-parameters upon the model performance.

5.1 Experiment Setup

The performance of FedWalk is measured through multi-label classification tasks. First, we deploy FedWalk to embed the vertices. Then, we randomly split the two datasets into a training dataset and a testing dataset based on a proportion T_R. T_R is the proportion of training data in full data. We set the vertex embedding vectors of the training data as features x and the labels of the training data as targets y and feed the (x, y) pairs into a one-vs-rest logistic regression. y is the indicator vector of the multiple labels for one vertex. Last we test the regression model using testing data.

We evaluate the performance of FedWalk based on two social network datasets: Blogcatalog[24] and Flickr[24]. Specifically,

- Blogcatalog dataset is crawled from a blog posting website Blogcatalog[2] representing the online social relationship between bloggers. The vertex labels are the interest of the bloggers.
- Flickr dataset is crawled from an image sharing website Flickr[3]. The vertices are the users and the edges represent that users add others to their contact lists. The labels are the interest groups of the users.

[2]http://www.blogcatalog.com
[3]http://www.Flickr.com

Blogcatalog dataset has 10312 vertices, 333983 edges, and 39 labels; Flickr dataset has 80513 vertices, 5899882 edges, and 195 labels. Two graphs are of different sizes so that the scalability of our algorithms can be examined.

Since we are the first to study the node-level federated unsupervised embedding problems, and the existing federated works cannot be applied in our scenario, we compare our FedWalk with DeepWalk[20], a classical node embedding method in centralized settings. It regards the random walk sequences as the word sequences and deploys Skipgram model to embed the nodes with vectors. In our experiments, both methods embed vertices with walk length $l = 40$, window size $w = 10$, embedding dimension $d = 128$ and $\gamma = 80$ walks per vertex. For FedWalk, we set the number of bins $k = \lfloor \ln |V| \rfloor$, the noise parameter $\epsilon = 2$ and the probability of triggering 2-hop neighbor predictor $p = 0.2$. The experiments are done with an Intel Core i7-10700 CPU clocked at 2.90GHz, an 8GB-memory NVIDIA RTX 3070, and 16GB memory.

To quantify the performance of multi-label classification, we use macro-F1 and micro-F1 to measure its accuracy.

- Macro-F1 is the average F1 score of all labels.

$$Macro - F1 = \frac{\sum_{l \in L} F1(l)}{|L|} \quad (14)$$

- Micro-F1 is calculated by counting the overall true positives, false negatives and false negatives for all labels.

$$Micro - F1 = \frac{2PR}{P+R} \quad (15)$$

where $P = \frac{\sum_{l \in L} TP(l)}{\sum_{l \in L} TP(l) + FP(l)}$ and $R = \frac{\sum_{l \in L} TP(l)}{\sum_{l \in L} TP(l) + FN(l)}$.

5.2 Results

Blogcatalog. For this dataset, we evaluate the performance of FedWalk and DeepWalk by setting the training dataset ratio T_R from 0.1 to 0.6. The results are illustrated in Fig. 4. FedWalk performs close to DeepWalk does both in Micro-F1 score and in Macro-F1 score. As T_R increases, the gap between these two methods closes. When $T_R = 0.6$, FedWalk only loses 1.8% Micro-F1 score and 1.0% Macro-F1 score compared to the scores of DeepWalk.
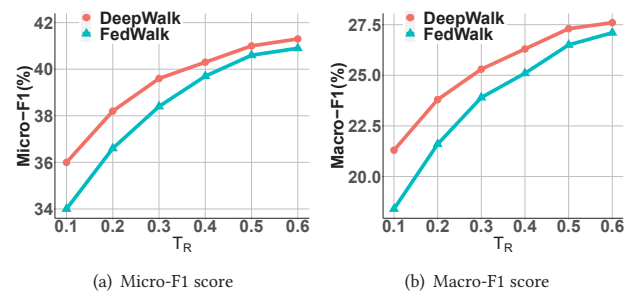

(a) Micro-F1 score   (b) Macro-F1 score

Figure 4: Blogcatalog dataset

Flickr. For Flickr dataset, we set the training dataset ratio T_R from 0.01 to 0.1 due to its large scale. The results are shown in Fig. 5. The results are consistent with those obtained from Blogcatalog dataset. Generally, DeepWalk has comparable performance to the

one of FedWalk. When T_R gets larger, the difference between the results of our method and the baseline method becomes smaller. When $T_R = 0.1$, DeepWalk only loses 1.5% Micro-F1 score and 4.4% Macro-F1 score.
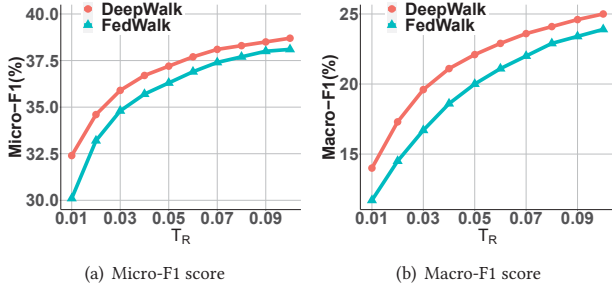

(a) Micro-F1 score

(b) Macro-F1 score

Figure 5: Flickr dataset

The classification results of these two methods show that our method FedWalk can achieve performance very close to the centralized method DeepWalk. This implies that our method has limited utility loss in the federated regime.
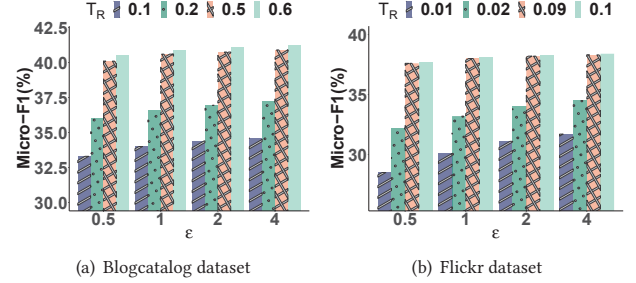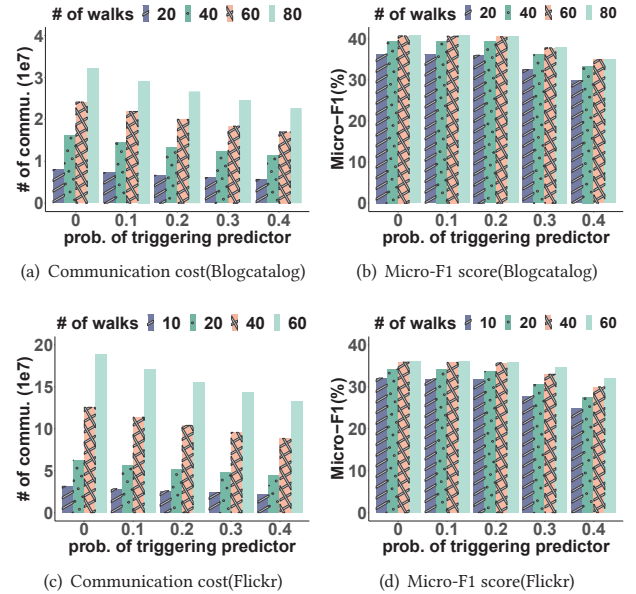
5.3 Sensitivity Analysis

Number of bins k. Figure 6 shows the effect of the number of bins k. Since Flickr Dataset has a lot more vertices than Blogcatalog Dataset, we test the performance upon different k values. For Blogcatalog dataset, increasing the number of bins from 3 to 9 results in 18.06%, 15.09%, 13.73% and 10.84% raise of Micro-F1 score respectively when $T_R = 0.1, 0.2, 0.5$ and 0.6. For Flickr dataset, increasing the number of bins from 5 to 11 results in 20.4%, 16.90%, 11.11% and 10.43% raise of Micro-F1 score respectively when $T_R = 0.01, 0.02, 0.09$ and 0.01. Both two bar plots show that larger k leads to better performance of FedWalk model given different training ratios T_R. A larger number of bins generates a more representative degree vector, thus leading to a better HCT. A good HCT provides an outcome with the high utility of good random walk sequence encoder and more accurate prediction of the two-hop neighbor predictor. This produces more representative vertex embedding.

(a) Blogcatalog dataset

(b) Flickr dataset
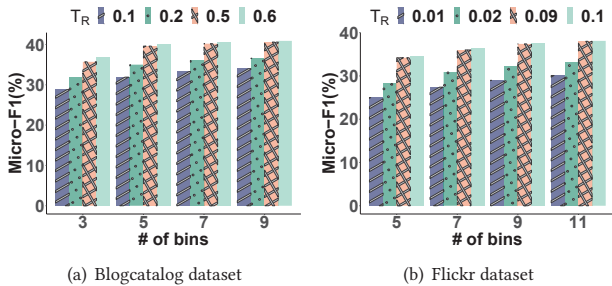
Figure 6: Sensitivity analysis of parameter k

Laplace noise parameter ϵ. Figure 7 shows the effect of Laplace noise parameter ϵ upon the performance of FedWalk model. For

Blogcatalog dataset, increasing ϵ from 0.5 to 4 results in 3.90%, 3.33%, 2.00% and 1.73% raise of Micro-F1 score respectively when $T_R = 0.1, 0.2, 0.5$ and 0.6. For Flickr dataset, the same increase of ϵ results in 11.23%, 7.14%, 1.86% and 1.85% raise of Micro-F1 score respectively when $T_R = 0.01, 0.02, 0.09$ and 0.01. Both two plots illustrate that the model with larger ϵ performs better for different training ratios T_R. ϵ represents the extent to which privacy is preserved. Small ϵ protects more privacy while losing more utility. In this way, the overall performance degrades as ϵ decreases.

(a) Blogcatalog dataset

(b) Flickr dataset

Figure 7: Sensitivity analysis of parameter ϵ

(a) Communication cost(Blogcatalog)

(b) Micro-F1 score(Blogcatalog)

(c) Communication cost(Flickr)

(d) Micro-F1 score(Flickr)

Figure 8: Sensitivity analysis of parameter p

Probability of triggering predictor p. We test the effect of the probability of triggering two-hop neighbor predictor p given different numbers of walks per vertex. Figure 8(a) and Figure 8(c) show the influence upon the number of inter-device communication. Increasing p from 0 to 0.4 results in approximately 23.80% decrease of the number of inter-device communication. For every 40-vertex-long random walk, it takes 39, 35.35, 32.30, 29.72 and 27.49 times of inter-device communication respectively when $p = 0, 0.1, 0.2, 0.3$

and 0.4. The figures show that larger p saves more inter-device communication as the predictor saves the communication cost by skipping the 1 hop neighbor and directly communicating to the predicted 2 hop neighbor.

Figure 8 displays the trade-off relationship between communication cost and accuracy of FedWalk model. Figure 8(b) and Figure 8(d) demonstrate that smaller p value leads to more representative vertex embedding. Increasing p from 0 to 0.4 results in 10.47%, 8.35%, 7.13% and 6.86% decrease of Micro-F1 score respectively when the numbers of walks per vertex are 20, 40, 60, and 80. For Flickr dataset, the same increase of p results in 13.08%, 10.64%, 8.08% and 3.88% decrease of Micro-F1 score respectively when the numbers of walks per vertex are 10, 20, 40 and 60. Especially when $p > 0.2$, the decrease of micro-F1 score is obvious.

6 CONCLUSION

In this paper, we propose the first unsupervised node embedding algorithm in the node-level federated graphs. Our framework, Fed-Walk, provides centralized competitive representation capability with reduced communication cost and enhanced data privacy. In FedWalk, we first propose a differentially private HCT constructor to capture structural information. We then propose a random walk generator which includes a sequence encoder to preserve privacy and a two-hop neighbor predictor to save communication cost. Fed-Walk is theoretically proved to have bounded dissimilarity value loss and quantifiable communication reduction while preserving $\epsilon-$ differential privacy. Extensive experiments on real-world datasets validate the effectiveness and efficiency of our design.

ACKNOWLEDGMENTS

REFERENCES

[1] Eugene Bagdasaryan, Peter Kairouz, Stefan Mellem, Adrià Gascón, Kallista Bonawitz, Deborah Estrin, and Marco Gruteser. 2021. Towards Sparse Federated Analytics: Location Heatmaps under Distributed Differential Privacy with Secure Aggregation. arXiv:2111.02356 [cs.CR]

[2] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series.. In the ACM SIGKDD Workshop. 359–370.

[3] Preston Bukaty. 2019. The California Consumer Privacy Act (CCPA): An implementation guide. IT Governance Publishing.

[4] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2018. Harp: Hierarchical representation learning for networks. In Proc. AAAI, Vol. 32.

[5] Cynthia Dwork. 2006. Differential privacy. In Proc. Colloq. on Automata, Lang., and Program. Springer, 1–12.

[6] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In Proc. Conf. on Theory of Cryptography. Springer, 265–284.

[7] European Commission. 2018. 2018 reform of EU data protection rules. https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf. Accessed: 2020-02-06.

[8] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. Knowledge-Based Systems 151 (2018), 78–94.

[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In Proc. ACM SIGKDD. 855–864.

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. Advances in Neural Inf. Processing Syst. 30 (2017).

[11] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In Proc. NeurIPS. 1025–1035.

[12] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In Proc. Conf. on Learn. Representations.

[13] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. IEEE Signal Process. Mag. 37, 3 (2020), 50–60.

[14] Yang Liu, Anbu Huang, Yun Luo, He Huang, Youzhi Liu, Yuanyuan Chen, Lican Feng, Tianjian Chen, Han Yu, and Qiang Yang. 2020. Fedvision: An online visual object detection platform powered by federated learning. In Proc. AAAI Conf. on Artif. Intell., Vol. 34. 13172–13179.

[15] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In Proc. Artif. Intell. Statist. 1273–1282.

[16] Frank McSherry and Kunal Talwar. 2007. Mechanism design via differential privacy. In Proc. IEEE Symp. on Found. of Comput. Sci. 94–103.

[17] Guangxu Mei, Ziyu Guo, Shijun Liu, and Li Pan. 2019. Sgnn: A graph neural network based federated learning approach by hiding structure. In Proc. IEEE Conf. on Big Data. 2560–2568.

[18] Chuizheng Meng, Sirisha Rambhatla, and Yan Liu. 2021. Cross-Node Federated Graph Neural Network for Spatio-Temporal Data Modeling. arXiv preprint arXiv:2106.05223 (2021).

[19] Hao Peng, Haoran Li, Yangqiu Song, Vincent Zheng, and Jianxin Li. 2021. Differentially Private Federated Knowledge Graphs Embedding. In Proc. ACM Conf. on Inf. and Knowl. Manage. 1416–1425.

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In Proc. ACM SIGKDD. 701–710.

[21] Daniel Ramage. 2020. Federated Analytics: Collaborative Data Science Without Data Collection. https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html Accessed: 2020-02-06.

[22] Erzsébet Ravasz, Anna Lisa Somera, Dale A Mongru, Zoltán N Oltvai, and A-L Barabási. 2002. Hierarchical organization of modularity in metabolic networks. Science 297, 5586 (2002), 1551–1555.

[23] Tianyang Sun, Chengchun Shu, Feng Li, Haiyan Yu, Lili Ma, and Yitong Fang. 2009. An efficient hierarchical clustering method for large datasets with map-reduce. In Proc. Conf. on Parallel and Distrib. Comput., Appl. and Technol. IEEE, 494–499.

[24] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In Proc. ACM SIGKDD. 817–826.

[25] Akira Ushioda. 1996. Hierarchical clustering of words and application to NLP tasks. In Proc. Workshop on Very Large Corpora.

[26] Binghui Wang, Ang Li, Hai Li, and Yiran Chen. 2020. Graphfl: A federated learning framework for semi-supervised node classification on graphs. arXiv preprint arXiv:2012.04187 (2020).

[27] Hongwei Wang, Fuzheng Zhang, Min Hou, Xing Xie, Minyi Guo, and Qi Liu. 2018. Shine: Signed heterogeneous information network embedding for sentiment link prediction. In Proc. ACM WSDM. 592–600.

[28] Zibo Wang, Yifei Zhu, Dan Wang, and Zhu Han. 2022. FedFPM: A Unified Federated Analytics Framework for Collaborative Frequent Pattern Mining. In Proc. IEEE Conf. on Comput. Commun.

[29] Stanley L Warner. 1965. Randomized response: A survey technique for eliminating evasive answer bias. J. Amer. Statistical Assoc. 60, 309 (1965), 63–69.

[30] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. 2021. Fedgnn: Federated graph neural network for privacy-preserving recommendation. arXiv preprint arXiv:2102.04925 (2021).

[31] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving google keyboard query suggestions. arXiv preprint arXiv:1812.02903 (2018).

[32] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. 2008. Wireless sensor network survey. Comput. Netw. 52, 12 (2008), 2292–2330.

[33] Changchuan Yin, Ying Chen, and Stephen S-T Yau. 2014. A measure of DNA sequence similarity by Fourier Transform with applications on hierarchical clustering. Journal of theoretical biology 359 (2014), 18–28.

[34] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In Proc. ACM SIGKDD. 2672–2681.

[35] Longfei Zheng, Jun Zhou, Chaochao Chen, Bingzhe Wu, Li Wang, and Benyu Zhang. 2021. ASFGNN: Automated separated-federated graph neural network. Peer-to-Peer Netw. and Appl. 14, 3 (2021), 1692–1704.

[36] Bin Zhou, Jian Pei, and WoShun Luk. 2008. A brief survey on anonymization techniques for privacy preserving publishing of social network data. Proc. ACM SIGKDD Explorations Newsletter 10, 2 (2008), 12–22.

[37] Jun Zhou, Chaochao Chen, Longfei Zheng, Huiwen Wu, Jia Wu, Xiaolin Zheng, Bingzhe Wu, Ziqi Liu, and Li Wang. 2020. Vertically Federated Graph Neural Network for Privacy-Preserving Node Classification. arXiv preprint arXiv:2005.11903 (2020).

[38] Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. 2020. Federated heavy hitters discovery with differential privacy. In Proc. AISTATS. 3837–3847.

A PROOFS

In this section, we provide proofs for the theorems mentioned in this paper.

A.1 Proof of Theorem 4.1

PROOF. Let the noised ordered degree matrix of vertex u be M_u and the original ordered degree matrix of vertex u be M'_u such that $M_{u,ij} = M'_{u,ij} + R_{u,ij}$. Recall that the dissimilarity value is calculated using Eq.6 and Eq.7. In this way,

$$\mathbb{E}[\text{dissim}(u,v) - \text{dissim}'(u,v)]$$
$$=\mathbb{E}\left[\sum ||c_{u_i} - c_{v_j}||_2 - \sum ||c'_{u_{i'}} - c'_{v_{j'}}||_2\right] \quad (16)$$

where (i,j) and (i',j') are some pairs selected from the $[1, |N(u)|] \times [1, |N(v)|]$ integer space according to Eq.6. Thus,

$$\mathbb{E}[\text{dissim}(u,v) - \text{dissim}'(u,v)]$$
$$=\mathbb{E}\left[\sum ||c_{u_i} - c_{v_j}||_1 - \sum ||c'_{u_{i'}} - c'_{v_{j'}}||_1\right]$$
$$\leq\mathbb{E}\left[\sum_{i=1}^{|N(u)|}\sum_{j=1}^{|N(v)|} ||c_{u_i} - c_{v_j}||_1 - \sum_{i'=1}^{|N(u)|}\sum_{j'=1}^{|N(v)|} ||c'_{u_{i'}} - c'_{v_{j'}}||_1\right]$$
$$=\mathbb{E}\left[\sum_{i=1}^{|N(u)|}\sum_{j=1}^{|N(v)|} ||c_{u_i} - c_{v_j}||_1 - ||c'_{u_i} - c'_{v_j}||_1\right]$$
$$=\mathbb{E}\left[\sum_{i=1}^{|N(u)|}\sum_{j=1}^{|N(v)|}\sum_{l=1}^{k} |M_{u,il} - M_{v,jl}| - |M'_{u,il} - M'_{v,jl}|\right]$$
$$\leq\mathbb{E}\left[\sum_{i=1}^{|N(u)|}\sum_{j=1}^{|N(v)|}\sum_{l=1}^{k} |R_{u,il} - R_{v,jl}|\right]$$
$$(17)$$

Since the noised item R follows an i.i.d. Laplace distribution, we have $f_{R_{u,il} - R_{v,jl}}(x) = \frac{\epsilon}{4}\left(e^{-\epsilon|x|} + |x|\epsilon e^{-\epsilon|x|}\right)$ which implies that $\mathbb{E}_{|x|}\left[f_{R_{u,il} - R_{v,jl}}(x)\right] = \frac{3}{2\epsilon}$. Hence, we bound the expectation

$$\mathbb{E}[\text{dissim}(u,v) - \text{dissim}'(u,v)]$$
$$\leq k|N(u)||N(v)|\frac{3}{2\epsilon}$$
$$\leq\frac{3k(\max_{v\in V} |N(v)|)^2}{2\epsilon} \quad (18)$$

□

A.2 Proof of Theorem 4.2

PROOF. Let the expected times of inter-device communication be $\mathbb{E}_l$ for one $l-$vertex-long random walk.

When $l = 1$, $\mathbb{E}_1 = 0$. When $l = 2$, $\mathbb{E}_2 = 1$. When $l \geq 3$, the following relationship holds

$$\mathbb{E}_l = p(\mathbb{E}_{l-2} + 1) + (1-p)(\mathbb{E}_{l-1} + 1) \quad (19)$$

which can be rewritten in the form of

$$\mathbb{E}_l - \mathbb{E}_{l-1} = 1 - p(\mathbb{E}_{l-1} - \mathbb{E}_{l-2}). \quad (20)$$

This gives us a recurrence relation of $\mathbb{E}_l - \mathbb{E}_{l-1}$, which implies that

$$\mathbb{E}_l = \frac{l-2}{1+p} + (2-2p - \frac{1}{1+p})\frac{1-(-p)^{l-2}}{1+p}. \quad (21)$$

When $p = 0$, it is expected to have $l - 1$ times of communication per random walk sequence. Thus, it is expected to save

$$l - 1 - \left\lceil \frac{l-2}{1+p} + (2-2p - \frac{1}{1+p})\frac{1-(-p)^{l-2}}{1+p}\right\rceil \quad (22)$$

times of communication per random walk sequence. In total, it is expected to save

$$|V|\gamma\mathcal{A} \quad (23)$$

where

$$\mathcal{A} = l - 1 - \left\lceil \frac{l-2}{1+p} + (2-2p - \frac{1}{1+p})\frac{1-(-p)^{l-2}}{1+p}\right\rceil \quad (24)$$

times of inter-device communication. □

B PSEDUO CODE

B.1 Pseduo code of HCT constructor

Algorithm 2 HCT constructor

Input: noise parameter ϵ, number of bins k, vertices V;
Output: a hierarchical clustering tree;
    Server operation
1: Group V into k bins randomly;
2: **for** each device **do**
3:     Server sends the group plan to the local device;
4:     Execute **Device operation 1**;
5:     Server receives the noised vector c_v;
6: **end for**
7: Collect all the vectors to a dictionary;
8: **for** each device **do**
9:     Server sends the dictionary to the local device;
10:     Execute **Device operation 2**;
11:     Server receives the degree matrix;
12: **end for**
13: Compute a dissimilarity matrix using DTW;
14: Construct an HCT based on the dissimilarity matrix using a hierarchical clustering algorithm;

    Device operation 1
15: Count the number of its neighbors belonging to each bins and get a k-dim vector;
16: Add noise $Lap(0, \frac{1}{\epsilon})$ to each element of c_v;
17: Send the noised vector c_v to the server;

    Device operation 2
18: Form its ordered degree matrix M_v according to Eq. 5;
19: Send the degree matrix to the server;