Data Sampling using Locality Sensitive Hashing for Large Scale Graph Learning

Sarath Shekkizhar*
University of Southern California
shekkizh@usc.edu

Neslihan Bulut
Google, Mountain View
neslihanbulut@google.com

Mohamed Farghal
Google, Munich
farghal@google.com

Sasan Tavakkol
Google, New York
tavakkol@google.com

MohammadHossein Bateni
Google, New York
bateni@google.com

Animesh Nandi
Google, Sunnyvale
animeshnandi@google.com

ABSTRACT

An important step in graph-based data analysis and processing is the construction of similarity graphs. Recent works, such as [7, 23], have focused on the semi-supervised setting to learn an optimal similarity function for constructing a task-optimal graph. However, in many scenarios with billions of data points and trillions of potential edges, the run-time and computational requirements for training the similarity model make these approaches impractical. In this work, we consider data sampling as a means to overcome this issue. Unlike typical sampling use-cases which only seek diversity, the similarity-learning for graph construction problem requires data samples that are both diverse and representative of highly similar data points. We present an efficient sampling approach by taking an adaptive partition view of *locality sensitive hashing*. Theoretically, we show that, though the samples obtained are correlated with sampling probabilities that do not sum to one, the training loss estimated for learning the graph similarity model using our approach is unbiased with a smaller variance compared to random sampling. Experiments on public datasets demonstrate the superior generalization of similarity models learned via our sampling. In a real large-scale industrial abuse-detection example, we observe $\approx 10\times$ increase in identifying abusive items while having lower false positive rate compared to the baseline.

KEYWORDS

Sampling, Graph learning, Locality sensitive hashing

ACM Reference Format:
Sarath Shekkizhar, Neslihan Bulut, Mohamed Farghal, Sasan Tavakkol, MohammadHossein Bateni, and Animesh Nandi. 2023. Data Sampling using Locality Sensitive Hashing for Large Scale Graph Learning. In *Proceedings of 19th International Workshop on Mining and Learning with Graphs (KDD Workshop '23).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

*Work done during internship at Google.

KDD Workshop '23, Aug 06–10, 2023, Long Beach, CA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Graphs provide a powerful abstraction for high-dimensional data with numerous applications in data mining [30], machine learning [8], and modern signal processing [33]. Given a collection of data points, a fundamental step in such graph-based methods is the *construction or learning* of the similarity graph [37, 38]. Here, each data point is represented as a node in the graph with weighted edges (denoting degree of similarity) connecting nodes in the graph. Despite its significance in real-world applications, especially with multi-modal data, the choice of similarity is often ad hoc [15, 31, 35].

To solve this problem, recently, Halcrow et al. [23] proposed Grale, a two-step graph learning procedure that works in a semi-supervised setting: (1) *Training* learns a similarity (two-tower deep neural network) model using the available data labels; and (2) *Construction* builds an appropriate graph on the entire data set using the similarity model obtained from previous step. Note that, for this approach to be of practical use in large-scale settings, the training and construction process needs to be efficient and not form a bottleneck in the larger execution pipeline (e.g., graph clustering [4, 16], embedding [22, 34], or label propagation [28, 40, 44]).

Grale and its extensions [7, 36] propose the use of data structures from nearest-neighbor search literature, namely locality sensitive hashing (LSH) [17, 18, 21], to limit the similarity model training and graph construction to relevant pairs of data points. However, in cases with billions of data points, one is often left with trillions of edges for training even after LSH. Consequently, the training step in Grale involves huge compute requiring several days of training, which hinders its usability. With increasing data sizes it is evident that training similarity models with only LSH is insufficient. Further, in situations where one does not have access to compute resources or time, it is imperative to design algorithms that can provide a summarized view of data, e.g., by sampling a small subset of the input data, for efficient learning of the similarity models. To clarify, we do not claim that LSH cannot scale. Rather, we make the case that relying only on LSH for graph learning in practice is not scalable: LSH generates training edges that are based on all-pairs comparison within each hash bucket which quickly adds up for large datasets. Theoretically, if the LSH parameters are chosen optimally, the size of each hash bucket would be small and this problem in graph learning will not exist. However, in practice, it is often difficult to find such an ideal parameter setting, making it necessary to implement additional techniques to improve scalability.

The use of existing sampling approaches [3, 41, 42] for training similarity models is hindered by two crucial factors. Firstly, a typical

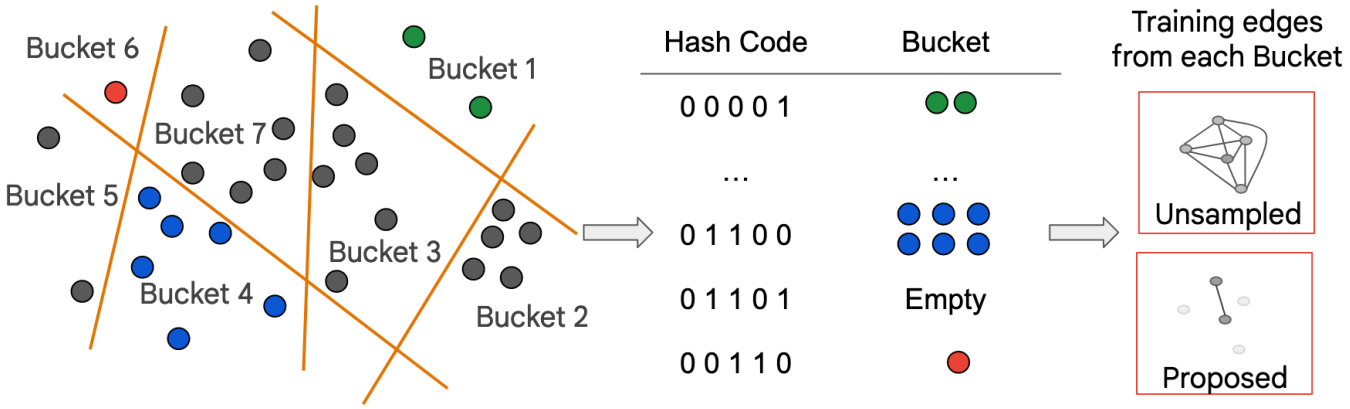

Figure 1: Grale Training and Sampling Problem. The left figure presents an example of LSH where each data point is hashed with a bit indicating the half space corresponding to each line that contain the data point. The data is then gathered in a table where pointers to data sharing the same code (bucket) are saved. Grale generates training edges between pairs of data point in a bucket for learning the similarity function for graph construction. In this work, we view the buckets as an adaptive partition of the input data. Our proposed sampling filters obtained buckets (e.g. buckets with at least 2 labeled examples) and allocates given sample budget uniformly across the filtered buckets.

sampling approach aims to select samples such that similar points are not sampled together [1, 2]. However, the samples in the graph learning setting need to be representative of the data distribution while *also capturing highly similar points* (edges) for obtaining a good graph. Secondly, the memory usage and computation of previous approaches scale poorly with the size of the sample set. For example, an experimental setting presented in this paper consists of a dataset with O(Billion) data points and sample set of size O(Million)[1]. In such scenarios, one resorts to randomly partitioning the dataset and sampling within each partition [19, 25]. Note that one can use a random sample of the dataset [32] as an efficient alternative. However, naive random selection and partition can discard important information and are often inadequate.

In this work, we consider the problem of sampling for semi-supervised learning of large-scale graphs. In this setting, we are given a dataset $X = \{x_1, x_2, \ldots x_N\}$ and an *oracle function* that gives a binary valued edge ground truth for a small fraction of data point pairs. In practice, one defines the oracle function based on labels in a subset of the data points. For example, consider a dataset which has binary labels of *spam* and *non-spam*. To construct a graph that captures *spam*, an appropriate oracle function would be one that assigns a value of one to pairs of data points with the same label of *spam*, and assigns a value of `zero` to all other data point pairs (one *spam* and one *non-spam* label, or both *non-spam* labels). Thus, the edges capture relationships between data points that are *spam* i.e., the graph is a network of *spam* nodes.

The goal of sampling is to obtain a sample set S that will be used to train a similarity model for constructing the entire graph. Additionally, we want the sampling to be (1) representative of the pairwise similarities in the input data, and (2) efficient, where the sampling process makes one pass over the data and does not incur severe computational overhead. It is important to emphasize that the sampling is only for learning the similarity model (*Training* step of Grale), and does not affect the subsequent step (*Construction*) for obtaining the graph using the learned similarity model on the entire dataset.

Our work differs from active learning in semi-supervised setting [20, 27, 43] on two accounts. Firstly, active learning approaches often assume a graph representation of the dataset exists. On the contrary, our work is aimed at sampling for the graph construction or link prediction. Secondly, the scale of our problem is much larger compared to that considered in typical active learning settings. We believe our use of LSH buckets to obtain an approximate graph structure for sampling presents a novel direction for sampling and can lead to new scalable ideas in related graph approaches. However, the immediate use of LSH in conjunction with previous sampling approaches is made fragile or inefficient due to the fact that there are no edge weights for data pairs in the LSH buckets. We leave this as an open problem for future research.

1.1 Contribution

We present a simple and efficient sampling framework for graph learning in the semi-supervised setting. The proposed approach views LSH as an adaptive, possibly overlapping, partition of input data points via hash buckets wherein similar points are grouped together. One then samples a few elements from *selected* LSH buckets (e.g., buckets with at least two labeled data point). The pairwise comparisons for training the similarity model are subsequently obtained using the LSH buckets and the sampled data. Thus, the approach improves upon the existing LSH-based graph-learning pipeline with minimal computational overhead. Figure 1 presents the high-level abstraction of the our sampling approach.

Here, we emphasize that while the majority of advances in LSH-based approaches focus on nearest-neighbor queries [18], the use of LSH for graph learning (where query set equals data set) allows for the development of unique techniques and optimizations specific

[1]We use O notation to denote the scale of the data problem, i.e., O(Billion) corresponds to a few tens or hundred billion data points.

to graphs. Conceptually, Carey et al. [7] pursue a similar direction as ours, where the number of comparisons for graph learning is minimized by selecting *leaders* in each bucket and constructing two-hop spanners instead of cliques in each LSH bucket.

2 BACKGROUND

2.1 Locality sensitive hashing

Locality sensitive hashing (LSH) [17, 18, 21] is a popular, sub-linear-time data structure for approximate nearest-neighbor search and graph construction. Briefly put, LSH consists of a family of hash functions $\mathcal{H}$ that map two points to the same *hash bucket* or value when the two points are similar (each $h \in \mathcal{H}$ is a function mapping the data points into non-negative integers). A hash collision occurs when two points have the same hash value, namely, $h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j)$. A typical LSH function $\mathcal{H}$ has collision probability that is monotonically increasing with the similarity of the data points,

$$p_{ij} = \Pr_{h \in \mathcal{H}} \left[h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j) \right] \propto \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{1}$$

where $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ corresponds to the measured similarity between points i and j.

Definition 2.1 ($(\lambda_1, \lambda_2, \delta, \rho)$-sensitive LSH Family)**.** Given similarity values $\lambda_1 < \lambda_2$ and a parameter $\rho \in [0, 1]$, a family $\mathcal{H}$ of functions is said to be $(\lambda_1, \lambda_2, \delta, \rho)$-sensitive if for every h in $\mathcal{H}$ the following holds for the dataset $\boldsymbol{X}$ of size N: if $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) > \lambda_2$ then $p_{ij} \geq \rho$ and if $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) < \lambda_1$ then $p_{ij} \leq \delta$.

The precise form of δ, ρ is defined by the choice of the LSH family used to build the hash tables. Given a family $\mathcal{H}$, one can amplify the sensitivity ρ using a combination of *AND* and *OR* constructions [26, 30] as follows. An *AND* family of hash functions is obtained by concatenating the hash value of r functions drawn from $\mathcal{H}$. The obtained functions are $(\lambda_1, \lambda_2, \delta^r, \rho^r)$-sensitive, i.e., two points with collision probability originally p_{ij} will belong to the same bucket with probability p_{ij}^r. Note that the hash table obtained with this construction partitions each data point *uniquely* into different hash buckets. In contrast, an *OR* construction turns a $(\lambda_1, \lambda_2, \delta, \rho)$-sensitive hash family into a $(\lambda_1, \lambda_2, 1 - (1 - \delta)^b, 1 - (1 - \rho)^b)$. Here, each data point is hashed into b hash tables and two points are considered neighbors if they belong to the same bucket in at least one of the b hash tables. This construction can be viewed as partitioning the input space into *overlapping* hash buckets with each data point assigned to b buckets.

In this work, we consider the LSH construction in [10, 23, 30] involving an *AND-OR* cascade, i.e., each data point is associated to b hash buckets with each hash value obtained using a concatenation of r hash functions. This construction reduces (1) the number of false positives because only valid neighbors are likely to match in all r hashes and (2) the number of false negatives by increasing the number of potential buckets (b) that could hold a valid neighbor. The resulting LSH is $(\lambda_1, \lambda_2, 1 - (1 - \delta^r)^b, 1 - (1 - \rho^r)^b)$-sensitive.

In our problem of graph learning, we will use a simplified Definition 2.1, namely, $(\lambda, \lambda, 0, 1 - (1 - \rho^r)^b)$ where we assume the false positive rate is tuned to be negligible, i.e., $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) < \lambda$ then $p_{ij} \approx 0$. This definition is suitable for our setting where we are interested in the graph edges with similarity greater than a threshold λ.

2.2 Locality sensitive sampling

LSH has traditionally been used to find similar data points or items based on a given query. However, it has more recently been used as a fast and adaptive sampling method in several applications. Charikar and Siminelakis [9] first utilized this view of LSH for hash-based kernel-density estimation; Spring and Shrivastava [39] leverage this idea for adaptive sparsification of neural networks; Chen et al. [10] accelerate stochastic gradient descent by selecting examples with LSH sampler; Coleman et al. [13] make use of LSH for estimating the probability of observing a genomic sequence streams and select diverse samples.

The LSH sampling algorithm that is used in LSH-based methods involves two steps: (1) Pre-processing, where items are inserted into hash tables using an LSH function, and (2) Query based sampling, where a subset of similar items are retrieved from the hash bucket corresponding to the given query for estimation. The probability of an item being returned as a candidate from a bucket in a (b, r)-parameterized LSH function is equal to its collision probability $1 - (1 - p^r)^b$. It is important to note that the sampling probabilities of the items do not sum up to one, i.e. non-normalized, and the sample set is likely to be correlated.

3 LSH SAMPLING FOR GRAPH LEARNING

In this section, we theoretically analyze the learning of a similarity model for graph construction using edges generated via locality sensitive hashing as detailed in Figure 1. We then present the impact of sampling in the training loss used for learning the model where the samples are obtained using random and LSH sampling. Our results show that both sampling procedures lead to unbiased estimators of the training loss but the variance associated with the LSH sampling is smaller than that obtained using random sampling.

3.1 Graph Learning with LSH

Given are N data points $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$. An oracle similarity function provides for any two points $\boldsymbol{x}_i, \boldsymbol{x}_j$ the ground-truth value $y_{ij} \in \{0, 1\}$ for the edge between them. The problem of graph learning is concerned with finding a function $f(\boldsymbol{x}_i, \boldsymbol{x}_j)$ that best predicts y_{ij}. In a typical learning problem, the oracle similarity is accessible for only some unordered pairs, e.g., in cases where a small subset of data is labeled, one can consider the similarity functions $y_{ij} = 1$ if the two points i, j share the same label and 0 otherwise.

The loss function associated with training is defined as

$$\begin{aligned} L(f, \boldsymbol{X}) &= \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \big[y_{ij} \log f(\boldsymbol{x}_i, \boldsymbol{x}_j) + \\ &\qquad (1 - y_{ij}) \log(1 - f(\boldsymbol{x}_i, \boldsymbol{x}_j)) \big] \\ &= \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} L_{ij} \end{aligned} \tag{2}$$

where L_{ij} is the loss corresponding to the input pair $\boldsymbol{x}_i, \boldsymbol{x}_j$.

Assumption 1:

$$L_{ij} = 0 \quad \forall \boldsymbol{x}_i, \boldsymbol{x}_j \mid \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq \lambda, \tag{3}$$

where $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ corresponds to the measured similarity between the two points. In other words, we are only interested in learning a graph for those points with similarity at least λ, i.e.,

$$L(f, \boldsymbol{X}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \mathbb{I}(\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) > \lambda)\ L_{ij}, \tag{4}$$

where $\mathbb{I} : \{False, True\} \rightarrow \{0, 1\}$ is the indicator function. Instead of explicitly calculating the similarity $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for all pairs, we will rely on (b, r)-parameterized cascade of LSH function to find relevant pairs for obtaining the training loss, namely,

$$L_{\mathrm{H}}(f, \boldsymbol{X}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \mathbb{I}(h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j))\ L_{ij}, \tag{5}$$

where h is the r-concatenated hash function (*AND*), H the hash table obtained with b randomly drawn r-concatenated hash functions (*OR*) from the LSH family $\mathcal{H}$. It is important to note that the loss in equation (5) is dependent on the random hash table H. We will assume that the hash table H for a given dataset is given and fixed.

Assumption 2: The number of LSH buckets ($|H|$) formed is smaller than b (number of duplicates obtained via *OR* construction) $\times\ M$ (size of sample set), namely,

$$|H| < bM. \tag{6}$$

This assumption relates the sample budget with the number of buckets formed, i.e., we assume that all LSH buckets are represented in the sampling. In our analysis, this allows us to ignore the case where one needs to select amongst the buckets formed for sampling.

Let us rewrite the loss function as

$$\begin{aligned} L_{\mathrm{H}}(f, \boldsymbol{X}) &= \sum_{i=1}^{N} \left(\frac{1}{N} \sum_{j=1}^{N} \mathbb{I}(h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j))\ L_{ij} \right) \\ &= \sum_{i=1}^{N} \mathrm{Z}^i \end{aligned} \tag{7}$$

where Z^i corresponds to the loss estimate at each data point.

3.2 Graph Learning with LSH Sampling

Assume that we sample a subset of the data points (each drawn independently) to use with training. Let S denote the random variable associated with the sample set. Thus, the loss function for training the network can be rewritten as

$$L_{\mathrm{H,S}}(f, \boldsymbol{X}) = \sum_{i=1}^{N} \mathbb{I}(\boldsymbol{x}_i \in S)\ \mathrm{Z}_S^i \tag{8}$$

where $\mathrm{Z}_{\mathrm{S}}^i$ is the loss estimate at $\boldsymbol{x}_i$ using the sampled set S.

Let us first consider the impact of sampling on each Z_S^i:

$$\mathrm{Z}_{\mathrm{S}}^i = \frac{1}{M} \sum_{j=1}^{N} \mathbb{I}(\boldsymbol{x}_j \in \mathrm{S}) \mathbb{I}(h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j))\ L_{ij}.$$

For the random sampling case, the summation follows a generalized binomial distribution [12] with N trials and with the probability of success $p_i = \frac{M}{N}$ where M is the sampling budget. Using the Central Limit Theorem, the estimate $\mathrm{Z}_{\mathrm{S}}^i$ can be approximated as normal distribution with mean and variance as

$$\mathbb{E}[\mathrm{Z}_{\mathrm{S}}^i] = \mathrm{Z}^i, \tag{9}$$

$$\begin{aligned} \mathrm{Var}[\mathrm{Z}_{\mathrm{S}}^i] &= \frac{1}{M^2} \frac{M}{N} \left(1 - \frac{M}{N}\right) \sum_{j=1}^{N} \mathbb{I}(h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j))\ L_{ij}^2 \\ &= \frac{1}{M^2} \frac{M}{N} \left(1 - \frac{M}{N}\right) \sum_{j=1}^{N} p_{ij}\ L_{ij}^2, \end{aligned} \tag{10}$$

where p_{ij} is the LSH collision probability of the sample $\boldsymbol{x}_j$ with respect to data point $\boldsymbol{x}_i$.

Now, for the case of our proposed sampling where each constituent's success follows different probabilities p_{ij}, the summation will follow a Poisson Binomial distribution [11, 14] where each constituent will be sampled according to its collision probability. Thus, the estimate $\mathrm{Z}_{\mathrm{S}}^i$ can be approximated as normal with

$$\begin{aligned} \mathbb{E}[\mathrm{Z}_{\mathrm{S}}^i] &= \frac{1}{M} \sum_{j=1}^{N} \mathbb{E}[\mathbb{I}(\boldsymbol{x}_j \in \mathrm{S})] \mathbb{I}(h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j))\ L_{ij} \\ &= \mathrm{Z}^i \qquad \text{because } \mathbb{I}(h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j) | \boldsymbol{x}_j \in S) = 1, \end{aligned} \tag{11}$$

$$\begin{aligned} \mathrm{Var}[\mathrm{Z}_{\mathrm{S}}^i] &= \frac{1}{M^2} \sum_{j=1}^{N} p_{ij}(1 - p_{ij})\ L_{ij}^2 \\ &\leq \frac{(1 - \rho^r)^b}{M^2} \sum_{j=1}^{N} p_{ij}\ L_{ij}^2, \end{aligned} \tag{12}$$

where we make use of the sensitivity of the LSH function considered for our problem (see Section 2.1). Thus, the variance in the loss estimate is smaller than the random sampling case, equation (10), when

$$\begin{aligned} &(1 - \rho^r)^b < p_i(1 - p_i) \\ \iff \quad &\rho > \left(1 - [p_i(1 - p_i)]^{\frac{1}{b}}\right)^{\frac{1}{r}}, \end{aligned} \tag{13}$$

where $p_i = \frac{M}{N}$ is the random sampling probability of drawing a data point used for estimating the training loss at data point i.

Remark 3.1. Here is an example to illustrate the above. Consider $b = 20, r = 5$, i.e., we create 20 LSH tables each made up of 5-dimensional hash codes. For a similarity threshold of $\lambda = 0.7$ and sensitivity parameter $\rho = \lambda$, we have $1 - (1 - \rho^r)^b = 1 - 0.025 = 0.975$. Thus, the proposed sampling leads to reduction in variance for sample sizes as small as $M = 0.025 \times N$.

Remark 3.2. The above sample complexity at each data point i can be further improved by considering the ratio of M, N to be only those samples that have L_{ij} nonzero. With slight abuse of notation, this corresponds to

$$p_i = \frac{\mathrm{nnz}(M)}{\mathrm{nnz}(N)}, \tag{14}$$

where $\mathrm{nnz}(M)$ indicates the number of sampled data with non-zero L_{ij} and $\mathrm{nnz}(N)$ equivalently represents that of the input.

Now, we will consider the impact of sampling on the variance of the entire training loss defined in equation (8).

Each Z_S^i can be approximately modeled as a Gaussian random variable, but they are not independent or identical. This is because a data point in S can contribute to multiple Z_S^i. Consequently, the moments are data-dependent and can have different number of non-zero values contributing to the estimated loss in the case of random sampling. The proposed sampling approach fixes the number of samples at each estimate i by dividing budget equally across each bucket, i.e., $\frac{M}{|H|}$, ensuring samples with stronger similarity are sampled. This allocation makes the loss estimates at each i identical, assuming the average of the data-dependent terms (L_{ij}) are similar.

Consider the variance associated with the sum of M correlated Gaussian random variables Z_S^i.

$$\text{Var}\left[\sum_{i=1}^{M} Z_S^i\right] = \sum_{i=1}^{M} \text{Var}[Z_S^i] \quad + \sum_{i,j=1, i\neq j}^{M} \text{Cov}[Z_S^i Z_S^j]. \tag{15}$$

As shown the first term on the right is smaller for the proposed sampling approach compared to that of random sampling when equation (13) holds.

For the term involving covariance, note that the covariance between Z_S^i, Z_S^j is positive when $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are similar, i.e., larger $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and equivalently negative when the points are not similar. Thus, for the case of proposed LSH-based sampling, the sum corresponding to the covariance contains largely negative terms. This is because in proposed LSH-based sampling, if $\boldsymbol{x}_i$ is sampled then $\boldsymbol{x}_j$ has less chance of being sampled and vice versa. On the other hand, with random sampling the number of similar data points that are sampled can be arbitrarily high. In other words, we can expect the overall variance of the LSH-based sampling to be much lower than the random sampling approach.

3.3 Proposed: LSH Filtering + Sampling

We now describe our proposed sampling approach for training the similarity model to be used for learning a graph. Given a LSH function constructed for identifying relevant candidate pairs in a dataset, our sampling approach begins by bucketing or *sketching* points in the dataset using b randomly drawn LSH functions. We cap the size of each bucket, randomly subdividing any sketch bucket that is larger. This additional step helps avoid blow-up in candidate pairs generated and mitigate the impact of noisy hashing. For each bucket obtained, our approach then calculates bucket-level statistics to obtain a smaller relevant set of buckets for sampling. The filtering helps overcome practical difficulties associated with *Assumption 2* (equation (6)) in our theoretical analysis. The algorithm then proceeds by sampling a random data point within each filtered bucket, repeating the procedure if the data point was already sampled. The obtained sample set is then used with the sketching to generate candidate edges for training the similarity model. The training and graph inference then follow the same setup as in Grale [23]. By reusing the same LSH buckets used for the training edge generation in Grale, our sampling approach avoids additional computation or memory overhead. Thus, the sampling can be integrated into existing pipeline and derive any LSH improvements made for graph learning, such as the SortingLSH introduced in [7].

Algorithm 1: Grale with proposed sampling

```
Function Sample(H, X, y, condition, M):
    Filtered Buckets H_f = {}
    for key h, value v in H do
        if condition(h, v) then
            Append h, v pair to H_f
    end
    Sample set S = [ ]
    while Size of S < M do
        for key h, value v in H_f do
            if v empty then
                continue
            s = Random sample a point from v
            Remove s from v
            \\Repeat sampling if s already in S
            Append s to S
        end
    end
    return S
Function Sketching(X):
    Buckets H = {}
    H(r) = r-hash LSH family from H \\AND construction
    H(r, b) = b random h from H(r) \\OR construction
    for x_i in X do
        for h in H(r, b) do
            if h(x_i) not in H then
                H[h(x_i)] = []
            Append i to bucket H[h(x_i)]
        end
    end
    Subdivide buckets in H of size larger than K
    return H
Function Grale(X, y, condition, M):
    Buckets H = Sketching(X)
    S = Sample(H, X, y, condition, M)
    for bucket in H do
        for i, j ∈ S ∩ bucket do
            Emit training edge (x_i, x_j), I(y_i = y_j)
        end
    end
    Train similarity model using obtained edges
    return Grale similarity model
```

3.4 System design and implementation

The proposed sampling approach is implemented as part of the Grale [23] graph building system, which is based on Adaptive Massive Parallel Computation model [5] of distributed framework. Each computation in this framework is automatically distributed across thousands of worker machines.

The sampling integrated graph learning system involves the following steps: (1) generating LSH tables H using LSH functions from $\mathcal{H}$; (2) obtaining allocated number of samples from each hash

bucket as outlined in section 3 and Algorithm 1. The edge pairs (sampled data sharing a bucket) are then passed as input to Grale similarity training model; and (3) scoring *all* edge pairs of the entire dataset using the trained similarity model.

The LSH tables save only a reference to each data point instead of the entire feature set. This allows for efficient memory usage, as each data point is replicated b times to improve LSH sensitivity. The edge pairs for training and inference are generated via lookups in a distributed hash table. Here, the entire dataset is cached across multiple machines, requiring $O(N)$ RAM, resulting in fast feature lookups without any costly disk I/O. This setup—used in [23] to handle large datasets (billions of points, large feature vectors)—allows for the proposed sampling to be materialized efficiently with the filtering and sampling performed in an online fashion as and when each bucket is processed.

To handle hash buckets with a large number of data points, [23] imposed a limit on LSH bucket sizes. Large buckets are randomly divided into smaller sub-buckets prior to edge pair generation, which reduces the number of edges eligible for comparison and balances worst-case running time with edge recall. For sampling, this step leads to drawing more data points from the original hash bucket and is equivalent to allocating more budget to dense partitions, provided all sub-buckets are selected for sampling.

4 EXPERIMENTS

We evaluate the performance of the graph obtained using similarity models trained with random and proposed data-sampling approaches on two public datasets as well as a large-scale abuse-detection setting. In all experiments, the sample budget is set to 5% of the input data.

4.1 Public datasets

We run experiments on two public datasets, MNIST [29] and Amazon2M [6] (also known as OGBN-Products [24]). The MNIST dataset contains 60k images corresponding to 10 digits, each of which is vectorized and normalized into 784-dimensional float vectors. Amazon2M contains 2,449,029 data points, each of which has a 100-dimensional float vector and a set of strings that indicates co-purchase relationships. In addition, each data points is given a class label corresponding to 47-top level categories. The data points are equally distributed among the target labels in MNIST while have a highly imbalanced distribution in Amazon2M.

For the graph learning problem, we simulate a semi-supervised setting by limiting ourselves to 10% of the class labels in the dataset as input with oracle similarity function that assigns 1 if the nodes belong to the same class and 0 otherwise. The sketching parameters used to obtain the hash buckets is selected as in [7, 23] where we use cosine-similarity hashing functions (*SimHash*) for feature vectors and use Jaccard similarity hashing (*MinHash*) between sets of strings with weights. The sketching dimension r is set to 12 for both datasets and the number of sketches b is set to 10. Each sketch bucket is limited to size 100 by randomly splitting larger sketches into smaller sub-buckets.

We evaluate the graphs obtained using our approach and baselines in terms of edge weights obtained at different weight thresholds and hierarchical average-linkage clustering [4]. We measure

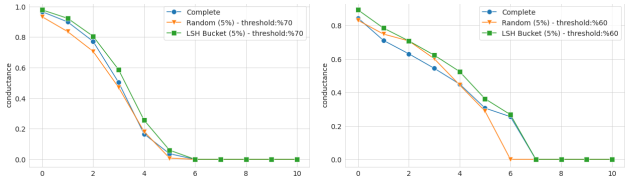

Figure 2: Conductance scores on clustering obtained with graphs learned on MNIST (Left) and Amazon2M (Right). Results are reported for the best performing edge weight model for each dataset and for 10-rounds of clustering.

clustering quality using *conductance*. The conductance of a graph $\mathcal{G}$ where each node has a binary class or cluster label is given by

$$C(S, S^c) = 1 - \sum_{i \in S, j \in S^c} \frac{w_{ij}}{\min w(S), w(S^c)}, \tag{16}$$

where w_ij is the weight of the edge between nodes i, j and $w(S) = \sum_{i \in S, j \in \mathcal{G}} w_{ij}$ is the sum of all edges incident on S (similarly S^c). Conductance score is in range $[0, 1]$ with higher conductance corresponding to better quality clusters.

We present our results in Figure 2. We observe that the proposed LSH bucket based sampling consistently outperform the random and complete baselines in terms of conductance evaluated at each round of hierarchical clustering. Note that the similarity model trained with the entire dataset has sub-optimal generalization as the number of *positive* edges (edges with oracle similarity 1) is far fewer than the number of *negative* edges (edges with oracle similarity 0). In contrast, our sampling approach naturally balances this imbalance in input for training the similarity model – the sampled data from each LSH buckets provide training edges that are representative and distributed across data space.

4.2 Case Study: Abuse detection

In this section we present a study on large scale abuse detection, a typical industry setting for Grale [23]. This scenario unlike the public datasets, involves features from multiple sources and modalities rather than a single dense feature space. We employ an *OR* style construction to combine the LSH outputs from each modality to obtain the sketch buckets for the graph learning. Note that in such setting our approach importance samples training points and corresponding edges that share buckets across multiple modalities. This is desirable as the similarity model learned using the edges obtained after sampling lead to better generalization, i.e. recall and precision of resulting graph edges, as seen in our results.

The input dataset to the abuse detection graph learning problem consists of O(Billion) points with labels available for a fraction of the dataset (< 10%). Each labeled data point belongs to the class of *abusive* items (due to policy violations) or the class of *safe* items (active, non-abusive, or verified). For the purpose of identifying unlabeled items that are abusive, we consider the learning objective in equation (2) and define the oracle similarity function as

$$y_{ij} = \begin{cases} 1 & \boldsymbol{x}_i \text{ and } \boldsymbol{x}_j \text{ are abusive} \\ 0 & \text{otherwise} \end{cases}.$$

Edges	Random	Proposed
Total	1	2
Abusive-Abusive	1	2
Safe-Safe	1	0.03

Table 1: Graph comparison on labeled items obtained using random and proposed sampling methods. We observe twice the number of edges, O(Billion), with the graph obtained using proposed sampling having better generalization i.e., the edges between abusive items, O(Billion), increased while edges between safe items, O(Thousand), decreased.

For our study, we consider a sample budget of 5% of the input dataset, as in the public dataset evaluation, equally split across abusive, safe, and unlabeled items for training the similarity function. Our approach is compared with the random sampling baseline previously used for abuse detection. We do not perform full dataset training as this requires massive compute resources and weeks of training. The graphs obtained using the trained similarity model is evaluated[2] in terms of (1) Precision and recall on edge connectivity on entire labeled set, and (2) Precision and recall in abuse detection using clustering [4] on obtained graphs. The clusters obtained are filtered for quality (cluster size, % abusive nodes in the clusters) before being used for flagging unlabeled items for abuse. Note that one typically does not have the true label for the unlabeled nodes. Thus, to evaluate the obtained clusters and methods we rely on abusive labels that were assigned to the items in the future and the false positive rate using the known safe items that were flagged in the cluster. In Table 2, we report the recall (known abusive nodes identified in cluster), expansion (nodes in the cluster that are not labeled abusive), future recall (expansion nodes that were flagged abusive in the following weeks), and known false positive rate (expansion nodes that have safe labels). Our sampling based model leads to better abuse detection where about 2.74× more abusive nodes are detected (recall) with about 2.4× higher future recall and 0.7× known false positive rate compared to the random baseline.

Sampling	Recall	Expansion	Future Recall	FP Rate
Random	1	1	1	1
Proposed	2.74	9.69	2.42	0.68

Table 2: Clustering performance comparison for abuse detection using random and proposed sampling approaches for training the similarity model. Note that recall, expansion, and future recall are in several O(Million) (Higher is better). The false positive (FP) rate captures the number of safe items in the identified clusters relative to the number of flagged items and is in the $O(1e^{-4})$ (Lower is better).

[2]We report relative improvements with respect to the random baseline in our comparison to avoid revealing any sensitive information related to the data.

5 CONCLUSION AND FUTURE WORK

As data sizes continue to grow, the need for reliably handling and scaling learning algorithms for large-scale datasets is imminent. In this paper, we tackle the problem of similarity learning for graph construction using data sampling. Our proposed approach, an efficient and distributed sampling strategy, leverages locality sensitive hashing as an adaptive partition of the input dataset to guide the sampling procedure and obtain both diverse and highly similar points. Our method significantly reduces the run time and memory requirements of the training procedure while leading to high quality graphs and downstream clustering outputs. As an example, in a large scale abuse detection dataset with billions of items, we demonstrate a 10-fold increase in identifying abusive patterns with a lower false positive rate than a random sampling baseline.

Moreover, the novel view of LSH buckets as data structures that can guide data sampling in graph learning presents interesting opportunities to revisit previous sampling approaches for scalability. As an extension of our proposed sampling, we plan to study a two-level LSH sampling where we consider the overlap of the buckets to better distribute the sampling budget. This extension is equivalent to a graph-set cover for identifying canonical samples in high-dimensional inputs, in particular those obtained from multiple modalities. This approach also mitigates the impact of the number of buckets formed with LSH on sampling, as in equation (6).

The high-level idea with the two-Level LSH sampling is as below.

- (First-level) We hash the features into LSH buckets, combining multiple weak similarity models using appropriate LSH functions for each *piecemeal* feature. This step is the same as that used in the sampling studied in this work (algorithm 1).
- (Second-level) We then construct a new feature-vector for each item in the dataset as the *set* of LSH-buckets that each piecemeal feature are hashed in the first-level. We then hash (MinHash) this set feature vector to determine data that have high Jaccard-Similarity for distributing the sample budget.

The second-level captures a LSH representation of items sharing similarity over first-level buckets which can help allocating more precise sample budgets in our proposed algorithm. We believe this approach can help overcome LSH outputs with very large hash buckets and maximizing the efficiency of our sampling procedure.

REFERENCES

[1] Sofiane Abbar, Sihem Amer-Yahia, Piotr Indyk, Sepideh Mahabadi, and Kasturi R Varadarajan. 2013. Diverse near neighbor problem. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*. 207–214.
[2] Pankaj K Agarwal, Sariel Har-Peled, Kasturi R Varadarajan, et al. 2005. Geometric approximation via coresets. *Combinatorial and computational geometry* 52, 1 (2005).
[3] Daniel Barbar'a, William DuMouchel, Christos Faloutsos, Peter J Haas, Joseph M Hellerstein, Yannis Ioannidis, HV Jagadish, Theodore Johnson, Raymond Ng, Viswanath Poosala, et al. 1997. The New Jersey data reduction report. In *IEEE Data Engineering Bulletin*. Citeseer.
[4] MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab Mirrokni. 2017. Affinity clustering: Hierarchical clustering at scale. *Advances in Neural Information Processing Systems* 30 (2017).
[5] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Łacki, Vahab Mirrokni, and Warren Schudy. 2021. Massively parallel computation via remote memory access. *ACM Transactions on Parallel Computing* 8, 3 (2021), 1–25.
[6] K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. 2016. The extreme classification repository: Multi-label datasets and code. http://manikvarma.org/downloads/XC/XMLRepository.html

[7] CJ Carey, Jonathan Halcrow, Rajesh Jayaram, Vahab Mirrokni, Warren Schudy, and Peilin Zhong. 2022. Stars: Tera-Scale Graph Building for Clustering and Learning. In *Advances in Neural Information Processing Systems*.

[8] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. 2022. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research* 23, 89 (2022), 1–64.

[9] Moses Charikar and Paris Siminelakis. 2017. Hashing-based-estimators for kernel density in high dimensions. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1032–1043.

[10] Beidi Chen, Yingchen Xu, and Anshumali Shrivastava. 2019. Fast and accurate stochastic gradient estimation. *Advances in Neural Information Processing Systems* 32 (2019).

[11] Sean X Chen and Jun S Liu. 1997. Statistical applications of the Poisson-binomial and conditional Bernoulli distributions. *Statistica Sinica* (1997), 875–892.

[12] Xiang-Hui Chen, Arthur P Dempster, and Jun S Liu. 1994. Weighted finite population sampling to maximize entropy. *Biometrika* 81, 3 (1994), 457–469.

[13] Benjamin Coleman, Benito Geordie, Li Chou, RA Leo Elworth, Todd Treangen, and Anshumali Shrivastava. 2022. One-Pass Diversified Sampling with Application to Terabyte-Scale Genomic Sequence Streams. In *International Conference on Machine Learning*. PMLR, 4202–4218.

[14] Constantinos Daskalakis, Ilias Diakonikolas, and Rocco A Servedio. 2012. Learning poisson binomial distributions. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. 709–728.

[15] C. A. R. De Sousa, S. O. Rezende, and G. Batista. 2013. Influence of graph construction on semi-supervised learning. Springer, Berlin, Heidelberg.

[16] Laxman Dhulipala, David Eisenstat, Jakub Lacki, Vahab Mirrokni, and Jessica Shi. 2021. Hierarchical agglomerative graph clustering in nearly-linear time. In *International Conference on Machine Learning*. PMLR, 2676–2686.

[17] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*. 577–586.

[18] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. 2019. Learning space partitions for nearest neighbor search. *arXiv preprint arXiv:1901.08544* (2019).

[19] Dan Feldman, Melanie Schmidt, and Christian Sohler. 2020. Turning big data into tiny data: Constant-size coresets for k-means, PCA, and projective clustering. *SIAM J. Comput.* 49, 3 (2020), 601–657.

[20] Akshay Gadde, Aamir Anis, and Antonio Ortega. 2014. Active semi-supervised learning using sampling theory for graph signals. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 492–501.

[21] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*, Vol. 99. 518–529.

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

[23] Jonathan Halcrow, Alexandru Mosoi, Sam Ruth, and Bryan Perozzi. 2020. Grale: Designing networks for graph learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2523–2532.

[24] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[25] Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S Mirrokni. 2014. Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 100–108.

[26] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[27] Ajinkya Jayawant and Antonio Ortega. 2022. Practical graph signal sampling with log-linear size scaling. *Signal Processing* 194 (2022), 108436.

[28] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[29] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[30] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2020. *Mining of massive data sets*. Cambridge university press.

[31] M. Maiker, U. von Luxburg, and M. Hein. 2009. Influence of graph construction on graph-based clustering measures. *Advances in Neural Information Processing Systems* (2009).

[32] Luca Martino, David Luengo, and Joaquín Míguez. 2018. *Independent random sampling methods*. Springer.

[33] Antonio Ortega. 2022. *Introduction to graph signal processing*. Cambridge University Press.

[34] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[35] Lishan Qiao, Limei Zhang, Songcan Chen, and Dinggang Shen. 2018. Data-driven graph construction and graph learning: A review. *Neurocomputing* 312 (2018), 336–351.

[36] Benedek Rozemberczki, Peter Englert, Amol Kapoor, Martin Blais, and Bryan Perozzi. 2021. Pathfinder discovery networks for neural message passing. In *Proceedings of the Web Conference 2021*. 2547–2558.

[37] Sarath Shekkizhar and Antonio Ortega. 2019. Neighborhood and Graph Constructions using Non-Negative Kernel Regression. *arXiv preprint arXiv:1910.09383* (2019).

[38] Sarath Shekkizhar and Antonio Ortega. 2020. Graph Construction from Data by Non-Negative Kernel Regression. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 3892–3896.

[39] Ryan Spring and Anshumali Shrivastava. 2017. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 445–454.

[40] Partha Talukdar and William Cohen. 2014. Scaling Graph-based Semi Supervised Learning to Large Number of Labels Using Count-Min Sketch. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 33)*, Samuel Kaski and Jukka Corander (Eds.). PMLR, 940–947.

[41] Steven K Thompson. 2012. *Sampling*. Vol. 755. John Wiley & Sons.

[42] Yves Tillé. 2006. *Sampling algorithms*. Springer.

[43] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

[44] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*. 912–919.