

Large Scale Distributed Semi-Supervised Learning Using Streaming Approximation

Sujith Ravi

Google Inc., Mountain View, CA, USA
sravi@google.com

Qiming Diao¹

Carnegie Mellon University, Pittsburgh, PA, USA
Singapore Mgt. University, Singapore
qiming.ustc@gmail.com

Abstract

Traditional graph-based semi-supervised learning (SSL) approaches are not suited for massive data and large label scenarios since they scale linearly with the number of edges $|E|$ and distinct labels m . To deal with the large label size problem, recent works propose sketch-based methods to approximate the label distribution per node thereby achieving a space reduction from $O(m)$ to $O(\log m)$, under certain conditions. In this paper, we present a novel streaming graph-based SSL approximation that effectively captures the sparsity of the label distribution and further reduces the space complexity per node to $O(1)$. We also provide a distributed version of the algorithm that scales well to large data sizes. Experiments on real-world datasets demonstrate that the new method achieves better performance than existing state-of-the-art algorithms with significant reduction in memory footprint. Finally, we propose a robust graph augmentation strategy using unsupervised deep learning architectures that yields further significant quality gains for SSL in natural language applications.

1 Introduction

Semi-supervised learning (SSL) methods use small amounts of labeled data along with large amounts of unlabeled data to train prediction systems. Such approaches have gained widespread usage in recent years

and have been rapidly supplanting supervised systems in many scenarios owing to the abundant amounts of unlabeled data available on the Web and other domains. Annotating and creating labeled training data for many predictions tasks is quite challenging because it is often an expensive and labor-intensive process. On the other hand, unlabeled data is readily available and can be leveraged by SSL approaches to improve the performance of supervised prediction systems.

There are several surveys that cover various SSL methods in the literature [25, 37, 8, 6]. The majority of SSL algorithms are computationally expensive; for example, transductive SVM [16]. Graph-based SSL algorithms [38, 17, 33, 4, 26, 30] are a subclass of SSL techniques that have received a lot of attention recently, as they scale much better to large problems and data sizes. These methods exploit the idea of constructing and smoothing a graph in which data (both labeled and unlabeled) is represented by nodes and edges link vertices that are related to each other. Edge weights are defined using a similarity function on node pairs and govern how strongly the labels of the nodes connected by the edge should agree. Graph-based methods based on label propagation [38, 29] work by using class label information associated with each labeled “seed” node, and propagating these labels over the graph in a principled, iterative manner. These methods often converge quickly and their time and space complexity scales linearly with the number of edges $|E|$ and number of labels m . Successful applications include a wide range of tasks in computer vision [36], information retrieval (IR) and social networks [34] and natural language processing (NLP); for example, class instance acquisition and relation prediction, to name a few [30, 27, 19].

Several classification and knowledge expansion type of problems involve a large number of labels in real-world scenarios. For instance, entity-relation classification over the widely used Freebase taxonomy requires learning over thousands of labels which can grow further by orders when extending to open-domain ex-

¹Work done during an internship at Google.

traction from the Web or social media; scenarios involving complex overlapping classes [7]; or fine-grained classification at large scale for natural language and computer vision applications [28, 13]. Unfortunately, existing graph-based SSL methods cannot deal with large m and $|E|$ sizes. Typically individual nodes are initialized with sparse label distributions, but they become dense in later iterations as they propagate through the graph. Talukdar and Cohen [28] recently proposed a method that seeks to overcome the label scale problem by using a Count-Min Sketch [10] to approximate labels and their scores for each node. This reduces the memory complexity to $O(\log m)$ from $O(m)$. They also report improved running times when using the sketch-based approach. However, in real-world applications, the number of *actual* labels k associated with each node is typically sparse even though the overall label space may be huge; i.e., $k \ll m$. Cleverly leveraging sparsity in such scenarios can yield huge benefits in terms of efficiency and scalability. While the sketching technique from [28] approximates the label space succinctly, it does not utilize the sparsity (a naturally occurring phenomenon in real data) to full benefit during learning.

Contributions: In this paper, we propose a new graph propagation algorithm for general purpose semi-supervised learning with applications for NLP and other areas. We show how the new algorithm can be run efficiently even when the label size m is huge. At its core, we use an approximation that effectively captures the sparsity of the label distribution and ensures the algorithm propagates the labels accurately. This reduces the space complexity per node from $O(m)$ to $O(k)$, where $k \ll m$ and a constant (say, 5 or 10 in practice), so $O(1)$ which scales better than previous methods. We show how to efficiently parallelize the algorithm by proposing a distributed version that scales well for large graph sizes. We also propose an efficient linear-time graph construction strategy that can effectively combine information from multiple signals which can vary between *sparse* or *dense* representations. In particular, we show that for graphs where nodes represent textual information (e.g., entity name or type), it is possible to robustly learn latent semantic embeddings associated with these nodes using only raw text and state-of-the-art deep learning techniques. Augmenting the original graph with such embeddings followed by graph SSL yields significant improvements in quality. We demonstrate the power of the new method by evaluating on different knowledge expansion tasks using existing benchmark datasets. Our results show that, when compared with existing state-of-the-art systems for these tasks, our method performs better in terms of space complexity and qualitative performance.

2 Graph-based Semi-Supervised Learning

Preliminary: The goal is to produce a soft assignment of labels to each node in a graph $G = (V, E, W)$, where V is the set of nodes, E the set of edges and W the edge weight matrix.² Every edge $(v, u) \notin E$ is assigned a weight $w_{vu} = 0$. Among the $|V| = n$ number of nodes, $|V_l| = n_l$ of them are labeled while $|V_u| = n_u$ are unlabeled. We use diagonal matrix S to record the seeds, in which $s_{vv} = 1$ if the node v is seed. L represents the output label set whose size $|L| = m$ can be large in the real world. \mathbf{Y} is a $n * m$ matrix which records the training label distribution for the seeds where $Y_{vl} = 0$ for $v \in V_u$, and $\hat{\mathbf{Y}}$ is an $n * m$ label distribution assignment matrix for all nodes. In general, our method is a graph-based semi-supervised learning algorithm, which learns $\hat{\mathbf{Y}}$ by propagating the information of \mathbf{Y} on graph G .

2.1 Graph SSL Optimization

We learn a label distribution $\hat{\mathbf{Y}}$ by minimizing the convex objective function:

$$\begin{aligned} \mathcal{C}(\hat{\mathbf{Y}}) = & \mu_1 \sum_{v \in V_l} s_{vv} \|\hat{\mathbf{Y}}_v - \mathbf{Y}_v\|_2^2 \\ & + \mu_2 \sum_{v \in V, u \in \mathcal{N}(v)} w_{vu} \|\hat{\mathbf{Y}}_v - \hat{\mathbf{Y}}_u\|^2 \\ & + \mu_3 \sum_{v \in V} \|\hat{\mathbf{Y}}_v - \mathbf{U}\|_2^2 \\ & s.t. \sum_{l=1}^L \hat{Y}_{vl} = 1, \forall v \end{aligned} \quad (1)$$

where $\mathcal{N}(v)$ is the (incoming) neighbor node set of the node v , and \mathbf{U} is the (uniform) prior distribution over all labels. The above objective function models that: 1) the label distribution should be close to the gold label assignment for all the seeds; 2) the label distribution of a pair of neighbors should be similar measured by their affinity score in the edge weight matrix; 3) the label distribution should be close to the prior \mathbf{U} , which is a uniform distribution. The setting of the hyperparameters μ_i will be discussed in Section 5.1.

The optimization criterion is inspired from [5] and similar to some existing approaches such as Adsorption [3] and MAD [29] but uses a slightly different objective function, notably the matrices have different constructions. In Section 5, we also compare our vanilla version against some of these baselines for completeness.

The objective function in Equation 1 permits an efficient iterative optimization technique that is repeated

²The graph G can be directed or undirected depending on the task. Following most existing works in the literature, we use undirected edges for E in our experiments.

until convergence. We utilize the Jacobi iterative algorithm which defines the approximate solution at the $(i + 1)th$ iteration, given the solution of the $(i)th$ iteration as follows:

$$\hat{Y}_{vl}^{(i)} = \frac{1}{M_{vl}}(\mu_1 s_{vv} Y_{vl} + \mu_2 \sum_{u \in \mathcal{N}(v)} w_{vu} \hat{Y}_{ul}^{(i-1)} + \mu_3 U_l) \quad (2)$$

$$M_{vl} = \mu_1 s_{vv} + \mu_2 \sum_{u \in \mathcal{N}(v)} w_{vu} + \mu_3$$

where i is the iteration index and $U_l = \frac{1}{m}$ which is the uniform distribution on label l . The iterative procedure starts with $\hat{Y}_{vl}^{(0)}$ which is initialized with seed label weight Y_{vl} if $v \in V_l$, else with uniform distribution $\frac{1}{m}$. In each iteration i , $\hat{Y}_{vl}^{(i)}$ aggregates the label distribution $\hat{Y}_{ul}^{(i-1)}$ at iteration $i - 1$ from all its neighbors $u \in \mathcal{N}(v)$. More details for deriving the update equation can be found in [5].

We use the name EXPANDER to refer to this vanilla method that optimizes Equation 1.

2.2 DIST-EXPANDER: Scaling To Large Data

In many applications, semi-supervised learning becomes challenging when the graphs become huge. To scale to really large data sizes, we propose DIST-EXPANDER, a distributed version of the algorithm that is directly suited towards parallelization across many machines. We turn to Pregel [20] and its open source version Giraph [2] as the underlying framework for our distributed algorithm. These systems follow a Bulk Synchronous Parallel (BSP) model of computation that proceeds in rounds. In every round, every machine does some local processing and then sends arbitrary messages to other machines. Semantically, we think of the communication graph as fixed, and in each round each node performs some local computation and then sends messages to its neighbors.

The specific systems like Pregel and Giraph build infrastructure that ensures that the overall system is fault tolerant, efficient, and fast. The programmer's job is simply to specify the code that each vertex will run at every round. Previously, some works have explored using MapReduce framework to scale to large graphs [31]. But unlike these methods, the Pregel-based model is far more efficient and better suited for graph algorithms that fit the iterative optimization scheme for SSL algorithms. Pregel keeps vertices and edges on the machine that performs computation, and uses network transfers only for messages. MapReduce, however, is essentially functional, so expressing a graph algorithm as a chained MapReduce requires passing the entire state of the graph from one stage to the next—in general requiring much more communication and associated serialization overhead which results in significant network cost (refer [20]

Algorithm 1 DIST-EXPANDER Algorithm

- 1: **Input:** A graph $G = (V, E, W)$, where $V = V_l \cup V_u$
 V_l = seed/labeled nodes, V_u = unlabeled nodes
 - 2: **Output:** A label distribution $\hat{Y}_v = \hat{Y}_{v1} \hat{Y}_{v2} \dots \hat{Y}_{vm}$ for every node $v \in V$ minimizing the overall objective function (1). Here, \hat{Y}_{vl} represents the weight of label l assigned to the node v .
 - 3: Let L be the set of all possible labels, $|L| = m$.
 - 4: Initialize \hat{Y}_{vl}^0 with seed label weights if $v \in V_l$, else $\frac{1}{m}$.
 - 5: (Graph Creation) Initialize each node v with its neighbors $\mathcal{N}(v) = \{u : (v, u) \in E\}$.
 - 6: Partition the graph into p disjoint partitions V_1, \dots, V_p , where $\bigcup_i V_i = V$.
 - 7: **for** $i = 1$ to max_iter **do**
 - 8: Process individual partitions V_p in parallel.
 - 9: **for** every node $v \in V_p$ **do**
 - 10: (Message Passing) Send previous label distribution \hat{Y}_v^{i-1} to all neighbors $u \in \mathcal{N}(v)$.
 - 11: (Label Update) Receive a message \mathcal{M}_u from its neighbor u with corresponding label weights \hat{Y}_u^{i-1} . Process each message $\mathcal{M}_1 \dots \mathcal{M}_{|\mathcal{N}(v)|}$ and update current label distribution \hat{Y}_v^i iteratively using Equation (2).
 - 12: **end for**
 - 13: **end for**
-

for a detailed comparison). In addition, the need to coordinate the steps of a chained MapReduce adds programming complexity that is avoided by DIST-EXPANDER iterations over rounds/steps. Furthermore, we use a version of Pregel that allows spilling to disk instead of storing the entire computation state in RAM unlike [20]. Algorithm 1 describes the details.

3 Streaming Algorithm for Scaling To Large Label Spaces

Graph-based SSL methods usually scale linearly with the label size m , and require $\mathcal{O}(m)$ space for each node. Talukdar and Cohen [28] proposed to deal with the issue of large label spaces by employing a Count-Min Sketch approximation to store the label distribution of each node. However, we argue that it is not necessary to approximate the whole label distribution for each node, especially for large label sets, because the label distribution of each node is typically sparse and only the top ranking ones are useful. Moreover, the Count-Min Sketch can even be harmful for the top ranking labels because of its approximation. The authors also mention other related works that attempt to induce sparsity using regularization techniques [32, 18] but for a very different purpose [11]. In contrast, our work tackles the exact same problem as [28] to scale graph-based SSL for large label settings. The method presented here does not attempt to enforce sparsity and instead focuses on efficiently storing and updating label distributions during semi-supervised learning with a streaming approximation. In addition, we also compare (in Section 5) against other relevant graph-

based SSL baselines [30, 1] that use heuristics to discard poorly scored labels and retain only top ranking labels per node out of a large label set.

EXPANDER-S Method: We propose a streaming sparsity approximation algorithm for semi-supervised learning that achieves constant space complexity and huge memory savings over the current state-of-the-art approach (MAD-SKETCH) in addition to significant runtime improvements over the exact version. The method processes messages from neighbors efficiently in a streaming fashion and records a sparse set of top ranking labels for each node and approximate estimate for the remaining. In general, the idea is similar to finding frequent items from data streams, where the item is the label and the streams are messages from neighbors in our case. Our Pregel-based approach (Algorithm 1) provides a natural framework to implement this idea of processing message streams. We replace the update Step 11 in the algorithm with the new version thereby allowing us to scale to both large label spaces and data using the same framework.

Preliminary: Manku and Motwani [21] presented an algorithm for computing frequency counts exceeding a user-specified threshold over data streams, and others have applied this algorithm to handle large amounts of data in NLP problems [15, 14, 35, 24]. The general idea is that a data stream containing N elements is split into multiple epochs with $\frac{1}{\epsilon}$ elements in each epoch. Thus there are ϵN epochs in total, and each such epoch has an ID starting from 1. The algorithm processes elements in each epoch sequentially and maintains a list of tuples of the form (e, f, Δ) , where e is an item, f is its reported frequency, and Δ is the maximum error of the frequency estimation. In current epoch t , when an item e comes in, it increments the frequency count f , if the item e is contained in the list of tuples. Otherwise, it creates a new tuple $(e, 1, t - 1)$. Then, after each epoch, the algorithm filters out the items whose maximum frequency is small. Specifically, if the epoch t ended, the algorithm deletes all tuples that satisfies the condition $f + \Delta \leq t$. This ensures that rare items are not retained at the end.

Neighbor label distributions as weighted streams: Intuitively, in our setting, each item is a label and each neighbor is an epoch. For a given node v , the neighbors pass label probability streams to node v , where each neighbor $u \in \mathcal{N}(v)$ is an epoch and the size of epochs is $|\mathcal{N}(v)|$. We maintain a list of tuples of the form (l, f, Δ) , in which the l is the label index, f is the weighted probability value, and Δ is the maximum error of the weighted probability estimation. For the current neighbor u_t (say, it is the t -th neighbor of v , $t \leq |\mathcal{N}(v)|$), the node v receives the label distribution $\hat{Y}_{u_t l}$ with edge weight w_{vu_t} . The al-

gorithm then does two things: if the label l is currently in the tuple list, it increments the probability value f by adding $w_{vu_t} \hat{Y}_{u_t l}$. If not, it creates new tuple of the form $(l, w_{vu_t} \hat{Y}_{u_t l}, \delta \sum_{i=1}^{t-1} w_{vu_i})$. Here, we use δ as a probability threshold (e.g., can be set as uniform distribution $\frac{1}{m}$), because the value in an item frequency stream is naturally 1 while ours is a probability weight. Moreover, each epoch t , which is neighbor u_t in our task, is weighted by the edge weight w_{vu_t} unlike previous settings [21]. Then, after we receive the message from the t -th neighbor, we filter out the labels whose maximum probability is small. We delete label l , if $f + \Delta \leq \delta \sum_{i=1}^t w_{vu_i}$.

Memory-bounded update: With the given streaming sparsity algorithm, we can ensure that no low weighted-probability labels are retained after receiving messages from all neighbors. However, in many cases, we want the number of retained labels to be bounded by k , i.e retain the top- k label based on the probability. In this case, for a node v , each of its neighbors $u \in \mathcal{N}(v)$ just contains its top- k labels, i.e. $\hat{\mathbf{Y}}_u = \hat{Y}_{u l_1}, \hat{Y}_{u l_2}, \dots, \hat{Y}_{u l_k}$. Moreover, we use $\delta_u = \frac{1.0 - \sum_{i=1}^k \hat{Y}_{u l_i}}{m - k}$ to record the average probability mass of the remaining labels. We then apply the previous streaming sparsity algorithm. The only difference is that when a label l does not exists in the current tuple list, it creates a new tuple of the form $(l, w_{vu_t} \hat{Y}_{u_t l}, \sum_{i=1}^{t-1} w_{vu_i} \delta_{u_i})$. Intuitively, instead of setting a fixed global δ as threshold, we vary the threshold δ_{u_i} based on the sparsity of the previous seen neighbors. In each epoch, after receiving messages $\hat{\mathbf{Y}}_{u_t}$ from the current (t -th) neighbor, we scan the current tuple list. For each tuple (l, f, Δ) , we increments its probability value f by adding δ_{u_t} , if label l is not within the top- k label list of the current t -th neighbor. Then, we filter out label l , if $f + \Delta \leq \sum_{i=1}^t w_{vu_i} \delta_{u_i}$. Finally, after receiving messages from all neighbors, we rank all remaining tuples based on the value $f + \Delta$ within each tuple (l, f, Δ) . This value represents the maximum weighted-probability estimation. Then we just pick the top- k labels and record only their probabilities for the current node v .

Lemma 1 *For any node $u \in V$, let y be the unnormalized true label weights and \hat{y} be the estimate given by the streaming sparsity approximation version of EXPANDER algorithm at any given iteration. Let N be the total number of label entries received from all neighbors of u before aggregation, $d = |\mathcal{N}(u)|$ be the degree of node u and k be the constant number of (non-zero) entries retained in \hat{y} where $N \leq k \cdot d$, then (1) the approximation error of the proposed sparsity approximation is bounded in each iteration by $\hat{y}_l \leq y_l \leq \hat{y}_l + \delta \cdot \frac{N}{k}$ for all labels l , (2) the space used by the algorithm at each node is $O(k) = O(1)$.*

The proof for the first part of the statement can be derived following a similar analysis as [21] using label weights instead of frequency. At the end of each iteration, the algorithm ensures that labels with low weights are not retained and for the remaining ones, its estimate is close to the exact label weight within an additive factor. The second part of the statement follows directly from the fact that each node retains at most k labels in every iteration. The detailed proof is not included here.

Next, we study various graph construction choices and demonstrate how augmenting the input graph using external information can be beneficial for learning.

4 Graph Construction

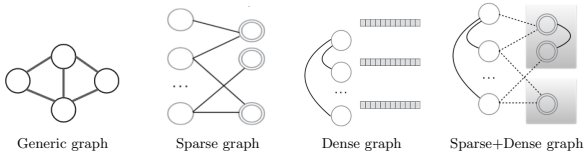


Figure 1: Graph construction strategies.

The main ingredient for graph-based SSL approaches is the input graph itself. We demonstrate that the choice of graph construction mechanism has an important effect on the quality of SSL output. Depending on the edge link information as well as choice of vertex representation, there are multiple ways to create an input graph for SSL—(a) *Generic* graphs which represent observed neighborhood or link information connecting vertices (e.g., connections in a social network), (b) graphs constructed from *sparse feature representations* for each vertex (e.g., a bipartite Freebase graph connecting entity nodes with cell value nodes that capture properties of the entity occurring in a schema or table), (c) graphs constructed from *dense representations* for each vertex, i.e., use dense feature characteristics per node to define neighborhood (discussed in more detail in the next section), and (d) *augmented* graphs that use a mixture of the above.

Figure 1 shows an illustration of the various graph types. We focus on (b), (c) and (d) here since these are more applicable to natural language scenarios. Sparse instance-feature graphs (b) are typically provided as input for most SSL tasks in NLP. Next, we propose a method to automatically construct a graph (c) for text applications using semantic embeddings and use this to produce an augmented graph (d) that captures both sparse and dense per-vertex characteristics.

4.1 Graph Augmentation with Dense Semantic Representations

In the past, graph-based SSL methods have been widely applied to several NLP problems. In many scenarios, the nodes (and labels) represent textual information (e.g., query, document, entity name/type, etc.) and could be augmented with semantic information from the real world. Recently, some researchers have explored strategies to enhance the input graphs [19] using external sources such as the Web or a knowledge base. However, these methods require access to structured information from a knowledge base or access to Web search results corresponding to a large number of targeted queries from the particular domain. Unlike these methods, we propose a more robust strategy for graph augmentation that follows a two-step approach using only a large corpus of raw text. First, we learn a dense vector representation that captures the underlying semantics associated with each (text) node. We resort to recent state-of-the-art deep learning algorithms to efficiently learn word and phrase semantic embeddings in a dense low-dimensional space from a large text corpus using unsupervised methods.

We follow the recent work of Mikolov et al. [22, 23] to compute continuous vector representations of words (or phrases) from very large datasets. The method takes a text corpus as input and learns a vector representation for every word (or phrase) in the vocabulary. We use the continuous skip-gram model [22] combined with a hierarchical softmax layer in which each word in a sentence is used as an input to a log-linear classifier which tries to maximize classification of another word within the same sentence using the current word. More details about the deep learning architecture and training procedure can be found in [22]. Moreover, these models can be efficiently parallelized and scale to huge datasets using a distributed training framework [12]. We obtain a 1000-dimensional vector representation (for each word) trained on 100 billion tokens of newswire text.³ For some settings (example dataset in Section 5), nodes represent entity names (word collocations and not bag-of-words). We can train the embedding model to take this into account by treating entity mentions (e.g., within Wikipedia or news article text) as special words and applying the same procedure as earlier to produce embedding vectors for entities. Next, for each node $v = w_1 w_2 \dots w_n$, we query the pre-trained vectors \mathcal{E} to obtain its corresponding embedding v_{emb} from words in the node text.

$$v_{emb} = \begin{cases} \mathcal{E}(v), & \text{if } v \in \mathcal{E} \\ \frac{1}{n} \sum_i \mathcal{E}(w_i), & \text{otherwise} \end{cases} \quad (3)$$

³It is also possible to use pre-trained embedding vectors: <https://code.google.com/p/word2vec/>

Following this, we compute a similarity function over pairs of nodes using the embedding vectors, where $sim_{emb}(u, v) = u_{emb} \cdot v_{emb}$. We filter out node pairs with low similarity values $< \theta_{sim}$ and add an edge in the original graph $G = (V, E)$ for every remaining pair.

Unfortunately, the above strategy requires $O(|V|^2)$ similarity computations which is infeasible in practice. To address this challenge, we resort to locality sensitive hashing (LSH) [9], a random projection method used to efficiently approximate nearest neighbor lookups when data size and dimensionality is large. We use the node embedding vectors v_{emb} and perform LSH to significantly reduce unnecessary pairwise computations that would yield low similarity values.⁴

5 Experiments

5.1 Experiment Setup

Data: We use two real-world datasets (publicly available from Freebase) for evaluation in this section.

Data Name	Nodes	Edges	Labels	Avg. Deg.
Freebase-Entity	301,638	1,155,001	192	3.83
Freebase-Relation	9,367,013	16,817,110	7,664	1.80

Freebase-Entity (referred as FB-E) is the exact same dataset and setup used in previous works [28, 30]. This dataset consists of cell value nodes and property nodes which are entities and Table properties in Freebase. An edge indicates that an entity appears in a table cell. The second dataset is Freebase-Relation (referred as FB-R). This dataset comprises entity1-relation-entity2 triples from Freebase, which consists of more than 7000 relations and more than 8M triples. We extract two kinds of nodes from these triples, entity-pair nodes (e.g., *<Barack Obama, Hawaii>*) and entity nodes (e.g., *Barack Obama*). The former one is labeled with the relation type (e.g., *PlaceOfBirth*). An edge is created if two nodes have an entity in common.

Graph-based SSL systems: We compare different graph-based SSL methods: **EXPANDER**, both the vanilla method and the version that runs on the graph with semantic augmentation (as detailed in Section 4.1), and **EXPANDER-S**, the streaming approximation algorithm introduced in Section 3.

For baseline comparison, we consider two state-of-art existing works **MAD** [29] and **MAD-SKETCH** [28]. Talukdar and Pereira [30] show that MAD outperforms traditional graph-based SSL algorithms. MAD-SKETCH further approximates the label distribution on each node using Count-Min Sketch to reduce the space and time complexity. To ensure a fair comparison, we obtained the MAD code directly from the authors and ran the exact same code on the

same machine as EXPANDER for all experiments reported here. We obtained the same MRR performance (0.28) for MAD on the Freebase-Entity dataset (10 seeds/label) as reported by [28].

Parameters: For the SSL objective function parameters, we set $\mu_1 = 1$, $\mu_2 = 0.01$ and $\mu_3 = 0.01$. We tried multiple settings for MAD and MAD-SKETCH algorithms and replicated the best reported performance metrics from [28] using these values, so the baseline results are comparable to their system.

Evaluation: Precision@K (referred as P@K) and Mean Reciprocal Rank (MRR) are used as evaluation metrics for all experiments, where higher is better. P@K measures the accuracy of the top ranking labels (i.e., at least one of the gold labels was found among the top K) returned by each method. MRR is calculated as $\frac{1}{|Q|} \sum_{v \in Q} \frac{1}{rank_v}$, where $Q \subseteq V$ is the test node set, and $rank_v$ is the rank of the gold label among the label distribution \hat{Y}_v .

For experiments, we use the same procedure as reported in literature [28], running each algorithm for 10 iterations per round (verified to be sufficient for convergence on these datasets) and then taking the average performance over 3 rounds.

5.2 Graph SSL Results

First, we quantitatively compare the graph-based SSL methods in terms of MRR and Precision@K without considering the space and time complexity. Table 1 shows the results with 5 seeds/label and 10 seeds/label on the Freebase-Entity dataset.

From the results, we have several findings: (1) Both EXPANDER-based methods outperform MAD consistently in terms of MRR and Precision@K. (2) Our algorithm on the enhanced graph using semantic embeddings (last row) produces significant gains over the original graph, which indicates that densifying the graph with additional information provides a useful technique for improving SSL in such scenarios.

5.3 Streaming Sparsity versus Sketch

In this section, we compare the MAD-SKETCH and EXPANDER-S algorithms against the vanilla versions. The former one uses Count-Min Sketch to approximate the whole label distribution per node, while the latter uses streaming approximation to capture the sparsity of the label distribution. For Freebase-Entity dataset, we run these two methods with 5 seeds/label.⁵ The Freebase-Relation dataset is too big to run on a single machine, so we sample a smaller dataset FB-R₂⁶ from it. For this new dataset, we only

⁵We observed similar findings with larger seed sizes.

⁶We create FB-R₂ by randomly picking 1000 labels and keeping only entity-pair nodes which belong to these labels and their corresponding edges (4.5M nodes, 7.4M edges).

⁴For LSH, we use $\theta_{sim}=0.6$, number of hash tables $D=12$, width $W=10$ in our experiments.

Methods	5 seeds/label					10 seeds/label				
	MRR	P@1	P@5	P@10	P@20	MRR	P@1	P@5	P@10	P@20
MAD	0.2485	0.1453	0.3127	0.4478	0.5513	0.2790	0.1988	0.3496	0.4663	0.5604
EXPANDER	0.3271	0.2086	0.4507	0.6029	0.7299	0.3348	0.1994	0.4701	0.6506	0.7593
EXPANDER (combined graph)	0.3511	0.2301	0.4799	0.6176	0.7384	0.3727	0.23436	0.5173	0.6654	0.7679

Table 1: Comparison of various graph transduction methods on the the Freebase-Entity graph.

Methods	MRR	P@1	P@5	P@10	P@20	compute time(s)	space(G)
MAD	0.2485	0.1453	0.3127	0.4478	0.5513	206.5	9.10
MAD-SKETCH (w=20 d=3)	0.2041	0.1285	0.2536	0.3133	0.4528	30.0	1.20
MAD-SKETCH (w=109 d=3)	0.2516	0.1609	0.3206	0.4266	0.5478	39.8	2.57
EXPANDER	0.3271	0.2086	0.4507	0.6029	0.7299	256.4	1.34
EXPANDER-S (k=5)	NA	0.2071	0.4209	NA	NA	78.2	0.62
EXPANDER-S (k=10)	NA	0.2046	0.4493	0.5923	NA	94.0	0.76
EXPANDER-S (k=20)	NA	0.2055	0.4646	0.5981	0.7221	123.14	0.82

Table 2: Comparison of various scalable methods based on MAD and EXPANDER on the Freebase-Entity graph.

Methods	MRR	P@1	P@5	P@10	P@20	compute time(s)	space(G)
MAD-SKETCH (w=20 d=3)	0.1075	0.0493	0.21572	0.2252	0.2902	294	12
EXPANDER-S (k=5)	NA	0.1054	0.2798	NA	NA	1092	0.91
EXPANDER-S (k=10)	NA	0.1057	0.2818	0.3745	NA	1302	1.02
EXPANDER-S (k=20)	NA	0.1058	0.2832	0.3765	0.4774	1518	1.14

Table 3: Comparison of MAD and EXPANDER methods on the FB-R₂ graph, a subgraph of Freebase-Relation.

compare the approximation methods MAD-SKETCH and our EXPANDER-S, by picking 20 seeds/label and taking average over 3 rounds. We just test MAD-SKETCH (w=20,d=3), since the setting MAD-SKETCH (w=109,d=3) runs out-of-memory using a single machine. We use protocol buffers (an efficient data serialization scheme) to store the data for EXPANDER-S. For the space, we report the memory taken by the whole process. For EXPANDER-S, as described in section 3, each node stores at most k labels, so the MRR and precision@K where $K > k$ are not available, and we refer it as NA.

Tables 2, 3 show results on Freebase-Entity and the smaller Freebase-Relation (FB-R₂) datasets, respectively. We make the following observations: (1) MAD-SKETCH (w=109,d=3) can obtain similar performance compared with MAD, while it can achieve about $5.2\times$ speedup, and $3.54\times$ space reduction. However, when the sketch size is small (e.g. w=20,d=3), the algorithm loses quality in terms of both MRR and Precision@K. For applications involving large label sizes, due to space limitations, we can only allocate a limited memory size for each node, yet we should still be able to retain the accurate or relevant labels within the available memory. On FB-R₂ data, we observe that MAD-SKETCH (w=109,d=3) is not executable, and the MAD-SKETCH (w=20,d=3) yields poor results. (2) Comparing the EXPANDER and EXPANDER-S, the latter one obtains similar performance in terms of Precision@K, while it achieves $3.28\times$ speedup for $k = 5$ and $2.16\times$ space reduction. Compared with the MAD-SKETCH, the speedup is not as steep mainly because the algorithm needs to go through the tuple list and filter out the ones below the threshold to ensure that we retain the “good” labels. However, we can easily execute EXPANDER-

S on the subset of Freebase-Relation (FB-R₂), due to low space requirements ($\sim 12\times$ lower than even MAD-SKETCH). Moreover, it outperforms MAD-SKETCH (w=20,d=3) in terms of Precision@K.

Frequency Thresholding vs. Streaming Sparsity: We also compare our streaming approximation algorithm (EXPANDER-S) against a simple frequency-based thresholding technique (FREQ-THRESH) used often by online sparse learning algorithms (zero out small weights after each update Step 11 in Algorithm 1). However, this becomes computationally inefficient $O(\text{degree} * k)$ in our case especially for nodes with high degree, which is prohibitive since it requires us to aggregate label distributions from all neighbors before pruning.⁷ In both cases, we can still maintain constant space complexity per-node by retaining only top-K labels after the update step. Table 4 shows that the streaming sparsity approximation produces significantly better quality results (precision at 5, 10) than frequency thresholding in addition to being far more computationally efficient.

Methods	P@1	P@5	P@10
FREQ-THRESH(k=5)	0.1921	0.4066	NA
EXPANDER-S (k=5)	0.2071	0.4209	NA
FREQ-THRESH(k=10)	0.2028	0.4216	0.5719
EXPANDER-S (k=10)	0.2046	0.4493	0.5923

Table 4: Comparison of sparsity approximation vs. frequency thresholding on Freebase-Entity dataset.

5.4 Graph SSL with Large Data, Label Sizes

In this section, we evaluate the space and time efficiency of our distributed algorithm DIST-EXPANDER (described in Section 2.2) coupled with

⁷We set threshold $\delta = 0.001$ for FREQ-THRESH based on experiments on a small heldout dataset.

the new streaming approximation update (Section 3). To focus on large data settings, we only use Freebase-Relation data set in subsequent experiments. Following previous work [28], we use the identity of each node as a label. In other words, the label size can potentially be as large as the size of the nodes $|V|$. First, we test how the computation time scales with the number of nodes, by fixing label size. We follow a straightforward strategy: randomly sample different number of edges (and corresponding nodes) from the original graph. For each graph size, we randomly sample 1000 nodes as seeds, and set their node identities as labels. We then run vanilla EXPANDER, EXPANDER-S ($k = 5$) and DIST-EXPANDER-S ($k = 5$) on each graph. The last one is a distributed version which partitions the graph and runs on 100 machines. We show the running time for different node sizes in Figure 2. EXPANDER runs out-of-memory when the node size

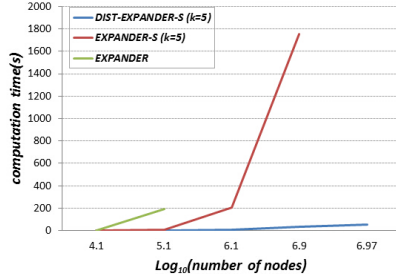


Figure 2: Running time vs. Data size for single-machine versus distributed algorithm.

goes up to 1M, and the running time slows down significantly when graph size increases. EXPANDER-S ($k=5$) can handle all five data sets on a single machine and while the running time is better than EXPANDER, it starts to slow down noticeably on larger graphs with 7M nodes. DIST-EXPANDER-S scales quite well with the node size, and yields a 50-fold speedup when compared with EXPANDER-S when the node size is ~ 7 M.

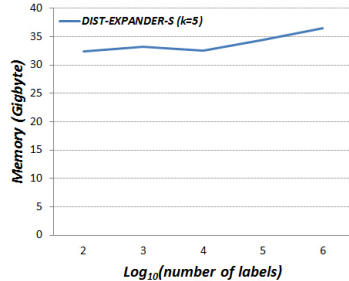


Figure 3: Memory usage vs. Label size.

Figure 3 illustrates how the memory usage scales with label size for our distributed version. For this scenario, we use the entire Freebase-Relation dataset and vary label size by randomly choosing different number of seed nodes as labels. We find that the overall

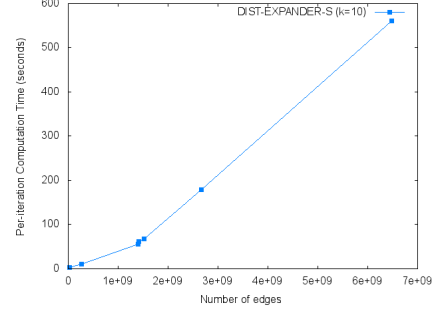


Figure 4: Per iteration runtime on massive graphs of varying sizes distributed across 300 machines.

space cost is consistently around 35GB, because our streaming algorithm captures a sparse constant space approximation of the label distribution and does not run out-of-memory even for large label sizes. Note that the distributed version consumes more than 30GB primarily because there will be redundant information recorded when partitioning the graph (including data replication for system fault-tolerance).

Finally, we test how the distributed sparsity approximation algorithm scales on massive graphs with billions of nodes and edges. Since the Freebase graph is not sufficiently large for this setting, we construct graphs of varying sizes from a different dataset (segmented video sequences tagged with 1000 labels). Figure 4 illustrates that the streaming distributed algorithm scales very efficiently for such scenarios and runs quite fast. Each computing iteration for DIST-EXPANDER-S runs to completion in just 2.3 seconds on a 17.8M node/26.7M edge graph and roughly 9 minutes on a much larger 2.6B node/6.5B edge graph.

6 Conclusion

Existing graph-based SSL algorithms usually require $O(m)$ space per node and do not scale to scenarios involving large label sizes m and massive graphs. We propose a novel streaming algorithm that effectively and accurately captures the sparsity of the label distribution. The algorithm operates efficiently in a streaming fashion and reduces the space complexity per node to $O(1)$ in addition to yielding high quality performance. Moreover, we extend the method with a distributed algorithm that scales elegantly to large data and label sizes (for example, billions of nodes/edges and millions of labels). We also show that graph augmentation using unsupervised learning techniques can provide a robust strategy to yield performance gains for SSL problems involving natural language.

Acknowledgements

We thank Partha Talukdar for useful pointers to MAD code and Kevin Murphy for providing us access to the Freebase-Relation dataset.

References

- [1] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the International World Wide Web Conference*, 2013.
- [2] Apache giraph. <http://giraph.apache.org/>, 2013.
- [3] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly. Video suggestion and discovery for Youtube: Taking random walks through the view graph. In *Proceedings of the 17th International Conference on World Wide Web, WWW '08*, pages 895–904, 2008.
- [4] M. Belkin, P. Niyogi, and V. Sindhwani. On manifold regularization. In *Proceeding of the Conference on Artificial Intelligence and Statistics (AISTATS)*, 2005.
- [5] Y. Bengio, O. Delalleau, and N. Le Roux. Label propagation and quadratic criterion. In O. Chapelle, B. Schölkopf, and A. Zien, editors, *Semi-Supervised Learning*, pages 193–216. MIT Press, 2006.
- [6] J. Blitzer and X. J. Zhu. Semi-supervised learning for natural language processing. In *ACL-HLT Tutorial*, June 2008.
- [7] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.
- [8] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.
- [9] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.
- [10] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
- [11] D. Das and N. A. Smith. Graph-based lexicon expansion with sparsity-inducing penalties. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 677–687. Association for Computational Linguistics, 2012.
- [12] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *Proceedings of NIPS*, pages 1232–1240, 2012.
- [13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [14] B. V. Durme and A. Lall. Streaming pointwise mutual information. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1892–1900. 2009.
- [15] A. Goyal, H. Daumé, III, and S. Venkatasubramanian. Streaming for large scale NLP: Language modeling. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, NAACL '09*, pages 512–520, 2009.
- [16] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 200–209, 1999.
- [17] T. Joachims. Transductive learning via spectral graph partitioning. In *Proceedings of ICML*, pages 290–297, 2003.
- [18] M. Kowalski and B. Torrèsani. Sparsity and persistence: mixed norms provide simple signal models with dependent coefficients. *Signal, Image and Video Processing*, 3(3):251–264, Sept. 2009.
- [19] Z. Kozareva, K. Voevodski, and S.-H. Teng. Class label enhancement via related instances. In *Proceedings of EMNLP*, pages 118–128, 2011.
- [20] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146, 2010.
- [21] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases, VLDB '02*, pages 346–357, 2002.
- [22] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at ICLR*, 2013.

- [23] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*, 2013.
- [24] M. Osborne, A. Lall, and B. V. Durme. Exponential reservoir sampling for streaming language models. In *Proceedings of The 52nd Annual Meeting of the Association for Computational Linguistics*, ACL '2014, pages 687–692, 2014.
- [25] M. Seeger. Learning with labeled and unlabeled data. Technical report, 2001.
- [26] A. Subramanya and J. A. Bilmes. Entropic graph regularization in non-parametric semi-supervised classification. In *Proceedings of NIPS*, pages 1803–1811, 2009.
- [27] A. Subramanya, S. Petrov, and F. Pereira. Efficient graph-based semi-supervised learning of structured tagging models. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 167–176, 2010.
- [28] P. Talukdar and W. Cohen. Scaling graph-based semi supervised learning to large number of labels using count-min sketch. In *Proceedings of AISTATS*, pages 940–947, 2014.
- [29] P. P. Talukdar and K. Crammer. New regularized algorithms for transductive learning. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*, ECML PKDD '09, pages 442–457, 2009.
- [30] P. P. Talukdar and F. Pereira. Experiments in graph-based semi-supervised learning methods for class-instance acquisition. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1473–1481, 2010.
- [31] P. P. Talukdar, J. Reisinger, M. Pasca, D. Ravichandran, R. Bhagat, and F. Pereira. Weakly-supervised acquisition of labeled class instances using graph random walks. In *EMNLP*, pages 582–590, 2008.
- [32] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288, 1996.
- [33] A. C. Tommi and T. Jaakkola. On information regularization. In *Proceedings of the 19th UAI*, 2003.
- [34] J. Ugander and L. Backstrom. Balanced label propagation for partitioning massive graphs. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, pages 507–516, 2013.
- [35] B. Van Durme and A. Lall. Efficient online locality sensitive hashing via reservoir counting. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 18–23, 2011.
- [36] Y. Wang, R. Ji, and S.-F. Chang. Label propagation from imagenet to 3d point clouds. In *Proceedings of CVPR*, pages 3135–3142. IEEE, 2013.
- [37] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.
- [38] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of ICML*, pages 912–919, 2003.