# Scalable Graph Embeddings via Sparse Transpose Proximities

Yuan Yin
yinyuan@ruc.edu.cn
Beijing Key Lab of Big Data Management and Analysis Method, MOE Key Lab DEKE, School of Information
Renmin University of China

Zhewei Wei*
zhewei@ruc.edu.cn
Beijing Key Lab of Big Data Management and Analysis Method, MOE Key Lab DEKE, School of Information
Renmin University of China

## ABSTRACT

Graph embedding learns low-dimensional representations for nodes in a graph and effectively preserves the graph structure. Recently, a significant amount of progress has been made toward this emerging research area. However, there are several fundamental problems that remain open. First, existing methods fail to preserve the out-degree distributions on directed graphs. Second, many existing methods employ random walk based proximities and thus suffer from conflicting optimization goals on undirected graphs. Finally, existing factorization methods are unable to achieve scalability and non-linearity simultaneously.

This paper presents an in-depth study on graph embedding techniques on both directed and undirected graphs. We analyze the fundamental reasons that lead to the distortion of out-degree distributions and to the conflicting optimization goals. We propose *transpose proximity*, a unified approach that solves both problems. Based on the concept of transpose proximity, we design STRAP, a factorization based graph embedding algorithm that achieves scalability and non-linearity simultaneously. STRAP makes use of the *backward push* algorithm to efficiently compute the sparse *Personalized PageRank (PPR)* as its transpose proximities. By imposing the sparsity constraint, we are able to apply non-linear operations to the proximity matrix and perform efficient matrix factorization to derive the embedding vectors. Finally, we present an extensive experimental study that evaluates the effectiveness of various graph embedding algorithms, and we show that STRAP outperforms the state-of-the-art methods in terms of effectiveness and scalability.

## CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**; • **Information systems → Data mining**; • **Computing methodologies → Learning latent representations**;

## KEYWORDS

Graph Embedding; Network Representation Learning; Personalized PageRank

*Zhewei Wei is the corresponding author.


*KDD '19, August 4–8, 2019, Anchorage, AK, USA*

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330860

**Table 1: Proximities used by existing methods.**

| Method | Proximity | Category |
|---|---|---|
| DeepWalk [31] | $\mathbf{s}_u \cdot \mathbf{s}_v \sim$ probability that a truncated random walk from $u$ visits $v$ | Random Walk |
| Node2Vec [19] | $\mathbf{s}_u \cdot \mathbf{s}_v \sim$ probability that a truncated 2nd order random walk from $u$ visits $v$ | Random Walk |
| LINE [35] | $\mathbf{s}_u \cdot \mathbf{s}_v \sim$ Adjacency relation between $u$ and $v$ | Random Walk |
| APP [44] | $\mathbf{s}_u \cdot \mathbf{t}_v \sim \text{PPR}(u, v)$ | Random Walk |
| VERSE [37] | $\mathbf{s}_u \cdot \mathbf{t}_v \sim \text{PPR}(u, v)$, $\text{SimRank}(u, v)$ | Random Walk |
| HOPE [29] | $\mathbf{s}_u \cdot \mathbf{t}_v \sim \text{PPR}(u, v)$, $\text{Katz}(u, v)$ | Factorization |
| AROPE [43] | $\mathbf{s}_u \cdot \mathbf{t}_v \sim$ Higher order proximity of form $\sum_{i=1}^{q} w_i A^i$ | Factorization |

**ACM Reference Format:**
Yuan Yin and Zhewei Wei. 2019. Scalable Graph Embeddings via Sparse Transpose Proximities. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330860

## 1 INTRODUCTION

Graphs are a fundamental tool for understanding and modeling complex physical, social, informational, and biological systems. In recent years, graph embedding has drawn increasing attention from the academic fields due to its applications in various machine learning tasks. The central idea of graph embedding is to learn a low-dimensional latent representation for nodes in the graph, such that the inherent properties and structures of the graph are preserved by the embedding vectors. These vectors can then be feed into well-studied machine learning methods in the vector space for common tasks on graphs such as classification, clustering, link prediction, and visualization.

In the past year, many methods have been proposed for learning node representations, and we summarize a few of recent ones in Table 1. In general, there are broadly two categories of approaches: methods which use random walks to learn the embedding vectors, and methods which use matrix factorization to directly derive the embedding vectors. Despite of their diversity, most of the existing methods adopt the following framework: 1) Determine a proximity measure $P(u, v)$; 2) Train embedding vector $\mathbf{s}_u$ for each node $u \in V$, such that $\mathbf{s}_u \cdot \mathbf{s}_v \sim P(u, v)$. For random walk methods, $\mathbf{s}_u$'s are trained by skip-gram model [27] with negative sampling or hierarchical softmax; For factorization methods, $\mathbf{s}_u$'s are directly derived from singular value decomposition (SVD) or eigen-decomposition. Recently, [37] and [44] propose that in order to capture the asymmetry of directed graphs, we should train two vectors $\mathbf{s}_u$ and $\mathbf{t}_u$ as content/contextrepresentations, and thus the goal becomes to train $\mathbf{s}$ and $\mathbf{t}$ such that $\mathbf{s}_u \cdot \mathbf{t}_v \sim P(u, v)$ for any $u, v \in V$.

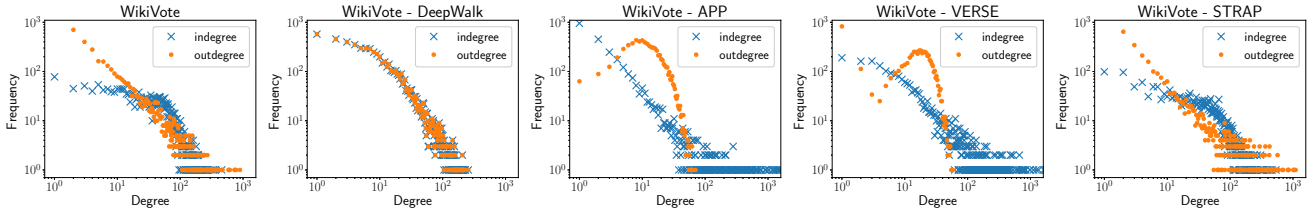


**Figure 1: Degree distributions of WikiVote.**

## 1.1 Motivations and Objectives

Although a significant amount of progress has been made toward the understanding of graph embedding, we believe that there are still several fundamental problems that remain unsolved. The goal of this paper is to analyze the mechanisms that cause these problems and to design principles and techniques that solve them. In particular, our objective is to design a graph embedding algorithm with the following desired properties.

**Objective 1: preserve both in- and out-degree distributions on directed graphs.** Consider a simple task of reconstructing the directed graph WikiVote with $n = 7,115$ nodes and $m = 103,689$ edges. We train embedding vectors for each node, rank pairs of nodes according to the inner products of their vectors, remove self-loop, and take the top-$m$ pairs of nodes to reconstruct the graph. Figure 1 shows the degree distributions of the original graph WikiVote and the reconstructed graphs by several state-of-the-art graph embedding algorithms: DeepWalk [31], APP [44], VERSE [37]. We exclude the results of some other methods such as HOPE [29] and Node2Vec [19], as the results are similar to those of DeepWalk or VERSE. We first observe that DeepWalk generates identical in/out-degree distributions. This is because DeepWalk (or Node2Vec) trains a single embedding vector $\mathbf{s}_u$ for each node $u$, and uses the same inner product $\mathbf{s}_u \cdot \mathbf{s}_v = \mathbf{s}_v \cdot \mathbf{s}_u$ to predict for the edge from $u$ to $v$ and the edge from $v$ to $u$. Therefore, DeepWalk (and Node2Vec) is only able to preserve structural information for undirected graphs.

The second observation is that VERSE and APP, the two recent embedding algorithms that are designed for directed graphs, fail to preserve the out-degree distribution of the original graphs. In particular, the reconstructed out-degree distributions do not follow power-law distribution: there are no nodes with large out-degrees, and most out-degrees concentrate on 14, the average out-degree of the original graphs. As it turns out, there is a fundamental reason for this phenomenon. Recall that an embedding algorithm determines a proximity measure $P(u, v)$, and tries to train $\mathbf{s}_u \cdot \mathbf{t}_v \sim P(u, v)$. For random walk based proximities such as Personalized PageRank or hitting probability, the proximities values of node $u$ to any nodes in the graph is normalized, i.e., $\sum_{v \in V} P(u, v) = 1$. Therefore, given a source node $u$, the number of pairs $(u, v)$ with large proximities $P(u, v)$ (and hence large inner products $\mathbf{s}_u \cdot \mathbf{t}_v$) is actually limited to a very small range. Consequently, these methods are inherently unable to reconstruct nodes with large out-degrees, and the out-degrees of the reconstructed graph will concentrate on the average out-degree of the original graph.

We note that the lack of ability to preserve degree distributions will hurt both the effectiveness of the embedding vectors. In particular, these methods are inherently unable to make predictions for nodes with many or very few out-neighbors. Therefore, our first objective is to study how to modify the proximity measure $P(u, v)$ to preserve both in- and out-degree distributions.

**Objective 2: avoid conflicting optimization goals for undirected graphs.** Another more subtle deficiency suffered by existing techniques is the conflicting optimization goals lead by the usage of asymmetric proximities. More precisely, recall that existing methods such as DeepWalk, Node2Vec and VERSE train a single embedding vector $\mathbf{s}_u$ for each node $u$ on undirected graphs, such that $\mathbf{s}_u \cdot \mathbf{s}_v \sim P(u, v)$ for some proximity measure $P(u, v)$. Consequently, the algorithms will train $\mathbf{s}_v \cdot \mathbf{s}_u$ to approximate the proximity $P(v, u)$. We note that the inner product $\mathbf{s}_u \cdot \mathbf{s}_v = \mathbf{s}_v \cdot \mathbf{s}_u$ is commutative, but the proximity $P(u, v)$ generally does not equal to $P(v, u)$, even on undirected graphs. For example, the probability that a random walk from $u$ visit $v$ in $t$ steps does not equal to the probability that a random walk from $v$ visit $u$ in $t$ steps; the Personalized PageRank of $v$ with respect to $u$ does not equal to the Personalized PageRank of $u$ with respect to $v$. As a consequence, these methods try to train $\mathbf{s}_u \cdot \mathbf{s}_v$ to approximate two conflicting values, which will hurt the quality of the embeddings vector.

On the other hand, HOPE and APP tries to solve this problem by training asymmetric content/context embedding vectors $\mathbf{s}_u$ and $\mathbf{t}_u$ for each node $u$ on undirected graphs, such that $\mathbf{s}_u \cdot \mathbf{t}_v \sim P(u, v)$ and $\mathbf{s}_v \cdot \mathbf{t}_u \sim P(v, u)$. However, this approach introduces another problem: since there may be a substantial difference between $\mathbf{s}_u \cdot \mathbf{t}_v$ and $\mathbf{s}_v \cdot \mathbf{t}_u$, we are unable to determine which to use to predict for edge $(u, v)$ in the task of graph reconstruction or link prediction. Therefore, it is desirable to use symmetric proximities on undirected graphs. At first glance, this requirement rules out all random walk based proximities. However, as we shall see, we can achieve symmetry by a simple modification.

**Objective 3: design factorization method that achieves scalability and non-linearity simultaneously.** The general goal of embedding algorithms is to optimizing both inductive (e.g. link prediction) and transductive (e.g. graph reconstruction) effectiveness, and to achieve high scalability on large graphs. Matrix factorization methods usually achieve good transductive effectiveness as they are designed to minimize the reconstruction error of the proximity matrix. However, they suffer from scalability problem, since it takes $\Theta(n^2)$ time to compute the proximity matrix. Recently, HOPE and AROPE [43] avoid the $\Theta(n^2)$ computation time by factorizing a sparse matrix that closely related to the proximity matrix. However, they do not explicitly compute the proximity matrix, and thus do not allow any non-linear operation (such as taking logarithm or softmax) on the proximity matrix. This approach limits their inductive strength due to the linear nature. In fact, it has been shown in [44] and [32] that skip-gram based algorithms implicitly factorize the logarithm of certain proximity matrix, where taking entry-wise

logarithm simulates the effect of the sigmoid function and improves the induction strength of the model. As a result, it is desirable to design a factorization method that achieves high scalability and allows non-linear operations on the proximity matrix.

### 1.2 Our Contributions

To remedy the deficiencies of existing techniques, this paper presents an in-depth study of graph embedding techniques on both directed and undirected graphs. First, given a normalized proximity measure $P(u, v)$, we propose that instead of training $\mathbf{s}_u \cdot \mathbf{t}_v \sim P(u, v)$, we should train $\mathbf{s}_u \cdot \mathbf{t}_v$ to approximate the *transpose proximity* $P(u, v) + P^T(v, u)$, where $P^T(v, u)$ is the proximity of $u$ with respect to $v$ in the transpose graph $G^T$. Here $G^T$ is obtained by reverting the edge direction of the original graph $G$. We show that by this simple modification, we solve the distortion of out-degree distributions and the conflicting optimization goals simultaneously.

Based on the concept of transpose proximity, we propose *STRAP* (graph embedding via Sparse TRAnspose Proximities), an embedding algorithm that provides both high predictive strength and scalability. See Figure 1 for the reconstructed degree distributions of WikiVote by STRAP. We use *Personalized PageRank (PPR)* as the normalized proximity measure $P(u, v)$ to demonstrate the superiority of transpose proximity. To avoid the $\Theta(n^2)$ barrier of computing pair-wise PPR, we employ the *backward push* algorithm [25] that computes approximate pair-wise PPR values with additive error $\varepsilon$ in $O(m/\varepsilon)$ time. Unlike HOPE or AROPE, we explicitly derive the proximity matrix $P$, a sparse matrix that consists of at most $O(n/\varepsilon)$ non-zero entries. The sparsity enables us to impose non-linear functions such as entry-wise logarithm to improve the predictive strength, as well as to use sparse SVD algorithm to efficiently decompose $P$ into the embedding vectors. We experimentally evaluate STRAP on a variety of benchmark datasets, and our results demonstrate that STRAP outperforms state-of-the-art methods for both transductive and inductive tasks.

## 2 RELATED WORK

In general, there are broadly two categories of approaches: methods which use random walks to learn the embeddings, and methods which use matrix factorization to directly derive the embeddings. We briefly review some of the relevant works in each category and refer readers to [14, 18, 42] for comprehensive surveys.

**Factorization methods.** A natural idea for preserving the high-order proximities is to perform explicit matrix factorization on the proximity matrices. Early efforts include LLE [34], Laplacian Eigenmaps [7] and Graph Factorization [4]. GRAREP [9] performs SVD on the $k$-th order transition matrix, and GEM-D [12] gives a unified approach to compute and factorize the proximity matrix for various proximity measures. [32] shows that existing random walk based methods are equivalent to factorizing high order proximity matrices. However, computing the proximity matrix for the above methods still takes $\Theta(n^2)$ time, and hence are inherently not scalable. HOPE [29] avoid the $\Theta(n^2)$ time by performing a special version of SVD on proximity matrix of form $M_g^{-1}M_\ell$, where both $M_g$ and $M_\ell$ are sparse. Recently, [43] proposes AROPE, a general framework for preserving arbitrary-order proximities that includes HOPE as its special case. However, HOPE and AROPE do not compute the explicit proximity matrix, and thus are unable to support any non-linear operation on the proximity matrix, which limits their inductive strength due to the linear nature.

**Table 2: Frequently used notations.**

| Notation | Description |
|---|---|
| $G$=($V$, $E$) | The input graph $G$ with node set $V$ and edge set $E$ |
| $n$, $m$ | The number of nodes and edges in $G$, respectively |
| $\mathcal{O}(v)$, $\mathcal{I}(v)$ | The set of out- and in-neighbors of node $v$ |
| $d_{out}(v)$, $d_{in}(v)$ | The out-degree and in-degree of node $v$ |
| $\mathbf{s}_u$, $\mathbf{t}_u$ | The content/context embedding vectors of $u$ |
| PPR($u$, $v$) | The Personalized PageRank of $v$ with respect to $u$ |
| $\alpha$ | The decay factor |
| $r(u, v)$, $\pi(u, v)$ | The reserve and residue of $v$ from $u$ in backward push |

**Random walk methods.** Random walks have been used to approximate many different proximities such as Personalized PageRank [30], Heat Kernel PageRank [23] and SimRank [20]. In the line of graph embedding research, DeepWalk [31] first proposed to train embedding vectors by feeding truncated random walks to the SkipGram model [27]. The model is optimized by Stochastic Gradient Descent (SGD). LINE [35] focuses on one-hop neighbor proximity, which essentially equals to random walks with step at most 1. Node2Vec [19] proposes to replace the truncated random walks with a higher order random walks that exploit both DFS and BFS nature of the graph. Recently, Verse [37] and APP [44] propose to train embedding vectors using Personalized PageRank, where the positive samples can be efficiently obtained by simulating $\alpha$-discounted random walks. Random walk methods are scalable than some of the factorization methods and generally achieve higher inductive effectiveness. However, they only allow normalized proximity measure, which, as we shall see, leads to the distortion of out-degree distributions on directed graphs.

**Other related work.** The growing research on deep learning has led to a deluge of deep neural networks based methods applied to graphs [8, 10, 28, 39]. Recently, Graph Convolutional Network (GCN) [22] and its variants have drawn increasing research attention. There are also various graph embeddings designed for specific graphs, such as signed graphs [21, 41], dynamic graphs [26, 45] and heterogeneous networks [11, 15]. In this paper, we focus on the most fundamental case where only the static network is available.

## 3 STRAP ALGORITHM

In this section, we present STRAP, a scalable graph embedding algorithm that achieves all three objectives. Table 2 summaries the frequently used notations used in this paper. We first show that a unified approach, *transpose proximity,* achieves **Objective 1** and **Objective 2** simultaneously.

### 3.1 Transpose Proximity

Suppose the goal of a graph embedding algorithm is to train content/context embedding vectors $\mathbf{s}_u$ and $\mathbf{t}_u$ for each node $u \in V$, such that $\mathbf{s}_u \cdot \mathbf{t}_v \sim P(u, v)$ for a predetermined proximity measure $P(u, v)$. We assume the proximities of any node with respect to a given node $u$ is normalized, i.e., $\sum_{v \in V} P(u, v) = 1$. This assumption holds for any random walk based proximities (e.g. Personalized PageRank, hitting probability etc), and thus is well-recognized by

various graph embedding algorithms. Let $G^T$ denote the transpose graph of $G$, that is, there is an edge from node $v$ to node $u$ in $G^T$ if and only if there is an edge from node $u$ to node $v$.

The key insight of transpose proximity is that instead of optimizing $\mathbf{s}_u \cdot \mathbf{t}_v \sim P(u,v)$, we should optimize $\mathbf{s}_u \cdot \mathbf{t}_v \sim P(u,v) + P^T(v,u)$, where $P^T(v,u)$ is the proximity of $u$ with respect to $v$ in the transpose graph $G^T$. We will show that 1) the transpose proximity preserves both in- and out-degree distributions for directed graphs and 2) the transpose proximity avoid conflicting optimization goals on undirected graphs.

**In-degree distribution and proximities.** To show that the transpose proximity preserves both in- and out-degree distributions for directed graphs, we first establish the connection between the in-degree distributions and normalized proximities. We observe from Figure 1 that although existing methods do not preserve the out-degree distribution, they generate power-law-shaped in-degree distributions that are similar to the one of the original graph. An intuitive explanation is that although the proximity sum $\sum_{v \in V} P(u,v)$ from a source node $u$ is normalized to 1, the proximity sum $\sum_{u \in V} P(u,v)$ to a target node $v$ is not normalized. In fact, $\sum_{u \in V} P(u,v)$ reflects the number of nodes that are similar to $v$ and thus is a good approximation to the indegree of $v$. For example, VERSE, APP and HOPE use Personalized PageRank of $v$ with respect to $u$ as the normalized proximity measure $P(u,v)$. For this particular proximity measure, we have $\sum_{u \in V} P(u,v) = n \cdot \mathrm{PR}(v)$ [30], where $n$ is the number of nodes in the graph, and $\mathrm{PR}(v)$ is the PageRank of $v$. It is shown in [6, 25, 40] that on scale-free networks, PageRank and in-degrees follow the same power-law distribution, which implies that $\sum_{u \in V} P(u,v) = n \cdot \mathrm{PR}(v) \sim d_{in}(v)$. Consequently, we claim the reason that existing method preserves the in-degree distribution is that they employ normalized proximity $P(u,v)$ that satisfies $\sum_{u \in V} P(u,v) \sim d_{in}(v)$.

**In-/out-degree distributions and transpose proximities.** With the insight that $\sum_{u \in V} P(u,v) \sim d_{in}(v)$, we now show that transpose proximity $P(u,v) + P^T(v,u)$ preserves both the in- and out-degree distribution. Consider the summation of transpose proximities to a target node $v$, we have $\sum_{u \in V} \left(P(u,v) + P^T(v,u)\right) = \sum_{u \in V} P(u,v) + \sum_{u \in V} P^T(v,u) = \sum_{u \in V} P(u,v) + 1 \sim d_{in}(v)$. Note that here we use the fact that $\sum_{u \in V} P(u,v) \sim d_{in}(v)$ and ignore the plus one since it does not change the relative order of the proximity summations. On the other hand, let $d^T_{in}(u)$ denote the in-degree of $u$ in the transpose graph $G^T$. Consider the summation of transpose proximities from a source node $u$ and we have $\sum_{v \in V} \left(P(u,v) + P^T(v,u)\right) = \sum_{v \in V} P(u,v) + \sum_{v \in V} P^T(v,u) = 1 + \sum_{v \in V} P^T(v,u) \sim d^T_{in}(u)$. Here we use the fact $\sum_{u \in V} P^T(v,u) \sim d^T_{in}(u)$ in the transpose graph $G^T$. We observe that $d^T_{in}(u)$, the in-degree of $u$ in the transpose graph $G^T$, equals to $d_{out}(u)$, the out-degree of $u$ in the original graph $G$. It follows that the summation $\sum_{v \in V} \left(P(u,v) + P^T(v,u)\right) \sim d_{out}(u)$. Therefore the summation of transpose proximities to a target node $v$ approximates the in-degree of $v$, while the summation of transpose proximities from a source node $u$ approximates the out-dgree of $u$. As a consequence, by employing the transpose proximity $P(u,v) + P^T(v,u)$, we preserve both the in- and out-degree distribution for directed graphs.

**Transpose proximity on undirected graphs.** Another advantage of transpose proximity is that it automatically avoids the conflicting optimization goals on undirected graphs. In particular, note that for undirected graphs, the transpose graph $G^T$ is identical to the original graph $G$, and thus $P^T(v,u)$ equals to $P(v,u)$. Therefore, the transpose proximity becomes $P(u,v) + P(v,u)$, which is a symmetric similarity measure for any proximity measure $P(u,v)$. Consequently, we train $\mathbf{s}_u \cdot \mathbf{s}_v = \mathbf{s}_v \cdot \mathbf{s}_u$ to approximate the same proximity $P(u,v) + P(v,u) = P(v,u) + P(u,v)$, and thus avoiding the conflicting optimization goals suffered by existing techniques.

## 3.2 Sparse Personalized PageRank

Although the concept of transpose proximity works for any normalized proximity measure $P(u,v)$, in this paper we focus on $P(u,v) = \mathrm{PPR}(u,v)$, the *Personalized PageRank (PPR)* [30] of node $v$ with respect to node $u$. Given a source node $u$, a target node $v$ on directed graph $G = (V,E)$, $\mathrm{PPR}(u,v)$ measures the importance of $v$ in the view of $u$. More precisely, we define an *$\alpha$-discounted random walk* from $u$ to be a traversal of $G$ that starts from $u$ and, at each step, either 1) terminates at the current node with $\alpha$ probability, or 2) proceeds to a randomly selected out-neighbor of the current node. For any node $v \in V$, $\mathrm{PPR}(u,v)$ of $v$ with respect to $u$ is the probability that an $\alpha$-discounted random walk from $u$ terminates at $v$. We choose PPR mainly because it has been widely used in graph embedding algorithms [29, 37, 44]. Moreover, as we stated before, the summation of $\sum_{u \in V} \mathrm{PPR}(u,v) = n\mathrm{PR}(v)$ equals the PageRank of $v$, and PageRank and in-degrees follow the same power-law distribution. Therefore, by employing transpose proximity matrix $P$ with $P_{uv} = \mathrm{PPR}(u,v) + \mathrm{PPR}^T(v,u)$, the resulting embeddings will preserve both the in- and out-degree distributions of the graphs.

However, directly computing PPR for any node pair $(u,v)$ takes at least $\Theta(n^2)$ time and memory usage. To make thing worse, it takes $O(n^3)$ time to decompose a $n \times n$ dense proximity matrix. Therefore, it is infeasible to compute exact PPR for all node pairs on large graphs. HOPE [29] proposes to decompose the PPR matrix $\mathrm{PPR} = M_g^{-1} M_\ell$ by performing a generalized SVD on sparse matrices $M_g$ and $M_\ell$, where $M_g = I - (1-\alpha)D^{-1}A$, $M_\ell = \alpha I$, $D$ is the diagonal degree matrix and $A$ is the adjacency matrix. However, this approach does not explicitly compute the PPR matrix and thus does not support decomposition of the transpose proximity matrix $P$ where $P_{uv} = \mathrm{PPR}(u,v) + \mathrm{PPR}^T(v,u)$. Furthermore, it does not allow non-linear operations before the decomposition, which is crucial for achieving satisfying predictive strength [32].

To explicitly compute PPR values for all pairs of nodes in the graph efficiently, we will use an approximate version of PPR called *Sparse Personalized PageRank (SPPR)*. Given an error parameter $\varepsilon \in (0,1)$ and two nodes $u$ and $v$ in the graph, $SPPR(u,v)$ is a real value that satisfies:1) $|SPPR(u,v) - PPR(u,v)| \leq \varepsilon$, for any $u, v \in V$; 2) For a fixed node $u$, there are at most $2/\varepsilon$ nodes $v$ with non-zero $SPPR(u,v)$. Note that the first condition guarantees that the sparse Personalized PageRank approximates the original Personalized PageRank with precision $\varepsilon$. This relaxation allows us to compute SPPR in time linear to the edge number $m$. On the other hand, the second condition ensures that the proximity matrix is sparse, which is crucial for efficient matrix decomposition. Combining the idea of transpose proximity, our final proximity measure is defined as $P_{uv} = SPPR(u,v) + SPPR^T(v,u)$.

### 3.3 Computing SPPR with Backward Push

**Algorithm 1:** Backward Push [25]

**Input**: Graph $G$, target node $v$, decay factor $\alpha$, threshold $r_{max}$
**Output**: Backward residue $r(x, v)$ and reserve $\pi(x, v)$ for all $x \in V$

1 $r(v, v) \leftarrow 1$ and $r(x, v) \leftarrow 0$ for all $x \neq v$; $\pi(x, v) \leftarrow 0$ for all $x$;
2 **while** $\exists x$ *such that* $r(x, v) > r_{max}$ **do**
3  **for** *each* $y \in \mathcal{I}(x)$ **do**
4  $r(y, v) \leftarrow r(y, v) + (1 - \alpha) \cdot \frac{r(x,v)}{d_{out}(y)}$
5  $\pi(x, v) \leftarrow \pi(x, v) + \alpha \cdot r(x, v)$;
6  $r(x, v) \leftarrow 0$;

**Backward Push.** We employ a local search algorithm called *Backward push* [25] to compute SPPR for any node pair $(u, v)$ in $O(m/\varepsilon)$ time. Given a *destination* node $v$, the backward push algorithm employs a traversal from $v$ to compute an approximation of $v$'s PPR value $\text{PPR}(u, v)$ with respect to any other node $u$. We sketch the algorithm in Algorithm 1 for completeness. The algorithm starts by assigning a residue $r(x, v)$ and reserve $\pi(x, v) = 0$ to each node $x$, and setting $r(v, v) = 1$ and $r(x, v) = 0$ for any $x \neq v$ (Lines 1-2). Subsequently, it traverses from $v$, following the incoming edges of each node. For any node $x$ that it visits, it checks if $x$'s residue $r(x, v)$ is larger than a given threshold $r_{max}$. If so, then it increases $x$'s reserve by $\alpha \times r(x, v)$ and, for each in-neighbor $y$ of $x$, increases the residue of $y$ by $(1 - \alpha) \cdot \frac{r(x,v)}{d_{out}(y)}$ (Lines 4-5). After that, it reset $x$'s residue $r(x, v)$ to 0 (Line 6). The following Lemma is proven by Lofgren et al. [25]:

Lemma 3.1 ([25]). *The amortized time of backward push over all possible target nodes $v \in V$ is $O\left(\frac{m}{n \cdot r_{max}}\right)$. When the algorithm terminates, it provides a reserve $\pi(u, v)$ for any node $u$, such that*

$$\text{PPR}(u, v) - r_{max} \leq \pi(u, v) \leq \text{PPR}(u, v).$$

**Algorithm 2:** STRAP

**Input**: Graph $G$, dimension $d$, error parameter $\varepsilon$, decay factor $\alpha$
**Output**: Embedding vectors $\mathbf{s}_u$ and $\mathbf{t}_u$ for each $u \in V$

1 Initialize sparse proximity matrix $P_{n \times n} \leftarrow 0$;
2 **for** *each node* $v \in V$ **do**
3  $BackwardPush(v, \alpha, \varepsilon/2, G)$;
4  **for** *each node* $u$ *with reserve* $\pi(u, v) \geq \varepsilon/2$ **do**
5  $P_{uv} \leftarrow \pi(u, v)$;
6 **for** *each node* $u \in V$ **do**
7  $BackwardPush(u, \alpha, \varepsilon/2, G^T)$;
8  **for** *each node* $v$ *with reserve* $\pi^T(v, u) \geq \varepsilon/2$ **do**
9  $P_{uv} \leftarrow P_{uv} + \pi^T(v, u)$;
10 Set sparse matrix $P \leftarrow \log\left(\frac{2}{\varepsilon} \cdot P\right)$ for non-zero entries;
11 $[U, \Sigma, V] \leftarrow$ Randomized SVD$(P, d)$;
12 **return** $U\sqrt{\Sigma}$ and $V\sqrt{\Sigma}$ as embedding vectors;

**STRAP algorithm.** Algorithm 2 illustrates the pseudocode of STRAP. Recall that our goal is to compute the proximity matrix $P$, which consists of entries $P_{uv} = \text{SPPR}(u, v) + \text{SPPR}^T(v, u)$. To compute $\text{SPPR}(u, v)$ for any node pair $u, v \in V$, we perform backward push on each target node $v \in V$ with $r_{max} = \varepsilon/2$ (Lines 2-3). This will give us a list of node-reserve pairs $(u, \pi(u, v))$. For each node $u$ with reserve $\pi(u, v) > \varepsilon/2$, we update the proximity matrix $P$ by $P_{uv} \leftarrow \pi(u, v)$ (Lines 4-5). We claim that $P_{uv} = \text{SPPR}(u, v)$ at this time point. We then perform the same process on each node $u$ in $G^T$ to compute $\text{SPPR}^T(v, u)$ (Lines 6-9). The only difference is that for each node $v$ with reserve $\pi^T(v, u) > \varepsilon/2$, we increment $P_{uv}$ (instead of $P_{vu}$) by $\pi^T(v, u)$ (Line 9). We have the following Lemma that shows that $P$ is a sparse matrix that approximates the transpose proximity for any node pair $(u, v)$:

Lemma 3.2. *The proximity matrix $P$ satisfies 1) There are at most $4n/\varepsilon$ non-zero entries in $P$; 2) For any $u, v \in V$, we have*

$$\text{PPR}(u, v) + \text{PPR}^T(v, u) - 2\varepsilon \leq P_{uv} \leq \text{PPR}(u, v) + \text{PPR}^T(v, u).$$

Proof. As target node $v$ iterates over all possible nodes in $V$, we ensure that for any node pair $(u, v)$, $P_{uv} = \pi(u, v)$ if $\pi(u, v) \geq \varepsilon/2$ and $P_{uv} = 0$ otherwise. We will show that $P_{uv}$ is a valid SPPR. By the property of backward push, we have

$$\text{PPR}(u, v) - \varepsilon/2 = \text{PPR}(u, v) - r_{max} \leq \pi(u, v) \leq \text{PPR}(u, v)$$

for any $u, v \in V$. Since we only take $P_{uv} = \pi(u, v)$ with $\pi(u, v) \geq \varepsilon/2$, it follows that $P_{uv} \leq \pi(u, v) \leq \text{PPR}(u, v)$, and similarly

$$P_{uv} \geq \pi(u, v) - \varepsilon/2 \geq \text{PPR}(u, v) - \varepsilon/2 - \varepsilon/2 = \text{PPR}(u, v) - \varepsilon.$$

Therefore, we have $\text{PPR}(u, v) - \varepsilon \leq P_{uv} \leq \text{PPR}(u, v)$, and the first condition of SPPR is satisfied. To see that $P_{uv}$ satisfies the sparsity condition, note that we only take $P_{uv} = \pi(u, v)$ with $\pi(u, v) \geq \varepsilon/2$, which means $P_{uv} \geq \varepsilon/2$ for any $u, v \in V$. Since the summation $\sum_{v \in V} P_{uv}$ from a source node $u$ satisfies $\sum_{v \in V} P_{uv} \leq \sum_{v \in V} \pi(u, v) \leq \sum_{v \in V} \text{PPR}(u, v) = 1$, it follows that there are at most $2/\varepsilon$ non-zero $P_{uv}$ entries for a given source node $u$. Consequently, each row of $P$ contains no more than $2/\varepsilon$ non-zero entries, adding to a total of $2n/\varepsilon$ non-zero entries. Let $P'_{uv}$ be the increment to $P_{uv}$ in line 9. By a similar argument, we have $\text{PPR}^T(v, u) - \varepsilon \leq P'_{uv} \leq \text{PPR}^T(v, u)$, and thus

$$\text{PPR}(u, v) + \text{PPR}^T(v, u) - 2\varepsilon \leq P_{uv} \leq \text{PPR}(u, v) + \text{PPR}^T(v, u).$$

Finally, there are at most $2/\varepsilon$ non-zero $P'_{uv}$'s for a given target node $v$, which means the backward pushes on $G^T$ adds at most $2n/\varepsilon$ non-zero entries, resulting at most $4n/\varepsilon$ non-zero entries in the final proximity matrix $P$. Note that despite its sparsity, the final proximity matrix is not row-sparse or column-sparse, and thus is able to capture nodes with large in- or out-degrees. □

**Achieving non-linearity.** After obtaining the sparse proximity matrix $P$, we perform logarithm to each non-zero entry in $P$ (Line 10). It has been shown in [44] and [32] that skip-gram based algorithms implicitly factorize the logarithm of certain proximity matrix, where taking entry-wise logarithm simulates the effect of the softmax function. We also multiply the proximity by $2/\varepsilon$ inside the logarithm, such that all entries of $P$ remains non-negative after we take entry-wise logarithm.

### 3.4 Sparse Randomized SVD

We perform truncated singular value decomposition (tSVD) to decompose the proximity matrix $P$ into three matrices $U$, $\Sigma$, and $V$, where $U$ and $V$ are $n \times d$ unitary matrices, and $\Sigma$ is a diagonal

matrix. It is folklore that the reconstruction matrix $U\Sigma V^T$ is the best-$d$ approximation to matrix $P$, i.e.

$$\|P - U\Sigma V^T\|_F = \min_{rank(B)\leq d} \|P - B\|_F,$$

where $\|A\|_F$ denote the Fobenius norm of matrix $A$, After the decomposition, we can return $U\sqrt{\Sigma}$ and $V\sqrt{\Sigma}$ as the content/context embedding vectors (Lines 11-12).

However, applying traditional truncated SVD to a $n \times n$ matrix requires $O(n^2 d)$ time, which is not feasible when $n$ is large. To reduce this time complexity, we make use of the fact that $P$ is a sparse matrix with at most $4n/\varepsilon$ non-zero entries. In particular, we use *Sparse Subspace Embedding* [13], which allows us to decompose $P$ into three matrices $U'$, $\Sigma'$ and $V'$, where $U'$ and $V'$ are $n \times d$ unitary matrices, and $\Sigma'$ is a diagonal matrix, such that

$$\|P - U'\Sigma' V'^T\|_F \leq (1+\delta)\|P - U\Sigma V^T\|_F = (1+\delta) \min_{rank(B)\leq d} \|P - B\|_F.$$

In other words, the reconstructed matrix $U'\Sigma' V'^T$ is an $(1+\delta)$-approximation to the best-$d$ approximation of $P$. [13] shows that this decomposition can be performed in $O\left(nnz(P) + nd^2/\delta^4\right)$ time. Therefore, the complexity of SRSVD on our proximity matrix $P$ is bounded by $O\left(\frac{n}{\varepsilon} + nd^2/\delta^4\right)$. We set $\delta$ to be a constant so the running time is bounded by $O\left(\frac{n}{\varepsilon} + nd^2\right)$. Finally, we return $U'\sqrt{\Sigma'}$ and $V'\sqrt{\Sigma'}$ as the content/context embedding vectors. Note that for undirected graph, the proximity matrix $P$ is a symmetric matrix, in which case SVD on $P$ is equivalent to eigendecomposition on $P$.

**Running time and parallelism.** By Lemma 3.1, the total running time for the backward push is $O\left(n \cdot \frac{m}{n \cdot r_{max}}\right) = O\left(\frac{m}{\varepsilon}\right)$. Combining the running time $O\left(\frac{n}{\varepsilon} + nd^2\right)$ for randomized SVD, it follows that the running time of STRAP is bounded by $O\left(\frac{m}{\varepsilon} + nd^2\right)$. We can provide tradeoffs between scalability and accuracy by manipulating the error parameter $\varepsilon$. In particular, as we decrease $\varepsilon$, we tradeoff running time for more accurate embeddings. In practice, the backward push part can be trivially parallelized by running backward push algorithms on multiple nodes at the same time. To parallelize the SVD part, we use frPCA [17], a parallel randomized SVD algorithm designed for sparse matrices.

## 4 EXPERIMENTS

This section experimentally evaluates STRAP against the states of the art. All experiments are conducted on a machine with a Xeon(R) E7-4809@2.10GHz CPU and 320GB memory.

### 4.1 Experimental Settings

**Datasets.** We employ seven widely-used datasets, as shown in Table 3. BlogCatalog, Flickr and YouTube are three undirected social networks where nodes represent users and edges represent relationships between users. WikiVote is the directed Wikipedia who-votes-on-whom network. Slashdot is the directed social network of Slashdot.com. Brazil and Euro [33] are two airport networks with nodes as airports and edges as commercial airlines. All data sets are obtained from public sources [1–3].

**Competitors and Parameter Setting.** Unless otherwise specified, we set the embedding dimensionality $d$ to 128 in line with previous research [31, 37, 43]. For STRAP, we set the error parameter $\varepsilon = 0.00001$, so that the running time of our method is comparable to that of the fastest competitor. The decay factor $\alpha$ is set to be 0.5 to balance the tradeoff between transductive and inductive effectiveness. We evaluate STRAP against several state-of-the-art graph embedding algorithms. We obtain the source code of these methods from GitHub and present their results with the authors' preferred parameters.

**Table 3: Data Sets.**

| Data Set | Type | $n$ | $m$ |
|---|---|---|---|
| BlogCatalog (BC) | undirected | 10,312 | 333,983 |
| Flickr (FL) | undirected | 80,513 | 5,899,882 |
| Youtube (YT) | undirected | 1,138,499 | 2,990,443 |
| WikiVote (WV) | directed | 7,115 | 103,689 |
| Slashdot (SD) | directed | 82,168 | 870,161 |
| Euro | undirected | 399 | 5,993 |
| Brazil | undirected | 131 | 1,003 |

- DeepWalk[1] [31] uses truncated random walks and the skip-gram model to learn embedding vectors. We use the parameters suggested in [31]: window size 10, walk length 40, and the number of walks from each node to be 80.
- Node2Vec[2] [19] generalizes DeepWalk by adopting potentially biased random walks. We set the bias parameters $p = q = 1$ and use the default settings for other parameters.
- HOPE[3] [29] uses sparse SVD to decompose the proximity matrix of form $M_g^{-1} M_\ell$. The default HOPE uses Katz similarity as its proximity measure. Since Katz does not converge on directed graphs with sink nodes, we evaluate HOPE with Personalized PageRank and set decay factor $\alpha = 0.5$ as suggested in [29].
- VERSE[4] [37] is a random walk method that uses Personalized PageRank and SimRank as the proximities. The paper presents two methods, VERSE, which simulates $\alpha$-discounted random walk to train the skip-gram model, and fVERSE, which directly computes pair-wise PPR and SimRank. We exclude fVERSE due to its $\Theta(n^2)$ complexity. Following the suggestion in [37], we set the number of iterations to be $10^5$. We set the decay factor $\alpha$ to the default value 0.85 [37], as we have experienced performance drop for $\alpha = 0.5$. VERSE supports directed graphs by producing asymmetric content/context embedding vectors $\mathbf{s}$ and $\mathbf{t}$ [37].
- APP[5] [44] uses $\alpha$-discounted random walk to train asymmetric content/context embedding vectors for each node using the skip-gram model. We set the number of samples per node to 200 and the decay factor $\alpha$ to 0.15, as suggested in [44].
- AROPE[6] [43] is a factorization method that preserves the polynomial-shaped proximity matrix $P = \sum_{i=1}^{q} w_i A^i$. AROPE includes HOPE and LINE as its special cases [43] and achieves high scalability as it only performs eigen-decomposition to the (sparse) adjacency matrix. However, AROPE does not allow non-linear operations on the proximity matrix $P$, and it only works for undirected graphs due to the usage of eigen-decomposition. We use one of the default set: $q = 3$ and $w = \{1, 0.1, 0.01\}$.

[1] https://github.com/phanein/deepwalk
[2] https://github.com/aditya-grover/node2vec
[3] https://github.com/ZW-ZHANG/HOPE
[4] https://github.com/xgfs/verse
[5] https://github.com/AnryYang/APP
[6] https://github.com/ZW-ZHANG/AROPE

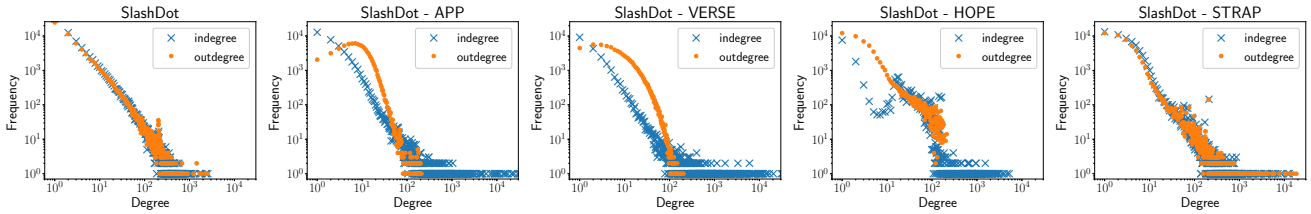


**Figure 2: Degree distributions of SlashDot.**

**Remark.** Note that Node2Vec, VERSE, and AROPE are able to generate multiple embedding vectors with varying parameters for each node. For example, AROPE sets $q = 1, 2, 3, -1$ and various weights $w$ to generate multiple vectors for each node, and chooses the best-fit embedding vector for a specific task using cross-validation. Our method also allows multiple embeddings by a grid search on the decay factor $\alpha$ and error parameter $\varepsilon$. However, we argue that a fair comparison should evaluate the embedding vectors generated by a single set of parameters over various tasks. As a counter-example, we note that one set of default parameters ($q = 1$) in AROPE is to directly decompose the adjacency matrix, which naturally gives the best graph reconstruction result. However, the adjacency matrix performs poorly for inductive tasks such as link prediction or node classification, and thus AROPE will select the embeddings generated by a larger $q$ for these tasks. Our method can achieve similar results by setting $\alpha$ close to 1 for graph reconstruction and a smaller $\alpha$ for link prediction or node classification (see Figure 3). However, this would not be fair to other methods with fixed parameters. Therefore, we believe that the only fair way to compare these methods is to evaluate them in both transductive and inductive tasks with consistent parameters. We also include ADJ-SVD, the method that directly applies SVD to the adjacency matrix, to demonstrate the above argument. For each task, we run each method ten times and report the average of its score.

## 4.2 Graph Reconstruction

We perform graph reconstruction task to see if the low-dimensional representation can accurately reconstruct the adjacency matrix. For each method, we train embedding vectors and rank pairs of nodes $(u, v)$ according to the inner product $\mathbf{s}_u \cdot \mathbf{t}_v$, where $\mathbf{s}_u$ and $\mathbf{t}_v$ are the content and context embedding vectors of node $u$ and $v$, respectively. We then take the top-$m$ pairs of nodes (removing self-loop) to reconstruct the graph, where $m$ is the number of edges in the original graph.

**Degree distributions on directed graphs.** Figure 2 shows the degree distribution of the directed graph SlashDot and the reconstructed graphs by HOPE, APP, VERSE and STRAP. We exclude Node2Vec and DeepWalk as they generate identical in- and out-degree distributions. Similar to the results on WikiVote, STRAP is the only method that can generate out-degree distribution similar to that of the original graph, which concurs with our theoretical analysis for transpose proximity.

**Reconstruction precision.** We calculate the ratio of real links in top-$m$ predictions as the reconstruction precision. Table 4 shows the results the reconstruction precision for each dataset. As expected, ADJ-SVD achieves the highest precision on all graphs. For other methods, we observe that STRAP significantly outperforms all existing methods. The advantage of STRAP becomes more obvious on directed graphs WikiVote and SlashDot, which demonstrates the effectiveness of transpose proximity.

**Table 4: Graph Reconstruction Precision (%).**

| Method | BC | FL | YT | WV | SD |
|---|---|---|---|---|---|
| DeepWalk | 5.08 | 4.86 | 0.68 | 1.64 | 3.45 |
| Node2Vec | 6.53 | 2.85 | 0.13 | 4.19 | 0.15 |
| HOPE | 21.85 | 14.90 | 8.78 | 10.98 | 8.61 |
| APP | 18.50 | 19.95 | 12.34 | 10.85 | 11.91 |
| VERSE | 40.03 | 20.22 | 6.09 | 20.89 | 10.73 |
| AROPE | 37.06 | 26.21 | 24.50 | NA | NA |
| STRAP | 52.32 | 34.92 | 27.18 | 55.29 | 24.42 |
| ADJ-SVD | 59.53 | 41.34 | 31.81 | 74.15 | 30.87 |

## 4.3 Link Prediction

An important inductive application of graph embedding is predicting unobserved links in the graph. To test the performance of different embedding methods on this task, we randomly hide 50% of the edges as positive samples for testing and sample the same number of non-existing edges as negative examples. We then train embedding vectors on the rest of the 50% edges and predict the most likely edges which are not observed in the training data from the learned embedding. Table 5 reports the precision of each method. We observe that STRAP is consistently the best predictor on all datasets except YouTube, on which VERSE takes the lead by 1%. We also note that STRAP significantly outperforms the state-of-the-art factorization methods AROPE and HOPE, and we attribute this quality to the non-linearity of our methods.

**Table 5: Link Prediction Precision (%).**

| Method | BC | FL | YT | WV | SD |
|---|---|---|---|---|---|
| DeepWalk | 53.59 | 70.26 | 63.46 | 66.72 | 65.42 |
| Node2Vec | 63.58 | 57.26 | 54.55 | 56.81 | 53.37 |
| HOPE | 79.97 | 86.75 | 67.23 | 85.67 | 84.11 |
| APP | 78.19 | 81.69 | 63.11 | 61.44 | 72.77 |
| VERSE | 87.99 | 90.13 | 67.51 | 86.39 | 83.99 |
| AROPE | 88.09 | 88.78 | 65.43 | NA | NA |
| STRAP | 88.92 | 91.49 | 66.86 | 91.79 | 84.47 |
| ADJ-SVD | 76.36 | 89.27 | 59.31 | 74.02 | 62.77 |

To demonstrate the effect of the training ratio, we also report the precisions of each method with varying training/testing ratio on BlogCatalog. We observe that our method consistently outperforms existing methods for all training ratios, with VERSE being the closest competitor.

## 4.4 Node Classification

Node classification aims to predict the correct node labels in a graph. Following the same experimental procedure in [31], we randomly sample a portion of labeled vertices for training and use the rest for testing. The training ratio is varied from 10% to

**Table 6: Link Prediction Precision (%) for BlogCatalog.**

| Method | 10% | 30% | 50% | 70% | 90% |
|---|---|---|---|---|---|
| DeepWalk | 61.77 | 54.62 | 53.59 | 53.53 | 53.41 |
| Node2Vec | 57.32 | 63.77 | 63.58 | 62.57 | 63.66 |
| HOPE | 68.24 | 72.67 | 79.97 | 81.63 | 83.45 |
| APP | 53.49 | 70.91 | 78.19 | 77.31 | 78.67 |
| VERSE | 83.73 | 86.38 | 87.99 | 88.74 | 89.52 |
| AROPE | 80.77 | 87.37 | 88.09 | 88.35 | 88.49 |
| STRAP | 84.78 | 87.40 | 88.92 | 89.92 | 90.42 |
| ADJ-SVD | 57.12 | 72.86 | 76.36 | 80.36 | 83.30 |

90%. We use LIBLINEAR [16] to perform logistic regression with default parameter settings. To avoid the thresholding effect [36], we assume that the number of labels for test data is given [31]. The performance of each method is evaluated in terms of average Micro-F1 and average Macro-F1 [38], and we only report Micro-F1 as we experience similar behaviors with Macro-F1. Table 7 and Table 8 show the node classification results on BlogCatalog and Flickr. Surprisingly, DeepWalk outperforms all successors other than STRAP on both datasets. On the other hand, STRAP is able to achieve comparable precision to that of DeepWalk. In particular, STRAP significantly outperforms HOPE and AROPE, which again demonstrates the effectiveness of the non-linearity.

**Table 7: Node Classification on BlogCatalog.**

| Method | 10% | 30% | 50% | 70% | 90% |
|---|---|---|---|---|---|
| DeepWalk | 35.93 | 39.65 | 40.86 | 41.93 | 43.31 |
| Node2Vec | 34.60 | 38.27 | 39.31 | 40.14 | 40.36 |
| HOPE | 16.68 | 17.85 | 17.92 | 19.23 | 20.18 |
| APP | 28.09 | 31.63 | 33.31 | 33.71 | 33.49 |
| VERSE | 31.48 | 35.96 | 38.32 | 39.64 | 40.49 |
| AROPE | 27.01 | 30.98 | 31.89 | 32.76 | 32.94 |
| STRAP | 36.42 | 40.29 | 41.59 | 42.68 | 42.55 |
| ADJ-SVD | 23.15 | 28.42 | 29.75 | 31.83 | 31.85 |

**Table 8: Node Classification on Flickr.**

| Method | 10% | 30% | 50% | 70% | 90% |
|---|---|---|---|---|---|
| DeepWalk | 38.96 | 40.83 | 41.54 | 41.85 | 42.08 |
| Node2Vec | 38.15 | 39.85 | 40.60 | 41.06 | 41.34 |
| HOPE | 16.39 | 16.59 | 16.59 | 16.67 | 16.56 |
| APP | 33.15 | 35.29 | 35.99 | 36.23 | 36.54 |
| VERSE | 34.54 | 37.10 | 38.07 | 38.57 | 38.83 |
| AROPE | 29.56 | 30.62 | 30.89 | 31.27 | 31.73 |
| STRAP | 39.32 | 41.00 | 41.47 | 41.77 | 42.06 |
| ADJ-SVD | 24.52 | 26.59 | 27.22 | 27.54 | 27.97 |

**Node Structural Role Classification.** We also perform node structural role classification task [33, 43] on Brazil and Euro, two airport networks with nodes as airports and edges as commercial airlines. The goal is to assign each node a label from 1 to 4 to indicate the level of activities of the corresponding airports. Due to the size of the graphs, we set the dimension $d = 16$ for this particular task. Table 9 and Table 10 shows the node structural role classification results on the two graphs, respectively. Again, our method performs comparably well. This suggests that STRAP preserves the structural role of the graphs. We also observe that DeepWalk, the main competitor in the previous task, achieves unsatisfying results, while our method performs consistently on two very different tasks.

**Table 9: Node Structural Role Classification on Brazil.**

| Method | 10% | 30% | 50% | 70% | 90% |
|---|---|---|---|---|---|
| DeepWalk | 25.42 | 32.61 | 27.27 | 25.00 | 21.43 |
| Node2Vec | 36.44 | 41.30 | 42.42 | 37.50 | 50.00 |
| HOPE | 22.88 | 20.65 | 21.21 | 32.50 | 28.57 |
| APP | 24.58 | 35.87 | 36.36 | 40.00 | 28.57 |
| VERSE | 30.51 | 32.61 | 31.82 | 42.50 | 35.71 |
| AROPE | 39.83 | 47.83 | 50.00 | 60.00 | 64.29 |
| STRAP | 37.29 | 51.74 | 52.42 | 59.50 | 70.71 |
| ADJ-SVD | 38.98 | 43.48 | 46.97 | 62.50 | 64.29 |

**Table 10: Node Structural Role Classification on Euro.**

| Method | 10% | 30% | 50% | 70% | 90% |
|---|---|---|---|---|---|
| DeepWalk | 26.94 | 26.79 | 24.00 | 30.00 | 35.00 |
| Node2Vec | 37.78 | 40.00 | 39.00 | 40.83 | 50.00 |
| HOPE | 25.00 | 27.50 | 20.50 | 23.33 | 30.00 |
| APP | 26.11 | 32.14 | 28.50 | 38.33 | 42.50 |
| VERSE | 33.89 | 39.29 | 43.50 | 45.83 | 42.50 |
| AROPE | 42.50 | 41.43 | 41.50 | 60.83 | 65.00 |
| STRAP | 47.56 | 44.79 | 48.75 | 61.42 | 65.50 |
| ADJ-SVD | 42.78 | 43.57 | 43.50 | 53.33 | 65.00 |

## 4.5 Running Time and Scalability

Table 11 reports the wall-clock time of each method, with thread number bounded by 24. In general, our method achieves the same level of scalability as AROPE does, and significantly outperforms all random walk methods.

**Table 11: Running time (s).**

| Method | BC | FL | YT | WV | SD |
|---|---|---|---|---|---|
| DeepWalk | 1.2e3 | 1.3e4 | 1.7e5 | 7.3e2 | 1.2e4 |
| Node2Vec | 2.8e2 | 6.4e4 | 3.4e4 | 1.1e2 | 6.2e3 |
| HOPE | 3.5e2 | 2.5e3 | 1.9e5 | 2.3e2 | 2.5e3 |
| APP | 8.9e2 | 7.2e3 | 1.7e5 | 6.2e2 | 9.3e3 |
| VERSE | 2.7e2 | 2.4e3 | 3.6e4 | 1.1e2 | 1.7e3 |
| AROPE | 2.4e1 | 1.3e2 | 1.0e3 | NA | NA |
| STRAP | 3.9e1 | 7.5e2 | 2.1e3 | 6.0e0 | 2.4e2 |

## 4.6 Parameter Analysis

We study the effect of decay factor $\alpha$ and error parameter $\varepsilon$. Figure 3 shows how graph reconstruction and link prediction precisions behave as we vary $\alpha$ from 1 to 0. The results show that $\alpha$ provides tradeoffs between inductive and transductive effectiveness: for $\alpha$ close to 1, the proximity matrix focuses on one-hop neighbors and thus preserves the adjacency information. As $\alpha$ approaches 0, the information of multi-hop neighbors will be added to the proximity matrix, trading transductive strength for inductive strength. Figure 4 shows how running time and graph reconstruction precisions behave as we vary error parameter $\varepsilon$. It shows that $\varepsilon$ controls the tradeoff between precision and running time. As we decrease $\varepsilon$, we tradeoff running time for more accurate embedding vectors.

# 5 CONCLUSION

In this paper, we propose transpose proximity, a unified approach that allows graph embeddings to preserve both in- and out-degree distributions on directed graphs and to avoid the conflicting optimization goals on undirected graphs. Based on the concept of transpose proximity, we present STRAP, a factorization method

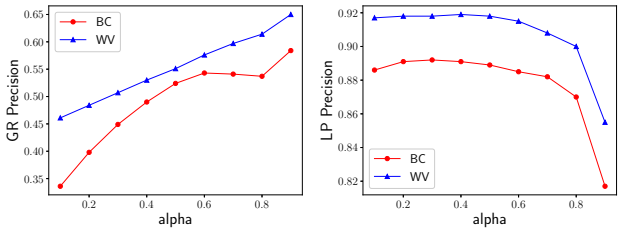


**Figure 3: Graph reconstruction and link prediction precisions with varying $\alpha$.**

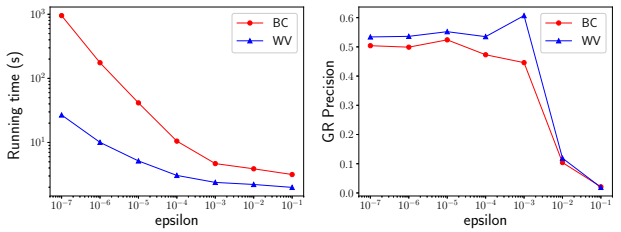


**Figure 4: Running time and graph reconstruction precisions with varying $\varepsilon$.**

that achieves both scalability and non-linearity on large graphs. The theoretical analysis shows that the running time of our algorithm is linear to the number of edges in the graph. The experimental results show by using transpose proximity, STRAP outperforms competitors in both transductive and inductive tasks, while achieving satisfying scalability. As future work, an interesting open problem is to study how to combine transpose proximity with the skip-gram model for better parallelism and predictive strength.

## 6 ACKNOWLEDGEMENTS

This research was supported in part by National Natural Science Foundation of China (No. 61832017 and No. 61732014) and by the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 18XNLG21.

## REFERENCES

[1] http://snap.stanford.edu/data/index.html.
[2] http://law.di.unimi.it/datasets.php.
[3] https://github.com/leoribeiro/struc2vec/.
[4] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *WWW*, pages 37–48. ACM, 2013.
[5] Rodrigo Aldecoa, Chiara Orsini, and Dmitri Krioukov. Hyperbolic graph generator. *Computer Physics Communications*, 196:492–496, 2015.
[6] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *VLDB*, 4(3):173–184, 2010.
[7] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, pages 585–591, 2002.
[8] Mansurul Bhuiyan and Mohammad Al Hasan. Representing graphs as bag of vertices and partitions for graph classification. *Data Science and Engineering*, 3(2):150–165, 2018.
[9] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *CIKM*, pages 891–900. ACM, 2015.
[10] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016.
[11] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. Heterogeneous network embedding via deep architectures. In *SIGKDD*, pages 119–128. ACM, 2015.
[12] Siheng Chen, Sufeng Niu, Leman Akoglu, Jelena Kovačević, and Christos Faloutsos. Fast, warped graph embedding: Unifying framework and one-click algorithm. *arXiv preprint arXiv:1702.05764*, 2017.
[13] Kenneth L Clarkson and David P Woodruff. Low-rank approximation and regression in input sparsity time. *JACM*, 63(6):54, 2017.
[14] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *TKDE*, 2018.
[15] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *SIGKDD*, pages 135–144. ACM, 2017.
[16] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *JMLR*, 9(Aug):1871–1874, 2008.
[17] Xu Feng, Yuyang Xie, Mingye Song, Wenjian Yu, and Jie Tang. Fast randomized pca for sparse data. In *Asian Conference on Machine Learning*, pages 710–725, 2018.
[18] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.
[19] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864. ACM, 2016.
[20] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *SIGKDD*, pages 538–543, 2002.
[21] Junghwan Kim, Haekyu Park, Ji-Eun Lee, and U Kang. Side: representation learning in signed directed networks. In *WWW*, pages 509–518. International World Wide Web Conferences Steering Committee, 2018.
[22] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.
[23] Kyle Kloster and David F Gleich. Heat kernel based community detection. In *SIGKDD*, pages 1386–1395. ACM, 2014.
[24] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguná. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010.
[25] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Personalized pagerank estimation and search: A bidirectional approach. In *WSDM*, pages 163–172, 2016.
[26] Jianxin Ma, Peng Cui, and Wenwu Zhu. Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks. AAAI, 2018.
[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.
[28] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023, 2016.
[29] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *SIGKDD*, pages 1105–1114. ACM, 2016.
[30] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.
[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *SIGKDD*, pages 701–710. ACM, 2014.
[32] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*, pages 459–467. ACM, 2018.
[33] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *SIGKDD*, pages 385–394. ACM, 2017.
[34] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.
[35] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.
[36] Lei Tang, Suju Rajan, and Vijay K Narayanan. Large scale multi-label classification via metalabeler. In *WWW*, pages 211–220. ACM, 2009.
[37] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *WWW*, pages 539–548. International World Wide Web Conferences Steering Committee, 2018.
[38] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. In *Data mining and knowledge discovery handbook*, pages 667–685. Springer, 2009.
[39] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *KDD*, pages 1225–1234. ACM, 2016.
[40] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. Topppr: top-k personalized pagerank queries with precision guarantees on large graphs. In *SIGMOD*, pages 441–456. ACM, 2018.
[41] Shuhan Yuan, Xintao Wu, and Yang Xiang. Sne: signed network embedding. In *PAKDD*, pages 183–195. Springer, 2017.
[42] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE transactions on Big Data*, 2018.
[43] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. Arbitrary-order proximity preserved network embedding. In *SIGKDD*, pages 2778–2786. ACM, 2018.
[44] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. Scalable graph embedding for asymmetric proximity. In *AAAI*, pages 2942–2948, 2017.
[45] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. 2018.