Scalable Graph Embedding for Asymmetric Proximity

Chang Zhou,[1*] **Yuqiong Liu,**[1] **Xiaofei Liu,**[2] **Zhongyi Liu,**[2] **Jun Gao**[1*]
[1]Key Laboratory of High Confidence Software Technologies, EECS, Peking University
{zhouchang,liuyuqiong,gaojun}@pku.edu.cn *Corresponding Authors
[2]Alibaba Group, {hsiaofei.hfl,zhongyi.lzy}@alibaba-inc.com

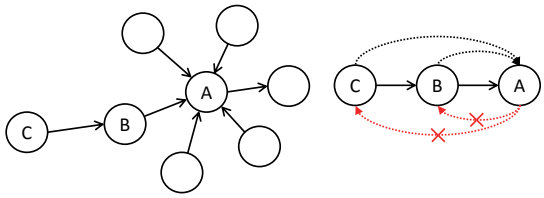
Abstract

Graph Embedding methods are aimed at mapping each vertex into a low dimensional vector space, which preserves certain structural relationships among the vertices in the original graph. Recently, several works have been proposed to learn embeddings based on sampled paths from the graph, *e.g.,* DeepWalk, Line, Node2Vec. However, their methods only preserve symmetric proximities, which could be insufficient in many applications, even the underlying graph is undirected. Besides, they lack of theoretical analysis of what exactly the relationships they preserve in their embedding space. In this paper, we propose an asymmetric proximity preserving *(APP)* graph embedding method via random walk with restart, which captures both asymmetric and high-order similarities between node pairs. We give theoretical analysis that our method implicitly preserves the Rooted PageRank score for any two vertices. We conduct extensive experiments on tasks of link prediction and node recommendation on open source datasets, as well as online recommendation services in Alibaba Group, in which the training graph has over 290 million vertices and 18 billion edges, showing our method to be highly scalable and effective.

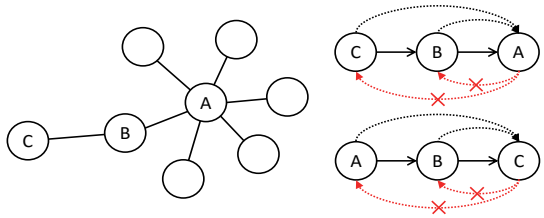
Introduction

In recent years, Graph embedding has drawn a lot of attentions from the academic fields, due to its wide usage in many real world applications such as Recommendations (Barkan and Koenigstein 2016; Zhang et al. 2016), Social Network Analysis (Perozzi, Al-Rfou, and Skiena 2014), Natural Language Processing (Tang et al. 2015) and Knowledge Bases (Lin et al. 2015; Wang et al. 2014). Graph Embedding techniques try to embed each vertex from a graph into a low dimensional vector space, which preserves the structural similarities or distances among the vertices in the original graph. By doing that, we can represent a graph vertex in a vector form, and use well-studied machine learning methods in the vector space to do further mining tasks like clustering, classification and prediction.

There are many ways of learning node representations recently, most of which adopt a random walk based sampling procedure to exploit the network structure, *e.g.,* Deepwalk (Perozzi, Al-Rfou, and Skiena 2014), Node2Vec (Grover and Leskovec 2016). However, none of them can preserve asymmetric proximities in both undirected and directed graphs, which may be critical in many tasks, *e.g.,* link prediction in social network, recommendation in e-commerce. In addition, they do not give theoretical analysis of what exactly the proximity they preserve by embedding the vertex from the original graph structure.



(a) Directed



(b) Undirected

Figure 1: Asymmetric Proximity in both Directed and Undirected Graphs. Intuitively, $Sim(A, C)$ is not equal with $Sim(C, A)$ in both cases, due to their asymmetric local structures. Even for undirected graphs, the learning model should treat the two sampled paths $C \rightarrow B \rightarrow A$ and $A \rightarrow B \rightarrow C$ differently as directed sequences, since the probabilities of this two sampled paths are quite different, which means the occurrence number of (C, A) and (A, C) in a graph could vary a lot in many real world applications that consider the asymmetric relations.

For instance, for the application of friends recommendation in social network like Twitter, the underlying directed graph naturally has an asymmetric similarity or closeness metric between any two nodes. As shown in Figure 1(a), A has the potential to be liked by C while it's not true the other way round. Even for the undirected graph shown in Figure 1(b), we can intuitively see that A is much more important

for C than C is for A, since the probability that A arrives at C is far less than the one that C arrives at A, due to their asymmetric local structures.

For the undirected graph in Figure 1(b), let p_1 be the path of $C \to B \to A$ sampled from C, and p_2 be the path of $A \to B \to C$ sampled from A. We can see that, the probabilities of the p_1 (0.5) and p_2 (0.08) are quite different, while the methods proposed in (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016) all regard these two sampled paths as the same, which take the path as a sequence of words, and use skip-gram model proposed in (Mikolov et al. 2013) to update the embedded node vectors. Since p_1 alone does not necessarily mean the probability that A can predict C deserves as high as the probability of sampling this path, the proximity of (A, C) could be over-estimated by their methods.

Another important question about these graph embedding methods is that, what exactly is the relationships they preserve by embedding each vertex in the new latent space? Node similarity has been studied for a very long time, and there are plenty of ways defining node similarities or closenesses, as can be found in (Liben-Nowell and Kleinberg 2007). A correlation with the traditional similarity measurement will give us a better understanding about what properties of the embeddings should have and how the method will behave.

In this paper, we propose a scalable asymmetric proximity preserving (*APP* for short) graph embedding method based on random walk with restart, by only allowing stochastic gradient updates on the forward direction of the sampled path. We give a theoretical analysis that the asymmetric structural relationship we try to preserve from the original graph is the Rooted PageRank proximity between any two vertices. We conduct extensive experiments to show the effectiveness of our method for link prediction tasks on open source real world datasets, and we also evaluate our method for the recommendation tasks in one of the online services of Alibaba Group, in which the training graph has over 290 million vertices and 18 billion edges.

Related Work

Graph Embedding Methods

Graph embedding can be viewed as an dimensionality reduction approach, which maps each vertex into a latent vector space while preserves the topological proximities, *e.g.,* similarities and distances, of vertex pairs introduced by the original graph.

These pair-wise proximities can be expressed as matrices, *e.g.,* the first-order adjacency matrix and the higher-order predefined node similarity matrix. While both linear and non-linear dimensionality reduction methods such as PCA and IsoMap have been studied extensively in the literature (Wold, Esbensen, and Geladi 1987; Tenenbaum, De Silva, and Langford 2000; Roweis and Saul 2000), they suffer from severe computational problems and therefore cannot scale to large graphs.

Recently, more computationally efficient models in word representation learning for natural language processing are proposed (Mikolov et al. 2013; Pennington, Socher, and Manning 2014). The Skip-Gram model introduced by (Mikolov et al. 2013) has been also used in the graph embedding community recently. DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) samples multiple paths from the graph, each of which is regarded as a word sequence. For each vertex in the sequence, it predicts the nearby vertices in both direction, and updates the vector according to the Skip-Gram model. It cannot capture asymmetric relationships in a graph, which restricts its applications. Line (Tang et al. 2015) introduces the 2-nd order proximity between a pair of vertices, which encodes the similarity measured by their local neighborhood. It samples individual edges in the graph, and updates the corresponding vector according to SGD in the Skip-Gram model. Line is also unable to capture the asymmetric relationships in undirected graphs since it will update node vectors from both sides of the sampled edges. In addition, node pairs from two hop away will be regarded as negative labels, which makes it fail to preserve the higher-order similarities. Higher order proximity is considered by many traditional similarity measurements, *e.g.,* SimRank (Jeh and Widom 2002), Rooted PageRank (Haveliwala 2002), Katz (Katz 1953), which have been proved to be effective in many real world tasks. Node2vec offers a flexible sampling strategy, with two parameters controlling the shape of the sampled paths. However, it still ignores the asymmetric nature of the path sampling procedure and trains the model symmetrically. Besides, none of them gives any theoretical analysis about what vertex-pair relations they preserve.

Asymmetric graph embedding is also studied in (Ou et al. 2016), which reformats and factorizes the node-similarity matrix in a graph, using a partial generalized SVD algorithm. However, the similarity metrics are pre-defined, *e.g.,* Katz, which is hard to generalize. In addition, the incremental update cannot be well supported by their matrix factorization method, which is crucial in highly dynamic applications, *e.g.,* Social Networks and Recommendation Systems.

Monte Carlo approach for Rooted PageRank

Rooted PageRank (Haveliwala 2002) can be regarded as a similarity measurement between two vertices, and it's widely studied in both information retrieving and data mining fields. Personalized PageRank vector $\overline{p}$ is defined as the solution of the following equation (Haveliwala 2002),

$$\overline{p} = (1 - c) \cdot \overline{p}A + c \cdot \overline{r} \tag{1}$$

Here A denotes the transition matrix of the graph with normalized rows and $c \in (0, 1)$ is the teleportation probability. In addition, $\overline{r}$ is a preference vector inducing a probability distribution over the vertices. For a Rooted PageRank vector w.r.t. v, $\overline{r}$ is a one-hot vector which is all zero except the position of v to be 1.

Due to the complexity of the iteration-style computation, Monte Carlo approach is usually adopted when calculating Rooted PageRank values (Gupta et al. 2013; Fogaras et al. 2005; Bahmani, Chowdhury, and Goel 2010; Avrachenkov et al. 2007). To compute a rooted pagerank vector for v, the Monte Carlo approach randomly samples N

independent paths started from v, with stoping probability of c. Then the rooted pagerank value can be approximated as,

$$ppr_v(u) = \frac{\#PathEndsAt(u)}{N} \tag{2}$$

The proof of the correctness and the bound can be found in (Fogaras et al. 2005).

Asymmetric Embedding Approach

To preserve the asymmetric proximity, each vertex v needs to have two different roles, the source role and the target role, represented by vector $\vec{s_v}$ and $\vec{t_v}$, respectively. For each sampled path which starts from u and ends with v, (u, v) is defined as a sampled vertex pair. We then define the probability that the source node u predicts the target node v as the softmax function:

$$p(v|u) = \frac{\exp(\vec{s_u} \cdot \vec{t_v})}{\sum_{n \in V} \exp(\vec{s_u} \cdot \vec{t_n})} \tag{3}$$

where V is the vertex set of the graph. However, the objective of Equation 3 is hard to optimize, due to the costly summation over all inner product with every vertex of the graph. To improve the training efficiency, we adopt the Skip-Gram with Negative Sampling(SGNS) method (Mikolov et al. 2013), which turns to optimize the following objective function for each observed (u, v) pair:

$$log\sigma(\vec{s_u} \cdot \vec{t_v}) + k \cdot E_{t_n \sim P_D}[log\sigma(-\vec{s_u} \cdot \vec{t_n})] \tag{4}$$

where we randomly samples k negative pairs according to a vertex distribution of $P_D(n)$. Let $\#Sampled_u(v)$ be the number of sampled pair (u, v), we can then write down the global objective as follows:

$$\ell = \sum_u \sum_v \#Sampled_u(v) \cdot (log\sigma(\vec{s_u} \cdot \vec{t_v}) + k \cdot E_{t_n \sim P_D}[log\sigma(-\vec{s_u} \cdot \vec{t_n})]) \tag{5}$$

where $\sigma(x) = 1/(1 + exp(-x))$ is the sigmoid function. Note that, we usually have $\#Sampled_u(v) \neq \#Sampled_v(u)$, which means the observed numbers of positive pairs of (u, v) and (v, u) are different. This asymmetric property is not captured by other random walk based embedding methods, in which the multipliers are the same for both (u, v) and (v, u). In graph, we sample a negative vertex according to the uniform distribution, $P_D(v) \sim \frac{1}{|V|}$, where $|V|$ is the total number of vertices.

Algorithm 1 describes our sampling and asymmetric learning strategy. We follow the Monte-Carlo End-Point sampling method mentioned in (Fogaras et al. 2005), $SampleEndPoint$ randomly samples a path p starting from v with a stopping probability of α, and returns the ending node u in p. This sampling method can be used to estimate Rooted PageRank value between each vertex pair, and we use this to simulate the probability of v arrives at u. Note that, this sampling procedure can be used in both undirected/directed and weighted/unweighted graphs. In weighted graphs, the transition probability between vertices is calculated using normalized weight.

Once we get the source and target vector of each vertex, we can represent the proximity of vertex pair (u, v) as the inner product of s_u and t_v.

Compared with methods like DeepWalk and Node2Vec, we treat each sampled path as a directed sequence, in which we observe positive vertex pairs only along the forward direction. This strategy strictly conforms to the nature of the asymmetric probabilities of path sampling, so that we can capture the asymmetric proximity caused by the difference of local graph structure, which will be further explained later in this section.

Algorithm 1 APP Embedding Algorithm

Input: G(V, E, W), Jumping Factor α, Learning Rate η
Output: Embedded Vector of $\vec{s_v}$, $\vec{t_v}$ for each $v \in V$
function PPREMBEDDING(G, α)
  Initialize: $\vec{s_v}$, $\vec{t_v}$, $\forall v \in V$
  for each $v \in V$ **do**
    for $i = 0; i < \#Sample; i++$ **do**
      u = SampleEndPoint(v)
      StochasticGradientDescent(v, u, 1)
      for $j = 0; j < k; j++$ **do**
        p = RandomUniform(V)
        StochasticGradientDescent(v, p, 0)
      end for
    end for
  end for
end function
function STOCHASTICGRADIENTDESCENT(v, u, $label$)
  $\vec{s_v}$ = $\vec{s_v}$ - η $(\sigma(\vec{s_v} \cdot \vec{t_u}) - label) \cdot \vec{t_u}$
  $\vec{t_u}$ = $\vec{t_u}$ - η $(\sigma(\vec{s_v} \cdot \vec{t_u}) - label) \cdot \vec{s_v}$
end function

Optimization with Path Sharing: The basic algorithm only produces one valid vertex pair in a single walk, which can be optimized by path sharing techniques. In fact, as a sampled path $p = v_1, v_2, ..., v_L$, any suffix subpath, *e.g.,* $v_2, ..., v_L$, can be viewed as an independently sampled path, thus producing (v_i, v_L) as positive samples, where $i = 1, 2, 3, ..., L-1$. Another strategy considers the prefix subpath as valid, *e.g.,* $v_1, ..., v_{L-1}$, which is proved to be effective in (Liu et al. 2016). These sampling-saving strategies can dramatically reduce the average number of samplings per vertex to $\frac{2R}{L(L-1)}$, where R is the samples number needed by Algorithm 1 and L is the expected walk length for each path, which is controlled by α.

Proof of Rooted PageRank Proximity Preserving: Now we prove that, our approach learns an embedding that implicitly preserves the Rooted PageRank score of any pair of vertices. Similar to (Levy and Goldberg 2014), for a sufficiently large vector dimensionality, we can regard the global objective as a function of each independent $\vec{s_u} \cdot \vec{t_v}$ term. And for any vertex u, the sum of rooted pagerank value ($Sim_u(v)$ here) over all the vertices equals 1, then we have,

$$
\begin{aligned}
\ell &= \sum_u \sum_v \#Sample \cdot Sim_u(v) \cdot (log\sigma(\vec{s_u} \cdot \vec{t_v}) \\
&\quad + \frac{k}{|V|} \sum_n log\sigma(-\vec{s_u} \cdot \vec{t_n})) \\
&= \#Sample\{\sum_u \sum_v Sim_u(v) \cdot log\sigma(\vec{s_u} \cdot \vec{t_v}) \\
&\quad + \sum_u \sum_n \sum_v Sim_u(v) \cdot \frac{k}{|V|} \cdot log\sigma(-\vec{s_u} \cdot \vec{t_n})\} \\
&= \#Sample\{\sum_u \sum_v Sim_u(v) \cdot log\sigma(\vec{s_u} \cdot \vec{t_v}) \\
&\quad + \sum_u \sum_n \frac{k}{|V|} \cdot log\sigma(-\vec{s_u} \cdot \vec{t_n})\} \\
&= \#Sample\{\sum_u \sum_v (Sim_u(v) \cdot log\sigma(\vec{s_u} \cdot \vec{t_v}) \\
&\quad + \frac{k}{|V|} \cdot log\sigma(-\vec{s_u} \cdot \vec{t_v}))\}
\end{aligned} \tag{6}
$$

To maximize this objective, we let the partial derivative of each independent $x = \vec{s_u} \cdot \vec{t_v}$ be zero,

$$
\begin{aligned}
\frac{\partial \ell}{\partial x} = \#Sample\{&-(Sim_u(v) + \frac{k}{|V|})\sigma(x) \\
&+ Sim_u(v)\}
\end{aligned} \tag{7}
$$

This gives us,

$$
x = log(\frac{|V| Sim_u(v)}{k}) \tag{8}
$$

According to Equation 8, since k and $|V|$ are both constants, we can see that $\vec{s_u} \cdot \vec{t_v}$ tries to preserve the logarithmic $Sim_u(v)$ with a constant shift, in which $Sim_u(v)$ in our case represents the Rooted PageRank value of v as for u.

Analysis: We only need $O(2|V|d + |E|)$ space overheads since we use the per-observation stochastic gradient updates on the fly as the random paths generate. And it's also easy to verify that the time complexity of our method is $O(|V|dRLk)$, where d is the dimension number, R is the number of samples per vertex, L is the expected sample length, and k is the iteration number. And if we use the path sharing strategy, the complexity drops at $O(|V|dRk/L)$. We can see that, our embedding method is both space and time efficient, which can scale to extremely large industrial graph data, as we evaluate in the experiment section.

An advantage of *APP* over the matrix factorization based ones is that, it does not need pre-computed proximity matrix, and it can naturally support incremental update, refine the model as the graph changes, which is a very nice property for the real world applications.

Extendability: We can see from Equation 6 that, if a proximity satisfies that $\sum_v Sim_u(v)$ is a constant number, and can also be approximated by Monte Carlo approach, then we can learn embedding which preserves this proximity within the same framework. We leave this topic as future works.

Experiment

In this section, we conduct extensive experiments to evaluate the performance of our method via tasks like link prediction and node recommendation.

Data Sets and Experiment Setting

Dataset. We collect four open-source data graphs to compare the performance of the graph embedding methods.

Arxiv [1]: Arxiv GR-QC is a collaboration network generated from the e-print arXiv. Nodes represent authors of papers and edges represent collaborations between authors.

Cora [2]: It is a citation network of academic papers where nodes represent academic papers and each directed edge indicates the citation relationship between papers.

Epinions [3]: This is the trust network from the online social network Epinions. Nodes are users of Epinions and directed edges represent trust between the users.

Amazon [4]: In Amazon network, nodes represent products and edges represent co-purchasing relation between products. We convert the original Amazon graph to an undirected graph, since the co-purchasing relation is symmetric.

We also evaluate our method on an extremely large private data AliItemGraph, which is an item graph converted by the item click sequence from the user browsing sessions. AliItemGraph has 290 million vertices (items) and over 18 billion edges. We learn the source and target embedded vectors for each item and use them for an online recommendation service in Alibaba Group, which we will describe in more detail later.

Some statistics about these graphs are summarized in Table 1.

DataSet	# Nodes	# Edges	Type
arxiv	5,242	28,980	undirected
cora	23,166	91,500	directed
epinions	75,879	508,837	directed
amazon	262,111	1,799,584	undirected
aliItemGraph	290million	18billion	directed

Table 1: Statistics of Graph DataSets

Competitors. We evaluate our method against the following methods that can measure the node-pair similarity: DeepWalk, Line, Node2Vec, Common Neighbors (CNbrs for short), Adamic Adar (Adar for short) and Jaccard Coefficience. For Deepwalk, Node2Vec and Line, we use the inner product of the embedded vectors to estimate the proximity of node pairs (u, v). And we use the inner product of s_u and t_v for *APP*. Note that, although Line also has two vectors learned for each vertex, namely a context vector and a vertex vector, we do not observe any improvement using both of them in our test, so we only use its vertex vectors. To have

[1] http://snap.stanford.edu/data/ca-GrQc.html

[2] http://konect.uni-koblenz.de/networks/subej_cora

[3] http://konect.uni-koblenz.de/networks/soc-Epinions1

[4] http://snap.stanford.edu/data/amazon0302.html

a fair comparison, the number of sampled paths of all these methods are made the same, and the number of dimensions is set to 128. In fact, we observe no significant improvement as the dimension size of our method goes beyond 32 for small datasets in both tasks. For Line, we use its 2-nd order proximity. For other methods, we calculate the similarity directly according to their similarity definitions, as illustrated in Table 2, where $N(v)$ represents the set of neighbors for v. For directed graphs, $N(v)$ could be the number of either incoming neighbors or outgoing neighbors.

Method	Definition
Common Nbrs	$\lvert N(u) \bigcap N(v) \rvert$
Jaccard	$\frac{\lvert N(u) \bigcap N(v) \rvert}{\lvert N(u) \bigcup N(v) \rvert}$
Adamic Adar	$\sum_{t \in N(u) \bigcap N(v)} \frac{1}{log \lvert N(t) \rvert}$

Table 2: Definitions of Traditional Node Similarity Metrics

Link Prediction

Given a network with some edges removed, link prediction aims to predict the occurrence of links, which is a fundamental problem in networks. For link prediction, we remove 30% of edges which are chosen randomly as ground truth in the test set, and take the remaining graph as the training set. We also randomly sample an equal number of node pairs that have no edge connecting them as negative samples in our test set.

We summarize the AUC scores for all methods in Table 3. We can see that, for link prediction task, the learning methods usually outperform the traditional methods in terms of the AUC metrics. This is because for Adar, CNbrs and Jaccard, there are too many zero similarities since they only exploit local structures, making it impossible to tell the negative pairs from the positive ones that are connected two more hops away. Among the learning approaches, our method *APP* achieves the best for all the datasets, which shows the benefit of preserving both asymmetric and higher-order proximities in the raw graph.

DataSet	arxiv	cora	epinions	amazon
Adar	0.8570	0.6431	0.7898	0.8079
CNbrs	0.8568	0.6430	0.7896	0.8078
Jaccard	0.8570	0.6428	0.7888	0.8078
DeepWalk	0.8865	0.9355	0.8226	0.9503
Line	0.7503	0.6936	0.8670	0.6844
Node2Vec	0.8101	0.7338	0.8647	0.9617
APP	**0.8873**	**0.9443**	**0.9256**	**0.9765**

Table 3: Area Under Curve (AUC) scores for Link Prediction

As the negative node pairs sampled above are so random that the impact of the asymmetric link may still be limited, we further change the way of generating negative samples to amplify the performance gaps among the models' abilities to predict asymmetric links. For directed graphs, *e.g.,* cora and epinions, we reverse the node pair in the positive samples and use it to replace an original negative sample if its edge is not bi-directional. After the replacement, 94% of the negative samples in *cora* are replaced with the non-existing reverse edges. And the number for *epinions* is 60%. The new results are listed in Table 4. Since the other three methods can only model the symmetric proximity, their AUC scores are near 0.5 as expected, especially for $cora$. And for APP, it's far better than the other three, as it considers asymmetric proximities.

DataSet	cora	epinions
DeepWalk	0.5360	0.6892
LINE	0.5130	0.6729
Node2Vec	0.5170	0.6889
APP	**0.7674**	**0.8076**

Table 4: Area Under Curve (AUC) scores on Biased Negative Samples

Node Recommendation

As for many applications like friends recommendation, instead of picking up the top-k highest intimate node-pairs from a large candidate random node-pairs, it's needed to calculate the top-k possible candidates for *each individual* person, to build up personalized services. So we evaluate the performance of these methods in node recommendation tasks. We remove 10% of the edges in the original graph as test set, and use the rest of the graph as the training data. We use $Precision@k$ and $Recall@k$ as the evaluation metrics for node recommendation methods, where

$$Precision@k = \frac{|PredSet \cap TestSet|}{|PredSet|}$$

$$Recall@k = \frac{|PredSet \cap TestSet|}{|TestSet|}$$

As illustrated in Table 5, we first observe that the existing embedding methods are not as good as traditional methods in node recommendation tasks. However, compared with other methods including the traditional ones, our method still achieves the overall best performance. And we can significantly outperform Line, DeepWalk and Node2vec, especially for the directed graphs. This shows the benefit of preserving asymmetric and high-order proximity in our method.

Note that, for traditional methods, *e.g.,* Adamic Adar, Commmon Neighbors, Jaccard, the size of their recommendation list could be smaller than k, since they cannot produce meaningful results when two nodes share no common neighbors, while embedding methods can always select the top-k value. This can explain that their precision is sometimes competitive with our method, especially when k becomes large, while our recall is significantly higher in most cases. In fact, we can setup a threshold for our method to remove the un-satisfied results in the top-k list, according to the ROC curve, which in practice may have a $10\% - 20\%$ improvement in precision.

DataSet	arxiv						amazon					
	P@10	R@10	P@20	R@20	P@50	R@50	P@10	R@10	P@20	R@20	P@50	R@50
CNbrs	0.092	0.493	0.065	0.635	0.044	0.793	0.080	0.551	0.052	0.677	0.034	0.773
Adar	0.110	0.587	**0.082**	0.712	**0.051**	0.819	0.084	0.579	0.056	0.686	0.036	0.753
Jaccard	0.053	0.259	0.042	0.319	0.033	0.364	0.085	0.579	0.056	0.686	**0.037**	0.735
DeepWalk	0.095	0.598	0.060	0.757	0.029	0.894	0.061	0.423	0.038	0.534	0.018	0.633
Line	0.080	0.551	0.054	0.659	0.024	0.772	0.021	0.100	0.012	0.172	0.011	0.350
Node2Vec	0.082	0.508	0.052	0.660	0.026	0.823	0.071	0.497	0.047	0.659	0.023	0.809
APP	**0.122**	**0.697**	0.065	**0.829**	0.035	**0.973**	**0.095**	**0.660**	**0.059**	**0.824**	0.027	**0.928**

Table 5: Precision and Recall for top-k Node Recommendation, Undirected Graph

DataSet	epinions						cora					
	P@10	R@10	P@20	R@20	P@50	R@50	P@10	R@10	P@20	R@20	P@50	R@50
CNbrs	0.023	0.061	0.017	0.093	0.011	0.148	0.023	0.142	0.017	0.198	0.011	0.268
Adar	**0.026**	0.072	**0.018**	0.103	0.012	0.161	0.023	0.142	0.017	0.198	0.011	0.268
Jaccard	0.021	0.057	0.016	0.086	0.011	0.137	0.021	0.134	0.016	0.185	0.011	0.259
DeepWalk	0.015	0.049	0.011	0.066	0.006	0.094	0.025	0.180	0.019	0.270	0.011	0.400
Line	0.004	0.014	0.004	0.025	0.003	0.049	0.012	0.084	0.009	0.124	0.005	0.184
Node2Vec	0.010	0.032	0.007	0.045	0.004	0.067	0.022	0.160	0.018	0.256	0.010	0.400
APP	0.024	**0.075**	**0.018**	**0.112**	**0.013**	**0.182**	**0.030**	**0.223**	**0.023**	**0.340**	**0.014**	**0.525**

Table 6: Precision and Recall for top-k Node Recommendation, Directed Graph

We also observe in our test that proximity measured by $\vec{s_u} \cdot \vec{t_v}$ can be 30% better than $\vec{s_u} \cdot \vec{s_v}$ for our method in this task, which further illustrates the importance of asymmetric proximity preserving.

Evaluation for Online Recommendation Services. We also evaluate our method on one of the personalized online recommendation services in Alibaba Group [5]. There are tens of millions of shops resided in Alibaba, and these shops will offer several item sets (S) on their homepages, each of which contains a small number (< 60) of candidate items according to their own marketing strategies. Our task is to expose the top 6 items within each item set to the customers with mobile devices, when they visit that shop. We train our method and the competitors with only the collaborative structural data, which is an item graph constructed by the item click sequences organized by user sessions, within the last 7 days. Note that, the graph contains both inter and inner shop item connections. Based on this offline training data, we use APP method to produce each vertex with two embedded vectors. We set the dimensions to be 200.

The online recommender follows the traditional item-based collaborative filtering (Sarwar et al. 2001; Adomavicius and Tuzhilin 2005), which takes the tested methods as different similarity measurements. For each customer u, u has a footprint item set T, consisting of the most recent items u has viewed. Then we score each item i in S according to the average proximity of each (t, i) pair, where $t \in T$.

Because the computing resources are limited, and the other embedding methods are not showing better performance than the traditional methods, *e.g.,* Adamic Adar, for node recommendation tasks in previous toy test, we only compare our method with traditional node-pair similarity measurement for their simplicity and efficiency. We directly compute the top-k list along with their proximity scores for each item where k is a small number, *e.g.,* 400.

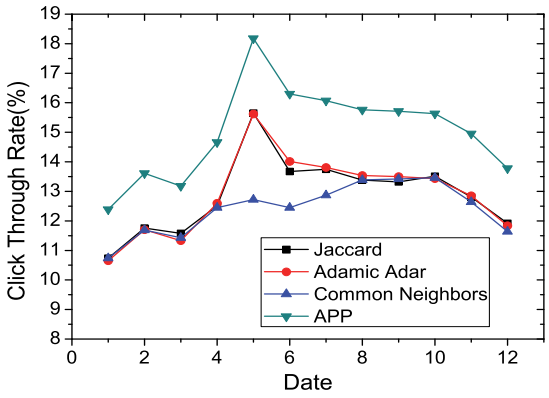


Figure 2: CTR for Item Recommendation

We run the A/B test for all these methods with the same traffic flow, and report the Click Through Rate(CTR) for a continuous 12 days, in Figure 2. We can see that our method is significantly better than the traditional similarity measurements for this service. Despite that our method can preserve asymmetric and higher-order proximities, APP can get much higher recall in this service, where traditional top-k methods suffer from severe problem that the top-k list is too small w.r.t. the number of shops such that each item in the footprint set has very low probability to match any of the item in the specific item set.

[5] https://www.taobao.com/

Conclusion

In this paper, we propose a scalable graph embedding method which can preserve asymmetric similarities between vertex pairs. We take the sampled path as a directed sequence which only observes positive vertex pair along the forward direction. Experiments show that our method is superior to the existing embedding methods in both link prediction and node recommendation tasks. And our method is highly scalable that it has been tested on an industrial-scale large graph with hundreds of millions of vertices. We also give the theoretical analysis that our method implicitly preserves the Rooted PageRank score, which is an asymmetric high-order metric.

Acknowledgement

This work was partially supported by NSFC under Grant No. 61272156 and 61572040, National Key Research and Development Program No. 2016YFB1000700, Shenzhen Gov Research Project under Grant No. JCYJ20151014093505032, and Alibaba-PKU jointed Program.

References

Adomavicius, G., and Tuzhilin, A. 2005. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17(6):734–749.

Avrachenkov, K.; Litvak, N.; Nemirovsky, D.; and Osipova, N. 2007. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM Journal on Numerical Analysis* 45(2):890–904.

Bahmani, B.; Chowdhury, A.; and Goel, A. 2010. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment* 4(3):173–184.

Barkan, O., and Koenigstein, N. 2016. Item2vec: Neural item embedding for collaborative filtering. *arXiv preprint arXiv:1603.04259*.

Fogaras, D.; Rácz, B.; Csalogány, K.; and Sarlós, T. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics* 2(3):333–358.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *International Conference on Knowledge Discovery and Data Mining*. ACM.

Gupta, P.; Goel, A.; Lin, J.; Sharma, A.; Wang, D.; and Zadeh, R. 2013. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*, 505–514. ACM.

Haveliwala, T. H. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, 517–526. ACM.

Jeh, G., and Widom, J. 2002. Simrank: a measure of structural-context similarity. In *International Conference on Knowledge Discovery and Data Mining*, 538–543. ACM.

Katz, L. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18(1):39–43.

Levy, O., and Goldberg, Y. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, 2177–2185.

Liben-Nowell, D., and Kleinberg, J. 2007. The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58(7):1019–1031.

Lin, Y.; Liu, Z.; Sun, M.; Liu, Y.; and Zhu, X. 2015. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, 2181–2187.

Liu, Q.; Li, Z.; Lui, J.; and Cheng, J. 2016. Powerwalk: Scalable personalized pagerank via random walks with vertex-centric decomposition. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 195–204. ACM.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Ou, M.; Cui, P.; Pei, J.; and Zhu, W. 2016. Asymmetric transitivity preserving graph embedding. In *International Conference on Knowledge Discovery and Data Mining*. ACM.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, 1532–43.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710. ACM.

Roweis, S. T., and Saul, L. K. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290(5500):2323–2326.

Sarwar, B.; Karypis, G.; Konstan, J.; and Riedl, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, 285–295. ACM.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, 1067–1077. ACM.

Tenenbaum, J. B.; De Silva, V.; and Langford, J. C. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290(5500):2319–2323.

Wang, Z.; Zhang, J.; Feng, J.; and Chen, Z. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 1112–1119. Citeseer.

Wold, S.; Esbensen, K.; and Geladi, P. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2(1-3):37–52.

Zhang, F.; Yuan, N. J.; Lian, D.; Xie, X.; and Ma, W. Y. 2016. Collaborative knowledge base embedding for recommender systems. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.