BTWalk: Branching Tree Random Walk for Multi-order Structured Network Embedding

Hao Xiong, and Junchi Yan* *Member, IEEE,*

Abstract—Multi-order proximity is useful for effective network embedding. In contrast to many previous works that only consider order-level weights, this paper proposes to explore a more expressive node-level weighting mechanism to encode the diverse local structure, with a scalable and theoretically justified sampling strategy for its learning. Specifically, we start with a formal definition of multi-order proximity matrix which leads to our new multi-order objective based on Laplacian Eigenmaps and Skip-Gram. Then we instantiate the node-specific multi-order weights in the objective with the help of neighborhood size estimation, which indicates node-specific multi-order information. For objective learning, it is implicitly fulfilled with our proposed branching tree-like random walk strategy termed by BTWalk, which differs from the dominant chain-like walk in existing sampling techniques. BTWalk is designed by a synergetic combination of BFS (breadth-first search) and DFS (depth-first search), which is modulated according to the weights of the considered proximity orders. We theoretically analyze its cost-efficiency, and further propose the so-called Vec4Cross framework that incorporates joint node embedding and network alignment for two partially overlapped networks based on the seed matchings, whereby BTWalk is also adopted for embedding. Promising experimental results are obtained on real-world datasets across popular tasks.

Index Terms—Network Embedding, Representation Learning, Network Alignment, Branching Tree Random Walk

1 Introduction

Compared with vector-like data as readily handled by a wide spectrum of existing learning methods, data in the form of networks is in general more challenging. One way to reuse the rich classic learning methods is to embed the nodes into vectorized features [1], [2]. Various embedding models have been proposed in recent years, and most focus on learning embedding on single networks. More recently, there is also an increasing demand for cross-network embedding, with emerging applications like knowledge graph alignment, network alignment (with joint link prediction) [3], [4], etc.

Among successful network embedding methods, existing efforts (see Table 1) have spanned two directions including the expressive objective design and effective learning procedure, mostly based on the random walk technique.

i) Expressive objective design: Pairwise node proximity is one of the principles for node representation learning objective design [2], [5], [6], [7], [8], [9], [10], [11]. In general, the proximity between nodes can be divided as 1st-order, 2nd-order, and higher-order proximity [9]. Some works [6], [7], [10] propose to model multi-order proximity. They simply give different weights to different orders for all the nodes, making the problem tractable through some mathematical tricks. However, such a simplification neglects the personalized multi-order features for the nodes, which may limit models' ability of expressiveness on local structures.

ii) Effective sampling and learning: Efforts have been made on learning the objective of multi-order proximity [7].

TABLE 1: Two lines of research for network embedding. This paper contributes to both parts, via multi-order proximity objective design and branching tree random walk.

Improvement	Related works		
	Single-order	Multi-order	Others
Objective design	[2], [8], [9]	[6], [7], [10]	[11]
Random-walk design	[9]	[12], [13], [14]	[15], [16]

Among them, scalable methods are often limited to specific forms [6], [7], while some more effective sampling techniques suffer from scalability issues due to the adopted complex random walk strategies [13], [16]. It is much needed for performing sampling and learning efficiently and effectively, especially towards a more general multi-order proximity preserving problem.

Although many efforts have been paid in the above two aspects, it still remains open for a principled way of addressing them both, which is the focus of this paper. Specifically, we devise a new objective to encode the multi-order proximity information effectively, and meanwhile develop new sampling and learning procedure to solve the objective. We resort to an approach based on light-weighted random walk and shallow embedding frameworks [2], [17] in contrast to the recent embedding models based on deep neural network e.g. SDNE [8] and its hyper version [18] that are hardly scalable for real-world datasets. More notably, our proposed BTWalk (namely Branching Tree Random Walk) possesses theoretically guaranteed cost-efficiency for learning rich structure information. We highlight the main merits of our approach as follows.

1) Expressive objective design for multi-order proximity. In contrast to matrix factorization based embedding methods [6], [7], [10] and deep learning based methods [8],

H. Xiong and J. Yan are with School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, 200240, China, and MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University. E-mail: yanjunchi@sjtu.edu.cn. *The work is partially supported by National Key Research and Development Program of China, No. 2018AAA0100704 and NSFC 61972250, U19B2035, U1609220.*
J. Yan is the correspondence author.

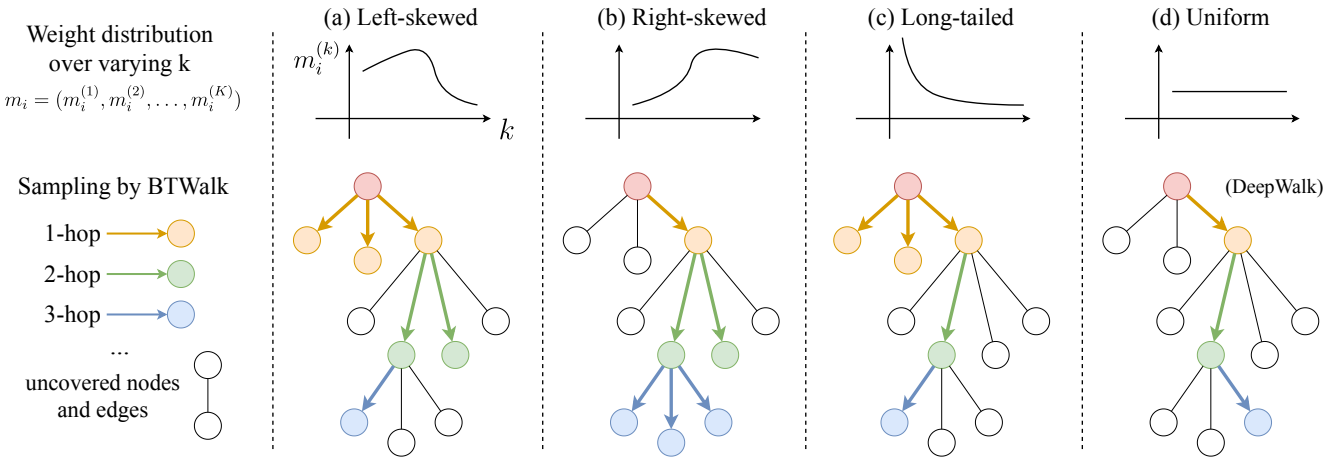


Fig. 1: Top row: the multi-order proximity weight $m_i^{(k)}$ over varying k where k is the proximity order i.e. k-hop neighborhood. Bottom row: the instantiated sampling sequences by DeepWalk. Note the proposed branching tree random walk method, i.e. BTWalk generates tree-like sampling with node-specific weights over different proximity orders. It can adaptively preserve the weights in different multi-order proximity settings, e.g. (a) left-skewed; (b) right-skewed; (c) long-tailed; (d) uniform, etc. This is fulfilled by repeating BFS sampling for different times upon the nodes on a single DFS path. In contrast, uniform random walk i.e. DeepWalk [12] samples one node per hop, unable to achieve the node-specific weights of multiple proximity orders. See Sec. 3.4.2 for more detailed comparison and discussion.

[18], an explicit objective is often missing in random walk methods [12], [13], [16], leading to a somehow ad-hoc learning procedure. Though this has been recently addressed in [9], [19] with new objectives, they are either limited to low-order proximity preserving [9], or neglecting node-specific nature of multi-order features [19]. In contrast, we aim to design a novel objective with node-specific multi-order weights on proximity matrix. The objective is further specified by a combination of Laplacian Eigenmaps (LE) [2] and Skip-Gram [17]. Both of them are shallow models with impressive performance and scalability.

2) Scalable multi-order proximity learning. One fundamental challenge in modeling multi-order proximity in existing embedding methods like [10] is the resulting $O(|\mathcal{V}|\overline{d}^k)$ time complexity of computing the multi-order proximity matrix in the pre-processing stage ($\mathcal{V}$ stands for the node set of the given network, and $\overline{d}$ stands for the average degree of nodes). Some embedding methods based on matrix factorization like [7] can avoid multiplication between transition matrix and adjacency matrix by mathematical tricks, while sacrificing the flexibility for personalized weighting.

To avoid computing the actual multi-order proximity matrix in the pre-processing stage, we propose Branching Tree Random Walk (namely BTWalk) as a sampling strategy. Notably, BTWalk gives a solution of sampling for multi-order weights not only in order-level, but also in node-level. Specifically, BTWalk is a neighbor sampling algorithm to search neighbors by both BFS and DFS with a time complexity of $O(N_w K)$ (N_w is the number of walks and K is the max hop). As Fig. 1 shows, BTWalk superimposes BFS on DFS in a synergetic manner. In our method, the sampling number of k-hop BFS is set proportional to the weight of k-th order proximity, and the total sampling depth is set to the max hop K. In this way, personalized multi-order proximity can be preserved. In Fig. 1, we give examples

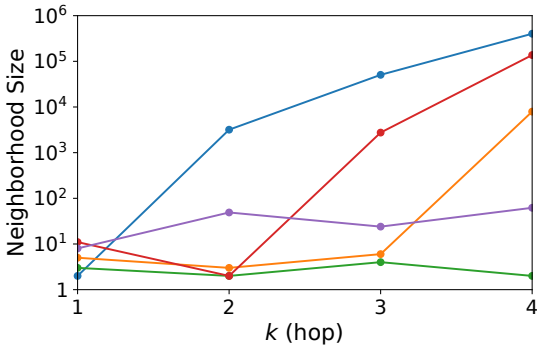


Fig. 2: The varying size (i.e. number of nodes) of k-hop neighborhoods of the five sampled node sequences from YouTube [20]. In the green and purple walking sequences, the neighborhood sizes keep stable, while the other three sequences grow explosively at 2-, 3-, and 4-hop respectively. Hence it calls for adaptive sampling by the number of nodes for different hops as their neighborhood sizes vary much.

of the personalized multi-order weights and how BTWalk may deal with them accordingly. Existing random walk methods [12], [19] are only able to deal with one distribution in a network, while BTWalk can handle all of them at the same time in one network. Moreover, we prove that BTWalk is equivalent to sampling directly over the multi-order proximity matrix while avoiding the tedious matrix computation (see Lemma 1). In other words, our approach is scalable for multi-order learning with theoretical guarantee for its equivalence as an alternative.

3) Node-specific multi-order weighting by neighborhood size estimation. How to determine proper and expressive node-specific multi-order weights in networks is another problem. In many real-world cases, the sizes of

k-hop neighborhoods can vary as shown in Fig. 2. Here we give a toy example: there is a small clique (including node i) where only very few nodes connect to the rest of the network. As the random walk from i continues, the neighborhoods of node i will hardly become larger since the random walk is almost constrained in the clique (like the green line in Fig. 2). In this case, a long random walk sequence can be unnecessary since most information is repeated. In comparison, when a low-degree node is connected to a high-degree node, the low-degree node will gain a large 2-hop neighborhoods (e.g. the blue line in Fig. 2). To learn more structural information, more times of sampling on 2-hop are needed in such a case. However, most random walk based methods [12], [13], [14] sample one node per hop, neglecting the varying size of neighborhoods.

To solve the problem, we propose to set the weight of k-hop neighbors according to the k-hop neighborhood size. As such, more flexible node sampling of BFS on k-hop can be achieved with different quantities. However, exactly counting this number can be intractable due to its time complexity of $O(|\mathcal{V}|\overline{d}^k)$. For this reason, we develop an embedding based estimation technique. Its better efficacy against vanilla random walk with uniform size setting is empirically verified in our experiments.

4) Cross-network embedding and alignment. An extension is made to the less studied cross-network setting, which is useful for tasks like network alignment. We resort to an end-to-end learning paradigm based on the expectation-maximization (EM) protocol, for node embedding and alignment. In an EM iteration, it jointly embeds two networks and aligns them by learned node embedding. While in contrast, some embedding-based network alignment methods [21], [22] train node embedding and learn cross-network mapping in two stages, which can be less optimal for the high non-linear structure of networks [8], [23] and lack of training data (10%-30% ratio) [24], [25]. Apart from this, there are other two important features of the cross-network embedding framework: i) it is fitted for most single-network embedding methods; ii) it is able to embed pairwise node attributes while most attribute-based embedding methods only preserve node-wise attributes.

Summarizing the above features, in a nutshell, this paper makes the following main contributions:

1) We define a new node-specific weighted objective. It incorporates multi-order proximity information and is further specified by the combination of Laplacian Eigenmaps and Skip-Gram models. To our best knowledge, this is the first multi-order proximity objective for network embedding that is node-specific. It provides a direct control to keep balance among the different orders of proximity.

2) For model learning, we propose our Branching Tree Random Walk (BTWalk, see Alg. 1), which exploits neighborhoods by a combination of BFS and DFS. Notably, our theoretical study proves to guarantee its equivalence to directly computing the multi-order proximity matrix (see Definition 3.3), which is intractable for large networks. We term the resulting embedding method BTVec (see Alg. 2).

3) To explore the cross-network information, we develop an end-to-end joint embedding and alignment method under the EM framework, namely Vec4Cross. This is in contrast to [21], [22] performing the two tasks separately. Its practical utility is improved as it can incorporate both handcrafted features and learned embedding features. Our Vec4Cross framework can also reuse the existing embedding models e.g. node2vec, DeepWalk as well as the proposed BTVec in an out-of-box manner, termed BTCross (see Alg. 3).

4) Experimental results on various real-world datasets with popular tasks show the state-of-the-art performance of our approach. The source code will be released.

TABLE 2: Main notations and description used in this paper. Notations with * are hyper-parameters.

Single-network model: BTVec	
$G = (\mathcal{V}, \mathcal{E}, \mathbf{A})$	network where $\mathcal{V}$ is vertex set, $\mathcal{E}$ is edge set, and $\mathbf{A}$ is adjacency matrix
K^*	maximum hop
$\mathcal{N}^{(k)}(v)$	k-hop neighbors of node v
$f(v)$, $g(v)$	vertex and context embedding for node v
z^*	dimension of latent vector space
$\mathbf{D}$	degree matrix
$\mathbf{P}$	1-hop transition probability matrix
$\mathbf{M}^{(k)}$	k-th order weight matrix
$\mathbf{A}^{(k)}$	k-th order proximity matrix
$\widetilde{\mathbf{A}}^{(k)}$	k-hop multi-order proximity matrix
α^*	weight of the Laplacian Regularizer term
w^*	used in estimating sizes of neighborhoods
Cross-network framework: Vec4Cross, and our BTCross	
G^s, G^t	source network and target network for cross-network embedding
T	set of seed matchings
UT	nodes outside the seed set for matching
$\mathbf{C}$	matching confidence matrix
N_m^*	maximum number of estimated matching nodes for each node
N_p^*	maximum number of newly estimated matchings
$t_i = \{t_{i1}, \ldots, t_{iS}\}$	set of estimated matchings for node v_i
$\xi_i = \{\xi_{i1}, \ldots, \xi_{iS}\}$	associated confidence of elements in t_i
$\phi(v)$	ground-truth matching of node v
Θ	parameters of the framework
S_{emb}	node embedding similarity
S_{add}	matching scores given by the additional classifier
γ^*	weight of non-common-neighbor penalty
β^*	weight of alignment loss

2 Related Works

We discuss related methods that basically span the following two threads of research in the field.

2.1 Single-network Embedding

Network embedding can be generally divided to three groups: shallow models [9], [12], [13], [16], deep models [8], [18], and those by matrix factorization [5], [26], [27], [28].

Shallow models are popular for large-scale networks. DeepWalk [12] first combines the random walks and neural language model i.e. Skip-Gram [29] to generate the node embedding. Then node2vec [13] designs a novel strategy to walk on a graph which explores neighborhoods by both breadth-first and depth-first strategies. It uses two parameters, namely 'return parameter' and 'in-out parameter', to control the walking procedure, which is actually a second-order Markov chain with $O(|\mathcal{V}|^2)$ time complexity to compute the transition probability matrix. In comparison,

our BTWalk proposes to sample BFS neighbors and DFS neighbors simultaneously without any extra efforts on the computation of transition probability matrix. It results in a time complexity linear to the number of walks. LINE [9] explicitly considers the first-order and second-order proximity. struc2vec [16] proposes to embed nodes with a similar structure closely by structure-targeted random walk.

GraRep [10] solves the embedding by matrix factorization for random walk and Skip-Gram while also taking high-order proximity information into consideration. However, it can not preserve the proximity information of different orders jointly. [26], [28] treat the network embedding task as sparse matrix factorization and [26] proposes to unify some shallow models within a matrix factorization framework. [6], [7] propose to preserve multi-order proximity by matrix factorization with some mathematical tricks.

There are also deep network methods like [8], [18], while they have scalability issues for large networks.

2.2 Network Alignment

Relevant methods can be mainly divided into handcrafted feature-based and embedding-based approaches.

Handcrafted feature-based methods can yield promising results. Given seed alignments, MNA [30] extracts pairwise features from multiple networks and then solves network alignment as a stable matching problem. The features include the number of common neighbors, Jaccard similarity of users' neighborhoods, and Adamic/Adar measure. BASS [25] jointly models consistencies for handcrafted features, including Jaccard similarity of users' neighborhoods, usernames' edit distance, and the features of social contents, and then employs EM algorithm to learn the classifier's parameter in a bootstrapping framework. In each EM iteration, it chooses the common neighbors as matching candidates and aligns them afterwards. These methods, in general, lack the capacity to fully explore the structure by learning. Hence recently learning based and especially embedding based methods are receiving more attention.

For embedding-based approaches, some works [31], [32] train node embeddings by forcing the identical nodes across networks to be the same. While some other works [21], [22] follow a specific pipeline: 1) Generate node representations based on matrix factorization [31] or Skip-gram [22], with the supervision of seed matchings; 2) Use a similarity function $sim : \mathcal{V} \times \mathcal{V} \rightarrow R$ to measure the pairwise embedding similarity between nodes, e.g. cosine similarity [21], [22] or Euclidean distance [33]; 3) Match node pairs for those with high similarity. However, all the above methods cannot achieve joint end-to-end learning for both embedding and similarity, making the pipeline less optimal for the alignment task. The extension of our proposed BTCross fills this gap for its unique differentiable nature.

2.3 Remarks

In summary, our proposed approach with the multi-order proximity model tries to satisfy the desirable properties for network embedding: 1) scalable and fast to deal with large networks; 2) learning with an explicit objective; 3) preserving rich information (structure, proximity, etc.); 4) end-to-end learning for cross-network embedding and alignment.

In the following, we first present our embedding approach for single network, then we show how to extend our technique to perform joint embedding and alignment.

3 BTVec: Multi-order Proximity Preserved Embedding via BTWalk

Before going to the details, we present the whole picture in Fig. 3. On one hand, we explicitly define the multi-order proximity matrix (Sec. 3.1), based on which a multi-order objective encoding node-specific weights is devised (Sec. 3.1 for its most general form; Sec. 3.2 for the objective used in our paper; Sec. 3.3 for node-specific weighting). On the other hand, the Branching Tree Random Walk (namely BTWalk) which incorporates the BFS/DFS neighborhood structure information is devised also based on the multi-order proximity matrix (Sec. 3.1), to learn the resulting embedding model regarding with the multi-order objective.

3.1 Preliminary

First, we formally define network and network embedding.

Definition 3.1. ***Network.*** *For network $G = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, $\mathcal{V} = \{v_1, v_2, \dots\}$ is the node set, $\mathcal{E} = \{(v_i, v_j)\}$ is the edge set between nodes, and $\mathbf{A}$ is the adjacency matrix of G. If there exists an edge from v_i to v_j, then the binary tuple $(v_i, v_j) \in \mathcal{E}$, and the corresponding weight $\mathbf{A}_{i,j} > 0$. Note $\mathbf{A}_{ij}$ is not necessarily equal to $\mathbf{A}_{j,i}$ for a directed edge.*

Definition 3.2. ***Network embedding.*** *Given network G, network embedding aims to encode each node $v \in \mathcal{V}$ into a low-dimensional latent space with a mapping function $f : v \rightarrow \mathbb{R}^z$, where z is the dimension of the latent space. In this paper, we also use the term 'node embedding' exchangeably.*

The adjacency matrix $\mathbf{A}$ mainly describes the low-order connections between nodes. To preserve multi-order proximity, a polynomial function of adjacency matrix is defined to account for different orders [6], [7]:

$$\widetilde{\mathbf{A}} = \sum_{k=1}^{K} m^{(k)} \mathbf{A}^k. \tag{1}$$

where K is the maximum hop, $m^{(k)} \geq 0$ is the weight for the k-th order proximity. However, such a simple weighted sum cannot directly model the node-level multi-order features which is important for fine-grained node representation.

In this paper, we propose and formally define the so-called multi-order proximity matrix $\widetilde{\mathbf{A}}^{(K)}$ as follows[1].

Definition 3.3. ***Multi-order proximity matrix $\widetilde{\mathbf{A}}^{(K)}$.*** *Given $G = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, we denote its out degree matrix as $\mathbf{D} = diag(d_1^{out}, d_2^{out}, ...)$, where $d_i^{out} = \sum_j \mathbf{A}_{i,j}$ is the out degree for node v_i, and $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ denotes transition probability matrix. Then the multi-order proximity matrix $\widetilde{\mathbf{A}}^{(K)}$ is defined as:*

$$\widetilde{\mathbf{A}}^{(K)} = \sum_{k=1}^{K} \mathbf{M}^{(k)} \mathbf{A}^{(k)}, \mathbf{A}^{(k)} = \mathbf{A}\mathbf{P}^{k-1}, \tag{2}$$

where the weight matrix $\mathbf{M}^{(k)} = diag(m_1^{(k)}, m_2^{(k)}, \dots, m_{|\mathcal{V}|}^{(k)})$ $(k = 1, 2, \cdots, K)$ controls weights of k-th order proximity

1. In this paper, we denote $\widetilde{\cdot}$ as multi-order related variables.

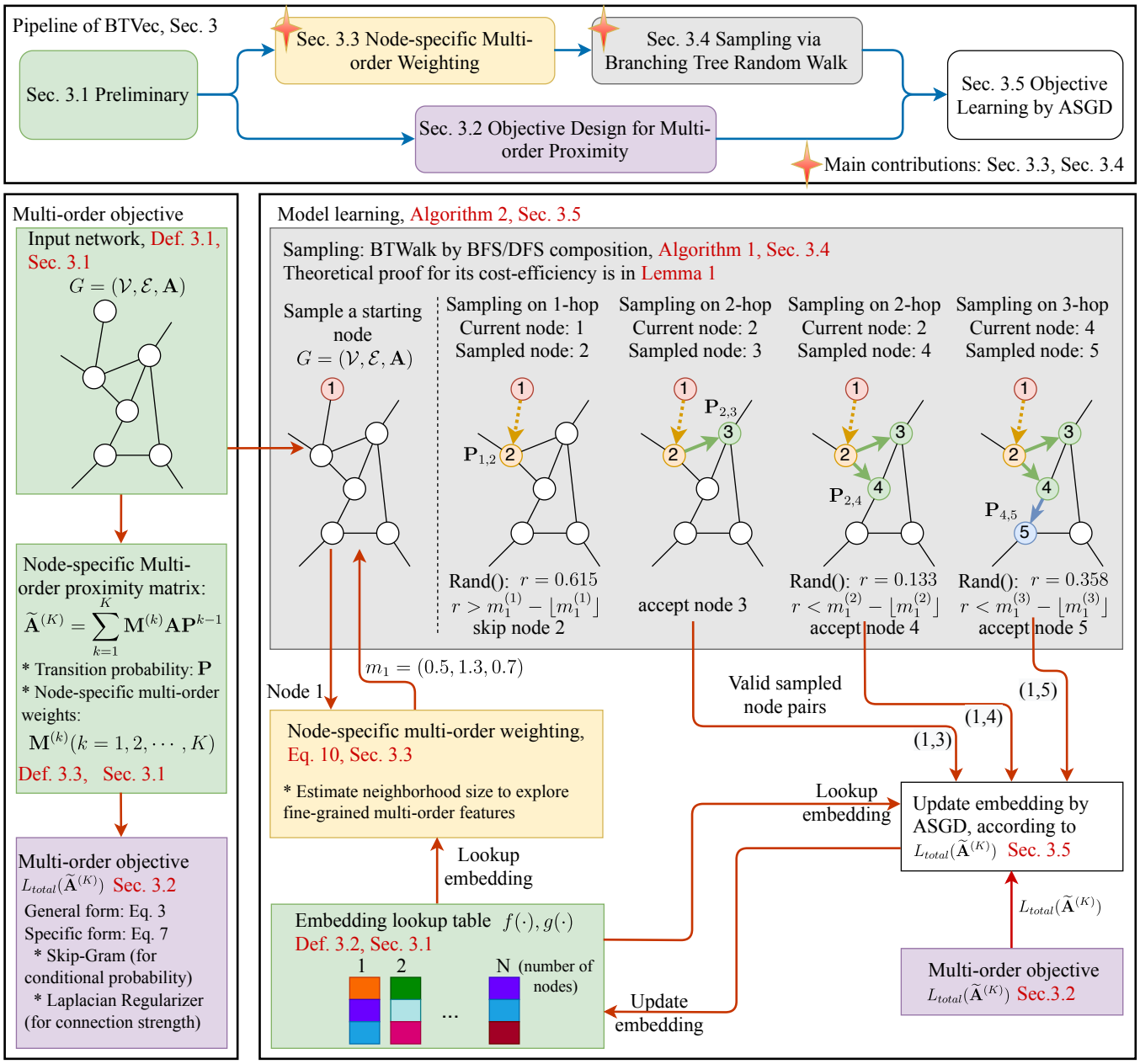


Fig. 3: Our single network embedding approach as described in Sec. 3 (see also in Alg. 2). Boxes in the same color denote functions/definitions from the same subsection, which are sketched at the top and detailed in the bottom, respectively. Beginning from the left part, we give some preliminary in Sec. 3.1, where we define a node-specific multi-order proximity matrix $\widetilde{\mathbf{A}}^{(K)}$ by adjacency matrix $\mathbf{A}$, transition probability $\mathbf{P}$, and node-specific multi-order weight matrices $\mathbf{M}^{(k)}$ $(k = 1, 2, \cdots, K)$. Then in Sec. 3.2, we introduce a new designed objective with a combination of Skip-Gram and Laplacian Regularizer. The bottom right part describes the model learning procedure. To conduct our BTWalk sampling strategy whose efficiency and effectiveness is proved in Lemma 1, we start with sampling a starting node from the origin network. Then in the yellow box, we get the corresponding node-specific multi-order weights through estimating the neighborhood size, which indicates the node-specific multi-order information. After that, BTWalk runs according to the sampled nodes' multi-order weights, as shown in the grey box. Node embedding is updated by ASGD according to the defined objective.

*in $\widetilde{\mathbf{A}}^{(K)}$, and K is the maximum hop. Matrix $\mathbf{M}^{(k)}$ is **node-specific** since the multi-order weights $m_i = (m_i^{(1)}, \cdots, m_i^{(K)})$ are personalized for different nodes. For nodes v_i and v_j, if the inside $\mathbf{A}_{i,j}^{(k)} > 0 (k \leq K)$, then the multi-order proximity matrix $\widetilde{\mathbf{A}}^{(K)}$ preserves the multi-order proximity between v_i and v_j. Nodes with high proximity should be close in embedding space.*

Based on the multi-order proximity matrix $\widetilde{\mathbf{A}}^{(K)}$, the objective of a general form can be reached by:

$$L_{total} = \sum_{i,j} \widetilde{\mathbf{A}}_{i,j}^{(K)} L(v_i, v_j), \tag{3}$$

where $L(v_i, v_j)$ is the loss function for a single positive node pair. We are going to discuss about the specific design of our

objective in the next subsection.

3.2 Objective Design for Multi-order Proximity

We aim to learn two mapping functions [9] $f, g : \mathcal{V} \to \mathbb{R}^z$, where f and g denotes the embedding for 'vertex' and 'context' representation respectively, following [9], [17]. Here 'vertex' nodes are used to predict 'context' nodes. We propose the embedding model which is based on Skip-Gram and meanwhile introducing the ideas from Laplacian Eigenmaps (LE) [2]. LE aims to embed nodes sharing proximity closely, while Skip-Gram aims to embed nodes with similar neighborhoods [9]. Both have been respectively used in existing embedding literature [8], [32] due to their good performance and scalability in network embedding. The combination of LE and Skip-Gram is a natural extension. Based on the node-specific multi-order proximity matrix $\widetilde{\mathbf{A}}^{(K)}$, we further design the whole objective as Eq. 7.

Laplacian Regularizer Term. Following [2], we adopt Laplacian Regularizer to model the connection strength. For each $\widetilde{\mathbf{A}}^{(K)}_{i,j} > 0$, we call v_i the 'vertex' node and v_j the 'context' node [9]. For proximity preservation, node v_i should be embedded as a 'vertex' close to v_j as a 'context'. The loss of connection strength is defined as:

$$L_{lap}(\widetilde{\mathbf{A}}^{(K)}) = \sum_{i,j} \widetilde{\mathbf{A}}^{(K)}_{i,j} \|f(v_i) - g(v_j)\|_2^2. \tag{4}$$

Especially, when we treat 'vertex' and 'context' in the same way, i.e. $f \equiv g$, the form is the same as LE [2].

Skip-Gram Term. Skip-Gram is originally used in language neural models [17], which then becomes popular in network embedding [9], [11], [12], [13]. Given vertex v_i and context node v_j, softmax is used to define the probability:

$$p(v_j|v_i) = \frac{\exp\left(f(v_i)^\top g(v_j)\right)}{\sum_{j'} \exp\left(f(v_i)^\top g(v_{j'})\right)}. \tag{5}$$

However, it is intimidating to traverse over all the negative nodes $v_{j'}$. To maximize the probability in a tractable way, the softmax probability is substituted with a sigmoid score function $\sigma(\cdot)$, and negative sampling [17] is adopted based on noise contrastive estimation [34]. For each sampled positive node pair (v_i, v_j), we sample B negative nodes denoted as v_n from a context-independent noise distribution $\mathbb{P}_n$. Specifically, we set $\mathbb{P}_n = (d_n^{out})^{3/4}$ which works empirically well [9], [17]. Then the objective becomes:

$$\begin{aligned} L_{sg}(\widetilde{\mathbf{A}}^{(K)}) = \sum_{i,j} \widetilde{\mathbf{A}}^{(K)}_{i,j} \Big[& \log \sigma\big(f(v_i)^\top g(v_j)\big) \\ & + B\mathbb{E}_{v_n \sim \mathbb{P}_n} \log \sigma\big(-f(v_i)^\top g(v_n)\big)\Big]. \end{aligned} \tag{6}$$

The overall loss can be obtained by adding Laplacian Regularizer and Skip-Gram model together weighted by α:

$$L_{total}(\widetilde{\mathbf{A}}^{(K)}) = \alpha L_{lap}(\widetilde{\mathbf{A}}^{(K)}) + L_{sg}(\widetilde{\mathbf{A}}^{(K)}). \tag{7}$$

3.3 Node-specific Multi-order Weighting

In this part, we propose an adaptive strategy for setting multi-order weights $\mathbf{M}^{(k)}$ $(k = 1, 2, \cdots, K)$. For node v_i, we define its k-hop neighborhood as $\mathcal{N}^{(k)}(v_i) = \{v_j | \mathbf{A}^{(k)}_{i,j} > 0\}$. Since the k-hop neighborhood size $|\mathcal{N}^{(k)}(v_i)|$ can somehow indicate the difficulty to fit k-th order proximity, we tend to give a larger $m_i^{(k)}$ $(k = 1, 2, \cdots, K)$ for a larger $|\mathcal{N}^{(k)}(v_i)|$. Considering that $|\mathcal{N}^{(k)}(v_i)|$ can grow irregularly, and accurate counting of $|\mathcal{N}^{(k)}(v_i)|$ is intractable, here we propose to estimate $|\mathcal{N}^{(k)}(v_i)|$ by the learned embedding functions f and g in the current iteration. Since the embeddings evolve during training, the weights $m_i = (m_i^{(1)}, m_i^{(2)}, \cdots, m_i^{(K)})$ also evolve dynamically and adaptively until it converges.

To start with, we estimate the k-hop connection strengths between nodes. In this paper, for tractable estimation, we assume there is little overlap among k-hop neighborhoods when the max hop K is small and the network is large. This is in fact a moderate condition as the neighborhood size usually grows quickly e.g. in power law. Suppose node v_j is sampled on k-hop for the first time by v_i, then we can assume $\widetilde{\mathbf{A}}^{(K)}_{i,j} \propto \mathbf{A}^{(k)}_{i,j}$, which is given by:

$$\widetilde{\mathbf{A}}^{(K)}_{i,j} = \frac{B\widetilde{d}_i^{out(K)}\widetilde{d}_j^{in(K)}}{\sum_l \widetilde{\mathbf{D}}^{(K)}_{l,l}} \exp\Big(f(v_i)^\top g(v_j)\Big), \tag{8}$$

where the notation $\widetilde{\cdot}$ denotes the versions of variables with multi-order information (akin to $\widetilde{\mathbf{A}}^{(K)}$ in Eq. 2). Note d_i^{in} (d_i^{out}) denotes the in (out) degree of node v_i.

To achieve an analytical solution, we directly borrow the optimal solution in Skip-Gram methods as given by [35]:

$$f(v_i)^\top g(v_j) = \log \frac{\widetilde{\mathbf{A}}^{(K)}_{i,j} \sum_l \widetilde{\mathbf{D}}^{(K)}_{l,l}}{\widetilde{d}_i^{out(K)}\widetilde{d}_j^{in(K)}} - \log B. \tag{9}$$

Then according to $|\mathcal{N}^{(k)}(v_i)| = \mathbf{D}^{(k)}_{i,i} / \mathbb{E}_{j \sim \mathbf{A}^{(k)}_{i,}} \mathbf{A}^{(k)}_{i,j}$, k-hop degree matrix $\mathbf{D}^{(k)} = \mathbf{D}$, and supposing $\widetilde{d}^{in(K)} \propto d^{in}$, we set the weights of proximity orders as Eq. 10, aiming to perform more sampling for a larger neighborhood size:

$$m_i^{(k)} = \left(\frac{|\mathcal{N}^{(k)}|}{|\mathcal{N}^{(1)}|}\right)^w \approx \left(\frac{d_j^{in} \exp\left(f(v_i)^\top g(v_j)\right)}{d_k^{in} \exp\left(f(v_i)^\top g(v_k)\right)}\right)^w, \tag{10}$$

where hyper-parameter w controls how much the weight $m_i^{(k)}$ depends on the estimated neighborhood size $|\mathcal{N}^{(k)}|$ and $|\mathcal{N}^{(1)}|$. In this way, the weights of proximity orders can be more precisely set for effective learning. Particularly, when $w = 0$, it becomes the vanilla random walk (i.e. the random walk strategy in DeepWalk [12]), which samples one node per hop.

3.4 Sampling via Branching Tree Random Walk

In this subsection, we introduce a random walk strategy, BTWalk, which serves for a general objective based on the multi-order proximity matrix $\widetilde{A}^{(K)}$ as given by Eq. 3, namely $L_{total} = \sum_{i,j} \widetilde{\mathbf{A}}^{(K)}_{i,j} L(v_i, v_j)$. One of the basic ideas to optimize the objective is to calculate the value of each element in $\widetilde{\mathbf{A}}^{(K)}$ and adopt edge sampling [9] over $\widetilde{\mathbf{A}}^{(K)}$ afterwards. However, since the computing of $\widetilde{\mathbf{A}}^{(K)}$ is extremely time-consuming, such an approach is impracticable for a big K. To address the problem, we propose an efficient sampling strategy, BTWalk, which runs in the style of BFS/DFS composition. The equivalence of optimizing Eq. 3 directly by sampling over $\widetilde{\mathbf{A}}^{(K)}$ and sampling through

Algorithm 1: BTWalk: Branching Tree Random Walk on network for node sampling

Input: starting node v_i; 1-hop transition matrix $\mathbf{P}$; weights of proximity order $\{m_i^{(1)}, \dots, m_i^{(K)}\}$; max hops K;
Output: Sampled node list $Walk$;

```
Initialize Walk = [];
for k = 1 : K do
    Float r = m_i^(k) e.g. by Eq. 10;
    // r is the expectation of the sample # on k-hop;
    while r > 0 do
        Sample v_j by probability P_{i,j};
        if r ≥ 1 then
            Append(Walk, v_j);    // Add v_j to Walk;
        else
            Generate a random value e ∈ (0, 1);
            // Accept the sample v_j by probability r;
            if e < r then
                Append(Walk, v_j);
        r = r − 1;
    v_i = v_j;
Return Walk;
```

BTWalk is proved in Lemma 1 while our BTWalk can be more efficient in magnitude as an alternative (see details in Lemma 1). For implementation, we use Eq. 7 as the objective and set the multi-order weights as discussed in Sec. 3.3.

3.4.1 BTWalk by BFS/DFS Composition

Given an objective in the form of Eq. 3, BTWalk achieves learning within a linear time complexity. Specifically, BTWalk adopts the following sampling pipeline which combines BFS and DFS: 1) Sample a starting node v_i by probability $\mathbf{D}_{i,i}/\sum_j \mathbf{D}_{j,j}$, whose weights of proximity order are $m_i = (m_i^{(1)}, m_i^{(2)}, \dots, m_i^{(K)})$; 2) Set $k = 1$, current node is v_i; 3) On k-hop, sample $\lceil m_i^{(k)} \rceil$ nodes from v_i's 1-hop neighbors by distribution $\mathbf{P}_{i,:}$; 4) Accept the nodes except the last one in step 3 as the random walk result; 5) Accept the last sampled nodes in step 3 by probability $m_i^{(k)} - \lfloor m_i^{(k)} \rfloor$, with the node denoted as v_j; 6) $k = k + 1$; 7) Set v_j as the current node, $v_i = v_j$, then repeat step 3 to step 6 until K hop is reached. The procedure of BTWalk with weights is also detailed in Alg. 1. Note that such a pipeline by BTWalk is equal to the former mentioned approach [9] directly doing sampling over $\widetilde{\mathbf{A}}^{(K)}$, as theoretically proved in Lemma 1.

Transition probability. We give BTWalk's transition probability mathematically in this paragraph. We denote a random walk sequence starting from node x_0 by BTWalk as $(x_{1,1}, x_{1,2}, \dots)$, where $x_{k,i}$ denotes the node by i-th sampling on k-hop, and x_0 is the starting node. The multi-order transition probability is defined as:

$$P(x_{k+1,i} = v_c | x_{k,1} = v_b, x_0 = v_a; \mathbf{M}^{(1),\dots,(K)}) = \mathbf{P}_{b,c}, \quad i = 1, 2, \dots, \lceil m_a^{(k+1)} \rceil, \tag{11}$$

from which we can find that the next sampled node is determined on both the starting node and the sampled node on the former hop, with node-specific weights of proximity orders as parameters. Different from DeepWalk where the sampled node depends on the former one step and node2vec where the sampled node depends on the former two steps, BTWalk runs in an order weight distribution insensitive way. Detailed comparison will be discussed.

Lemma 1. *Given network $G = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, maximum hop K, and the multi-order proximity matrix $\widetilde{\mathbf{A}}^{(K)}$, BTWalk is ensured to achieve that for any positive node pair (v_i, v_j), it is sampled with the same probability as performing edge sampling [9] on $\widetilde{\mathbf{A}}^{(K)}$, while avoiding its complex computation. In this way, BTWalk reduces the $O(|\mathcal{V}|\bar{d}^K)$ time complexity to $O(|\mathcal{V}|K)$.*

Proof. Given $\widetilde{\mathbf{A}}^{(K)}$, if we directly sample from $\widetilde{\mathbf{A}}^{(K)}$, we have $\widetilde{\mathbf{P}}_{i,j}^{(K)}$ denoting the probability (v_i, v_j) is sampled:

$$\widetilde{\mathbf{P}}_{i,j}^{(K)} = \widetilde{\mathbf{A}}_{i,j}^{(K)} / \mathbf{D}_{i,i}^{(K)}, \mathbf{D}_{i,i}^{(K)} = \sum_{j=1}^{|\mathcal{V}|} \widetilde{\mathbf{A}}_{i,j}^{(K)}, \tag{12}$$

where $\mathbf{D}^{(K)}$ is the degree matrix by $\widetilde{\mathbf{A}}^{(K)}$.

Let $P(j|i;K)$ denote the probability that node v_j can be reached within K-hops BTWalk from v_i. It is defined as:

$$P(j|i;K) = E_{i,j}^{(K)} / N^{(K)}, \tag{13}$$

where $E_{i,j}^{(K)}$ is expectation of the times that node v_j shows in K-hop BTWalk from v_i, and $N^{(K)}$ denotes the expectation of total number of sampled nodes within K-hop BTWalk.

Proving the lemma is equal to prove:

$$P(j|i;K) = \widetilde{\mathbf{P}}_{i,j}^{(K)}. \tag{14}$$

When $K = 1$, we have $E_{i,j}^{(1)} = N^{(1)} \mathbf{A}_{i,j} / \mathbf{D}_{i,i}$. Since $\widetilde{\mathbf{A}}^{(1)} = m_i^{(1)} \mathbf{A}$ and $\widetilde{\mathbf{D}}^{(1)} = m_i^{(1)} \mathbf{D}$, it is easy to find that Eq. 14 holds in this case. Assuming that Eq. 14 holds with $(K \geq 1)$, we need to prove that it also holds for $K = K + 1$. We denote $\mathbf{P}^{(k)} = \mathbf{P}^k$ as k-hop transition matrix, then for $P(j|i;K+1)$ we have:

$$P(j|i;K+1) = \frac{E_{i,j}^{(K+1)}}{N^{(K+1)}} = \frac{E_{i,j}^{(K)} + \Delta N^{(K+1)} \sum_{p=1}^{|\mathcal{V}|} \mathbf{P}_{i,p}^{(K)} \mathbf{P}_{p,j}}{N^{(K)} + \Delta N^{(K+1)}}, \tag{15}$$

where $\Delta N^{(K)} = N^{(K)} - N^{(K-1)}$ $(N^{(0)} = 0)$ denotes the sampling number of BTWalk on the K-hop. Then we try to figure out $\widetilde{\mathbf{P}}_{i,j}^{(K+1)}$. For $\widetilde{\mathbf{A}}_{i,j}^{(K+1)}$ and $\widetilde{\mathbf{D}}_{i,i}^{(K+1)}$, we have:

$$\begin{aligned}
\widetilde{\mathbf{A}}_{i,j}^{(K+1)} &= \widetilde{\mathbf{A}}_{i,j}^{(K)} + m_i^{(K+1)} \mathbf{A}_{i,j}^{(K+1)} \\
&= \widetilde{\mathbf{A}}_{i,j}^{(K)} + m_i^{(K+1)} \sum_{p=1}^{|\mathcal{V}|} \mathbf{A}_{i,p}^{(K)} \mathbf{P}_{p,j}, \\
\widetilde{\mathbf{D}}_{i,i}^{(K+1)} &= \widetilde{\mathbf{D}}_{i,i}^{(K)} + m_i^{(K+1)} \sum_{k=1}^{|\mathcal{V}|} \mathbf{A}_{i,k}^{(K+1)} \\
&= \widetilde{\mathbf{D}}_{i,i}^{(K)} + m_i^{(K+1)} \sum_{k=1}^{|\mathcal{V}|} \sum_{l=1}^{|\mathcal{V}|} \mathbf{A}_{i,l}^{(K)} \mathbf{P}_{l,k} \\
&= \widetilde{\mathbf{D}}_{i,i}^{(K)} + m_i^{(K+1)} \sum_{l=1}^{|\mathcal{V}|} \mathbf{A}_{i,l}^{(K)}
\end{aligned} \tag{16}$$

Algorithm 2: BTVec: Branching Tree Random Walk for node vector representation.

Input: network $G = (\mathcal{V}, \mathcal{E}, \mathbf{A})$; max hop K; number of walks N_w; number of negative samples B;
Output: vertex embedding function f;

Degree matrix $\mathbf{D} = diag(\sum_j \mathbf{A}_{1,j}, \sum_j \mathbf{A}_{2,j}, ...)$;
Transition matrix $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$;
Randomly initialize embedding mapping f, g;
Learning rate $init\_lr = 0.025$ (same as in [9]);
$lr = init\_lr$;
for $t_1 = 1 : N_w$ **do**
  Sample a starting node v_i from $\mathcal{V}$ by probability $\mathbf{D}_{i,i}/\sum_k \mathbf{D}_{k,k}$;
  Estimate k-hop specific weight $m_i^{(k)}$ by Eq. 10 in Sec. 3.3;
  $Walk = \text{BTWalk}(v_i, \mathbf{P}, m_i^{(1),\cdots,(K)}, K)$ by Alg. 1;
  for $v_j \in Walk$ **do**
    Update_pos(v_i, v_j) (obj. in Eq. 7, Sec. 3.2);
    for $t_2 = 1 : B$ **do**
      Sample v_n from $\mathbb{P}_n$;
      Update_neg(v_i, v_n) (obj. in Eq. 7, Sec. 3.2);
  $lr = init\_lr \cdot (N_w - t_1)/N_w$;

By $\widetilde{\mathbf{P}}_{i,j}^{(K+1)} = \widetilde{\mathbf{A}}_{i,j}^{(K+1)}/\widetilde{\mathbf{D}}_{i,i}^{(K+1)}$, we can reach that:

$$\begin{aligned}\widetilde{\mathbf{P}}_{i,j}^{(K+1)} =& \frac{\widetilde{\mathbf{A}}_{i,j}^{(K)}/\widetilde{\mathbf{D}}_{i,i}^{(K)} + m_i^{(K+1)} \sum_{p=1}^{|\mathcal{V}|} (\mathbf{A}_{i,p}^{(K)} \mathbf{P}_{p,j}/\widetilde{\mathbf{D}}_{i,i}^{(K)})}{1 + m_i^{(K+1)} \sum_{l=1}^{|\mathcal{V}|} \mathbf{A}_{i,l}^{(K)}/\widetilde{\mathbf{D}}_{i,i}^{(K)}} \\ =& \frac{\widetilde{\mathbf{P}}_{i,j}^{(K)} + m_i^{(K+1)} \sum_{p=1}^{|\mathcal{V}|} \left(\frac{\mathbf{A}_{i,p}^{(K)} \mathbf{P}_{p,j}}{\sum_{q=1}^{K} m_i^{(q)} \mathbf{D}_{i,i}} \right)}{1 + m_i^{(K+1)} \frac{\sum_{l=1}^{|\mathcal{V}|} \mathbf{A}_{i,l}^{(K)}}{\sum_{q=1}^{K} m_i^{(q)} \mathbf{D}_{i,i}}} \\ =& \frac{P(j|i;K) + \frac{m_i^{(K+1)}}{\sum_{q=1}^{K} m_i^{(q)}} \sum_{p=1}^{|\mathcal{V}|} \mathbf{P}_{i,p}^{(K)} \mathbf{P}_{p,j}}{1 + \frac{m_i^{(K+1)}}{\sum_{q=1}^{K} m_i^{(q)}}},\end{aligned} \tag{17}$$

where the proof makes use of the following obvious result:

$$\begin{cases} \sum_{p=1}^{|\mathcal{V}|} \mathbf{A}_{i,p}^{(k)} = \sum_{p=1}^{|\mathcal{V}|} \mathbf{A}_{i,p} = \mathbf{D}_{i,i}, \\ \widetilde{\mathbf{D}}_{i,i}^{(k)} = \sum_{q=1}^{k} m_i^{(q)} \mathbf{D}_{i,i}, \\ \mathbf{A}_{i,p}^{(k)} = \mathbf{D}_{i,i} \mathbf{P}_{i,p}^{(k)}. \end{cases} \tag{18}$$

So if set $\Delta N(k) \propto m_i^{(k)}$ $(k = 1, \ldots, K)$ when starting from node v_i, Eq. 14 also holds for $K+1$ by mathematical induction. That means BTWalk can get the same result without time-consuming matrix multiplication for $\widetilde{\mathbf{A}}^{(K)}$. Now we have finished the proof. □

3.4.2 Comparison with Other Random-walk Methods

We compare BTWalk with popular random walk methods mainly on their transition probability. We denote a random walk sequence starting from node x_0 by peer methods as $\{x_1, x_2, \ldots, x_K\}$.

TABLE 3: Time complexity comparison. Denote N_w for number of walks ($N_w \propto |\mathcal{V}|$), K for the max hop, and $\overline{d}$ for average node degree.

Methods	Random walk complexity	Node-specific multi-order proximity
Deepwalk	$O(N_w K)$	×
LINE	$O(N_w)$	×
node2vec	$O(N_w K) + O(\lvert\mathcal{V}\rvert \overline{d}^2)$	×
BTWalk	$O(N_w K)$	✓

DeepWalk [12]. Its random walk strategy can be seen as a special case of BTWalk, with multi-order weight matrix $\mathbf{M}^{(k)} = \mathbf{I}$. The transition probability is defined as:

$$P(x_{k+1} = v_b | x_k = v_a) = \mathbf{P}_{a,b}. \tag{19}$$

LINE [9]. It only takes 1-hop neighbors into consideration. It is a special case of BTWalk with $K = 1$ and $\mathbf{M}^{(1)} = \mathbf{I}$. The transition probability is defined as:

$$P(x_1 = v_b | x_0 = v_a) = \mathbf{P}_{a,b}. \tag{20}$$

node2vec [13]. It explores BFS/DFS structure in a second-order style. First, it turns the adjacency matrix $\mathbf{A}$ into a second-order adjacency tensor $\overline{\mathbf{A}} \in \mathcal{R}^{|\mathcal{V}| \times |\mathcal{V}| \times |\mathcal{V}|}$ with two parameters, namely a 'return parameter' p and an 'in-out parameter' q. Such a preprocessing procedure may incur a complexity of $O(|\mathcal{V}|\overline{d}^2)$ for both time and space, where $\overline{d}$ is the average degree of the nodes. Formally the tensor $\mathbf{A}$ is given by (as indexed by 'a', 'b', 'c'):

$$\overline{\mathbf{A}}_{a,b,c} = \begin{cases} \frac{1}{p}\mathbf{A}_{b,c} & c = a \\ \mathbf{A}_{b,c} & \mathbf{P}_{a,b}\mathbf{P}_{b,c}\mathbf{P}_{a,c} > 0, a \neq c \\ \frac{1}{q}\mathbf{A}_{b,c} & \mathbf{P}_{a,b}\mathbf{P}_{b,c} > 0, \mathbf{P}_{a,c} = 0, a \neq c \\ 0 & \text{otherwise} \end{cases}. \tag{21}$$

Then a second-order transition probability tensor $\overline{\mathbf{P}}_{a,b,c} = \overline{\mathbf{A}}_{a,b,c}/\sum_{i=1}^{|\mathcal{V}|} \overline{\mathbf{A}}_{a,b,i}$ can be reached. The second-order transition probability is defined as:

$$P(x_{k+2} = v_c | x_{k+1} = v_b, x_k = v_a) = \overline{\mathbf{P}}_{a,b,c}. \tag{22}$$

Note that node2vec does not explicitly preserve multi-order proximity and the weights of proximity order $\mathbf{M}^{(k)}$ are not defined in that method. Compared with the above three popular random walk strategies, BTWalk achieves to preserve node-specific multi-order proximity in a compact and efficient way as described in the former subsection. The comparison is given in Table 3.

3.5 Objective Learning by ASGD

In this subsection, we summarize how the learning pipeline incorporates with the techniques discussed in the former subsections. To begin with, we use Eq. 7 as the objective (Sec. 3.2) and set multi-order weights by neighborhood-aware weighting (Sec. 3.3). The whole optimization procedure is in a typical multi-thread ASGD [36] style: In one

iteration, each thread first samples a starting node v_i. Then it estimates the k-hop specific weight $m_i^{(k)}$ $(k = 1, 2, \cdots, K)$ and conducts BTWalk according to the weights. Once a walk sequence is generated, it updates the embedding by gradient descent. We have described the detailed algorithms in Alg. 2. The whole working pipeline is shown in Fig. 3.

4 VEC4CROSS: CROSS-NETWORK EMBEDDING

In the task of network alignment, two networks are disconnected but the nodes in the two networks have a deeper relationship – they can be aligned. Two nodes are aligned means they are actually identical nodes in different networks. Single-network embedding methods usually fail to deal with such a situation. However, methods of single-network embedding can be extended to cross-network for network alignment (e.g. LINE in [22]).

In this section, we introduce such a framework, **Vec4Cross**, that models cross-network matching links by Gaussian Mixture Model (GMM) and learn up-to-date node matching by Expectation-Maximization (EM). The framework is designed to be general, which means it can incorporate any single-network embedding methods, including BTVec, which leads to the resulting cross-network embedding approach called **Vec4Cross**, as described in Alg. 3.

4.1 Overview and Preliminaries

We first introduce the task of network alignment and then give the overview and basic concepts of **Vec4Cross**.

Network alignment (node matching). Given a source network $G^s = (\mathcal{V}^s, \mathcal{E}^s, \mathbf{A}^s)$, a target network $G^t = (\mathcal{V}^t, \mathcal{E}^t, \mathbf{A}^t)$,[2] and partial node matching set $T = \{(u, v)|u \in \mathcal{V}^s, v \in \mathcal{V}^t\}$ as seed alignments by certain means, network alignment aims to predict the rest node matchings. We use such an injective mapping function $\phi : \mathcal{V}^s \cup \mathcal{V}^t \to \mathcal{V}^s \cup \mathcal{V}^t$ to describe node matching. Note that we assume $T \neq \emptyset$.

Framework overview. Given two networks G^s and G^t, our approach follows a common protocol in network alignment that the correspondences are established in a progressive manner. Specifically, a few node correspondences are set either by ground truth (in semi-supervised learning) or by estimation with an existing matching algorithm [25], [37]. Starting with these seed alignments, the EM procedure is performed: 1) In the E-step, we first define an **alignment confidence matrix** $\mathbf{C} \in \mathbb{R}^{|\mathcal{V}^s| \times |\mathcal{V}^t|}$. $\mathbf{C}_{i,j} \in [0, 1]$ denotes the matching certainty for the correspondence of pair (v_i^s, v_j^t), which is measured by the pairwise node similarity across G^s and G^t. The generation and updating of $\mathbf{C}$ will be detailed in Sec. 4.2.2. In the later stage of E-step, according to $\mathbf{C}$, we select N_p pairs of nodes with the highest matching confidence as candidate matching pairs, and each node has no more than N_m choices. We denote the alignment choices for node v_i as $t_i = \{t_{i1}, \ldots, t_{iN_m}\}$ with corresponding confidences $\xi_i = \{\xi_{i1}, \ldots, \xi_{iN_m}\}$. 2) In M-step, we update node embedding by cross-network embedding procedure according to the known and newly estimated node matchings. The parameters of the additional classifier will also be updated, when the extended node attributes are given.

2. In this paper, we use superscript ^{s} and ^{t} to denote variables for source networks and target networks respectively.

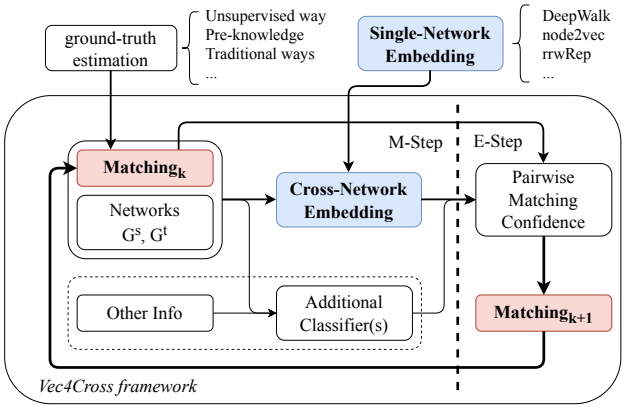


Fig. 4: Our Vec4Cross network alignment framework which can incorporate the proposed BTVec as the single-network embedding building block. Moreover, additional hand-crafted features in dash box can also be incorporated.

Algorithm 3: BTCross (under our Vec4Cross framework): Branching Tree Random Walk based cross-network embedding and alignment.

Input: networks G^s and G^t; seed matchings T; max hop K; number of walks N_w for each iteration; maximum number of newly estimated matchings N_p; maximum number of matchings for one node N_m; total number of iterations I; learning rate $lr = 0.025$;
Output: vertex embedding function f; matching confidence matrix $\mathbf{C}$;

Initialize embedding function f and classifier S_{add};
for $i = 1 : I$ **do**
  // E-step:
  Update $\mathbf{C}$ by Eq. 26 (input f, S_{emb}, S_{add});
  Estimate $(i + 1) * N_p$ pairs of node matchings (input $\mathbf{C}$, output estimated matching sets t_k and associated confidence sets c_k for $v_k \in \mathcal{V}^s \cup \mathcal{V}^t$);
  // M-step:
  Obtain embedding function f by BTVec (Alg. 2) by the joint learning objective Eq. 30 (or other embedding models e.g. node2vec, deepWalk by corresponding learning algorithms in [12], [13]);
  Train optional classifier S_{add} using extended node attribute features if available;

4.2 The Vec4Cross Framework

We show how to employ GMM and EM in our approach.

4.2.1 Gaussian Mixture Model for Matching Estimation

Given an unmatched node $v_i^s \in \mathcal{V}^s$, supposing we already have its top-N_m candidate matchings $t_i = \{t_{i1}, \ldots, t_{iN_m}\}$ with corresponding confidence $\xi_i = \{\xi_{i1}, \ldots, \xi_{iN_m}\}$, then we use the Gaussian kernel function as a classifier to determine whether two nodes are matched. With $f(\cdot)$ denoted as the embedding function and $\phi(\cdot)$ denoted as the alignment mapping function, the probability that node $v_j^t \in \mathcal{V}^t$ is

matched with v_i^s is defined as:

$$p_a(\phi(v_i^s) = v_j^t | v_i^s) = \exp(-\|f(v_j^t) - f(v_i^s)\|_2^2), \quad (23)$$

Assuming that the true alignment node $\phi(v_i^s) \in t_i$, we define the distribution of alignment nodes for v_i by GMM:

$$P\big(\phi(v_i^s)|\Theta\big) = \sum_{k=1}^{N_m} w_{ik} p_a(\phi(v_i^s) = t_{ik}|v_i^s), \quad (24)$$

where $w_{ik} = \xi_{ik} / \sum_{j=1}^{N_m} \xi_{ij}$, controls the weight of the k-th Gaussian classifier. Θ refers to all parameters in the model, including the node embedding function f and other learnable parameters of the (optional) classifier. Supposing that $\phi(\cdot)$ is independent for different nodes (as assumed in the paper), the joint distribution of $\phi(\cdot)$ becomes the follows:

$$P(\phi|\Theta) = \prod_{v \in \mathcal{V}^s \cup \mathcal{V}^t} P\big(\phi(v)|\Theta\big). \quad (25)$$

4.2.2 Expectation-Maximization Learning

The estimated node matchings are generated by Eq. 24 and Eq. 25, then our model is learned by the following steps.

1) Initialization. In the beginning, the information about node embedding and the score of the classifier are in deficiency. To start the whole cross-network embedding procedure, we first initialize the confidence matrix according to the given seed matchings T. If handcrafted features are given, we train additional classifiers (see Sec. 4.2.3) with given seed alignments as positive samples and those randomly matched pairs as negative samples. In this way, the EM procedure is started. We denote the model parameters in i-th iteration as $\Theta^{(i)}$ in the following parts. In particular, the initialized parameters are denoted as $\Theta^{(0)}$.

2) E-step. This step updates the confidence matrix and then select node pairs with top confidences as new matchings.

Confidence matrix updating. Both the embedding and additional classifiers (if exists) are used to update $\mathbf{C}$, where the confidences are treated as latent variables in our model. To combine the embedding and additional classifiers, we use the max function to fuse the embedding similarity function $S_{emb}(\cdot,\cdot)$ and normalized prediction scores by the additional classifier $S_{add}(\cdot,\cdot)$ as follows:

$$\mathbf{C}_{i,j} = \begin{cases} 1, & (v_i^s, v_j^t) \in T \\ 0, & (v_i^s, \cdot) \in T \wedge (\cdot, v_j^t) \in T \wedge (v_i^s, v_j^t) \notin T \\ \max\big(S_{emb}(v_i^s, v_j^t), S_{add}(v_i^s, v_j^t)\big), & \text{otherwise} \end{cases}. \quad (26)$$

Note that the classifier S_{add} is designed orthogonal to structure features and is also optional as shown in Fig. 4.1.

Node similarity with non-common-neighbor penalty. We use cosine similarity with the penalty to measure cross-network node embedding similarity:

$$S_{emb}(v_i^s, v_j^t) = \big(1 + (\mathbb{1}_{CA} - 1)\gamma\big) CosSim(v_i^s, v_j^t), \quad (27)$$

where $\mathbb{1}_{CA}$ is a signal function that will be set to 1 if v_i^s and v_j^t share a common pair of matched nodes in neighborhood and 0 otherwise. γ is the hyper-parameter which indicates penalty weight to punish the node pairs with $\mathbb{1}_{CA} = 0$. In extreme cases, if we set $\gamma = 1$, then all node pairs without a common matched node in the neighborhood will be regarded as impossible to be matched. If γ is set to 0, then $\mathbb{1}_{CA}$ will make no effect on node similarity measurement. The cosine similarity between nodes v_i, v_j is defined as:

$$CosSim(v_i, v_j) = \frac{f(v_i)^\top f(v_j)}{\|f(v_i)\|_2 \|f(v_j)\|_2}. \quad (28)$$

We calculate the confidence matrix and select node pairs with relatively high confidence as newly estimated matchings, then set ξ_{ik} with the corresponding confidence.

Estimated matchings selection. Obtaining the confidence matrix $\mathbf{C}$, we select node pairs with top confidences as newly estimated matchings. In the matching procedure, we limit the total number of estimated matchings to no more than N_p, and each node has no more than N_m matching candidates. For nodes in seed matchings, we do not generate corresponding newly estimated matchings. As such, estimated matchings are updated to learn parameters $\Theta^{(i)}$.

At last, we give the expectation of log probability of Eq. 25 as follows, preparing for the coming M-step:

$$Q(\Theta, \Theta^{(i)}) = E\Big[\log P(\phi|\Theta^{(i)}) \big| T, G^s, G^t\Big]. \quad (29)$$

3) M-step. To embed network structures, the loss of source network L^s and that of target L^t as constraints (e.g. Eg. 7) are added. While alternatives can be other single-network embedding methods. Accordingly, the objective becomes:

$$\Theta_l^{(i+1)} = \arg\min_{\Theta_l} \left(\underbrace{L^s + L^t}_{\text{embedding loss}} - \underbrace{\beta Q(\Theta, \Theta^{(i)})}_{\text{alignment loss}} \right), \quad (30)$$

where β controls the weight of the expectation term in the final objective. Note our Vec4Cross framework is orthogonal to the specific design of the embedding model, and the embedding terms L^s and L^t in Eq. 30 can take different forms. When the loss in Eq. 7 is used, the whole loss function can be readily solved by BTWalk and ASGD, similar to the procedure in Sec. 3.4, which leads to our alignment method namely BTCross. Alternatively, our framework can also incorporate other embedding models e.g. node2vec and DeepWalk by replacing the embedding terms L^s and L^t, accordingly. In such cases, the Vec4Cross framework can also derive other joint embedding and alignment algorithms by adopting the corresponding learning algorithm in [12], [13]. In this paper, we term them n2v-Cross and dw-Cross, respectively, which will also be evaluated in experiments.

4.2.3 Extended Features for Matching Estimation

Many handcrafted features have been devised for different types of networks. For example, for PPI networks, graphlets [38] show good capacity and Jaccard Coefficient is commonly used in social network alignment [25], [30]. This part serves as an additional classifier as mentioned in the confidence matrix updating step in Sec. 4.2.2.

Structural features. Traditional human-defined structural features can also contribute to accurate social network alignment [25], [30]. Following the state-of-the-art network alignment method using traditional features [25], we use the

revised Jaccard Coefficient to indicate the tie consistency of two nodes across networks. For a node pair (v_i^s, v_j^t), revised Jaccard Coefficient is defined as follows:

$$J_r(u,v) = \frac{|\phi(\mathcal{N}(v_i^s)) \cap \mathcal{N}(v_j^t)|}{|\mathcal{N}(v_i^s)| + |\mathcal{N}(v_j^t)| - |\phi(\mathcal{N}(v_i^s)) \cap \mathcal{N}(v_j^t)|}. \quad (31)$$

Attribute features. Attributes can be diverse. For instance, in social networks, username is supplementary to the information beyond network structure. Intuitively, in a small social group, people often choose different names to distinguish themselves [31]. Edit distance $ed(v_i, v_j)$ between usernames of v_i and v_j is a popular measurement of similarity between two strings.

Prediction scores. We train a linear SVM to predict the probability that two nodes v_i and v_j are matched:

$$S_1(v_i^s, v_j^t) = SVM\left(J_r(v_i^s, v_j^t), ed(v_i^s, v_j^t)\right). \quad (32)$$

4.3 Incorporating Single-Network Embedding Models

As shown in Line 7 of Alg. 3, our **Vec4Cross** framework can incorporate most single-network embedding methods, including **BTVec**, DeepWalk, node2vec. These variants will be evaluated and termed as BTCross, dw-Cross, and n2v-Cross, in the experiments. The learning procedure is also identical to ASGD as described in Sec. 3.5.

5 Experiments and Discussion

Experiments are performed on real-world datasets, with single-network embedding tasks i.e. multi-label classification and network reconstruction, as well as cross-network embedding applications e.g. network alignment. Most of the baselines, including LINE, DeepWalk, node2vec, FINAL, IsoRank, REGAL, our proposed BTVec, and BTCross are tested on a single machine using 16 threads with 128G memory, 4 physical CPU each with 12 cores (Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz). While the deep learning method SDNE runs on a single GPU (NVIDIA Corporation GP102 [GeForce GTX 1080 Ti]).

5.1 Single-network Tasks

5.1.1 Datasets, baselines, metrics and hyperparameters

Statistics of the tested social networks are given in Table 4. And the distributions of nodes' k-hop neighborhood sizes w.r.t node degrees are given in Fig. 5. All of the datasets are anonymized for privacy protection.

YouTube [20]: Users can add others to the friend list and join interest groups as treated as user labels. It is a large scale network with millions of nodes being very sparse.

YouTube-Cut [20]: In line with the protocol in [8], to make it a smaller network that can be handled by the high complexity model e.g. SDNE [8], we remove the unlabeled nodes from raw YouTube. It is also quite sparse.

Flickr [39]: On the website of Flickr, users can tag photos and join different interest groups. There are in total 195 interest groups used as labels in the raw dataset, many of which have few nodes for the classification. To make the experiment more convincing, we only retain 10 labels with the largest number of nodes while making no change to the network structure. It is a dense network.

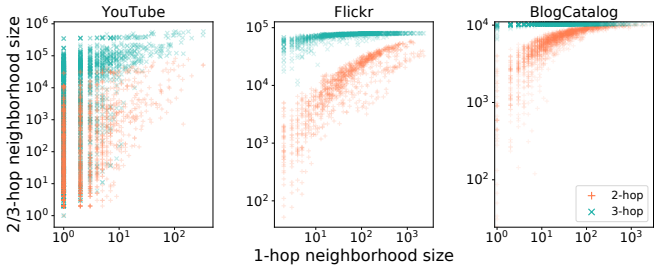


Fig. 5: The 2/3-hop neighborhood size w.r.t the 1-hop neighborhood size (i.e. node degree) in YouTube, Flickr, and BlogCatalog. It shows that nodes' neighborhood sizes usually grow unexpectedly, especially for low-degree nodes.

TABLE 4: Statistics for the single-network experiment.

Data	YouTube	YT-Cut	Flickr	BlogCtlog	Weibo	Twitter
$\lvert\mathcal{V}\rvert$	1,138,499	22,579	80,513	10,312	3,154	2,458
$\lvert\mathcal{E}\rvert$	2,990,443	95,506	5,899,882	333,983	241,736	95,034
#Labels	47	47	10	39	/	/
Avg. Deg.	5	8	146	65	77	39

BlogCatalog [39]: each blog is organized by some specific categories, which are the labels of a blogger, and bloggers have social connections with each other.

Weibo [24]: Weibo (https://www.weibo.com) is the largest micro-blog in China. Users can post their thoughts and photos, and can also follow or be followed by others.

Twitter [24]: Twitter is one of the most popular online social network platforms where users post 'tweets' to share their life moments. Social connections exist between users.

We compare with state-of-the-art embedding methods.

DeepWalk [12][3]: combines random walks and the Skip-Gram language model for embedding.

Node2vec [13][4]: substitutes the random walk procedure in DeepWalk as biased random walk.

SDNE [8][5]: a deep model capturing network structure and exploit the 1st-order and 2nd-order proximity jointly.

LINE [9][6]: models the first-order and second-order proximity by 1-hop vanilla random walk. We denote LINE-1st as the first-order proximity version.

ProNE [40][7]: initializes embedding by sparse matrix factorization, with enhancement via spectral propagation.

VERSE [11][8]: uses node similarity explicitly by Personalized PageRank (PPR), Adajcency Similarity, and SimRank, and it is based on Skip-Gram.

The following metrics are used for different tasks.

Multi-label classification. Popular metrics [8], [9], [12], [13], micro-F1 and macro-F1, are used in this paper.

Network reconstruction. $precision@N$ is adopted for performance evaluation. Given a network G, it is defined:

$$precision@N = \frac{\left|\{(i,j)|\Delta_{i,j} = 1, rank(i,j) < N, i < j\}\right|}{N}, \quad (33)$$

3. https://github.com/phanein/deepwalk
4. https://github.com/aditya-grover/node2vec
5. https://github.com/suanrong/SDNE
6. https://github.com/tangjianpku/LINE
7. https://github.com/THUDM/ProNE
8. https://github.com/xgfs/verse

TABLE 5: Networks used on multi-label classification.

Networks	1-hop α	2-hop α	w	N_w	#threads
Youtube	0.1	0.03	0.2	1e10	16
Flickr	0.02	0.03	0.2	1e9	
BlogCatalog	0.03	0.05	0.2	1e9	
Youtube-Cut	0.1	0.1	0.2	1e8	

where $\Delta_{i,j} = 1$ indicates that $(v_i, v_j) \in \mathcal{E}$ or $(v_j, v_i) \in \mathcal{E}$, $rank(i, j)$ is the rank of Euclidean distance between node (v_i, v_j) among $\{(v_i, v_j)|v_i, v_j \in \mathcal{V}, i < j\}$.

The hyperparameters of the compared methods are either recommended by the original paper or fine-tuned to achieve its best performance. The detailed settings are:

- DeepWalk: $window\_size = 10$, walk length $K = 40$, number of walks $N_w = 10$.
- Node2vec: $p = 2$, $q = 0.5$, $window\_size = 10$, and the rest parameters are the same as DeepWalk.
- SDNE: $\alpha = 100$, $\beta = 10$, $\gamma = 1$, $reg = 1$, and the layer size settings are $[10312 - 1000 - 128]$ on BlogCatalog, $[22579 - 2000 - 128]$ on YouTube-Cut as recommended in [8], and $[3154 - 800 - 128]$ on Weibo, $[2458 - 800 - 128]$ on Twitter.
- LINE: the only parameter N_w is the same as in BTVec.
- ProNE: the default setting, i.e. the term number of the Chebyshev expansion $k = 10$, $\theta = 0.5$, and $\mu = 0.2$. We use the enhanced embedding as the result.
- The proposed BTVec: see details in Table 5.

5.1.2 Experiments on Multi-Label Classification

For multi-label classification [41], each node has one or more labels for prediction. First, we generate the representations for nodes in the networks by BTVec and the peer methods. Then we adopt the LIBLINEAR package [42] to train the one-vs-rest Logistic regression classifiers. We randomly sample a portion of the labeled nodes, whose representations are set as training data and the rest for testing. Specifically, we randomly sample 1% to 10% of vertexes for training for YouTube, YouTube-Cut and Flickr, 10% to 90% for BlogCatalog. For each method, we repeat the procedure for 10 trials, and the average results are reported.

As shown in Fig. 6, BTVec outperforms the compared baselines consistently. We provide discussion as follows.

1) On the sparse networks (YouTube, YouTube-Cut) and dense networks (BlogCatalog, Flickr), BTVec performs the best, proving BTVec's adaptability to different densities.

2) Compared with methods that only use 1st- and 2nd-order proximity (LINE, SDNE), BTVec(1-hop) keeps outperforming them. Even compared with DeepWalk and node2vec that make use of high-order proximity, BTVec(1-hop) still outperforms them with limited proximity information, indicating the superiority of our proposed objective.

3) BTVec also outperforms node2vec which searches the neighborhoods by both BFS and DFS. It shows the greater competence of BTWalk to preserve local structure in a much more simple and faster way.

4) BTVec(2-hop) performs more stable than BTVec(1-hop). It indicates that richer neighborhood information (multi-order proximity modeling) can improve embedding.

We also conduct some further experiments in below.

Trade-off between speed and classification performance. Fig. 7 shows time cost and performance on BlogCatalog. LINE is the fastest with high performance. node2vec ($p = 0.25$ and $q = 0.25$ [13]) spends most of the time on calculating the transition probability, which slows it down severely compared with DeepWalk – a similar Random Walk based algorithms. SDNE is a deep model with high time complexity. In comparison, BTVec achieves the best performance with a relatively low time cost.

Effect of K-hop Branching Tree random walk. As shown in Fig. 6, BTVec(2-hop) outperforms its 1-hop version. Further ablation study is performed on Flickr to evaluate the effect of the number of hops K, whose result is shown in Fig. 8: 2-hop outperforms 1-hop significantly, while the F1 score of 3-hop drops slightly compared with 2-hop. It also proves the robustness of our model against a considerable part of noise (3-hop neighbors).

Scalability. Fig. 9 show the performance on BlogCatalog by adopting a parallel implementation. In Fig. 9, the relative speed of our method grows linearly with the number of threads. Fig. 9 shows that the F1-scores are not affected by the increased number of threads. These results verify the scalability of our ($\alpha = 0.03$ and $N_w = 1e8$).

5.1.3 Experiments on Network Reconstruction

A good network embedding method can preserve nodes' original local structure, which can be evaluated by network reconstruction. Given a network and the node representations, we reconstruct the network according to the distance between node embeddings with the graph's edges serving as ground-truth. $precision@N$ is used for evaluation.

The results are shown in Fig. 10. BTVec outperforms consistently across all datasets. Our analysis is as follows:

1) For dense networks (BlogCatalog, Weibo) whose complex structures bring a challenge to local structure preserving, BTVec performs well. It also works robustly on the relatively sparser network (Twitter).

2) Only using raw adjacency matrix, BTVec (1-hop) outperforms SDNE and LINE, showing the advantage of combining Laplacian Regularizer and Skip-Gram.

3) It can also be found that BTVec (2-hop) sometimes performs worse than BTVec (1-hop). It suggests that higher-order proximity might not contribute to local structure preserving but weight more on global structure.

4) Note DeepWalk performs also competitively with BTVec in most cases. It can be possibly interpreted as a result of long-range neighborhood modeling.

5.2 Cross-network Alignment

5.2.1 Datasets, baselines, metrics and hyperparameters

The statistics of datasets are shown in Table 6. For privacy, sensitive information like usernames is anonymized.

Facebook-Twitter (Fb-Tt) [24]: The ground truth alignment information is crawled from the website *About.Me*, a third-party for associating users' online accounts. To have a clean and direct evaluation, we remove the nodes (and their edges) without ground-truth matching information and those whose degree is less than 5 from the datasets.

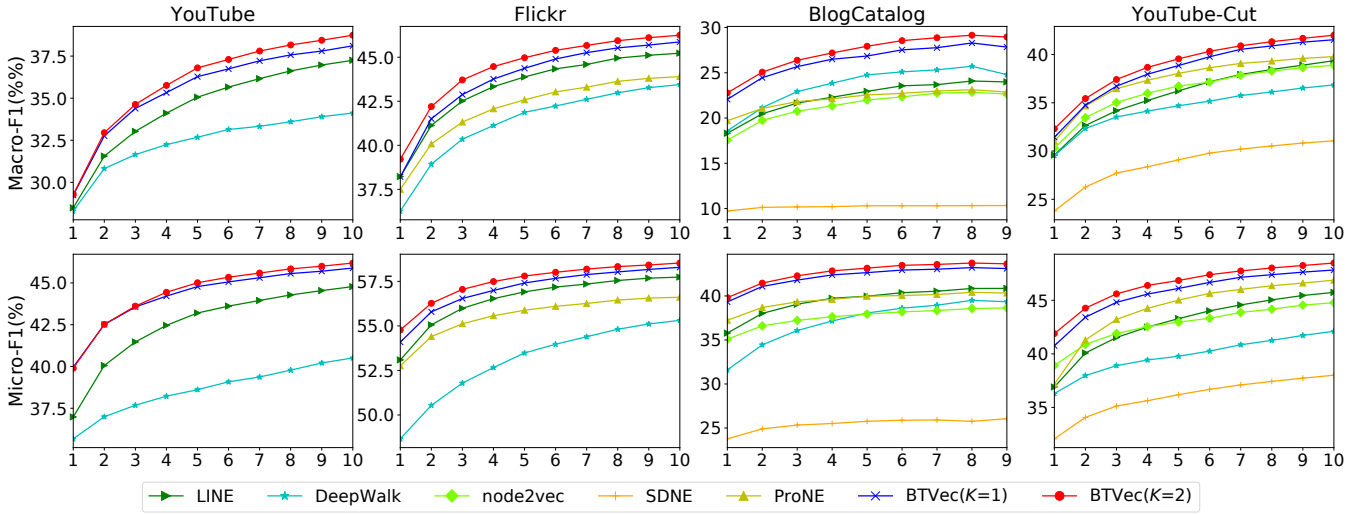


Fig. 6: BTVec outperforms on multi-label classification where x-axis denotes data portion (%) and y-axis is the score.

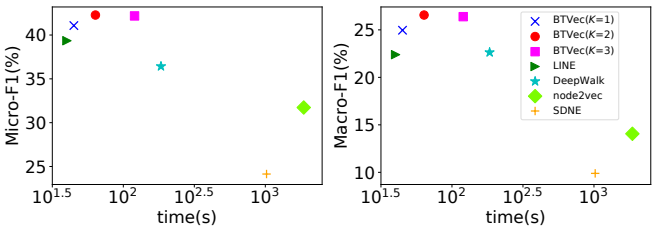


Fig. 7: Comparison on time cost and F1-score on BlogCatalog with 40% training data.

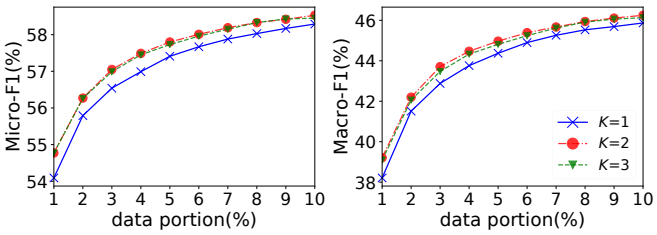


Fig. 8: Comparison of 1,2,3-hop on Flickr dataset.

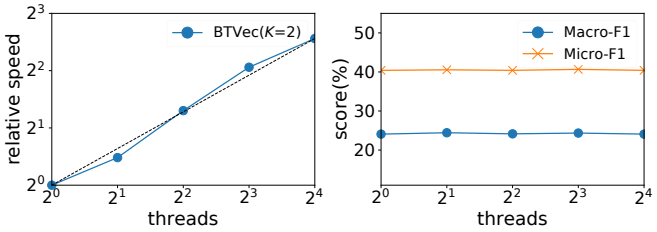


Fig. 9: Parallelizing performance on BlogCatalog dataset.

Douban-Weibo (Db-Wb) [24]: Weibo and Douban (https://www.douban.com/) are the largest microblog site and movie-rating site in China respectively. The alignment information is crawled from user pages of Douban and network information from each platform respectively. Akin to Fb-Tt, we cut nodes without ground-truth matching and whose degrees are less than 30 from the datasets.

TABLE 6: Statistics for the cross-network experiment.

cross-network	Fb-Tt	Db-Wb	Flickr
$\|\mathcal{V}^S\|$	2,458(Fb)	3,154(Db)	1,191
$\|\mathcal{V}^T\|$	2,458(Tt)	3,154(Wb)	
$\|\mathcal{E}^S\|$	40,298	301,074	509,816
$\|\mathcal{E}^T\|$	95,034	241,736	
Avg. Deg. of G^S	16	95	428
Avg. Deg. of G^T	39	77	
Overlap	0.28	0.36	/

Flickr [39]: To make a more comprehensive evaluation by varying the degree of overlapping between two networks and their density, we sample two sub-networks from Flickr for alignment. The density level and overlapping level of two sub-networks G^s and G^t are controlled by hyper-parameters δ_s (the larger the denser) and δ_c (the larger the more overlapping) respectively.

We perform the following procedure in line with [22], to generate G^s, G^t satisfying the probability constraints by Eq. 34: 1) For each edge in the raw network G, generate a random value r with the uniform distribution in $[0, 1]$. 2-a) If $r \leq 1 - 2\delta_s + \delta_s\delta_t$, the edge is discarded; 2-b) If $1 - 2\delta_s + \delta_s\delta_t < r \leq 1 - \delta_s$, the edge is preserved in G^s; 2-c) If $\leq 1 - \delta_s < r \leq 1 - \delta_s\delta_c$, the edge is preserved in G^t; 2-d) Otherwise, it is preserved in both G^s and G^t. Through this sampling procedure, we are able to ensure that:

$$p(edge \in G^s | edge \in G) = p(edge \in G^t | edge \in G) = \delta_s$$
$$p(edge \in G^t | edge \in G^s) = p(edge \in G^s | edge \in G^t) = \delta_c. \quad (34)$$

We compare three kinds of baselines: 1) Single-network embedding methods within our framework; 2) State-of-the-art network alignment approaches that accept similar input (seed matchings) and output a score matrix; 3) Our cross-network embedding approach (with handcrafted features).

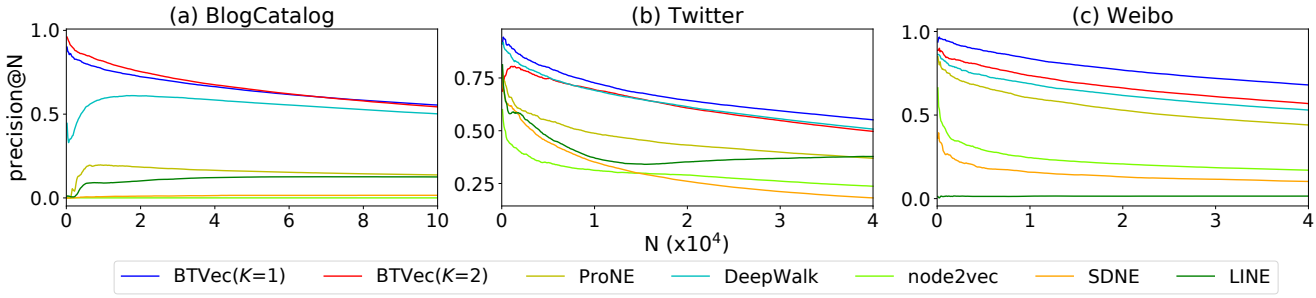


Fig. 10: BTVec consistently outperforms on single network reconstruction across different datasets.

Variants under Vec4Cross framework. We compare three single-network embedding methods under the Vec4Cross framework, namely **dw-Cross** (for DeepWalk), **n2v-Cross** (for node2vec), and **BTCross** (for BTVec). They are all trained without extended features. For dw-Cross and n2v-Cross, we set $f(u) \equiv f(v)$ when nodes u and v are matched, as a simplified alternative.

Network alignment methods. We compare with several existing network alignment algorithms including **FINAL** [43][9], **IsoRank** [44], and **REGAL** [33][10]. For FINAL and IsoRank, we input seed matchings as prior alignment knowledge and output an alignment matrix for evaluation. For REGAL, we use seed matchings as node attributes, and evaluate embedding similarity by cosine similarity (Eq. 28). Specifically, **FINAL** introduces a family of algorithms optimizing quadratic objective functions while **IsoRank** solves a version of the integer quadratic program with relaxed constraints. The prior alignment information is input in the form of a matrix $\mathbf{H}$. For our partial alignment settings, we generate $\mathbf{H}$ by the following process: For each node pair (v_i^s, v_j^t), a) if they are matched, then $\mathbf{H}_{i,j} = 1$; b) if one of them is matched with some other node, then $\mathbf{H}_{i,j} = 0$; c) otherwise, $\mathbf{H}_{i,j}$ is set to the expectation of probability that they will be matched, i.e. $\mathbf{H}_{i,j} = \frac{|UT|}{(|\mathcal{V}^s|-|T|)\times(|\mathcal{V}^t|-|T|)}$, where UT denotes the set of unknown node matching and T denotes set of existing matching seeds known as prior. Similarly, $\mathbf{H}$ serves as attribute inputs for **REGAL**.

BTCross with extended features. The version of BTCross with additional handcrafted features is termed by **BTCross+**. See Sec. 4.2.3 for details.

We use standard $P@N$ for node matching [21], [31], [32]:

$$P@N = \left|Success(v)@N | v \in \mathcal{V}^{UT}\right| / |\mathcal{V}^{UT}| \qquad (35)$$

where $\left|Success(v)@N | v \in \mathcal{V}^{UT}\right|$ is the number of nodes which can find the correct match in top-N choices. Here $\mathcal{V}^{UT}$ is set of nodes in all of the ground-truth matching.

For hyperparameters of IsoRank and FINAL, we set $\alpha = 0.82$, $maxiter = 30$, $tol = 1e-4$. The parameters of dw-Cross and n2v-Cross are the same as those in single-network tasks. For BTCross and BTCross+, see parameters in Table 7.

9. Both FINAL and IsoRank: https://github.com/maffia92/FINAL-network-alignment-KDD16

10. https://github.com/Allen517/alp-baselines

TABLE 7: Parameter settings for cross-network alignment.

	α	β	γ	w	N_w	K	N_m	N_p
Fb-Tt	0.08	1.0	0.07	0.2	3e8	2	2	200
Db-Wb	0.005	2.0	0.9	0.2				
Flickr	0.075	2.0	0.9	0.2				

5.2.2 Results on Network Alignment

Performance on real-world data. We use Facebook-Twitter (Fb-Tt) and Douban-Weibo (Db-Wb). We evaluate the methods with $10\%/50\%$ ground-truth matchings as seed matchings, where the seeds for the smaller ratio is a subset of a larger one. The results in Fig. 11 show that BTCross outperforms baselines mostly. We provide analysis as follows:

1) For alignment over both dense networks (Db-Wb) and sparse networks (Fb-Tt), methods within Vec4Cross give a stable performance. While in contrast, FINAL and IsoRank perform worse on Db-Fb which is several times denser than Fb-Tt, suggesting they do not adapt well to dense networks. And REGAL does not show good performance over both of the real-world datasets. It proves the stability of Vec4Cross framework over networks with different levels of density.

2) For all methods, accuracy grows as the number of seed matchings increases. Among the family of Vec4Cross, our BTCross keeps its superior advantages consistently, especially when the portion of seed matchings is small (10%), showing its robustness for challenging cases.

Performance on simulated data. To verify the conclusion above, we further study on the controlled synthetic Flickr datasets. We first vary the density level δ_s as $0.1/0.4$ while fixing overlapping level $\delta_c = 0.3$, which is a common value in real-world condition according to Table 6. Then we fix $\delta_s = 0.15$, which guarantees the nodes in networks with a proper degree, and set δ_c as $0.1/0.4$. We randomly sample 30% from ground truth alignments as seeds. We analyze the results in Fig. 12 as follows:

1) As shown in Fig. 12(a),(b), it becomes more challenging as the networks turn denser. The Vec4Cross family outperforms FINAL and IsoRank consistently by a notable margin especially on denser networks. RGEAL shows good capacity when the network goes denser compared to other methods. These results further verify the conclusion made in the above experiments that FINAL and IsoRank suit sparse networks better compared with dense ones, while the Vec4Cross family works well for both.

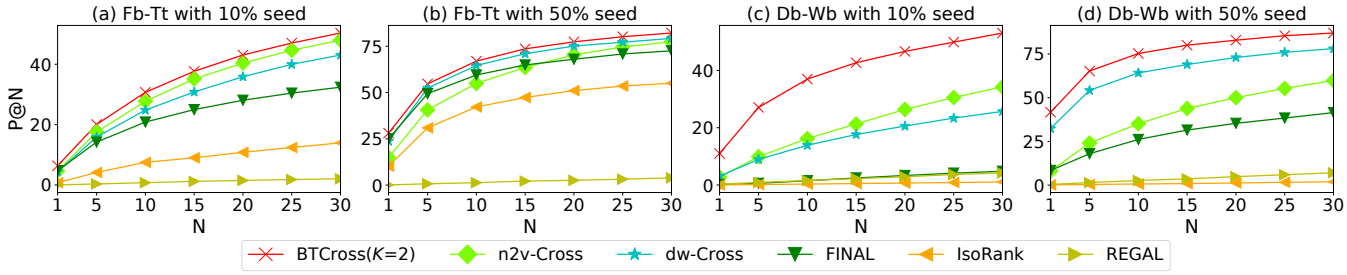


Fig. 11: Performance of network alignment on two real-world datasets Facebook-Twitter (Fb-Tt) and Douban-Weibo (Db-wb). Our proposed cross-network embedding method outperforms baselines in most cases.

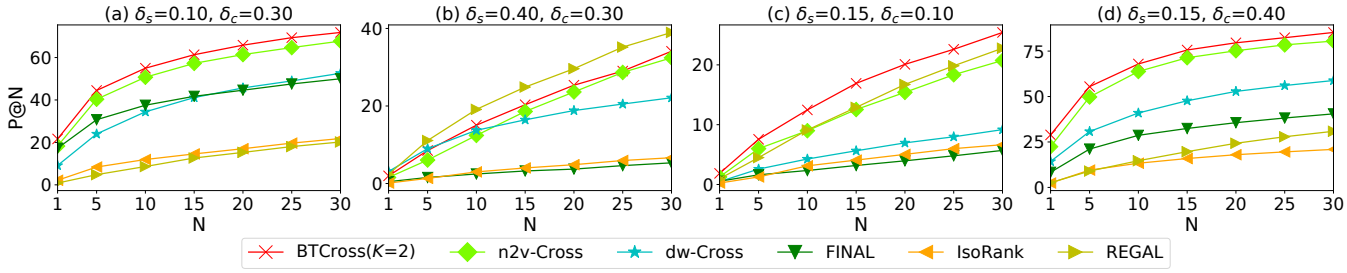


Fig. 12: Performance of network alignment on Flickr. δ_s describes the density level of the two synthetic networks and δ_c indicates the overlapping level between them. Our proposed BTCross outperforms the baselines mostly.

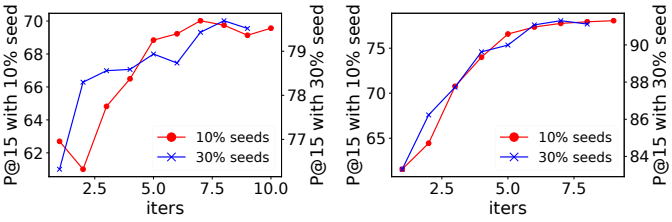
2) From Fig. 12(c),(d), we can observe that a higher overlapping level between networks makes the alignment task easier for most baselines. However, compared with others, REGAL works relatively better under a lower overlapping level. For BTCross, its $P@N$ keeps more than 400% of FINAL's scores when the overlapping level is very low ($\delta_c = 0.1$), and also maintain its supremacy at a high overlapping level. The results demonstrate the capability of Vec4Cross for all the overlapping levels.

The above results suggest that our proposed BTCross works well on networks with a low overlapping ratio and dense structure, which is common in practice while challenging. The results also show that methods under our Vec4Cross framework perform stably whenever networks are sparse/dense and the overlapping level of the source network and target network are low/high.

Boosting by EM algorithms. We show how our method improves basic network alignment methods. Take BTCross+ as an example, we run experiments on Fb-Tt and Db-Wb with the portion of 10% and 30% ground-truth as seeds. Fig. 13 shows the convergence process of $P@15$ over iterations. We provide analysis as follows:

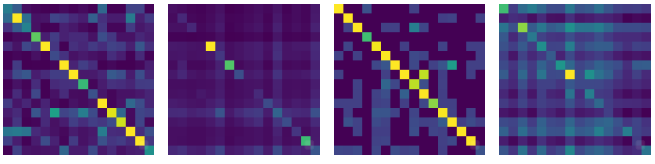
1) In Fig. 13(a), BTCross+ boosts over iterations. The newly estimated matchings enrich the alignment information thus improving the performance significantly even with a very low portion of alignments (10%). Though the newly estimated matchings might bring noise (the little drop of 10% seeds at iteration 2 in Fig. 13(a)), as the EM process continues, it converges to a relatively high performance.

2) Compared with BTCross in Fig. 11, the extended version with additional human-defined features i.e. BTCross+ outperforms baselines notably, suggesting the advance of incorporating additional human-defined features.



(a) Fb-Tt (b) Db-Wb

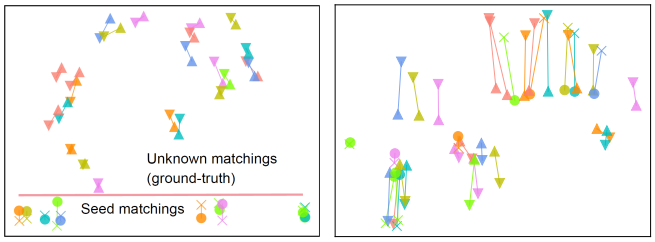
Fig. 13: Convergence of BTCross+ on Fb-Tt and Db-Wb.



(a) BTCross+ (b) FINAL (c) BTCross+ (d) FINAL

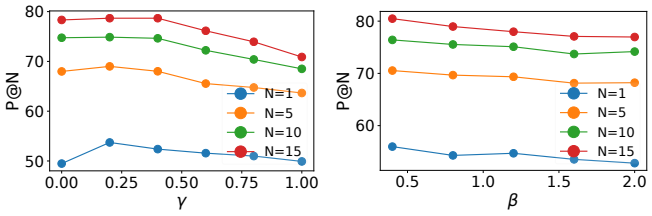
Fig. 14: Confusion matrix for node matching on Fb-Tt (left two) and Db-Wb (right two) with 30% portion of ground truth matchings as seeds. A lighter color indicates a higher confidence for the corresponding two nodes to be matched.

Heatmap analysis. We visualize the alignment result by heatmaps in Fig. 14. We randomly sample 16 pairs of nodes in ground-truth matching but not included in seed alignments, and visualize the pairwise confidence (BTCross+) or the probability score (FINAL) that they are aligned after iterations. From the figure, we can safely claim that our



(a) Karate networks by BTCross (b) Karate networks by BTVec

Fig. 15: PCA projection for embedding results of cross-karate networks. Nodes that are linked to each other are true matching across networks. Different shapes are used to distinguish seed matching from unknown matchings.



(a) $P@N$ w.r.t. γ (b) $P@N$ w.r.t. β

Fig. 16: Sensitivity w.r.t. weight of non-common-neighbor penalty γ, and weight of the expectation term, β in M-step.

vec4Cross+ yields a notably better matching result than FINAL. In Fig. 14(a) and Fig. 14(c), our BTCross+ can effectively distinguish the ground-truth matchings among all of the nodes, while FINAL fails in Fig. 14(b) and Fig. 14(d).

Visualization for cross-Karate networks. We use Karate network [45] and its copy as the target network, and then conduct the cross-network embedding experiment with 30% ground truth as seed matchings. The results are visualized in Fig. 15, for which we give analysis as follows:

1) As shown in Fig. 15(a), seed matchings and unknown ground truth matchings are divided clearly by the split line, while BTVec in Fig. 15(b) mixes them together.

2) Our BTCross performs competitively where the matchings are projected closer than BTVec, proving the capability of Vec4Cross framework to extend single-network embedding methods for network alignment.

Parameter Sensitivity. We study the performance of our proposed cross-network embedding model BTCross+ w.r.t. some main parameters, γ and β. We fix the other parameters the same as Table 7. Results are shown in Fig. 16:

1) Best performance is achieved with $0.4 > \gamma > 0$, which indicates that non-common-neighbor penalty can help moderately improve the alignment performance.

2) With β changing, performance stays quite stable, which indicates that the hyper-parameter β does not make big influence on the final matching precision.

5.3 Further Analysis

In this subsection, we do some further analysis on BTVec, including BTWalk and Laplacian regularizer.

Source analysis for improvement. We analyze the source of the improved performance by evaluating both the

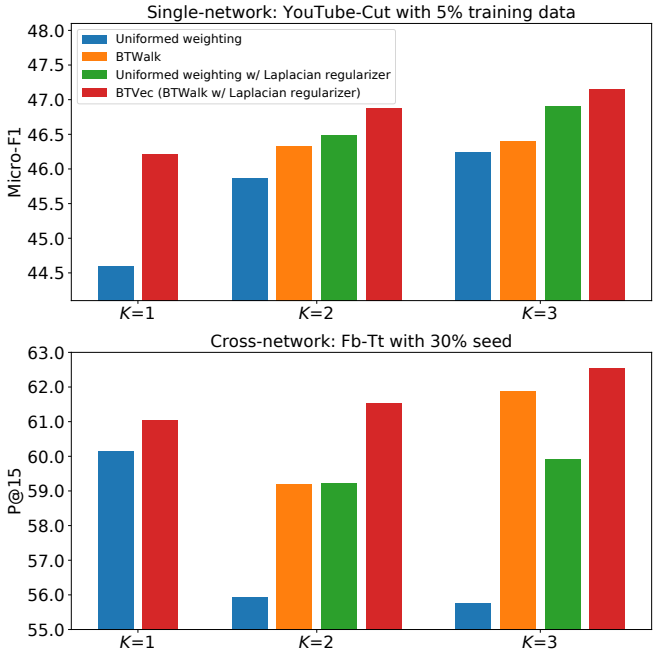


Fig. 17: Comparison by using different hops. Top: results on single-network embedding; Bottom: results on cross-network embedding. Among the baselines, the uniformed weighting sets the multi-order weight $m^{(k)} = 1$ for all the nodes, while BTWalk sets node-specific weights by Eq. 10. We compare the uniformed weighting and BTWalk to the versions with Laplacian regularizer according to Sec. 3.2.

single-network embedding and cross-network embedding on two real-world datasets, YouTube-Cut and Fb-Tt, for the tasks of node classification and network alignment respectively. The results are given in Fig. 17. For all the methods, $1e8$ random walk sequences are sampled and it runs with 16 threads, with the embedding dimension $z = 128$. On YouTube-Cut, we set $w = 0.3$ for BTWalk and $\alpha = 0.1$ for Laplacian regularizer. On Fb-Tt, we set $w = 0.2$ for BTWalk and $\alpha = 0.01$ for Laplacian regularizer.

On single-network embedding, we find that the classification results of all the methods improve as K grows larger. Among them, BTVec keeps outperforming the other baselines. Also, we find that both BTWalk and Laplacian regularizer have positive effects on the experimental results compared with uniformed weighting w/o Laplacian regularizer. And somehow, since BTVec, which can be seen as a combination of BTWalk and Laplacian regularizer on skip-gram model, outperforms both BTWalk w/o Laplacian regularizer and uniformed weighting w/ Laplacian regularizer, BTWalk and Laplacian regularizer can have orthogonal positive effects on the vanilla skip-gram model.

For cross-network embedding, the performance given by uniformed weighting declines for larger K. That might be caused by an improper weighting on multi-order proximity as neighborhoods grow. In comparison, the other three keep improving and outperforming uniformed weighting as K grows. It indicates that BTVec and BTWalk are able to deal with larger and more complex neighborhoods.

Laplacian regularizer on other methods. We conduct

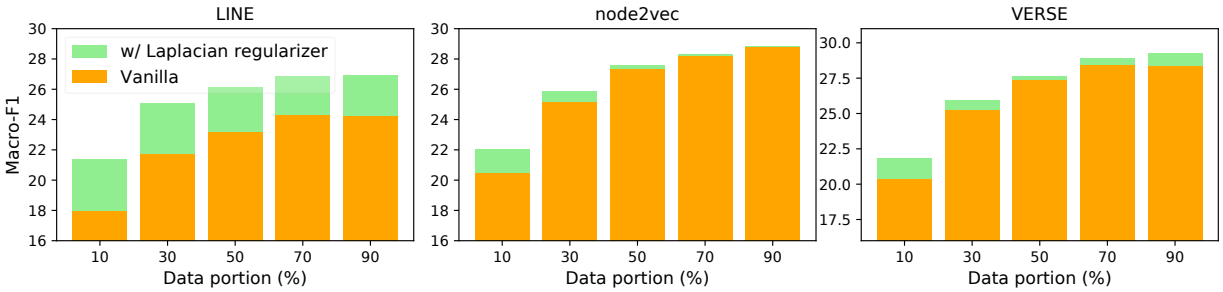


Fig. 18: Comparison of several skip-gram based single-network embedding methods w/ and w/o Laplacian regularizer.

experiments by adding Laplacian regularizer to several skip-gram based embedding methods, LINE, node2vec[11] and VERSE [11]. We set $\alpha = 0.1$ for LINE, $\alpha = 0.01$ for node2vec and VERSE. LINE is run with sampling number as $1e8$. node2vec is run with window size as 10, $p = 1$ and $q = 1$. VERSE is run with PPR similarity, where damping factor is set as 0.85. Embedding dimension is set as $z = 128$. The results are shown in Fig. 18, where the Macro-F1 scores on BlogCatalog are given. We find that Laplacian regularizer can improve the performance.

6 Conclusion

We have presented a network embedding approach with the following highlights: i) a clear weighted multi-order proximity model fulfilled by a combination of Skip-Gram probabilistic model and Laplacian Eigenmaps; ii) a fast and theoretically justified BTWalk technique based on branching tree-like BFS and DFS sampling; iii) an extension and adaption to the cross-network setting whereby embedding and alignment function can be learned jointly end to end. Experimental results show the efficacy of our approach.

References

[1] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[2] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in neural information processing systems*, 2002, pp. 585–591.

[3] X. Du, J. Yan, and H. Zha, "Joint link prediction and network alignment via cross-graph embedding," in *International Joint Conference on Artificial Intelligence*, 201.

[4] X. Du, J. Yan, R. Zhang, and H. Zha, "Cross-network skip-gram embedding for joint network alignment and link prediction," *IEEE Transactions on Knowledge and Data Engineering*, vol. PP, pp. 1–1, 05 2020.

[5] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[6] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1105–1114.

[7] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu, "Arbitrary-order proximity preserved network embedding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2778–2786.

11. https://github.com/xgfs/node2vec

[8] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 1225–1234.

[9] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[10] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM international on conference on information and knowledge management*. ACM, 2015, pp. 891–900.

[11] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller, "Verse: Versatile graph embeddings from similarity measures," in *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 2018, pp. 539–548.

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

[13] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.

[14] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena, "Don't walk, skip! online learning of multi-scale network embeddings," in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, ser. ASONAM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 258–265. [Online]. Available: https://doi.org/10.1145/3110025.3110086

[15] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

[16] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 385–394.

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

[18] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, "Structural deep embedding for hyper-networks," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[19] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. Alemi, "Watch your step: Learning node embeddings via graph attention," in *Neural Information Processing Systems*, 2018.

[20] L. Tang and H. Liu, "Scalable learning of collective behavior based on sparse social dimensions," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 1107–1116.

[21] F. Zhou, L. Liu, K. Zhang, G. Trajcevski, J. Wu, and T. Zhong, "Deeplink: A deep learning approach for user identity linkage," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1313–1321.

[22] T. Man, H. Shen, S. Liu, X. Jin, and X. Cheng, "Predict anchor links across social networks via an embedding approach," in *Pro-*

ceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, ser. IJCAI'16. AAAI Press, 2016, p. 1823–1829.

[23] D. Luo, F. Nie, H. Huang, and C. H. Ding, "Cauchy graph embedding," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 553–560.

[24] X. Cao and Y. Yu, "Asnets: A benchmark dataset of aligned social networks for cross-platform user modeling," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, ser. CIKM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1881–1884. [Online]. Available: https://doi.org/10.1145/2983323.2983864

[25] ——, "Bass: A bootstrapping approach for aligning heterogenous social networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2016, pp. 459–475.

[26] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 2018, pp. 459–467.

[27] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[28] J. Qiu, Y. Dong, H. Ma, J. Li, C. Wang, K. Wang, and J. Tang, "Netsmf: Large-scale network embedding as sparse matrix factorization," in *The World Wide Web Conference*. ACM, 2019, pp. 1509–1520.

[29] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[30] X. Kong, J. Zhang, and P. S. Yu, "Inferring anchor links across multiple heterogeneous social networks," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. ACM, 2013, pp. 179–188.

[31] S. Tan, Z. Guan, D. Cai, X. Qin, J. Bu, and C. Chen, "Mapping users across networks by manifold alignment on hypergraph," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[32] L. Liu, W. K. Cheung, X. Li, and L. Liao, "Aligning users across social networks using network embedding," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI'16. AAAI Press, 2016, p. 1774–1780.

[33] M. Heimann, H. Shen, T. Safavi, and D. Koutra, "Regal: Representation learning-based graph alignment," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2018, pp. 117–126.

[34] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 297–304.

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

[36] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

[37] Y. Zhang, J. Tang, Z. Yang, J. Pei, and P. S. Yu, "Cosnet: Connecting heterogeneous social networks with local and global consistency," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1485–1494.

[38] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Artificial Intelligence and Statistics*, 2009, pp. 488–495.

[39] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 817–826.

[40] J. Zhang, Y. Dong, Y. Wang, J. Tang, and M. Ding, "Prone: Fast and scalable network representation learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 4278–4284. [Online]. Available: https://doi.org/10.24963/ijcai.2019/594

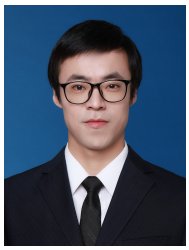
[41] W. Zhang, J. Yan, X. Wang, and H. Zha, "Deep extreme multi-label learning," in *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. ACM, 2018, pp. 100–107.

[42] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

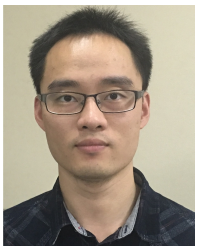
[43] S. Zhang and H. Tong, "Final: Fast attributed network alignment," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1345–1354.

[44] R. Singh, J. Xu, and B. Berger, "Global alignment of multiple protein interaction networks with application to functional orthology detection," *Proceedings of the National Academy of Sciences*, vol. 105, no. 35, pp. 12 763–12 768, 2008.

[45] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of anthropological research*, vol. 33, no. 4, pp. 452–473, 1977.



Hao Xiong is currently a Graduate Student with School of Electronic Information and Electrical Engineering from Shanghai Jiao Tong University, Shanghai, China. Before that, he received the B.E. degree in Cyber Science and Engineering (with honor) from Shanghai Jiao Tong University in 2019. He has been a research intern in Amazon AI Labs, Shanghai from 2019 to 2020. His research interests include machine learning, data mining, and network analysis.



Junchi Yan (M'10) is currently an Associate Professor with Department of Computer Science and Engineering, Shanghai Jiao Tong University. Before that, he was a Senior Research Staff Member and Principal Scientist with IBM Research, Shanghai, China, where he started his career in April 2011. He obtained the Ph.D. at the Department of Electrical Engineering of Shanghai Jiao Tong University, China in 2015. His research interests are machine learning and visual computing. He is a Member of IEEE.