ELTRA: An Embedding Method based on Learning-to-Rank to Preserve Asymmetric Information in Directed Graphs

Masoud Reyhani Hamedani
masoud@hanyang.ac.kr
Hanyang University, Seoul, Korea

Jin-Su Ryu
jinsu97@hanyang.ac.kr
Hanyang University, Seoul, Korea

Sang-Wook Kim*
wook@hanyang.ac.kr
Hanyang University, Seoul, Korea

ABSTRACT

Double-vector embedding methods capture the asymmetric information in directed graphs first, and then preserve them in the embedding space by providing *two* latent vectors, i.e., source and target, per node. Although these methods are known to be *superior* to the single-vector ones (i.e., providing a *single* latent vector per node), we *point out* their three drawbacks as inability to preserve asymmetry on NU-paths, inability to preserve global nodes similarity, and impairing in/out-degree distributions. To address these, we first propose *CRW*, a *novel* similarity measure for graphs that considers contributions of *both* in-links and out-links in similarity computation, *without* ignoring their directions. Then, we propose *ELTRA*, an *effective* double-vector embedding method to preserve asymmetric information in directed graphs. ELTRA computes *asymmetry preserving proximity scores* (AP-scores) by employing CRW in which the contribution of out-links and in-links in similarity computation is *upgraded* and *downgraded*, respectively. Then, for every node u, ELTRA selects its top-t *closest* nodes based on AP-scores and *conforms* the *ranks* of their corresponding target vectors w.r.t u's source vector in the embedding space to their *original* ranks. Our extensive experimental results with *seven real-world* datasets and *sixteen* embedding methods show that (1) CRW *significantly* outperforms Katz and RWR in computing nodes similarity in graphs, (2) ELTRA *outperforms* the existing state-of-the-art methods in graph reconstruction, link prediction, and node classification tasks.

CCS Concepts

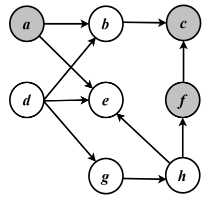
• **Information systems** → **Social networks**.

Keywords

directed graph embedding, link-based similarity, learning-to-rank

ACM Reference Format:
Masoud Reyhani Hamedani, Jin-Su Ryu, and Sang-Wook Kim. 2023. ELTRA: An Embedding Method based on Learning-to-Rank to Preserve Asymmetric Information in Directed Graphs. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3614862

*Sang-Wook Kim is the corresponding author.



Node-pairs	CRW	RWR	Katz	AP-score
(a, c)	0.0485	0.1847	0.0100	0.0165
(c, a)	0.0257	0	0	0.0037
(a, f)	0.0102	0	0	0.0013
(f, a)	0.0139	0	0	0.0011
(d, g)	0.1065	0.1025	0.1000	0.0606
(g, d)	0.3367	0	0	0

Figure 1: A sample graph.

1 INTRODUCTION

Graph embedding methods[1] (in short, embedding methods) aim to map nodes in a graph into low-dimensional latent vectors while preserving the topological information within the graph structure [5, 10, 18, 29]. The obtained latent vectors can be utilized by various machine learning tasks such as graph reconstruction [15, 28], link prediction [15, 28, 39, 43, 44], and node classification [42, 47, 48]. In general, embedding methods can be applied to both directed and undirected graphs; however, the graph embedding is *more challenging* for directed graphs than undirected ones since contrary to the latter, the former has a *asymmetric nature* [15, 28, 34]. In directed graphs, links represent the relation between objects (i.e., nodes) in a domain (e.g., social networks, citation networks, and protein networks) and their *directions* do asymmetric information, which should be preserved in the embedding space [15, 28, 34, 45, 48]; for example, in a citation graph, a directed link $\overrightarrow{uv}$ denotes a citation relationship between papers u and v in which u cites v but *not* inversely. In this paper, we focus on graph embedding methods for directed graphs (simply, *graphs from now on*).

Single-vector embedding methods such as DeepWalk [29], DWNS [3], FREDE [38], GELTOR [8], Gravity [33], NetMF [31], node2vec [5], and VERSE [37] provide a *single* latent vector $\vec{u}$ per node u in a graph, thereby having limitations in preserving the asymmetric information in the embedding space [15, 28, 34, 48]; the value of $\vec{u}\cdot\vec{v}$ indicates *only* the probability of having a link or path between u and v in the graph, but *unable* to represent the direction. To alleviate this problem, *double-vector* embedding methods *capture* the asymmetric information in graphs first, and then try to *preserve* them in the embedding space as follows. They consider *two* roles, i.e., *source* and *target*, for any nodes u in the graph based on an assumption that a directed link or path exists from a source to a target. Then, they provide two respective latent vectors $\overrightarrow{u_s}$ and $\overrightarrow{u_t}$ as a source vector and a target vector where the value of $\overrightarrow{u_s}\cdot\overrightarrow{v_t}$ indicates the probability of having a link or a path *only* from u as a source to v as a target. Random-walk-based methods such as ATP [34], DIVINE [44], DNE [48], and NERD [15] try to preserve the neighborhood of nodes in the graph among their corresponding latent vectors in the embedding space, while similarity-based ones such as HOPE

[1]In this paper, we focus on embedding methods that exploit *only* the topological information of the graph structure in the embedding process.

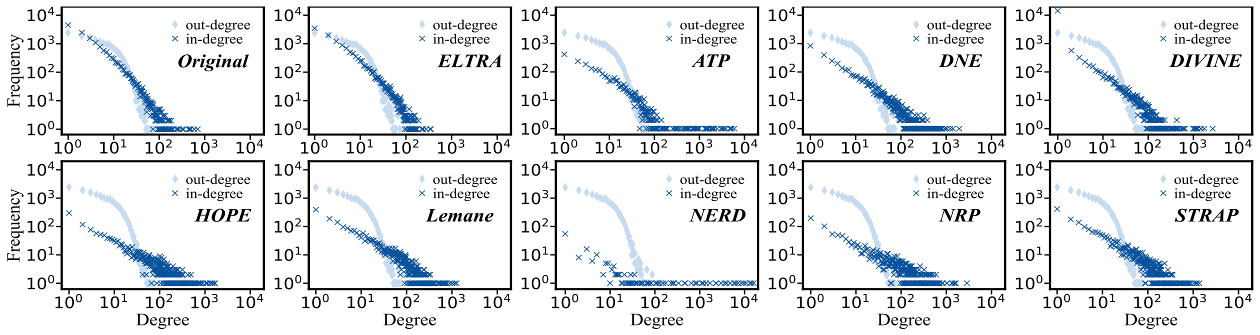


Figure 2: The original in/out-degree distributions of the DBLP dataset and the ones reconstructed by different methods.

[28], Lemane [47], NRP [42], and STRAP [43] try to conform the similarity distribution of latent vectors in the embedding space to that of nodes in the graph. Although double-vector methods are known to be superior to the single-vector ones in various machine learning tasks [15, 28, 34, 44, 48], we point out their three drawbacks. Before proceeding, let us introduce the following three notions by referring to Figure 1 that shows a sample graph and the similarity scores of four node-pairs computed by different measures[2].

Definition 1: In a graph, we define a *unidirectional path* (**U-path**) from node u to v as a sequence of nodes $u=v_0, v_1, v_2, ..., v_l=v$ where directed links exist *only* from v_{i-1} to v_i $(1 \leq i \leq l)$; e.g., $a \rightarrow b \rightarrow c$ is a U-path in Figure 1.

Definition 2: In a graph, we define a *non-unidirectional path* (**NU-path**) from node u to v as a sequence of nodes $u=v_0, v_1, v_2, ..., v_l=v$ where, contrary to U-paths, *at least* one directed link exists from v_i to v_{i-1} $(1 \leq i \leq l)$; e.g., $a \rightarrow b \rightarrow c \leftarrow f$ is a NU-path in Figure 1.

Definition 3: To capture the asymmetric information in a graph, most of the existing methods exploit *only* out-links based on an assumption that U-paths exists from a source to a target. However, it has been shown that the asymmetric information exist *not only* between direct neighbors (e.g., d and g in Figure 1) and any pairs of nodes connected by U-paths (e.g., a and c) *but also* between those pairs of nodes connected by NU-paths (e.g., a and f) where *NU-paths from a source to a target contain a more number of out-links than in-links* [48]; we call this an **asymmetry on NU-paths**.

Now, we point out three drawbacks of existing double-vector embedding methods as follows:

D1. Inability to Preserve Asymmetry on NU-paths: ATP and DIVINE have limitations to capture asymmetry on NU-paths due to employing normal random walks where *only* out-links are traversed. In NERD, although random walks traverse both in-links and out-links, only the paths with *alternately* changing link directions are considered (e.g., path $a \longrightarrow e \longleftarrow h$ is considered, while $a \longrightarrow b \longrightarrow c \longleftarrow f$ is not), thereby still having limitations to completely solve the problem. DNE solves this problem by considering NU-paths; however, it still *suffers* from drawbacks **D2** and **D3**. HOPE employs Katz [13], and NRP, STRAP, and Lemane employ random walk with restart (RWR) [36] to compute asymmetric similarity scores of nodes in the graph and utilize them in the embedding process. To compute the asymmetric similarity score of a node-pair (u, v), these two measures consider the contributions of *only* U-paths from u to v [8, 46], thereby having limitations in capturing the asymmetry on NU-paths; consequently, HOPE, NRP, STRAP, and Lemane *cannot* address the problem. For example, in Figure 1, Katz and RWR assign *non-zero* and *zero* similarity scores to node-pairs (d, g) and (g, d), respectively, and the same respective circumstances are observed for (a, c) and (c, a); these two measures capture the asymmetric information for directly connected nodes d and g and also that for a and c connected by a U-path. However, they assign zero similarity scores to *both* (a, f) and (f, a), showing they cannot capture the asymmetry between a and f connected by NU-paths.

D2. Inability to Preserve Global Nodes Similarity: ATP, DIVINE, DNE, and NERD employ random walks to capture the neighborhood of nodes in the graph. They utilize an *implicit* and *local* neighborhood views of nodes in the embedding process, thus unable to preserve the *global similarity* of nodes in the graph. This limits the capability of latent vectors for graph inference and analysis in diverse machine learning tasks [8, 14, 37, 38].

D3. Impairing In/Out-Degree Distributions: In ATP and DIVINE, random walks traverse only out-links, thereby having *difficulties* in sampling nodes with *zero* out-degrees or *low* in-degrees [15, 44, 48]. In NERD and DNE, although random walks traverse both out-links and in-links, NRED has limitations in *covering* NU-paths and DNE traverses out-links and in-links *uniformly* by equal probabilities (e.g., in Figure1, the probabilities that a random walk on h visits e and f are regarded equal to that of visiting g) *without* following in/out-degree distributions in the graph. In HOPE, Lemane, NRP, and STRAP similarity scores are transformed into a *distribution* via normalization (i.e.,$\sum_{v \in V} S(u, v)=1$ for any node u); therefore, the reconstructed graph based on the latent vectors may *not* follow power-law distributions [8]. Consider Figure 2 that illustrates the original in/out-degree distributions of the DBLP [6, 8] dataset and those reconstructed by the above eight embedding methods and our ELTRA; the degree distributions obtained by the eight competitors are *quite* different from the original one.

In this paper, we first propose *CRW*, a new similarity measure for graphs based on **C**omprehensive **R**andom **W**alks. CRW employs a *novel* random walk strategy, which enables to consider contributions of *not only* U-paths *but also* those of NU-paths in similarity computation, *without* ignoring the link directions. We also propose a *matrix form* for CRW, which *dramatically* accelerates its computation while providing *exact* similarity scores.

Then, we propose *ELTRA*, a *novel* and *effective* similarity-based double-vector **E**mbedding method based on listwise **L**earning-**T**o-**R**ank (LTR) [1, 40, 41] that preserves **A**symmetric information in

[2]CRW and RWR scores are computed in five iterations; the value of C in CRW is set as 0.6, and those of α and β in RWR and Katz are set as 0.15 and 0.10 by following [36] and [28], respectively.

Table 1: Comparing embedding methods in terms of the three drawbacks

	ATP	DIVINE	DNE	HOPE	Lemane	NERD	NRP	STRAP	ELTRA
D1	✗	✗	✓	✗	✗	✗	✗	✗	✓
D2	✗	✗	✗	✓	✓	✗	✓	✓	✓
D3	✗	✗	✗	✗	✗	✗	✗	✗	✓

graphs. To this, ELTRA assigns an *asymmetry preserving proximity score* (AP-score) to any node-pair (u, v) in the graph by employing CRW in which the contribution of out-links in similarity computation is *upgraded*, while that of in-links is *downgraded*; these asymmetric AP-scores enable ELTRA to identify the relative roles of u and v based on the out-links existing in the paths from u to v by following [48]. Contrary to the existing methods, ELTRA does *not* preserve the neighborhood of nodes among their corresponding latent vectors in the embedding space nor conforms the similarity distribution of latent vectors to that of nodes in the graph. Instead, for every node u, ELTRA selects top-t closest nodes to u in a descending order, τ_u, based on their AP-scores w.r.t u. Then, it conforms the ranks of τ_u's corresponding target vectors w.r.t $\overrightarrow{u_s}$ in the embedding space (i.e., obtained by dot product) to the original ranks of τ_u obtained based on the AP-scores. ELTRA *not only* preserves the global similarity of nodes in the graph *but also* well preserves the asymmetry on NU-paths, thanks to CRW and AP-scores; also, it does *not* suffer from the impairing in/out-degree distributions since it does not consider AP-scores as a distribution, thanks to our listwise LTR lost function. Table 1 outlines the *ability* (i.e., ✓) and *inability* (i.e., ✗) of nine double-vector methods in alleviating the three aforementioned drawbacks; as we observe in the table, ELTRA addresses *all* the three drawbacks.

We carefully evaluate the effectiveness of both CRW and ELTRA by conducting extensive experiments with *seven real-world* datasets; the results demonstrate that (1) CRW *significantly* outperforms *both* Katz and RWR in computing nodes similarity in graphs, (2) our ELTRA *significantly* outperforms sixteen single-vector and double-vector embedding methods in graph reconstruction, link prediction, and node classification tasks.

We summarize our contributions in this paper as follows:

1. We point out three drawbacks of existing double-vector embedding methods as *inability to preserve asymmetry on NU-paths*, *inability to preserve global nodes similarity*, and *impairing in/out-degree distributions*.
2. We propose CRW, a *novel* similarity measure for graphs:
    - CRW employs an effective random walk strategy to exploit *both* U-paths and NU-paths in similarity computation, *without* ignoring link directions.
    - We provide a matrix form that *dramatically* accelerates the CRW computation while providing the *exact* scores.
3. We propose ELTRA, a novel double-vector embedding method to preserve asymmetric information in graphs:
    - ELTRA preserves *both* the global similarity of nodes in a graph and asymmetry on NU-paths.
    - ELTRA employs the AP-scores for ranking rather than considering them as a distribution, thereby being *robust* against impairing in/out-degree distributions drawback.
4. We conduct extensive experiments with *seven* real-world datasets and *sixteen* various embedding methods to carefully evaluate our proposed method.

2 RELATED WORK

As explained in Section 1, single-vector embedding methods such as DeepWalk [29], DWNS [3], FREDE [38], GELTOR [8], Gravity [33], NetMF [31], node2vec [5], and VERSE [37] have limitations in preserving the asymmetric information in graphs due to providing a single latent vector per node [15, 28, 34, 48]. To alleviate this problem, double-vector embedding methods provide two latent vectors, i.e., source and target, in the embedding space per node; we classify them into two following categories:

Random-Walk-based Methods: ATP [34] constructs an asymmetric proximity matrix by incorporating the graph hierarchy and its reachability; then, the non-negative matrix factorization (NMF) [2] is applied to the matrix. NERD [15] samples the nodes with high out/in-degree distributions as sources/targets and their neighborhoods are obtained via a random walk strategy where the directions of links are alternately changed; a skip-gram model [25] is applied to obtain the latent vectors. DNE [48] also employs a skip-gram model where the neighborhood of nodes obtained by a random walk strategy in which both in-links and out-links are traversed uniformly with equal probabilities. DIVINE [44] constructs a signed graph by adding virtual negative links to the original graph and then utilizes random walks to define the neighborhood of nodes; finally, it applies SIDE [16] to obtain the latent vectors.

Similarity-based Methods: HOPE [28] utilizes Katz [13] to compute similarity scores among nodes in the graph and then applies the singular value decomposition (SVD) [4] to the similarity matrix to obtain latent vectors. STRAP [43] applies SVD to a similarity matrix where the similarity score of a node-pair is considered as a summation of its RWR score in the original graph and that in the transpose graph. In NRP [42], instead of computing the whole similarity matrix and then factorizing it, the similarity matrix is factorized by applying randomized SVD [27] after each step of RWR computation while assigning higher similarity scores to those pairs of nodes having higher in/out-degrees. Lemane [47] applies a trainable RWR where the restart probability α is learnt for a target machine learning task based on sampled sub-graphs of the original graph. Then, SVD is applied to the similarity matrix.

3 PROPOSED METHOD

3.1 CRW

In this section, we propose our novel similarity measure, *CRW*, and present its component and matrix forms in details.

3.1.1 Preliminaries To compute the similarity score of a node-pair (u, v), both Katz [13] and RWR [36] employ a *single* random walk traversing the graph recursively via *only* out-links by starting from u with *no* capability to change the walking directions. Practically, they consider the contributions of only U-paths from u to v in similarity computation; u and v are regarded dissimilar if there are no such paths. As observed in Figure 1, both Katz and RWR assign non-zero similarity scores (i.e., 0.0100 and 0.1847, respectively) to (a, c) due to the U-path $a \rightarrow b \rightarrow c$. However, they both assign *zero* as similarity scores to (a, f) since there are no U-paths from a to f although they are connected via multiple NU-paths such as $a \rightarrow b \rightarrow c \leftarrow f$; the same circumstances are observed for (c, a) and (f, a). As a result, these two measures incur *limitations in accurate*

Table 2: A sample of similarity computation by CRW

Init.	$\forall i, j \in V,\ S_0(i,j)=1$ if $i=j$; otherwise $S_0(i,j)=0$
IT. 1	$O_h=\{e,f\}; I_h=\{g\} \Rightarrow S_1(h,f)=\frac{C}{2}\cdot\big(\frac{S_0(e,f)+S_0(f,f)}{2}+S_0(g,f)\big)=\frac{C}{4}$
IT. 1	$O_c=\{\}; I_c=\{b,f\} \Rightarrow S_1(c,f)=\frac{C}{2}\cdot\big(\frac{S_0(b,f)+S_0(f,f)}{2}\big)=\frac{C}{4}$
IT. 2	$O_e=\{\}; I_e=\{a,d,h\} \Rightarrow S_2(e,f)=\frac{C}{2}\cdot\big(\frac{S_1(a,f)+S_1(d,f)+S_1(h,f)}{3}\big)=\frac{C^2}{24}$
IT. 2	$O_b=\{c\}; I_b=\{a,d\} \Rightarrow S_2(b,f)=\frac{C}{2}\cdot\big(S_1(c,f)+\frac{S_1(a,f)+S_1(d,f)}{2}\big)=\frac{C^2}{8}$
IT. 3	$O_a=\{b,e\}; I_a=\{\} \Rightarrow S_3(a,f)=\frac{C}{2}\cdot\big(\frac{S_2(b,f)+S_2(e,f)}{2}\big)=\frac{C^3}{24}$

similarity computation in real-world graphs; they regard a *large* number of node-pairs (u,v) not similar at all.

3.1.2 *Component Form*

Suppose that $G(V,E)$ is a graph, V is a set of nodes, $E \subseteq V \times V$ is a set of links among nodes, O_u is a set of nodes directly connected to u via its *out-links*, and I_u is a set of nodes directly connected to u via its *in-links*. To compute accurate similarity scores in graphs, we propose CRW, a novel similarity measure based on *comprehensive random walks*; CRW employs a random walk strategy to traverse *not* only U-paths *but* also NU-paths in the graph effectively, *without* ignoring link directions as follows. To compute the CRW score of (u,v), $S(u,v)$, two random walks r_o and r_i starting from u traverse the graph *recursively* via out-links and in-links, respectively, where u and v are regarded similar if any of r_o and r_i visits v; by employing the two random walks, we can easily change the *walking directions* to traverse *any* of out-links and in-links involving in a NU-path. Formally, $S(u,v)$ is computed by the following recursive component form:

$$S(u,v)=\frac{C}{2}\cdot\Big(\frac{\sum_{i\in O_u}S(i,v)}{|O_u|}+\frac{\sum_{j\in I_u}S(j,v)}{|I_u|}\Big) \tag{1}$$

where $C \in (0,1)$ is a damping factor, $|O_u|$ denotes the size of O_u, and $S(u,v)=1$ if $u=v$, as the base case of the recursive computation.

The above recursion can be solved by an iteration to a fixed-point for $k=1,2,...$ over $S(u,v)$ as follows: the computation starts with $S_0(u,v)=1$ if $u=v$, otherwise $S_0(u,v)=0$; $S_k(u,v)=1$ for any k, if $u=v$; otherwise

$$S_k(u,v)=\frac{C}{2}\cdot\Big(\frac{\sum_{i\in O_u}S_{k-1}(i,v)}{|O_u|}+\frac{\sum_{j\in I_u}S_{k-1}(j,v)}{|I_u|}\Big) \tag{2}$$

Table 2 summarizes the CRW score computation for node-pair (a,f) in our sample graph (Init. and IT. k are abbreviations for initialization and iteration k, respectively); *for simplicity*, we show the computation process in three iterations only for some required node-pairs. As observed in the table, CRW considers *both* possible NU-paths with length three from a to f (i.e., $a \rightarrow b \rightarrow c \leftarrow f$ and $a \rightarrow e \leftarrow h \rightarrow f$) to compute $S(a,f)$. More specifically, on each iteration k, similarity scores are computed for those node-pairs (u,v) where u is connected to v via U-paths/NU-paths of length k by utilizing the similarity scores computed on iteration $k-1$.

Component Form Properties: (1) CRW exploits *all* paths in the graph, thereby covering those paths *neglected* by Katz and RWR in similarity computation and assigning accurate similarity scores to *all* node-pairs in the graph; contrary to Katz and RWR, CRW assigns non-zero similarity scores to (c,a), (a,f), (f,a), and (g,d) in Figure 1. (2) CRW scores are *asymmetric*, *bounded*, *monotonic*, *unique*, and *always existence*[3]. (3) The time and space complexities are $O(|V|^3)$ and $O(|V|^2)$, respectively, in the worst cases. (4) Although in this paper we focus on directed graphs, CRW is *applicable* to undirected graphs as well *without* requiring any changes.

3.1.3 *Matrix Form*

Now, we provide a matrix form for CRW to *accelerate* its computation with large graphs. To compute the CRW score of (u,v), instead of considering only the nodes in O_u and I_u, we can consider *all* the nodes in the graph by rewriting Equation (1) as follows:

$$S(u,v)=\frac{C}{2}\cdot\Big(\frac{\sum_{i\in V}[A]_{u,i}\cdot S(i,v)}{|O_u|}+\frac{\sum_{j\in V}[A]_{j,u}\cdot S(j,v)}{|I_u|}\Big) \tag{3}$$

where $A_{|V|\times|V|}$ is the adjacency matrix of the graph, $[A]_{u,i}=1$ if $i \in O_u$, and $[A]_{j,u}=1$ if $j \in I_u$.

Note that $\frac{[A]_{u,i}}{|O_u|}$ and $\frac{[A]_{j,u}}{|I_u|}$ in Equitation (3) are identical to $[Q]_{u,i}$ and $[W]_{j,u}$, respectively, where Q and W are respective *row* and *column* normalized versions of A; therefore, we provide the following matrix form for CRW:

$$S(u,v)=\frac{C}{2}\cdot(Q\cdot S+W^T\cdot S)\vee I \tag{4}$$

where $S \in \mathbb{R}^{|V|\times|V|}$ is a matrix whose entry $[S]_{u,v}$ contains the CRW score of node-pair (u,v), W^T is the transpose of W, $I_{|V|\times|V|}$ is an identity matrix, and $\vee$ is a disjunction operator selecting its maximum operand to guarantee that $S(u,v)=1$ if $u=v$.

Matrix Form Properties: (1) As observed above, we transform the component form into a matrix form via a straightforward mathematical process *without* applying any approximation; the matrix form provides *exact* CRW scores. (2) The space complexity is $O(|V|^2)$. (3) S is implemented as a dense matrix, while Q and W are both implemented as *compressed sparse row* (CSR) matrices[4][32]; the time complexity of each of the two matrix multiplications is $O(|V||E|)$ in the worst case where $|E|$ is the number of edges in the graph.

3.2 ELTRA

Now, we propose *ELTRA*, a novel and effective embedding method to preserve the asymmetric information in graphs.

3.2.1 *Asymmetry Preserving Proximity Score*

CRW may *not* accurately capture the asymmetric information since it considers contributions of in-links and out-links *equally significant* in computing the CRW score of a node-pair (u,v), thereby being *unable* to identify the relative roles of the two nodes based on their similarity score. For example, in Figure 1, the CRW score of (d,g) (i.e., 0.1065) is smaller than that of (g,d) (i.e., 0.3367), while there is a link from d to g but not inversely; CRW cannot capture the asymmetric information for d and g. The same circumstance is observed for nodes a and f connected via six NU-paths where in *all* paths from a to f (i.e., $a \rightarrow b \rightarrow c \leftarrow f$, $a \rightarrow e \leftarrow h \rightarrow f$, $a \rightarrow b \leftarrow d \rightarrow e \leftarrow h \rightarrow f$, $a \rightarrow e \leftarrow d \rightarrow g \rightarrow h \rightarrow f$, $a \rightarrow b \leftarrow d \rightarrow g \rightarrow h \rightarrow f$, and $a \rightarrow e \leftarrow d \rightarrow b \rightarrow c \leftarrow f$), the number of out-links is *larger* than that of in-links, implying that a is a source and f is a target following [48]; however, the CRW score of (a,f) is *less* than that of (f,a). To address this problem, we define the notion of an *asymmetry*

[3]The proof is available via ELTRA's GitHub page.

[4]Note that CRW assigns non-zero similarity scores to all possible node-pairs in the graph after some iterations; therefore, implementing S as a CSR matrix may result inefficiency in both computation time and space after a few iterations.

preserving proximity score (AP-score), which is computed by employing CRW; however, the contribution of out-links in similarity computation is *upgraded*, while that of in-links is *downgraded* by following the assumption that having a more number of out-links than in-links in NU-paths from u to v indicates that u is a source and v is a target. The AP-score of a node-pair (u, v) is computed by the following component form:

$$AP\text{-}score(u,v) = \begin{cases} 0 & v \in I_u \\ prx(u,v) & otherwise \end{cases}$$

$$prx(u,v) = \frac{C}{2} \cdot \Big(\frac{\omega \cdot \sum_{i \in O_u} prx(i,v)}{|O_u|} + \frac{(1-\omega) \cdot \sum_{j \in I_u} prx(j,v)}{|I_u|} \Big) \quad (5)$$

where $prx(u, v) = 1$ if $u = v$ and $\omega \in (0.5, 1)$ is an *importance factor* for out-links. Note that setting ω to any value in range $(0.5, 1)$ makes AP-scores to capture the asymmetric information; however, in order to prevent very small AP-scores, we set ω as 0.6.

We apply the similar process explained in Section 3.1.3, to transform the above component form into the following matrix form:

$$\begin{aligned} AP &= \bar{A} \odot X \\ X &= \frac{C}{2} \cdot \big(\omega \cdot Q \cdot X + (1-\omega) \cdot W^T \cdot X \big) \vee I \end{aligned} \quad (6)$$

where $AP \in \mathbb{R}^{|V| \times |V|}$ and $X \in \mathbb{R}^{|V| \times |V|}$ are two matrices in which their entries $[AP]_{u,v}$ and $[X]_{u,v}$ contain $AP\text{-}score(u, v)$ and $prx(u, v)$, respectively, $\bar{A}$ is the *invert* of A where 0 and 1 are replaced, and $\odot$ is the Hadamard product.

AP-scores Properties: (1) As observed in Figure 1, contrary to CRW scores, the AP-score of (a, f) is larger than that of (f, a), and the same circumstance is observed for (d, g) and (g, d); AP-scores capture the asymmetric information *not only* for directly connected neighbors and pairs of nodes connected by U-paths *but also* for those connected by NU-paths. (2) Contrary to DNE [48], NU-paths are *not* exploited by traversing out-links and in-links uniformly under an equal probability; instead, the are traversed under *their corresponding probabilities* as $1/(|O_u|)$ and $1/(|I_u|)$, respectively; this enables ELTRA to follow in/out-degree distributions in the graph. (3) AP-scores are asymmetric likewise the CRW scores.

3.2.2 Listwise Learning-to-Rank Listwise LTR has been widely used in various domains such as sentiment analysis, information retrieval and collaborative filtering where *sorting* objects based on certain factors is the main intention [1, 41]. The listwise LTR takes a number of *queries*, each of which is associated with a *ranked* list of objects in the descending order of their *relevance degrees* to the query as the training samples, and trains a model by minimizing a loss function defined on the *whole* predicted lists conditioned on the ground truth lists [1, 41, 49]. In the literature, various listwise LTRs such as ListMLE [41], ListNet [1], and RankCosine [30] have been proposed where ListMLE fits the noise in the training data *well* due to having a tighter generalization bond [17]. Inspired by the fact that, in practical applications, correct ranking at top-t positions is much *more* important than that in the whole list, top-t True Loss (a variant of ListMLE) [40] minimizes the errors occurring *only* at the top-t positions of a ranked list; this variant is *successfully* applied to single-vector graph embedding [8]. These findings motivated us to employ a listwise loss function based on ListMLE in our ELTRA.

Now, to make our paper self-contained, let us provide a formal definition of the listwise LTR based on top-t True Loss as follows. Suppose that $\mathbf{X}$ is an input space whose elements are sets of objects, $\mathbf{Y}$ is an output space whose elements are permutations of objects, and $\mathbf{\Gamma} : \mathbf{X} \longrightarrow \mathbf{Y}$ is a ranking model. Given i.i.d a training set $\{(\mathbf{x}^{(j)}, \mathbf{y}^{(j)})\}_{j=1}^{n}$ where $\mathbf{x}^{(j)} \in \mathbf{X}$, $\mathbf{x}^{(j)} = \{x_1^{(j)}, x_2^{(j)}, \ldots, x_{m_j}^{(j)}\}$, $\mathbf{y}^{(j)} \in \mathbf{Y}$, $y_i^{(j)}$ is the *index* of an object ranked in position i of $\mathbf{y}^{(j)}$, and a scoring function $\mathbf{f} : \mathbf{x}^{(j)} \longrightarrow \mathbb{R}^{m_j}$, $\mathbf{\Gamma}$ is obtained by minimizing the following loss function where the index of training samples, letter j, is ignored here for simplicity:

$$\frac{1}{n} \sum_{j=1}^{n} - \log p(\mathbf{y}|\mathbf{x}; \mathbf{f}) \quad (7)$$

$$p(\mathbf{y}|\mathbf{x}; \mathbf{f}) = \prod_{i=1}^{t} \frac{\exp\big(\mathbf{f}(x_{y_i})\big)}{\sum_{r=i}^{m} \exp\big(\mathbf{f}(x_{y_r})\big)} \quad (8)$$

where the loss function is defined as a negative log likelihood of observing ground truth $\mathbf{y}$ conditioned on the predicted scores $\mathbf{f}(\mathbf{x})$ by considering a *permutation probability* on $\mathbf{f}(\mathbf{x})$, which follows the Plackett-Luce model [23] focusing on the top-t objects in $\mathbf{y}$.

Weighted-ListMLE [11] is also a useful variant of ListMLE that *influences* the likelihood by a score reflecting the *quality* of the sorting order in a ground truth list. The training set is defined as $\{(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}), w^{(j)}\}_{j=1}^{n}$ where $w^{(j)}$, indicated by the user, is the corresponding score of a training sample $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)})$ and the loss function is defined as follows:

$$\frac{1}{n} \sum_{j=1}^{n} -w \cdot \log p(\mathbf{y}|\mathbf{x}; \mathbf{f}) \quad (9)$$

where it is obtained by applying a simple modification to the loss functions for ListMLE proposed in [40] and [41] by adding weights to the gradient computation [11].

3.2.3 Learning Model Given a graph $G = (V, E)$, ELTRA learns two functions $\Delta, \phi \longrightarrow \mathbb{R}^d$ that represent each node u in the graph as a latent source vector $\overrightarrow{u_s}$ and a latent target vector $\overrightarrow{u_t}$, respectively, in a d-dimensional space where $d \ll |V|$. We first define a function $\mathbf{F} : V \longrightarrow (N_0^t, \mathbb{R})$ that maps each node u onto a tuple (τ_u, w_u) in which τ_u contains top-t closest nodes $v_{u,i}$ $(1 \le i \le t)$ to u based on their AP-scores w.r.t u and w_u is the score defined in Equation (9); we consider w_u as the discounted cumulative gain (DCG)[5] [12] of the AP-scores of all nodes w.r.t u sorted in the descending order. Then, we train a model that conforms the ranks of $\overrightarrow{v_{u,i}}_t$ based on their dot products w.r.t to $\overrightarrow{u_s}$ to the ranks of τ_u. To this end, we maximize a Bayesian posterior probability by following [8], where the prior has a *normal distribution* and τ_u of any node u is regarded *independent* from those of others:

$$L = - \sum_{u \in V} \log\big(p(\tau_u|\Theta)\big) + \frac{\lambda}{2} \|\Theta\|^2 \quad (10)$$

where Θ is the model parameter, $p(\tau_u|\Theta)$ is our likelihood, and the second term is the L2 regularizing term.

Our scoring function $\mathbf{f}$ is a dot product of two latent vectors and τ_u contains *only* the top-t closest nodes to u; consequently, by

[5]DCG is a well-known metric in the information retrieval area to evaluate the quality of retrieved relevant objects to a query [28, 35].

replacing **f** and changing the parameters in Equation (8), we obtain the following likelihood:

$$p(\tau_u|\Theta) = \prod_{i=1}^{t} \frac{\exp(\overrightarrow{u_s} \cdot \overrightarrow{v_{u,i_t}})}{\Psi - \sum_{r=1}^{i-1} \exp(\overrightarrow{u_s} \cdot \overrightarrow{v_{u,r_t}})} \quad (11)$$

where $\Psi = \sum_{v \in V} \exp(\overrightarrow{u_s} \cdot \overrightarrow{v_t})$; following Equation (9), by incorporating w_u into our likelihood and applying it to Equation (10), we have the following *final* loss function:

$$L = \sum_{u \in V} w_u \cdot \sum_{i=1}^{t} \Big(\log \big(\Psi - \sum_{r=1}^{i-1} \exp(\overrightarrow{u_s} \cdot \overrightarrow{v_{u,r_t}})\big) - \overrightarrow{u_s} \cdot \overrightarrow{v_{u,i_t}} \Big) + \frac{\lambda}{2} \| \Theta \|^2 \quad (12)$$

3.2.4 Implementation and Algorithm ELTRA is implemented by a *simple* deep neural network consisting of a single hidden layer and an output layer where the input is a *one-hot* vector of size $|V|$, the hidden layer of size d is equipped with a linear activation function (i.e., acting as a *projection layer* to fetch $\overrightarrow{u_s}$), and the output layer contains the dot products between $\overrightarrow{u_s}$ and $\overrightarrow{v_t}$ (i.e., $\forall v \in V$) equipped with the ReLU activation function. The latent vectors $\overrightarrow{u_s}$ and $\overrightarrow{u_t}$ are the *row u* and *column u* of the hidden layer and output layer weigh matrices, respectively. It is worth to note that ELTRA is *applicable to undeirected graphs* as well.[6]

Algorithm 1: ELTRA Computation

```
   Input  : $G(V,E)$, $d$, $t$, $\omega$, $itr$
   Output : $[\mathbb{S}]_{|V| \times d}$, $[\mathbb{T}]_{|V| \times d}$
1  $[T]_{|V| \times t}, [\Omega]_{|V| \times 1} = \mathsf{F}(G, itr, t, \omega)$
2  initialize $W_0$, $W_1$
3  for $u \leftarrow 1$ to $|V|$ do
4      $I_u$ = one_hot($u$)
5      $[\Upsilon]_{1 \times |V|} = train(I_u)$
6      $loss$ = LossFunction($[T]_{u,*}$, $[\Omega]_u$, $\Upsilon$)
7      $W_0, W_1 \longleftarrow backpropagation\_update(loss)$
8  return $W_0$, $W_1^T$
9  Function LossFunction($\tau_u$, $w_u$, $\Upsilon$):
10     $\Upsilon = \Upsilon - \max(\Upsilon)$
11     $\Psi = \mathrm{sum}(\exp(\Upsilon))$
12     $[\Upsilon']_{1 \times t} = \mathrm{gather}(\Upsilon, \tau_v)$
13     $[\Upsilon'']_{1 \times t} = \mathrm{ECS}(\exp(\Upsilon'))$
14     $loss = \mathrm{sum}\big(w_u \cdot (\log(\Psi - \Upsilon'') - \Upsilon')\big)$
15     return $loss$
```

Given graph G, embedding dimensionality d, the number of top closest nodes t, the importance factor ω, and the number of iterations itr to compute AP-scores, Algorithm 1 performs the ELTRA computation as follows. Line 1 computes function **F**. In line 2, the weight matrix of the hidden layer, W_0, and that of the output layer, W_1, are randomly initialized. In lines 3 to 8, the training process proceeds (for simplicity, the batch size and the epoch are considered as one): in line 4, the input vector is constructed; in line 5, the model's output is fetched; in line 6, the loss value is computed, and W_0 and W_1 are updated in line 7. Lines 9 to 15 implement our loss function: line 10 is to guarantee the mathematical stability for the exp() function; line 11 applies exp() to the model's output and sums them up; in line 12, values of $\overrightarrow{u_s} \overrightarrow{v_{u,i_t}}$ are extracted for all nodes $u_{v,i} \in \tau_u$ from the model's output; line 13 calculates $\exp(\overrightarrow{u_s} \cdot \overrightarrow{v_{u,i_t}})$ for nodes in τ_u and applies an *exclusive cumulative sum* (**ESC**) function to them (e.g., **ESC**$([a,b,c]) = [0, a, a+b]$); line 14 computes the final loss value. The complexity of our loss function is $O(|V|)$.

[6]Please refer to ELTRA's GitHub page to check the details.

Table 3: Some statistics about our datasets

	$\lvert V \rvert$	$\lvert E \rvert$	#Labels
Cit_PH	34,546	421,578	-
CoCit	44,034	195,361	15
Cora	23,166	91,500	70
DBLP	21,177	124,065	11
Facebook	4,039	88,234	-
TREC	42,302	374,701	16
Twitter	47,260	168,964	-

4 EXPERIMENTAL EVALUATION

Research Questions. We carefully design our experimental evaluation to answer the following questions:

- **Q1:** Is the CRW matrix form more efficient than its component form?
- **Q2:** Is CRW really effective in *similarity computation* in graphs?
- **Q3:** Is *AP-score* beneficial to preserving asymmetric information?
- **Q4:** Is considering the contribution of NU-paths beneficial to directed graph embedding?
- **Q5:** Is ELTRA more effective in directed graph embedding than the existing state-of-the-art methods?

4.1 Experimental Settings

Datasets. We use *seven* real-world datasets summarized in Table 3:

- **Cit_PH** [19, 34] is an unlabeled citation graph in high energy physics (phenomenology) area from the arXiv open-access archive.
- **CoCit** [8, 37, 38] is a fully labeled citation graph in data mining, databases, and machine learning areas where node labels denote publication venues.
- **Cora** [8, 28, 50] is a fully labeled citation graph in computer science where node labels denote research topics.
- **DBLP** [6, 8] is a partially labeled citation graph in data mining and databases areas where node labels represent research topics.
- **Facebook** [5] is an unlabeled social graph where nodes represent users and edges do a "friendship" relation between them.
- **TREC** [6, 7] is a partially labeled hyperlink graph based on TREC 2003[7] where node labels indicate query topics for webpages.
- **Twitter** [21, 48] is an unlabeled social graph where nodes represent users and edges do a "following" relation between them.

Competitors. To answer **Q1**, we compare the computation times of the two forms of `CRW` with all datasets. To answer **Q2**, we compare the accuracy of `CRW` with those of `Katz` and `RWR` in computing similarity of nodes with our *labeled datasets*. To answer **Q5**, we compare the effectiveness of our ELTRA (`ELT`) with those of *eight* single-vector embedding methods (i.e., DeepWalk (`DWK`) [29], DWNS (`DWN`) [3], FREDE (`FRD`) [38], GELTOR (`GLT`) [8], Gravity (`GRV`) [33], node2vec (`N2V`) [5], NetMF (`NMF`) [31], and VERSE (`VRS`) [37]) and *eight* double-vector ones (i.e., `ATP` [34], `DNE` [48], DIVINE (`DVN`) [44], HOPE (`HPE`) [28], Lemane (`LMN`) [47], NERD (`NRD`) [15], `NRP` [42], and STRAP (`STP`) [43]). For all these methods, we use their source codes publicly available with the optimal parameter settings suggested by their original papers; we set the embedding dimensionality d to 128 by following [28, 42, 43, 47].

ELTRA Variants. To answer **Q3**, we compare the effectiveness of `ELT` with that of its variant, `ELT-C` where the original `CRW` scores are employed instead of the AP-scores in `ELT`. To answer **Q4**, we

[7]http://trec.nist.gov/data.html

Table 4: Execution time (minutes) of two CRW forms

	Cit_PH	CoCit	Cora	DBLP	Facebook	TREC	Twitter
Component	526.09	406.09	105.17	114.72	12.30	601.54	405.18
Matrix	1.80	1.87	0.52	0.46	0.03	2.11	1.88

compare the effectiveness of ELT with those of its two variants, ELT-K and ELT-R where the Katz and RWR scores are employed instead of the AP-scores in ELT, respectively.

Machine Learning Tasks. We employ following three machine learning tasks: (1) graph reconstruction where we aim to rebuild a graph's adjacency matrix [15, 28, 43], (2) link prediction where we try to predict missing links in a graph [28, 34, 47, 48], and (3) node classification where we try to predict the correct labels of nodes [15, 42, 43]. For each task, we run an embedding method five times and take their average value in effectiveness as the final one.

Environment. All codes[8] are implemented with Python 3.10 and TensorFlow 2.9 on an Intel machine equipped with i9-9900K CPU, 128 GB RAM, and a 64-bit Fedora Core 35 operating system.

4.2 Results and Analyses: CRW

In this section, we provide answers **A1** and **A2** to **Q1** and **Q2**, respectively.

4.2.1 Efficiency of Two CRW Forms As explained in Section 3.1, we proposed two forms for CRW (i.e., component and matrix forms), both of which provide *exact* similarity scores, while the latter one accelerates the CRW computation. Table 4 shows the computation times (in minutes) of the two forms with our datasets only in *five* iterations. The time complexity of the component form depends *not* only on the number of nodes but *also* on the average number of in/out-neighbors in the graph (refer to Equation (1)); although the TREC dataset is *smaller* than CoCit, it shows *longer* computation time since the average number of its in-neighbors (i.e., 10.96) and out-neighbors (i.e., 9.78) are larger than those of CoCit (i.e., 5.89 and 5.48, respectively). Likewise, the computation time of the matrix form also depends on the number of links in the graph; again, TREC shows *longer* computation time than CoCit since the number of links in TREC (i.e., 374,701) is larger than that of CoCit (i.e., 195,361).

A1. As observed in Table 4, *the matrix form of CRW is dramatically faster than its component form.*

4.2.2 Effectiveness of CRW We compare the accuracy of CRW with those of Katz and RWR in similarity computation on the four *labeled datasets* in terms of MAP (mean average precision) and F-score [22] as evaluation metrics. In each dataset, we use every node with a label l as a *query* and retrieve its top-t ($t = 10, 20, 30$) similar nodes by employing each of the three measures; if a retrieved node is labeled with l, it is regarded as relevant, otherwise irrelevant. The *average* value of each metric over all values of t and all labels l in the dataset is regarded as the *final* accuracy. The values of β and α in Katz and RWR are set as 0.10 and 0.15 by following [28] and [36], respectively; we set the value of C in CRW as 0.6 since our experimental results show that the accuracy with different values of C (i.e., 0.4, 0.6, 0.8) are *not* tangible with all datasets. We consider the *best* accuracy of CRW and RWR obtained in their executions for 10 iterations; RWR shows its best accuracy on iteration 10 with all datasets, while CRW shows that on iterations 6, 5, 6, and 7 with CoCit, Cora, DBLP, and TREC, respectively. Table 5 shows our experimental results where the best accuracy are highlighted per dataset.

A2. As observed in Table 5, *our CRW significantly outperforms Katz and RWR in terms of both metrics with all datasets.*

[8]All the codes and data are publicly available via ELTRA's GitHub page.

Table 5: Accuracy comparison of the three measures

		CoCit	Cora	DBLP	TREC
MAP	Katz	0.00050	0.01196	0.03148	0.00830
	RWR	0.00050	0.01217	0.03621	0.01096
	CRW	0.00092	0.02346	0.05145	0.03523
F-score	Katz	0.00193	0.02947	0.06728	0.01769
	RWR	0.00192	0.02999	0.07332	0.02482
	CRW	0.00338	0.05626	0.12304	0.04657

4.3 Results and Analyses: ELTRA

In this section, we first explain the parameter tuning of ELT, and then we provide answers **A3**, **A4**, and **A5** to **Q3**, **Q4**, and **Q5**, respectively, for *each* machine learning task.

4.3.1 Parameter Tuning To find the best value of t for Algorithm 1, we do *not* take a completely heuristic parameter tuning by following [8]; instead, we exploit the structural property of a target graph $G(V, E)$ by defining $t = \frac{|E|}{|V|} \times t'$, finding the best value of t' as follows. For each dataset, we set the value of t' in the range of $[10, 45]$ in step of 5, construct the latent vectors for each case, and evaluate its effectiveness in the three machine learning tasks (the parameter sensitivity of ELT is explained in Section 4.3.5). For the *sake of brevity*, we set the value of t', the number of iterations to compute AP-scores, the number of epochs, the number of batches, a learning rate, and a regularizing parameter to 20, 3, 150, 5% of the dataset size (in power of 2), 0.0025, and 0.001, with *all* datasets, respectively. For the four ELT variants, we set their parameters identical to those of ELT; also, the number of iterations for CRW and RWR in ELT-C and ELT-R are set as 3 and 10, respectively.

4.3.2 Graph Reconstruction First, we construct latent vectors by applying each embedding method to our datasets. Then, with all single-vector embedding methods except for GELTOR [8], for any node u, we compute dot products of u's latent vector and those of other nodes, sort the nodes in descending order of their corresponding dot product scores w.r.t. u, and pick up the top nodes as many as the u's *out-degree* in the graph; we perform the same process for GELTOR but we pick up the top nodes as many as u's *in-degree* by following [8], since GELTOR exploits in-links in the graph embedding. With all double-vector embedding methods, for any node u, we compute dot products of $\overrightarrow{u_s}$ and corresponding target vectors of other nodes, sort them, and pick up the top nodes as many as to the u's out-degree. In all cases, the accuracy is considered as the *average* ratio of *correctly* selected neighbors for all nodes. Table 6 shows the results where GRV *cannot* run on Twitter due to the issue of the dataset size; for each dataset, underlined and *italic underlined* scores indicate the best accuracy among single-vector and double-vector competitor methods, respectively, while highlighted scores do the best accuracy among *all* methods. Figure 3 depicts original in/out-degree distributions of the Cit_PH dataset and the ones rebuilt by different double-vector embedding methods (i.e., NERD is excluded for a space issue here since it shows the worst result); except our ELT, the degree distributions obtained by other methods are quite *different* from those of the original one. Now, we provide **A3**, **A4**, and **A5** as follows:

Table 6: Results of graph reconstruction task

	Cit_PH	CoCit	Cora	DBLP	Facebook	TREC	Twitter
DWK	0.0781	0.0845	0.1249	0.0977	0.0630	0.1928	0.2473
DWN	0.1000	0.1918	0.2152	0.1311	0.3188	0.1343	0.3109
FRD	0.0261	0.0277	0.0410	0.0446	0.1444	0.1622	0.0601
GLT	0.3322	0.2870	0.3114	0.2916	0.3204	0.2240	0.3944
GRV	0.0025	0.0002	0.0012	0.0024	0.0190	0.0021	NA
N2V	0.0783	0.0867	0.1226	0.0990	0.0565	0.1903	0.2453
NMF	0.0050	0.0019	0.0285	0.0118	0.0091	0.0658	0.0784
VRS	0.1913	0.2214	0.2476	0.1738	0.4402	0.4211	0.4256
ATP	0.0503	0.0473	0.0605	0.0189	0.1432	0.0294	0.0180
DNE	0.1027	0.0775	0.1560	0.1401	0.2123	0.3264	0.2047
DVN	0.0069	*0.3461*	*0.3399*	0.2442	0.2672	0.1811	*0.4062*
HPE	0.2145	0.1260	0.2041	0.2128	0.4537	0.3024	0.0620
LMN	0.2012	0.1324	0.1602	0.1684	0.4592	0.2916	0.0967
NRD	0.0137	0.0096	0.0111	0.0200	0.0617	0.0193	0.0018
NRP	*0.2355*	0.1419	0.2146	*0.2478*	0.4813	0.3890	0.0767
STP	0.2150	0.1368	0.1938	0.1915	*0.4828*	*0.3967*	0.0889
ELT-C	0.6520	0.6835	0.6892	0.7140	0.4528	0.4279	0.4936
ELT-K	0.0257	0.1870	0.2905	0.2940	0.0273	0.0127	0.0423
ELT-R	0.3953	0.3091	0.3906	0.4264	0.3942	0.4005	0.4009
ELT	0.7248	0.7209	0.7050	0.7907	0.4642	0.4331	0.4117

A3. As observed in the table, with *all* datasets except for Twitter, ELT shows *better* accuracy than ELT-C; it implies that *our AP-scores are beneficial to preserving asymmetric information.*

A4. ELT outperforms *both* ELT-K and ELT-R with *all* datasets; also, ELT-C outperforms the two other variants with *all* datasets. These results imply that *considering the contribution of NU-paths is beneficial to directed graph embedding.*

A5. ELT *significantly* outperforms *all* single-vector and double-vector embedding methods with all datasets except for Facebook and Twitte: in the former dataset, STP shows *slightly* better accuracy than ELT; in the latter one, ELT-C, a variant of ELT, shows the highest accuracy. These results show that *our ELT is more effective in directed graph embedding than the existing state-of-the-art methods.*

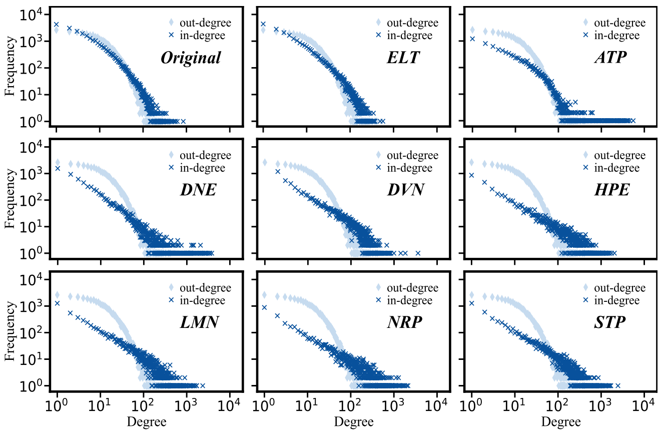


Figure 3: In/out-degree distributions with the Cit_PH dataset.

4.3.3 *Link Prediction* With each graph G, we randomly remove 10% of links as a test set, TS, (i.e., the size of TS is N) while the remained graph $\widehat{G}$ is connected except for Cit_PH and CoCit datasets, which are both originally not connected. Then, the N links in TS are considered as *positive* samples, and N *non-existing* links in the original graph G are randomly selected as *negative* ones; note that, in directed graphs, each link $\overrightarrow{uv}$ is ordered where we aim to predict whether there is a link only from u to v. Finally, we apply each of our embedding methods to graph $\widehat{G}$ to obtain latent vectors. With double-vector embedding methods, following [15, 42, 47], for each positive or negative sample $\overrightarrow{uv}$ in TS, we compute $\overrightarrow{u_s} \cdot \overrightarrow{v_t}$, sort them in descending order, and then evaluate the effectiveness by the Area Under Curve (AUC) [24]. With single-vector embedding methods, following [42, 43, 47], we first construct a train set, TR, with the same size as TS, by randomly selecting N existing links in $\widehat{G}$ as positive samples. Then, we randomly select N non-existing links in G as negative samples. For each link $\overrightarrow{uv}$ in both TR and TS, we concatenate the latent vector of u and that of v. Finally, $2d$-length vectors in TR are utilized to train a logistic regression classifier, and it is used to do link prediction on TS, which is evaluated by AUC. Table 7 shows the results where underlined, *italic underlined*, and highlighted scores have the same meanings as in Table 6; now, let us provide **A3**, **A4**, and **A5** as follows:

Table 7: Results of link prediction task

	Cit_PH	CoCit	Cora	DBLP	Facebook	TREC	Twitter
DWK	0.7619	0.7025	0.7413	0.7700	0.7178	0.7341	0.6354
DWN	0.7001	0.7227	0.7351	0.7824	0.7717	0.8211	0.5911
FRD	0.7209	0.7511	0.6443	0.6846	0.8156	0.7125	0.6842
GLT	0.7949	0.7725	0.8419	0.8440	0.8063	0.7983	0.6707
GRV	0.7082	0.5950	0.5922	0.6500	0.6665	0.5960	NA
N2V	0.7558	0.7033	0.7417	0.7705	0.7185	0.7338	0.6287
NMF	0.8189	0.7828	0.7746	0.7781	0.8093	0.7606	0.6951
VRS	0.7604	0.6294	0.6605	0.6897	0.7706	0.7100	0.7042
ATP	0.8113	0.8050	0.8192	0.8261	0.8724	0.6669	0.6943
DNE	0.9650	*0.9549*	*0.9681*	*0.9609*	0.9784	*0.9872*	*0.9466*
DVN	0.9032	0.7457	0.8494	0.7308	0.9073	0.8797	0.8469
HPE	0.9647	0.9019	0.9289	0.9380	0.9855	0.9758	0.8821
LMN	*0.9774*	0.9268	0.9480	0.9489	*0.9870*	0.9831	0.9122
NRD	0.8666	0.8513	0.8547	0.8807	0.8439	0.9021	0.7292
NRP	0.9345	0.8335	0.8582	0.8923	0.9494	0.9390	0.7769
STP	0.9754	0.9136	0.9330	0.9454	0.9849	0.9709	0.8994
ELT-C	0.9880	0.9608	0.9773	0.9702	0.9712	0.9901	0.9566
ELT-K	0.4066	0.9226	0.9413	0.9459	0.3910	0.3552	0.3448
ELT-R	0.9824	0.9473	0.9601	0.9578	0.9522	0.9832	0.9388
ELT	0.9892	0.9622	0.9802	0.9756	0.9763	0.9925	0.9597

A3. As observed in the table, with *all* datasets, ELT consistently outperforms ELT-C, which means our AP-scores are beneficial to preserving asymmetric information.

A4. ELT and its variant ELT-C outperform *both* ELT-K and ELT-R with *all* datasets; these results again show that considering the contribution of NU-paths is beneficial to directed graph embedding.

A5. ELT *outperforms all* single-vector and double-vector embedding methods with *all* datasets except for a single dataset, Facebook, where LMN shows the best accuracy. These results acknowledge that our ELT is more effective in directed graph embedding than the existing state-of-the-art methods.

4.3.4 *Node Classification* First, we obtain latent vectors by applying embedding methods to CoCit and Cora, our *fully* labeled datasets. With each of them, we randomly select $q\%$, $q = \{1, 3, 5, 7, 9\}$, of nodes as a test set and the ones remained as a training set. Following [42, 47, 48], with single-vector embedding methods, for any node u, we consider its latent vector as a node's feature, while with double-vector ones, we consider the concatenation of $\overrightarrow{u_s}$ and $\overrightarrow{u_t}$ as the node's feature. Then, we apply the logistic regression classifier to the nodes' features. We evaluate the accuracy in terms of Micro-F1 [9]; Table 8 presents the results where underlined,

Table 8: Results of node classification task

	CoCit					Cora				
	1%	3%	5%	7%	9%	1%	3%	5%	7%	9%
DWK	0.1950	0.1974	0.2134	0.2102	0.2112	0.2371	0.2273	0.2295	0.2318	0.2283
DWN	0.2336	0.2443	0.2507	0.2478	0.2571	0.3836	0.3914	0.4029	0.4149	0.4053
FRD	0.2109	0.2103	0.2203	0.2206	0.2243	0.2500	0.2273	0.2252	0.2324	0.2225
GLT	0.4444	0.4236	0.4337	0.4428	0.4420	0.6034	0.5942	0.5807	0.5808	0.5818
GRV	0.3424	0.3396	0.3411	0.3471	0.3494	0.4483	0.4288	0.4236	0.4353	0.4336
N2V	0.1859	0.1936	0.2048	0.2060	0.2094	0.2328	0.2129	0.2200	0.2238	0.2221
NMF	0.1383	0.1513	0.1594	0.1589	0.1635	0.0603	0.0748	0.0811	0.0832	0.0830
VRS	0.4331	0.4304	0.4319	0.4447	0.4432	0.5560	0.5712	0.5867	0.5999	0.6082
ATP	0.2268	0.2504	0.2634	0.2663	0.2674	0.2155	0.2273	0.2148	0.2164	0.2125
DNE	*0.4626*	*0.4614*	*0.4682*	*0.4715*	*0.4702*	*0.5216*	*0.5209*	*0.5194*	*0.5314*	*0.5372*
DVN	0.4399	0.4085	0.4133	0.4132	0.4132	0.4957	0.4921	0.5082	0.5185	0.5199
HPE	0.2857	0.2844	0.2975	0.3030	0.3007	0.4052	0.3942	0.3822	0.3915	0.3799
LMN	0.3810	0.3820	0.3969	0.3990	0.4026	0.5129	0.4978	0.4789	0.4827	0.4835
NRD	0.1293	0.1536	0.1717	0.1719	0.1784	0.0431	0.0576	0.0595	0.0647	0.0600
NRP	0.2132	0.2163	0.2216	0.2241	0.2306	0.2112	0.2029	0.2028	0.2121	0.2058
STP	0.3923	0.3873	0.3974	0.3990	0.4006	0.4914	0.4921	0.4909	0.4951	0.4940
ELT-C	0.4444	0.4372	0.4485	0.4528	0.4511	0.6078	0.5942	0.6005	0.6091	0.6149
ELT-K	0.4603	0.4440	0.4409	0.4376	0.4306	0.5216	0.5180	0.5220	0.5376	0.5487
ELT-R	0.4376	0.4266	0.4387	0.4259	0.4319	0.5000	0.5295	0.5496	0.5469	0.5506
ELT	0.5057	0.4667	0.4814	0.4736	0.4735	0.6164	0.6173	0.6152	0.6184	0.6129

italic underlined, and highlighted scores have the same meanings as in Tables 6 and 7; now, we provide **A3**, **A4**, and **A5** as follows:

A3. As observed in the table, with *both* datasets, ELT outperforms ELT-C for all values of q, except for one case when $q = 9\%$ with the Cora dataset; these results show that our AP-scores are beneficial to preserving asymmetric information.

A4. ELT outperforms ELT-K and ELT-R with both datasets for *all* values of q; also, ELT-C outperforms ELT-K and ELT-R in all cases except for CoCit where ELT-K outperforms it for $q = \{1\%, 2\%\}$; these results imply that considering the contribution of NU-paths is beneficial to directed graph embedding.

A5. ELT *outperforms all* single-vector and double-vector methods with *both* datasets for all values of q; it shows that our ELT is more effective in directed graph embedding than all the existing methods.

4.3.5 Parameter Sensitivity In our ELT, k (i.e., the number of iterations to compute AP-scores) and $t = \frac{|E|}{|V|} \times t^{'}$ (i.e., the number of top closest nodes to a any node) are two *important* parameters. Let us look over the sensitivity of ELT to different values of k and $t^{'}$; as sample cases, Figure 4 illustrates behaviors of ELT with the three machine learning tasks (i.e., for node classification, $q = 5\%$ as a sample) on different datasets where the values of k and $t^{'}$ are set to $\{2, 3, 4, 5\}$ and $\{10, 15, 20, ..., 45\}$, respectively.

Sensitivity to k. With all the three tasks, ELT shows *highest* accuracy when $k = \{3, 4\}$; we observe the same circumstances with other datasets as well. These results imply that for any node u, considering those nodes connected to u via short paths (i.e., $k < 2$) or too long paths (i.e., $k > 5$) are *not* beneficial to graph embedding.

Sensitivity to $t^{'}$. For graph reconstruction, ELT shows highest accuracy when $t^{'}$ is set to *small* values (i.e., $\{10, 15\}$); for example, with the Facebook dataset, the accuracy of ELT is 0.6430 when $k = 3$ and $t^{'} = 10$, which is *quite higher* than that of the best observed case in Table 6 (i.e., 0.4828 obtained by STP). However, for the two other tasks, ELT shows better effectiveness when $t^{'}$ is set to *middle* values (i.e., $\{20, 25\}$). We observe the same circumstances with other datasets as well, which means our *ELT is a flexible embedding method that can be trained for a specific machine learning task by only changing its parameters' values*; note, however, as stated in Section 4, for the sake of brevity, we set the values of k and $t^{'}$ to 3 and 20 for *all* tasks with *all* datasets, respectively.

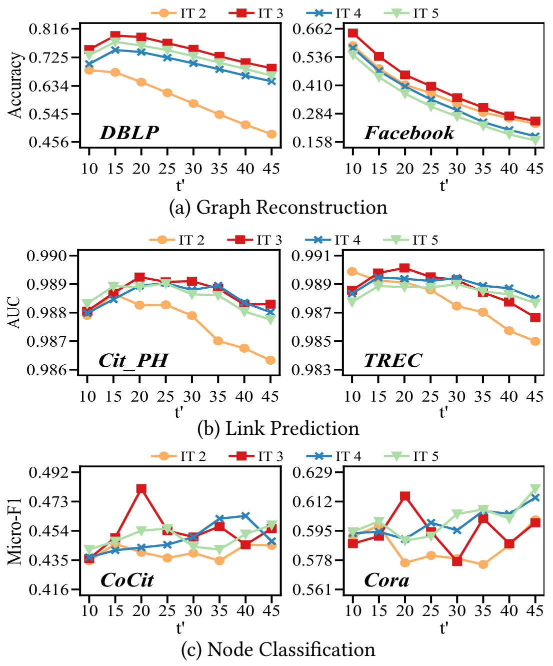


Figure 4: Parameter sensitivity for three sample tasks.

5 CONCLUSIONS

Observation. In this paper, we first pointed out the three drawbacks of existing double-vector embedding methods: inability to preserve asymmetry on NU-paths, inability to preserve global nodes similarity, and impairing in/out-degree distributions.

CRW. We then proposed CRW, a novel similarity measure for graphs that considers contributions of both U-paths and NU-paths in similarity computation, without ignoring link directions.

ELTRA. We proposed ELTRA that captures the asymmetric information by utilizing the AP-scores computed based on CRW where the respective contributions of out-links and in-links are upgraded and downgraded in similarity computation. Then, it preserves the asymmetric information by using a learning-to-rank loss function.

Experiments. We demonstrated that (1) CRW outperforms Katz and RWR in nodes similarity computation with all datasets, (2) ELTRA significantly outperforms all state-of-the-art single-vector and double-vector embedding methods in three machine learning tasks with seven real-world datasets.

Future Work. We plan to improve the ELTRA's efficiency by (1) applying acceleration techniques [20, 46] and also single-source computations [46] to AP-scores, and (2) utilizing negative sampling techniques [26] in our loss function.

6 ACKNOWLEDGMENTS

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00352 and No. RS-2022-00155586).

References

[1] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to Rank: from Pairwise Approach to Listwise Approach. In *Proceedings of the 24th International Conference on Machine Learning, ICML*. 129–136.

[2] Peizhe Cheng, Shuaiqiang Wang, Jun Ma, Jiankai Sun, and Hui Xiong. 2017. Learning to Recommend Accurate and Diverse Items. In *Proceedings of the 26th International Conference on World Wide Web, WWW*. 183–192.

[3] Quanyu Dai, Xiao Shen, Liang Zhang, Qiang Li, and Dan Wang. 2019. Adversarial Training Methods for Network Embedding. In *Proceedings of the 28th International Conference on World Wide Web, WWW*. 329–339.

[4] Gene H. Golub and Charles F. Van Loan. 2013. *Matrix Computations*. Johns Hopkins Univ. Press, 4th Edition.

[5] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*. 855–864.

[6] Masoud Reyhani Hamedani and Sang-Wook Kim. 2021. AdaSim: A Recursive Similarity Measure in Graphs. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management, CIKM*. 1528–1537.

[7] Masoud Reyhani Hamedani and Sang-Wook Kim. 2021. JacSim*: An Effective and Efficient Solution to the Pairwise Normalization Problem in SimRank. *IEEE Access* 9 (2021), 146038–146049.

[8] Masoud Reyhani Hamedani, Jin-Su Ryu, and Sang-Wook Kim. 2023. GELTOR: A Graph Embedding Method based on Listwise Learning to Rank. In *Proceedings of the ACM Web Conference 2023, WWW*. 6–16.

[9] Jiawei Han, Micheline Kamber, and Jian Pei. 2006. *Data Mining: Concepts and Techniques, Second Edition*. Morgan Kaufmann, San Francisco.

[10] Mingliang Hou, Jing Ren, Da Zhang, Xiangjie Kong, Dongyu Zhang, and Feng Xia. 2020. Network Embedding: Taxonomies, Frameworks and Applications. *Computer Science Review* 38 (2020), 100296.

[11] Swayambhoo Jain, Akshay Soni, Nikolay Laptev, and Yashar Mehdad. 2017. Rank-to-Engage: New Listwise Approaches to Maximize Engagement. arXiv:1702.07798 [stat.ML]

[12] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.

[13] Leo Katz. 1953. A New Status Index Derived from Sociometric Analysis. *Psychometrika* 18, 1 (1953), 39–43.

[14] Moein Khajehnejad. 2019. SimNet: Similarity-Based Network Embeddings with Mean Commute Time. *Plos ONE* 14, 8 (2019), 1102–1110.

[15] Megha Khosla, Jurek Leonhardt, Wolfgang Nejdl, and Avishek Anandm. 2019. Node Representation Learning for Directed Graphs. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Database, ECML-PKDD*. 395–411.

[16] Junghwan Kim, Haekyu Park, Ji-Eun Lee, , and U Kang. 2018. SIDE: Representation Learning in Signed Directed Networks. In *Proceedings of the International Conference on World Wide Web, WWW*. 509–518.

[17] Yanyan Lan, Tie-Yan Liu, Zhiming Ma, and Hang Li. 2009. Generalization Analysis of Listwise Learning-to-Rank Algorithms. In *Proceedings of the 26nd International Conference on Machine Learning, ICML*. 577–584.

[18] Yeon-Chang Lee, Nayoun Seo, and Sang-Wook Kim. 2020. Are Negative Links Really Beneficial to Network Embedding?: In-Depth Analysis and Interesting Results. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management, CIKM*. 2113–2116.

[19] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*. 177–187.

[20] Dmitry Lizorkin, Pavel Velikhov, Maxim Grinev, and Denis Turdakov. 2008. Accuracy Estimate and Optimization Techniques for SimRank Computation. In *Proceedings of the VLDB Endowment*. 422–433.

[21] Tiancheng Lou, Jie Tang, John Hopcroft, Zhanpeng Fang, and Xiaowen Ding. 2013. Learning to predict Reciprocity and Triadic Closure in Social Networks. *ACM Transactions on Knowledge Discovery from Data* 7, 2 (2013), 1–25.

[22] Christopher.D. Manning, Prabhakar Raghavan, and Hinrich Schutze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

[23] John I. Marden. 1995. *Analyzing and Modeling Rank Data*. Chapman and Hall/CRC.

[24] Donna Katzman McClish. 1989. . Analyzing a portion of the ROC curve. *Medical Decision Making* 9, 3 (1989), 190–195.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs.CL]

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems, NIPS*. 3111–3119.

[27] Cameron Musco and Christopher Musco. 2015. Randomized Block Krylov Methods for Stronger and Faster Approximate Singular Value Decomposition. In *Proceedings of the Advances in Neural Information Processing Systems, NeurIPS*. 1396–1404.

[28] Mingdong Ou, Peng Cui, Jian Pei, and Ziwei Zhang. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining, KDD*. 1105–1114.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*. 701–710.

[30] Tao Qin, Xu-Dong Zhang, Ming-Feng Tsai, De-Sheng Wang, Tie-Yan Liu, and Hang Li. 2008. Query-Level Loss Functions for Information Retrieval. *The Journal of Information Processing and Management* 44, 2 (2008), 838–855.

[31] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *Proceedings of the 11st ACM International Conference on Web Search and Data Mining, WSDM*. 459–467.

[32] Y. Saad. 2003. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[33] Guillaume Salha, Stratis Limnios, Romain Hennequin, Viet Anh Tran, and Michalis Vazirgiannis. 2019. Gravity-Inspired Graph Autoencoders for Directed Link Prediction. In *Proceedings of the 28th ACM Conference on Information and Knowledge Management, CIKM*. 589–598.

[34] Jiankai Sun, Bortik Bandyopadhyay, Armin Bashizade, Jiongqian Liang, P. Sadayappan, and Srinivasan Parthasarathy. 2019. ATP: Directed Graph Embedding with Asymmetric Transitivity Preservation. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI*. 265–272.

[35] Thibaut Thonet, Yagmur Gizem Cinar, Eric Gaussier, Minghan Li, and Jean-Michel Renders. 2022. Listwise Learning to Rank Based on Approximate Rank Indicators. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI*. 8494–8502.

[36] Hanghang Tong, Christos Faloutsos, and Jia yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *Proceedings of the 6th IEEE International Conference on Data Mining, ICDM*. 613–622.

[37] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *Proceedings of the International Conference on World Wide Web, WWW*. 539–548.

[38] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. 2021. FREDE: Anytime Graph Embeddings. *Proceedings of the VLDB Endowment* 14, 6 (2021), 1102–1110.

[39] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo1. 2018. GraphGAN: Graph Representation Learning with Generative Adversarial Nets. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, AAAI*. 2508–2515.

[40] Fen Xia, Tie-Yan Liu, and Hang Li. 2009. Statistical Consistency of Top-k Ranking. In *Proceedings of the 22th International Conference on Neural Information Processing Systems, NIPS*. 2098–2106.

[41] Fen Xia, Tie-Yan Liu, Jue Wang, and Wensheng Zhang Hang Li. 2008. Listwise Approach to Learning to Rank: Theory and Algorithm. In *Proceedings of the 25th International Conference on Machine Learning, ICML*. 1192–1199.

[42] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav Bhowmick. 2020. Homogeneous Network Embedding for Massive Graphs via Reweighted Personalized PageRank. *Proceedings of the VLDB Endowment* 13, 5 (2020), 670–683.

[43] Yuan Yin and Zhewei We. 2019. Scalable Graph Embeddings via Sparse Transpose Proximities. In *Proceedings of the 25nd ACM International Conference on Knowledge Discovery and Data Mining, KDD*. 1429–1437.

[44] Hyunsik Yoo, Yeon-Chang Lee, Kijung Shin, and Sang-Wook Kim. 2022. Directed Network Embedding with Virtual Negative Edges. In *Proceedings of the 15th ACM International Conference on Web Search and Data Mining, WSDM*. 1291–1299.

[45] Hyunsik Yoo, Yeon-Chang Lee, Kijung Shin, and Sang-Wook Kim. 2023. Disentangling Degree-related Biases and Interest for Out-of-Distribution Generalized Directed Network Embedding. In *Proceedings of the ACM Web Conference 2023, WWW*. 231–239.

[46] Weiren Yu, Xuemin Lin, Wenjie Zhang, Jian Pei, and Julie A. McCann. 2019. Simrank*: Effective and Scalable Pairwise Similarity Search Based on Graph Topology. *The VLDB Journal* 28, 3 (June 2019), 401–426.

[47] Xingyi Zhang, Kun Xie, Sibo Wang, and Zengfeng Huang. 2021. Learning Based Proximity Matrix Factorization for Node Embedding. In *Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*. 2243–2253.

[48] Sheng Zhou, Xin Wang, Martin Ester, Bolang Li, Chen Ye, Zhen Zhang, Can Wang, and Jiajun Bu. 2021. Direction-Aware User Recommendation Based on Asymmetric Network Embedding. *ACM Transactions on Information Systems* 40, 2 (2021), 1–23.

[49] Xiaofeng Zhu and Diego Klabjan. 2020. Listwise Learning to Rank by Exploring Unique Ratings. In *Proceedings of the 13th International Conference on Web Search and Data Mining, WSDM*. 798–806.

[50] Lovro Šubelj and Marko Bajec. 2013. Model of Complex Networks based on Citation Dynamics. In *Proceedings of the 22nd International Conference on World Wide Web, WWW Companion*. 527–530.