On Embedding Uncertain Graphs

Jiafeng Hu Reynold Cheng Zhipeng Huang Yixiang Fang Siqiang Luo
Department of Computer Science, The University of Hong Kong
{jhu, ckcheng, zphuang, yxfang, sqluo}@cs.hku.hk

ABSTRACT

Graph data are prevalent in communication networks, social media, and biological networks. These data, which are often noisy or inexact, can be represented by uncertain graphs, whose edges are associated with probabilities to indicate the chances that they exist. Recently, researchers have studied various algorithms (e.g., clustering, classification, and k-NN) for uncertain graphs. These solutions face two problems: (1) *high dimensionality*: uncertain graphs are often highly complex, which can affect the mining quality; and (2) *low reusability*, where an existing mining algorithm has to be redesigned to deal with uncertain graphs on different tasks. To tackle these problems, we propose a solution called URGE, or UnceRtain Graph Embedding. Given an uncertain graph $\mathcal{G}$, URGE generates $\mathcal{G}$'s *embedding*, or a set of low-dimensional vectors, which carry the proximity information of nodes in $\mathcal{G}$. This embedding enables the dimensionality of $\mathcal{G}$ to be reduced, without destroying node proximity information. Due to its simplicity, existing mining solutions can be used on the embedding. We investigate two low- and high-order node proximity measures in the embedding generation process, and develop novel algorithms to enable fast evaluation.

To our best knowledge, there is no prior study on the use of embedding for uncertain graphs. We have further performed extensive experiments for clustering, classification, and k-NN on several uncertain graph datasets. Our results show that URGE attains better effectiveness than current uncertain data mining algorithms, as well as state-of-the-art embedding solutions. The embedding and mining performance is also highly efficient in our experiments.

1 INTRODUCTION

Graph data are often found in important and emerging domains, including social media, communication networks, and biological databases. This rich information source, which describes complex relationships among objects, has attracted a lot of interest from research and industry communities. Mining and analysis solutions, such as clustering [32], classification [3], and embedding [13, 30, 36], allow important knowledge and insight to be discovered from graphs.

CIKM'17, November 6–10, 2017, Singapore.
 ACM ISBN 978-1-4503-4918-5/17/11...$15.00
DOI: http://dx.doi.org/10.1145/3132847.3132885

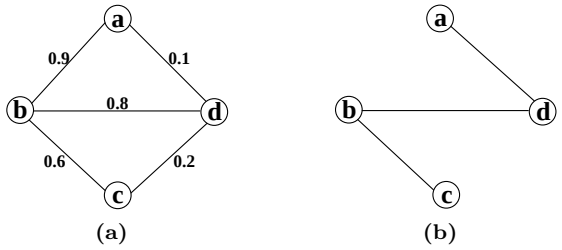


Figure 1: (a) An uncertain graph $\mathcal{G}$. (b) A possible world $G = \{(a,d),(b,d),(b,c)\}$ of $\mathcal{G}$ with probability $Pr[G] = \mathbf{P}_{ad}\mathbf{P}_{bd}\mathbf{P}_{bc}(1-\mathbf{P}_{ab})(1-\mathbf{P}_{cd}) = 0.1 \times 0.8 \times 0.6 \times 0.1 \times 0.8 = 0.00384$.

In many situations, graph data are noisy, inexact, and inaccurate. This can be due to various reasons, such as the error occurred during the data collection process, imperfection of machine-learning techniques for generating a graph, or deliberate hiding of information for privacy protection [21]. For example, in the Protein-Protein Interaction (PPI) networks, each protein is represented as a node, and an edge between two nodes depicts an interaction between them. However, whether an interaction exists is not certain, because it is derived through error-prone experiments or statistical models [22]. As another example, in influence maximization, an influence graph is used to indicate whether a social network user (node) can affect another one [20]. An edge from node a to b indicates that a can exert influence on b (e.g., to buy an electronic product). Since it is not entirely clear whether a can indeed influence b, the existence of edge (a,b) is probabilistic. Other examples of graph uncertainty include the modeling of reliability between nodes in mobile ad-hoc networks [4], as well as the use of graph obfuscation for privacy protection [5].

A common model that captures the inexact information of graph data is the *uncertain graph model*. As illustrated in Fig. 1a, each edge is associated with a probability value, which specifies the chance that the edge exists. Let us suppose that Fig. 1a is a PPI network. Then, the probability on the edge (a,b) indicates that proteins a and b interact with a probability of 0.9. In recent years, researchers have studied various problems on the uncertain graph model, including clustering [21, 24], classification [8], k-NN queries [31], structural-similarity computation [10, 45], dense subgraph retrieval [6, 15, 27], and frequent subgraph mining [44, 46]. In general, this kind of solutions takes an existing non-probabilistic (or deterministic) graph algorithm and modify it to address the probability information of graph edges. Although they have been shown to outperform a graph solution that does not handle uncertain graphs, they have two shortcomings.

• *High dimensionality.* An uncertain graph, containing numerous nodes, edges, and probabilities, is highly complex. If the graph is also sparse, it could carry a large amount of noise, and the quality of mining the graph can be affected.
• *Low reusability.* As discussed before, to develop a mining algorithm (say, uncertain graph clustering [21]) on uncertain graphs, a typical way is to modify its deterministic version. This methodology is highly non-trivial; for instance, new index structures and algorithms have to be developed to compute probabilistic information efficiently. Moreover, it is not easy to develop an uncertain graph mining algorithm by extending another one. For example, an index developed for uncertain graph clustering may not be used for uncertain graph classification. Hence, given a new mining algorithm $\mathcal{A}$ for deterministic graphs, extending $\mathcal{A}$ to support uncertain graphs can incur a huge amount of effort.

The URGE solution. To tackle the aforementioned problems, we propose an uncertain data mining paradigm, called UnceRtain Graph Embedding (or URGE in short). Given an uncertain graph $\mathcal{G}$, URGE generates its respective *embedding*, essentially a set of low-dimensional vectors that carry the proximity information of nodes in $\mathcal{G}$. A salient feature of URGE is that node proximity is preserved under the *Possible World Semantics* (or PWS), which is a correct interpretation of uncertain graphs [8, 10, 21, 24, 31, 45]. Fig. 1b illustrates a possible world G of $\mathcal{G}$, which only contains edges (a, d), (b, d), and (b, c), and exists with a probability of 0.00384. The proximity between two nodes (say, a and b) can then be computed based on the sum of the probabilities of the possible worlds in which a and b are close to each other. Through the use of proximity measures designed for uncertain graphs, we are able to preserve the similarity between a and b in the embedding space.

Because URGE represents $\mathcal{G}$ as a set of low-dimensional vectors, the high-dimensionality issue of $\mathcal{G}$ is alleviated. As shown in our experiments, the mining effectiveness of URGE is better than other uncertain data mining algorithms. Moreover, URGE demonstrates higher reusability than its uncertain data mining counterparts, because its vector-based representation is readily used by existing deterministic graph mining solutions (e.g., [3, 32]).

The topic of embedding graphs has been recently studied [13, 30, 36]. However, these solutions are not designed for uncertain graph models. A core part of URGE is to compute the *proximity matrix* of $\mathcal{G}$, which is an $n \times n$ matrix that captures the similarity between each pair of n nodes in $\mathcal{G}$. Here we consider two major classes of proximity measures, namely *Expected Jaccard Similarity* (or EJS) [16] and *Probabilistic Random Walk with Restart* (or PRWR) [31]. The EJS is a second-order measure, which is suitable for dense graphs whose nodes are close to each other, while the PRWR is a high-order metric, which captures the global structure of a graph. Both measures follow the PWS notion. Due to the huge overhead of computing the proximity matrix, we have designed efficient and approximate algorithms with accuracy guarantees. Particularly, our solution for computing the proximity matrix for EJS is $\mathcal{O}(d)$ times faster than a current solution in [45], where d is the maximum node degree of $\mathcal{G}$. Our algorithm for the PRWR proximity matrix also executes $\mathcal{O}(d/h)$ times faster than an existing algorithm [10], where h is the hash set size, with $h \ll d$. We also develop a matrix factorization method based on negative sampling, in order to derive the embedding from the proximity matrix.

We have conducted experiments on several uncertain graph datasets for three common applications (namely clustering, classification, and k-NN search). We compare our solutions with state-of-the-art graph embedding solutions (namely DeepWalk [30], LINE [36], and node2vec [13]) and uncertain data mining algorithms. Our results show that URGE consistently outperforms these solutions. Our algorithms developed to compute proximity matrices for EJS and PRWR are also more efficient than the existing ones [10, 45].

The rest of the paper is structured as follows. We review the related work in Section 2. Section 3 formulates the problem of uncertain graph embedding and presents the URGE model. In Section 4, we study proximity matrices. We optimize the training process in Section 5. In Section 6, we report our experimental results. Section 7 concludes.

2 RELATED WORK

We now discuss the literature related to (1) querying and mining of uncertain graphs (Section 2.1) and (2) graph embedding (Section 2.2).

2.1 Uncertain Graphs

Querying and mining over uncertain graphs has been widely studied in recent years, including query processing, e.g., k-NN queries [31], structural-similarity computing [10, 45] and dense subgraph retrieval [6, 15, 27], as well as graph mining tasks, e.g., frequent subgraph mining [44, 46], clustering [21, 24] and classification [8]. [19] is a survey on mining uncertain graphs.

Potamias et al. [31] study the k-NN problem over uncertain graphs. They propose several distance functions between nodes that extend shortest path distances from deterministic graphs and devise sampling based algorithms to answer k-NN queries efficiently.

Zou and Li [45] study several structural-context similarities for uncertain graphs, including the Jaccard similarity [16], the Dice similarity [9] and the cosine similarity. In this paper, we also investigate the Jaccard similarity for uncertain graphs, and propose an $\mathcal{O}(nd^2)$ algorithm which is $\mathcal{O}(d)$ times faster than directly applying their algorithm, where n is the number of nodes in the uncertain graph $\mathcal{G}$ and d is the maximum node degree of $\mathcal{G}$.

Kollios et al. [21] focus on the node clustering task on uncertain graphs, and propose a new definition of clustering based on expected edit distance, as well as algorithms for clustering. In addition, Dallachiesa et al. [8] focus on the node classification task on uncertain graphs and propose two algorithms based on iterative probabilistic labeling which incorporate the uncertainty of edges in their operation.

Different from existing works, we adopt the embedding paradigm, i.e., learning a low-dimensional vector for each node, for uncertain graphs, which can serve a wide range of tasks, e.g., clustering, classification, k-NN search, link prediction, etc. Experimental results in Section 6 demonstrate that our solution, URGE, performs better than those algorithms customized for clustering [21], classification [8] and k-NN [31].

2.2 Graph Embedding

Another kind of work related is graph embedding (a.k.a., dimension reduction), which aims at learning latent representations associated with nodes. Graph embedding can be used for many graph analysis tasks, e.g., node classification, clustering, link prediction, etc.

Traditional techniques [2, 33, 38] usually rely on computing the leading eigenvectors of the affinity matrices constructed from the feature vectors of nodes. As pointed out in [13], these methods suffer from both computational and statistical performance drawbacks.

Recently, inspired by Skip-Gram [26], a widely adopted language model which optimizes the co-occurrence likelihood among words that appear within a window of a sentence, many graph embedding models have been proposed to learn effective representations, e.g, DeepWalk [30], LINE [36], node2vec [13], etc. Particularly, DeepWalk considers a graph as a document, and truncated random walks on the graphs as sentences in the document. Hence, the Skip-Gram model can be adopted to learning representations on graphs. node2vec extends DeepWalk by designing a flexible biased random walk procedure based on second-order random walks. Different from DeepWalk and node2vec which utilize random walk to capture the proximities between nodes, LINE uses an explicit objective function to preserve both first-order and second-order proximities between nodes in the representation space. Specifically, LINE learns two vectors for the first-order and second-order proximities separately and then concatenates them to form the final representation.

However, all those existing embedding models are not designed for uncertain graphs. In this paper, we propose an embedding method, URGE, customized to uncertain graphs, by treating uncertainty as a first-class citizen.

Other related works like HOPE [28] which focuses on preserving asymmetric transitivity in directed graph embedding and SDNE [41] which is a semi-supervised deep model that captures the highly non-linear graph structure, cannot support uncertain graphs.

3 UNCERTAIN GRAPH EMBEDDING

An uncertain graph is denoted by $\mathcal{G} = (V, E, \mathbf{P})$, with node set V, edge set E, and probability set $\mathbf{P}$. $\mathbf{P}_e$ (or $\mathbf{P}_{uv}$ where $e = (u, v)$) is the probability associated with the edge $e \in E$, which is the probability that edge e exists in the graph.

Under the *Possible World Semantics* (or PWS) [18, 21, 31, 43], an uncertain graph $\mathcal{G}$ represents a probability distribution over all its possible worlds. In particular, a possible world

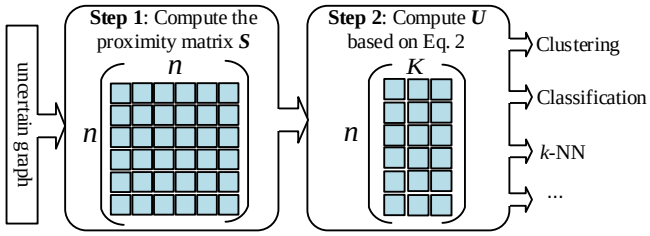


Figure 2: Flowchart of URGE

of $\mathcal{G}$ is a deterministic graph $G = (V, E_G)$, where $E_G \subseteq E$. Following previous work for uncertain graphs [31, 44], we also assume that the existence probabilities of edges are mutually independent. Hence, the probability of a possible world G is defined as:

$$Pr[G] = \prod_{e \in E_G} \mathbf{P}_e \prod_{e \in E \setminus E_G} (1 - \mathbf{P}_e) \tag{1}$$

As shown in Fig. 1 (in Section 1), Fig. 1a is an uncertain graph $\mathcal{G}$ and Fig. 1b is one of its possible worlds G.

Definition 3.1 (Uncertain Graph Embedding). Given an undirected uncertain graph [1] $\mathcal{G}$, the problem of uncertain graph embedding aims to represent each node $v \in V$ by a vector in a low-dimensional space $\Re^K$, i.e., learning an embedding matrix $\mathbf{U} \in \Re^{n \times K}$, where $n = |V|$ is the number of nodes in $\mathcal{G}$ and $K \ll n$ is the number of dimensions in the embedded space and the i-th row, u_i, is the embedding vector of the i-th node. In the embedded space $\Re^K$, the proximities among nodes in the original uncertain graph $\mathcal{G}$ are preserved.

3.1 The URGE Model

In this subsection, we present our solution, URGE, for uncertain graph embedding, which can preserve the proximities among nodes well in the embedded space.

Given an uncertain graph $\mathcal{G}$, let $\mathbf{S} \in \Re^{n \times n}$ be a proximity matrix of $\mathcal{G}$, where $\mathbf{S}_{uv}$ is the proximity between node u and v. Yang et al. [42] have proved that DeepWalk, a leading embedding method, is inherently a matrix factorization process whose power on embedding has been shown in [1, 28, 39, 42]. In this paper, we adopt the matrix factorization model to preserve the proximity. That is, we wish to find matrices $\mathbf{U} \in \Re^{n \times K}$ and $\tilde{\mathbf{U}} \in \Re^{n \times K}$ such that:

$$\min_{\mathbf{U}, \tilde{\mathbf{U}}} \|\mathbf{S} - \mathbf{U}\tilde{\mathbf{U}}^T\|^2 + \frac{\lambda}{2}(\|\mathbf{U}\|^2 + \|\tilde{\mathbf{U}}\|^2), \tag{2}$$

where $\|\cdot\|$ is the L_2 norm, and the factor λ controls the weight of the regularization term. In this paper, as used in [39], we take the learnt representations $\mathbf{U}$ as features.

Fig. 2 is the flowchart of the URGE model. URGE consists of two steps, described as follows:

(1) compute the proximity matrix $\mathbf{S}$ for the given uncertain graph $\mathcal{G}$.
(2) apply matrix factorization to get the embedded space $\mathbf{U}$.

[1] In this paper, we focus on undirected uncertain graphs. Our solution can be extended to directed uncertain graphs easily.

Next, we investigate two common proximity measures for uncertain graphs, as well as their efficient computation, in Section 4. We then describe how to optimize the loss function (Eq. 2) efficiently in Section 5.

4 PROXIMITY MATRIX FOR UNCERTAIN GRAPHS

Both low- and high-order node proximity measures have been studied in graph embedding. Specifically, LINE [36] focuses on preserving the low-order proximity, e.g., first-order relationship (edges in graph) and second-order relationship (2-hop relationship), since it can reflect the local structure among nodes well. Meanwhile, other works like DeepWalk [30] and node2vec [13] focus on preserving the high-order proximity to capture the global structure among nodes. In this section, we investigate both low- and high-order proximity measures. In particular, we study a second-order proximity (the expected Jaccard similarity) in Section 4.1 and a high-order proximity (probabilistic random walk with restart) in Section 4.2. Other proximity measures can also be adopted in the URGE model.

4.1 Expected Jaccard Similarity

The Jaccard similarity [16] has been widely used to measure the similarity between nodes of graphs. Particularly, given a deterministic undirected graph G, the Jaccard similarity $(\mathbf{S}^{\mathrm{J}}_{uv})_G$ between two nodes u and v of the graph G, is defined as follows[2]:

$$(\mathbf{S}^{J}_{uv})_G = \frac{|N_G(u) \cap N_G(v)|}{|N_G(u) \cup N_G(v)|}, \tag{3}$$

where $N_G(x)$ denotes the neighbor set of node x on G.

The Jaccard similarity defined on deterministic graphs does not make sense on uncertain graphs. Given an uncertain graph $\mathcal{G}$, the Jaccard similarity between two nodes u and v of $\mathcal{G}$ may be different on different possible worlds of $\mathcal{G}$ due to the difference of neighbor sets.

Definition 4.1 (Expected Jaccard Similarity [45]). Given an undirected uncertain graph $\mathcal{G}$, the expected Jaccard similarity (EJS for short) between two nodes u and v of $\mathcal{G}$ is defined as follows:

$$\begin{aligned}\mathbf{S}^{\mathtt{EJS}}_{uv} &= \sum_{G \in \Omega(\mathcal{G})} (\mathbf{S}^{\mathrm{J}}_{uv})_G Pr[G] \\ &= \sum_{G \in \Omega(\mathcal{G})} \frac{|N_G(u) \cap N_G(v)|}{|N_G(u) \cup N_G(v)|} Pr[G]\end{aligned} \tag{4}$$

where $\Omega(\mathcal{G})$ is the set of all possible worlds of $\mathcal{G}$.

In other words, $\mathbf{S}^{\mathtt{EJS}}_{uv} = E[(\mathbf{S}^{\mathrm{J}}_{uv})_G] = E\left[\frac{|N_G(u) \cap N_G(v)|}{|N_G(u) \cup N_G(v)|}\right]$, where the expectation is computed over all possible worlds G that are chosen at random from $\Omega(\mathcal{G})$ with probability $Pr[G]$.

Zou and Li [45] proposed an approximate algorithm to compute $\mathbf{S}^{\mathtt{EJS}}_{uv}$, the EJS between node u and v, in $\mathcal{O}(d)$ time, where d is the maximum node degree of $\mathcal{G}$, based on following two lemmas.

[2]The similarity is required to be 0 if the denominator is 0.

Lemma 4.2. *[35] Let X and Y be two nonnegative random variables such that $P\{X \le Y\} = 1$ and $E[Y] > 0$. If $Y = 0$, we require $X/Y = 0$. Then based on the second order Taylor expansion we have:*

$$E\left[\frac{X}{Y}\right] \approx \frac{E[X]}{E[Y]} - \frac{Cov(X,Y)}{E[Y]^2} + \frac{E[X]\,Var(Y)}{E[Y]^3}. \tag{5}$$

Lemma 4.3. *[45] Given an uncertain graph $\mathcal{G}$ and two nodes u and v of $\mathcal{G}$, let $X_{uv} = |N_G(u) \cap N_G(v)|$ and $Y_{uv} = |N_G(u) \cup N_G(v)|$, where G is a possible world of $\mathcal{G}$ chosen at random from $\Omega(\mathcal{G})$ with probability $Pr[G]$. Then,*

$$E[X_{uv}] = \sum_{w \in (N_{\mathcal{G}}(u) \cap N_{\mathcal{G}}(v))} \mathcal{P}_{u \wedge v}(w), \tag{6a}$$

$$E[Y_{uv}] = \sum_{w \in (N_{\mathcal{G}}(u) \cup N_{\mathcal{G}}(v))} (1 - \mathcal{P}_{\bar{u} \wedge \bar{v}}(w)), \tag{6b}$$

$$Var(Y_{uv}) = \sum_{w \in (N_{\mathcal{G}}(u) \cup N_{\mathcal{G}}(v))} \mathcal{P}_{\bar{u} \wedge \bar{v}}(w)(1 - \mathcal{P}_{\bar{u} \wedge \bar{v}}(w)), \tag{6c}$$

$$Cov(X_{uv}, Y_{uv}) = \sum_{w \in (N_{\mathcal{G}}(u) \cap N_{\mathcal{G}}(v))} \mathcal{P}_{u \wedge v}(w) \mathcal{P}_{\bar{u} \wedge \bar{v}}(w), \tag{6d}$$

$$\mathbf{S}^{\mathtt{EJS}}_{uv} = E\left[\frac{X_{uv}}{Y_{uv}}\right] \approx \frac{E[X_{uv}]}{E[Y_{uv}]} - \frac{Cov(X_{uv}, Y_{uv})}{E[Y_{uv}]^2} + \frac{E[X_{uv}]\,Var(Y_{uv})}{E[Y_{uv}]^3}, \tag{6e}$$

where $N_{\mathcal{G}}(x)$ denotes the neighbor set of node x on $\mathcal{G}$, $\mathcal{P}_{u \wedge v}(w) = \mathbf{P}_{uw}\mathbf{P}_{vw}$ is the probability that both edges (u,w) and (v,w) exist, and $\mathcal{P}_{\bar{u} \wedge \bar{v}}(w) = (1 - \mathbf{P}_{uw})(1 - \mathbf{P}_{vw})$ is the probability that both edges (u,w) and (v,w) do not exist.

Given any pair (u,v) of nodes, the EJS between u and v, $\mathbf{S}^{\mathtt{EJS}}_{uv}$, can be computed by visiting all neighbors of u and v once. Since [45] only focuses on computing the EJS for a pair of nodes, it is still not clear on how to compute the whole EJS matrix, $\mathbf{S}^{\mathtt{EJS}}$, efficiently. A naive way is to enumerate all 2-hop neighbors, v, for each node u on $\mathcal{G}$ and compute $S^{\mathtt{EJS}}_{uv}$. The time complexity is $\mathcal{O}(nd^3)$, where n is the number of nodes in $\mathcal{G}$, since there are up to $\mathcal{O}(nd^2)$ pairs of nodes on $\mathcal{G}$.

In the following, we propose an algorithm called `FastEJS`, which can compute the EJS for all pair of nodes simultaneously (i.e., the whole EJS matrix $\mathbf{S}^{\mathtt{EJS}}$) in $\mathcal{O}(nd^2)$ time, which is $\mathcal{O}(d)$ times faster than directly applying the solution proposed in [45].

The key idea of `FastEJS` is to transform union operators in Eq. 6b and Eq. 6c to intersection operators, which can be done by replacing $\mathcal{P}_{\bar{u} \wedge \bar{v}}(w)$ with $(1 - \mathbf{P}_{uw})(1 - \mathbf{P}_{vw})$ in Eq. 6b and Eq. 6c. Then, we have [3]:

$$\begin{aligned}E[Y_{uv}] &= \sum_{w \in (N_{\mathcal{G}}(u) \cup N_{\mathcal{G}}(v))} (\mathbf{P}_{uw} + \mathbf{P}_{vw} - \mathcal{P}_{u \wedge v}(w)) \\ &= \sum_{w \in N_{\mathcal{G}}(u)} \mathbf{P}_{uw} + \sum_{w \in N_{\mathcal{G}}(v)} \mathbf{P}_{vw} - \sum_{w \in (N_{\mathcal{G}}(u) \cap N_{\mathcal{G}}(v))} \mathcal{P}_{u \wedge v}(w) \\ &= \sum_{w \in N_{\mathcal{G}}(u)} \mathbf{P}_{uw} + \sum_{w \in N_{\mathcal{G}}(v)} \mathbf{P}_{vw} - E[X_{uv}]\end{aligned} \tag{7}$$

[3]For any node $u \in V$ on $\mathcal{G}$, if node $w \notin N_{\mathcal{G}}(u)$, $\mathbf{P}_{uw} = 0$.

and,

$$Var(Y_{uv}) = \underbrace{\sum_{w\in N_{\mathcal{G}}(u)} \mathbf{P}_{uw} + \sum_{w\in N_{\mathcal{G}}(v)} \mathbf{P}_{vw} - \sum_{w\in N_{\mathcal{G}}(u)} \mathbf{P}^2_{uw} - \sum_{w\in N_{\mathcal{G}}(v)} \mathbf{P}^2_{vw}}_{\text{part A}} - \underbrace{\sum_{w\in(N_{\mathcal{G}}(u)\cap N_{\mathcal{G}}(v))} \mathcal{P}_{u\wedge v}(w)(3 - 2\mathbf{P}_{uw} - 2\mathbf{P}_{vw} + \mathcal{P}_{u\wedge v}(w))}_{\text{part B}}. \quad (8)$$

Furthermore, for each node $w \in V$, we observe that for each pair (u, v) with $\{u, v\} \subseteq N_{\mathcal{G}}(w)$ and $u \neq v$, $\mathcal{P}_{u\wedge v}(w)$ contributes to $E[X_{uv}]$ (Eq. 6a), $E[Y_{uv}]$ (Eq. 7) and $Var(Y_{uv})$ (Eq. 8), and $\mathcal{P}_{u\wedge v}(w)$ $\mathcal{P}_{\bar{u}\wedge\bar{v}}(w)$ contributes to $\mathrm{Cov}(X_{uv}, Y_{uv})$ (Eq. 6d). In addition, for each node $u \in V$, $\sum_{w\in N_{\mathcal{G}}(u)} \mathbf{P}_{uw}$ and $\sum_{w\in N_{\mathcal{G}}(u)} \mathbf{P}^2_{uw}$ can be computed in $\mathcal{O}(d)$ time.

Based on the above analysis, the pseudocode of `FaseEJS` is shown in Alg. 1. `FastEJS` computes $\mathbf{S}^{\text{EJS}}$ as follows: after initialization (lines 1-2), for each node $w \in V$, we first record the sum ($\Psi(w)$) and square sum ($\Upsilon(w)$) of probabilities over all edges starting from w (line 5); then we enumerate all possible pairs (u, v) with $\{u, v\} \subseteq N_{\mathcal{G}}(w)$ and $u \neq v$, and for each candidate pair $e = (u, v)$, update the corresponding values of $\mu_x(e)$ (i.e. $E[X_{uv}]$), $\sigma_y(e)$ (i.e. $Var(Y_{uv})$) and $\sigma_{xy}(e)$ (i.e. $Cov(X_{uv}, Y_{uv})$) according to Eq. 6a, Eq. 8 part B, and Eq. 6d respectively (lines 7-11). Finally, for each candidate pair $e = (u, v)$ of nodes being computed, we calculate $\mu_y(e)$ (i.e. $E[Y_{uv}]$), update $\sigma_y(e)$ (i.e. $Var(Y_{uv})$) and compute $\mathbf{S}^{\text{EJS}}_{uv}$ based on Eq. 7, Eq. 8 (part A) and Eq. 6e respectively (lines 13-15). It is easy to prove that the time complexity of Alg. 1 is $\mathcal{O}(nd^2)$.

Algorithm 1: `FastEJS`($\mathcal{G}$)

Input: An uncertain graph $\mathcal{G} = (V, E, \mathbf{P})$
Output: $\mathbf{S}^{\text{EJS}}$, the expected Jaccard similarity matrix

1 $\mathbf{S}^{\text{EJS}} = \mathbf{0}_{n,n}$, $\mu_x = \emptyset$, $\mu_y = \emptyset$, $\sigma_y = \emptyset$, $\sigma_{xy} = \emptyset$;
2 $\Psi = \emptyset$, $\Upsilon = \emptyset$;
3 **for** *each node* $w \in V$ **do**
4 $T = N_{\mathcal{G}}(w)$; `/* the neighbor set of w */`
5 $\Psi(w) = \sum_{u\in T} \mathbf{P}_{wu}$, $\Upsilon(w) = \sum_{u\in T} \mathbf{P}^2_{wu}$;
6 **for** *each pair* $e = (u, v) \in T \times T$ *and* $u \neq v$ **do**
7 **if** $e \notin \mu_x$ **then**
8 Initialize $\mu_x(e), \mu_y(e), \sigma_y(e), \sigma_{xy}(e)$ to 0;
9 $\mu_x(e) = \mu_x(e) + \mathcal{P}_{u\wedge v}(w)$;
10 $\sigma_y(e) = \sigma_y(e) - \mathcal{P}_{u\wedge v}(w)(3 - 2\mathbf{P}_{uw}$ $-2\mathbf{P}_{vw}+\mathcal{P}_{u\wedge v}(w))$;
11 $\sigma_{xy}(e) = \sigma_{xy}(e) + \mathcal{P}_{u\wedge v}(w)\mathcal{P}_{\bar{u}\wedge\bar{v}}(w)$;
12 **for** *each pair* $e = (u, v) \in \mu_x$ **do**
13 $\mu_y(e) = \Psi(u) + \Psi(v) - \mu_x(e)$;
14 $\sigma_y(e) = \sigma_y(e) + \Psi(u) + \Psi(v) - \Upsilon(u) - \Upsilon(v)$;
15 $\mathbf{S}^{\text{EJS}}_{uv} = \frac{\mu_x(e)}{\mu_y(e)} - \frac{\sigma_{xy}(e)}{\mu_y(e)^2} + \frac{\mu_x(e)\sigma_y(e)}{\mu_y(e)^3}$;
16 **return** $\mathbf{S}^{\text{EJS}}$;

4.2 Probabilistic Random Walk with Restart

In this subsection, we first introduce the transition procedure between nodes on uncertain graphs and then present the probabilistic random walk with restart proximity.

4.2.1 Probabilistic Transition Matrix. Since each edge is associated with an existence probability in $\mathcal{G}$, the definition of transition probability from one node to another is different from the one on deterministic graphs. In this paper, we follow the definition of the transition procedure on uncertain graphs proposed in [31]. Specifically, for each node u, the transition procedure is defined as follows: 1) generate a possible world G for u; 2) walk to a neighbor uniformly at random if there exists any neighbors of u in G, otherwise stay at u.

Based on the above transition procedure, we now give the formal definition of the probabilistic transition matrix which describes the transition probability from one node to another.

Definition 4.4 (Probabilistic Transition Matrix [10, 31]). Given an uncertain graph $\mathcal{G} = (V, E, \mathbf{P})$ and its possible world set $\Omega(\mathcal{G})$, the probabilistic transition matrix (PTM for short) $\mathbf{W}$ is defined as:

$$\mathbf{W}_{uv} = \begin{cases} \prod_{(u,v)\in E}(1 - \mathbf{P}_{uv}), & u = v \\ \sum_{G\in\Omega(\mathcal{G})\wedge(u,v)\in E_G} \frac{1}{d^G_u} Pr[G], & u \neq v \end{cases} \quad (9)$$

where E_G is the edge set of the possible world G and d^G_u is the out-degree of node u in G.

Since the number of possible worlds is exponential, the time cost for computing the PTM of $\mathcal{G}$, $\mathbf{W}$, is extremely high if we directly follow Definition 4.4. How to calculate the PTM efficiently? Next, we first show an existing solution proposed in [10]. Then, we propose a faster algorithm to further improve the efficiency.

Let $\mathtt{St}(\mathtt{u})$ be a star graph of node u extracted from $\mathcal{G}$ by remaining edges associated with u, $|\mathtt{St}(\mathtt{u})|$ be the number of edges in $\mathtt{St}(\mathtt{u})$, and the combination probability $\Phi\binom{\mathcal{E}}{i}$ denote the probability that i edges exist in an edge set $\mathcal{E}$ with $|\mathcal{E}|$ edges. The summation of transition probabilities from node u to v on those possible worlds of $\mathcal{G}$ with out-degree being $i + 1$ ($0 \leq i \leq |\mathtt{St}(\mathtt{u})| - 1$) is $\frac{1}{i+1}\mathbf{P}_{uv}\Phi\binom{\mathtt{St}(\mathtt{u})\setminus(u,v)}{i}$. Based on this observation, Du et al. [10] derived another formula for $\mathbf{W}_{uv}$ ($u \neq v$) from the perspective of summing up the transition probabilities over all different values of out-degree from 1 to $|\mathtt{St}(\mathtt{u})|$:

$$\mathbf{W}_{uv} = \mathbf{P}_{uv} \sum_{i=0}^{|\mathtt{St}(\mathtt{u})|-1} \frac{1}{i+1} \times \Phi\binom{\mathtt{St}(\mathtt{u}) \setminus (u, v)}{i}. \quad (10)$$

Thus, the key problem of computing $\mathbf{W}_{uv}$ is to compute the combination probability Φ efficiently. For any edge set $\mathcal{E}$=$\{e_1, \cdots, e_{|\mathcal{E}|}\}$, let $\mathcal{E}_j$=$\{e_1, \cdots, e_j\}$ ($1 \leq j \leq |\mathcal{E}|$). [10] devised a dynamic programming method to compute $\Phi\binom{\mathcal{E}}{i}$ in quadratic time based on the following recursion equation:

$$\Phi\binom{\mathcal{E}_j}{i} = \begin{cases} \Phi\binom{\mathcal{E}_{j-1}}{i}(1 - \mathbf{P}_{e_j}), & i = 0 \\ \Phi\binom{\mathcal{E}_{j-1}}{i-1}\mathbf{P}_{e_j}, & i = j \\ \Phi\binom{\mathcal{E}_{j-1}}{i-1}\mathbf{P}_{e_j} + \Phi\binom{\mathcal{E}_{j-1}}{i}(1 - \mathbf{P}_{e_j}), & others \end{cases} \quad (11)$$

where $\mathbf{P}_{e_j}$ is the existence probability of edge e_j on $\mathcal{G}$, $0 \leq i \leq j$, and $1 \leq j \leq |\mathcal{E}|$. By initializing $\Phi\binom{\emptyset}{0}$ to 1, $\Phi\binom{\mathtt{St(u)} \setminus (u,v)}{i}$ $(i = 0, \ldots, |\mathtt{St(u)}| - 1)$ can be computed in $\mathcal{O}(d^2)$ time, where d is the out-degree of node u on $\mathcal{G}$.

After getting $\Phi\binom{\mathtt{St(u)} \setminus (u,v)}{i}$ for $i = 0, \ldots, (|\mathtt{St(u)}| - 1)$, $\mathbf{W}_{uv}$ can be computed in linear time based on Eq. 10. Then, the probabilistic transition probabilities of all edges starting from node u, i.e., $\mathbf{W}_{u*}$, can be computed straightforwardly in $\mathcal{O}(d^3)$ time by computing $\mathbf{W}_{uv}$ for each pair of edges [4].

In what follows, we propose a hash-based algorithm called `DP_hash`, which computes approximate $\mathbf{W}_{u*}$, denoted as $\widetilde{\mathbf{W}}_{u*}$, in $\mathcal{O}(hd^2)$ time where h is the size of the hash set, with $h \ll d$. The key idea is that, for two edges (u, v_k) and (u, v_t) which start from node u, if their existence probabilities are very close, then their probabilistic transition probabilities ($\mathbf{W}_{uv_k}$ and $\mathbf{W}_{uv_t}$) will also be very close. In the extreme case, if their existence probabilities are the same ($\mathbf{P}_{uv_k} = \mathbf{P}_{uv_t}$), then their probabilistic transition probabilities are the same too ($\mathbf{W}_{uv_k} = \mathbf{W}_{uv_t}$). Hence, after dividing the existence probability interval (i.e. (0, 1]) into h slots, when computing $\mathbf{W}_{uv_k}$, we first check whether there is an edge (u, v_t) with similar existence probability (e.g., mapping to the same slot) has been computed. If so, we directly set $\widetilde{\mathbf{W}}_{uv_k}$ as $\mathbf{W}_{uv_t}$; otherwise compute $\widetilde{\mathbf{W}}_{uv_k}$ based on Eq. 11 and Eq. 10. Alg. 2 is the pseudocode of the `DP_hash` algorithm. Since each hash slot is computed at most once, the time complexity of Alg. 2 is $\mathcal{O}(hd^2)$.

Algorithm 2: `DP_hash(St(u), h)`

Input: The star graph of node u in $\mathcal{G}$ with edges $(u, v_1), \cdots, (u, v_{|\mathtt{St(u)}|})$ and h, the size of the hash set
Output: $\mathbf{W}_{u*}$, the probabilistic transition probabilities of all edges starting from u

```
1  if St(u) contains only one edge (u, v_1) then
2      return W̃_{uv_1} = P_{uv_1};
3  Initialize H[0...h-1] = false, Ψ = ∅;
4  for each edge (u, v_k) ∈ St(u) do
5      id = Round(P_{uv_k} · h); /* map P_{uv_k} to a hash slot */
6      if H[id] is true then
7          W̃_{uv_k} = Ψ[id];
8      else
9          Compute Φ(St(u)\(u,v_k) choose i) (i=0...|St(u)|-1) based on
           Eq. 11;
10         Compute W̃_{uv_k} = W_{uv_k} based on Eq. 10;
11         H[id] = true, Ψ[id] = W̃_{uv_k};
12 return W̃_{u*};
```

THEOREM 4.5. *For any edge $(u, v_k) \in \mathtt{St(u)}$, suppose the approximate value of the probabilistic transition probability from node u to v_k returned by Alg. 2 is $\widetilde{\mathbf{W}}_{uv_k}$. The accuracy loss can be bounded as: $|\mathbf{W}_{uv_k} - \widetilde{\mathbf{W}}_{uv_k}| < \frac{1}{h}$.*

[4] [10] also proposed an incremental dynamic programming algorithm. However, practically it returns answers with unacceptable precision loss. Hence, we omit the comparison with it.

PROOF. There are two cases to obtain $\widetilde{\mathbf{W}}_{uv_k}$ based on whether $\mathcal{H}[id]$ is true or not (line 6). If $\mathcal{H}[id]$ is false, $\widetilde{\mathbf{W}}_{uv_k}$ is obtained based on Eq. 10 directly (lines 9-11), i.e., $\widetilde{\mathbf{W}}_{uv_k} = \mathbf{W}_{uv_k}$. Then, the inequation holds.

Otherwise, $\widetilde{\mathbf{W}}_{uv_k}$ is obtained from other recorded probabilistic transition probability (say, $\mathbf{W}_{uv_t}$) since $\mathbf{P}_{uv_t}$ and $\mathbf{P}_{uv_k}$ are mapped to the same hash slot (line 7). Then, let e_k (resp. e_t) denote the edge (u, v_k) (resp. edge (u, v_t)) and $m = |\mathtt{St(u)}|$. We have:

$$
\begin{aligned}
&|\mathbf{W}_{uv_k} - \widetilde{\mathbf{W}}_{uv_k}| = |\mathbf{W}_{uv_k} - \mathbf{W}_{uv_t}| \\
&= \left| \mathbf{P}_{uv_k} \sum_{i=0}^{m-1} \frac{1}{i+1} \Phi\binom{\mathtt{St(u)} \setminus e_k}{i} - \mathbf{P}_{uv_t} \sum_{i=0}^{m-1} \frac{1}{i+1} \Phi\binom{\mathtt{St(u)} \setminus e_t}{i} \right| \\
&= \left| \mathbf{P}_{uv_k} \left(\sum_{i=0}^{m-2} \frac{1}{i+1} \Phi\binom{\mathtt{St(u)} \setminus \{e_k, e_t\}}{i} (1 - \mathbf{P}_{uv_t}) \right.\right. \\
&\qquad \left. + \sum_{i=1}^{m-2} \frac{1}{i+1} \Phi\binom{\mathtt{St(u)} \setminus \{e_k, e_t\}}{i} \mathbf{P}_{uv_t} \right) \\
&\quad - \mathbf{P}_{uv_t} \left(\sum_{i=0}^{m-2} \frac{1}{i+1} \Phi\binom{\mathtt{St(u)} \setminus \{e_k, e_t\}}{i} (1 - \mathbf{P}_{uv_k}) \right. \\
&\qquad \left.\left. + \sum_{i=1}^{m-2} \frac{1}{i+1} \Phi\binom{\mathtt{St(u)} \setminus \{e_k, e_t\}}{i} \mathbf{P}_{uv_k} \right) \right| \\
&= \left| (\mathbf{P}_{uv_k} - \mathbf{P}_{uv_t}) \sum_{i=0}^{m-2} \frac{1}{i+1} \Phi\binom{\mathtt{St(u)} \setminus \{e_k, e_t\}}{i} \right|
\end{aligned} \tag{12}
$$

Since $\sum_{i=0}^{m-2} \Phi\binom{\mathtt{St(u)} \setminus \{e_k, e_t\}}{i} = 1$, we have:

$$
\sum_{i=0}^{m-2} \frac{1}{i+1} \Phi\binom{\mathtt{St(u)} \setminus \{e_k, e_t\}}{i} \leq 1, \text{ and} \tag{13a}
$$

$$
|\mathbf{W}_{uv_k} - \widetilde{\mathbf{W}}_{uv_k}| \leq |(\mathbf{P}_{uv_k} - \mathbf{P}_{uv_t})| < \frac{1}{h}. \tag{13b}
$$

□

Based on Alg. 2, the PTM of $\mathcal{G}$ can be computed in $\mathcal{O}(hnd^2)$ time by invoking Alg. 2 for each node $u \in V$.

4.2.2 Probabilistic Random Walk with Restart. Random walk with restart (RWR) [29] is a widely adopted proximity measure, due to its ability to capture the global structure of the graph. RWR proximity from node u to node v, is the stationary probability for a random walker starting from u to reach v after infinite time; at any transition the random walk restarts at u with probability α $(0 < \alpha < 1)$ and moves to a random neighbor with probability $(1 - \alpha)$. The stationary probability of reaching another node v from u reflects how close u is to v with respect to the graph structure.

Given an uncertain graph $\mathcal{G}$, the probabilistic random walk with restart (PRWR) is defined as the RWR based on the PTM. Let $\mathbf{S}^{\mathtt{PRWR}}$ be the PRWR proximity matrix, it can be computed iteratively using the following formula:

$$
\mathbf{S}^{\mathtt{PRWR}} = (1 - \alpha)\mathbf{S}^{\mathtt{PRWR}}\mathbf{W} + \alpha\mathbf{I}, \tag{14}
$$

where $\mathbf{W}$ is the PTM and $\mathbf{I}$ is an identity matrix.

Given the PTM, Monte Carlo methods can be used to simulate the random walk process [11, 23]. Particularly, we

simulate R independent random walkers starting from u and taking L steps, where L follows a geometric distribution $Pr[L = i] = \alpha(1-\alpha)^i, i = \{0, 1, 2...\}$ with α as the restart probability. Denote by R_x the number of times the walker ends at x. Then, the PRWR of node x in the view of u ($\mathbf{S}^{\texttt{PRWR}}_{ux}$) can be approximated by the probability that the random walker ends at x, i.e., $\widetilde{\mathbf{S}}^{\texttt{PRWR}}_{ux} = \frac{R_x}{R}$. Based on the Hoeffding Inequality [14], the approximate PRWR can be bounded as: $\Pr[|\mathbf{S}^{\texttt{PRWR}}_{ux} - \widetilde{\mathbf{S}}^{\texttt{PRWR}}_{ux}| \geq \epsilon] \leq 2\exp(-2R\epsilon^2)$, for any $\epsilon > 0$.

Given the PTM, $\mathbf{S}^{\texttt{PRWR}}$ can be computed in $\mathcal{O}(nR\frac{1}{\alpha})$ time using the above Monte Carlo method, where R is the number of walkers, and $1/\alpha$ is the expected length of random paths.

5 OPTIMIZING THE TRAINING PROCESS

Given the proximity matrix $\mathbf{S}$, directly optimizing the loss function (Eq. 2) is time consuming, because there are n^2 entries in $\mathbf{S}$ and we need to calculate the gradient for each entry (a pair of nodes (u, v), $\{u, v\} \subseteq V$). To speed up the training process, inspired by the negative sampling approach proposed in [26], which samples a small number of negative objects to enhance the influence of positive objects, we define the loss function for each pair (u, v) of objects with *non-zero* proximity $\mathbf{S}_{uv}$ as follows:

$$(\mathbf{S}_{uv} - \mathbf{U}_u\tilde{\mathbf{U}}_v^T)^2 + \frac{\lambda}{2}(\|\mathbf{U}_u\|^2 + \|\tilde{\mathbf{U}}_v\|^2) + \sum_{i=1}^{k} E_{u_i \sim P_n(u)}((\mathbf{S}_{uu_i} - \mathbf{U}_u\tilde{\mathbf{U}}_{u_i}^T)^2 + \frac{\lambda}{2}(\|\mathbf{U}_u\|^2 + \|\tilde{\mathbf{U}}_{u_i}\|^2)) \quad (15)$$

where k is the times of sampling, $P_n(u) \propto d_u^{3/4}$ as proposed in [26], and d_u is the out-degree of node u. We adopt the asynchronous stochastic gradient algorithm (ASGD)[17] for optimizing Eq. 15. In each step the ASGD algorithm samples a mini-batch of edges and then updates the model parameters.

6 EXPERIMENTAL RESULTS

We have evaluated URGE over three common uncertain graph mining tasks: node clustering, node classification, and k-NN search. For all these tasks, we consider the following embedding algorithms:

• DeepWalk [30][5], which learns K-dimensional feature representations by simulating uniform random walks.

• LINE [36][6] is an embedding method that preserves first- and second-order proximity between nodes. For each node, it learns two vectors for the first-order ($K/2$ dimensions) and second-order ($K/2$ dimensions) proximities separately, and then concatenates them.

• node2vec^q_p [13][7] extends DeepWalk to exploit homophily and structural roles for node embedding based on a biased random walk procedure. Here p and q are the return and in-out parameters respectively. Note that DeepWalk is a special case of node2vec with $p = 1$ and $q = 1$. We perform experiments for both node2vec$^{0.25}_{0.25}$ and node2vec$^1_4$, which are the best values of p and q as suggested by the authors of [13].

• URGE$_{\texttt{EJS}}$ is our URGE solution based on EJS.

• URGE$_{\texttt{PRWR}}$ is our URGE algorithm based on PRWR. To compute $\mathbf{S}^{\texttt{PRWR}}$, we set the size of hash set $h = 100$, walkers per node $R = 5000$ and restart probability $\alpha = 0.15$.

In the methods above, the dimension K is 128 by default, as used in [13, 30, 36]. For DeepWalk and node2vec, we use the typical values used in [13], i.e., 10 walkers per node, walk length $l = 80$, context size $k = 10$. For LINE, we use the default training settings, i.e., starting value of the learning rate $\rho_0 = 0.025$ and total number of training samples $T = 100M$ for both first-order and second-order representations. Unless otherwise stated, the number of negative samples is 3, and the total number of training samples $T = 100M$.

We have also compared the effectiveness of the above solutions with representative uncertain data mining solutions, namely, MCL [40] and pKwikCluster [21] (for clustering); uBayes$^+$ [8] (for classification); and MostProbPath [31] (for k-NN). We will describe settings of these solutions later.

All the experiments are conducted on a 16GB memory machine with Intel(R) Core(TM) i7 CPU@2.3 GHz.

6.1 Clustering

We first conduct clustering tasks for four real Protein-Protein Interaction (PPI) networks[8] (two proteins are linked if it is likely that they interact and all interactions are labeled with probabilities (confidence) by biologists): 1) Krogan_core: the core interaction dataset from [22] with all edges have probability no less than 0.27 and about one fourth of the edges with probability greater than 0.98; 2) Krogan_extend: the extended interaction dataset from [22] which contains less reliable interactions than the Krogan_core dataset, but its coverage is higher; 3) Collins: the weighted interaction map of [7], with mostly high-probability edges; 4) Gavin: the dataset of [12], with the majority of edges having low probabilities. The detailed statistics of PPI datasets are summarized into Table 1.

The PPI networks are one of the benchmark datasets used in previous works on uncertain graph clustering [21, 24]. In PPI networks, proteins can be grouped into different complexes. Proteins in the same group will interact with each other stably. In this group of experiments, we validate the output of different methods with respect to a known ground truth. We use the complex-memberships lists from the MIPS database [25] as the ground truth. MIPS complexes define co-complex relationships among proteins. Here, a co-complex relationship is a pair of proteins that both belong to the same complex. For each PPI network $\mathcal{G}$, during the evaluation, we only keep the proteins that occur in both $\mathcal{G}$ and MIPS with complex size no less than 3, while the input to the clustering algorithms is the entire graph $\mathcal{G}$. The detailed

[5] https://github.com/phanein/deepwalk
[6] https://github.com/tangjianpku/LINE
[7] https://github.com/aditya-grover/node2vec

[8] http://www.nature.com/nmeth/journal/v9/n5/full/nmeth.1938.html

Name	#Nodes	#Edges	Avg. Prob.
Krogan_core	2,708	7,123	0.68
Krogan_extend	3,672	14,317	0.42
Collins	1,622	9,074	0.78
Gavin	1,855	7,669	0.36

Table 1: Statistics of the PPI networks.

Name	#Nodes	#Complexes	#GT Pairs
MIPS∩Krogan_core	639	166	13,492
MIPS∩Krogan_extend	885	178	17,101
MIPS∩Collins	765	153	13,552
MIPS∩Gavin	754	148	13,206

Table 2: Ground truth of the PPI networks.

Algorithm	Krogan_core	Krogan_extend	Collins	Gavin
DeepWalk	39.21	33.43	55.15	47.33
LINE	38.73	33.07	48.28	44.14
node2vec$^{1}_{4}$	39.30	33.06	52.42	46.17
node2vec$^{0.25}_{0.25}$	38.96	33.75	53.23	46.13
MCL	36.01	30.83	57.55	47.84
pKwikCluster	16.94	12.88	24.59	5.65
URGE$_{\texttt{EJS}}$	38.39	30.08	55.61	**54.54**
URGE$_{\texttt{PRWR}}$	**44.86**	**35.58**	**58.16**	52.59

Table 3: F1 scores (%) for clustering tasks.

statistics of the ground truth for different PPIs are listed in Table 2 (GT is short for Ground Truth).

In addition to the embedding algorithms mentioned, we compare URGE with the following two clustering algorithms:

• MCL [40][9] is a clustering algorithm for deterministic graphs based on random walks and matrix multiplications. MCL is used for comparison in previous works on uncertain graphs [21, 24].

• pKwikCluster [21] is a clustering algorithm for uncertain graphs.

We evaluate the output clusters in terms of the confusion matrix, i.e., true positive, false positive, true negative and false negative. We report the F1 score based on the confusion matrix. For embedding algorithms, after getting the K-dimensional representations for all nodes, hierarchical clustering (Agglomerative with pooling-func being "mean") [32] is adopted to group proteins into different clusters. The number of clusters for hierarchical clustering is equal to the number of complexes in MIPS.

Results. Table 3 shows the results of clustering under different algorithms. Note that all existing embedding methods for deterministic graphs, i.e., DeepWalk, LINE and node2vec have similar performance. Moreover, MCL performs better than pKwikCluster on all the PPI networks. We also observe that URGE$_{\texttt{PRWR}}$ perform better than all other methods, because it keeps the high-order proximity of these uncertain graphs well. Meanwhile, URGE$_{\texttt{EJS}}$ is worse than URGE$_{\texttt{PRWR}}$ on most datasets except Gavin, since it only considers the second-order proximity.

6.2 Classification

Next, we evaluate our solutions for node classification. We conduct experiments on synthetic uncertain graphs generated by the obfuscating algorithm proposed in [5][10] on deterministic graphs. The authors in [5] studied the problem of injecting uncertainty (adding suitable edge probabilities) to turn a deterministic graph G into an uncertain graph $\mathcal{G}$ for the purpose of identity obfuscation. Individual nodes of the original graph G can no longer be identified in $\mathcal{G}$, while $\mathcal{G}$ preserves some global properties of G, such as degree distribution, diameter, and clustering coefficient.

Metric	Algorithm	20%	40%	60%	80%
Micro-F1(%)	DeepWalk	40.81	49.71	54.33	57.27
	LINE	40.87	47.96	53.26	56.52
	node2vec$^{1}_{4}$	41.25	51.29	55.67	59.27
	node2vec$^{0.25}_{0.25}$	40.16	49.33	53.53	56.98
	uBayes^{+}	32.43	45.13	44.57	57.44
	URGE$_{\texttt{EJS}}$	**58.00**	**63.31**	**66.36**	**69.45**
	URGE$_{\texttt{PRWR}}$	52.16	58.08	61.12	63.97
Macro-F1(%)	DeepWalk	38.12	48.22	52.95	55.98
	LINE	39.02	46.37	51.70	55.08
	node2vec$^{1}_{4}$	39.42	49.21	53.93	57.69
	node2vec$^{0.25}_{0.25}$	38.11	47.64	51.93	55.37
	uBayes^{+}	31.07	42.02	45.03	55.33
	URGE$_{\texttt{EJS}}$	**55.48**	**61.45**	**64.49**	**67.50**
	URGE$_{\texttt{PRWR}}$	49.86	56.41	59.64	62.42

Table 4: Results of classification on DBLP under different training ratio(%).

We generate an uncertain graph for each of the following real-world deterministic graphs with default values of parameters: 1) DBLP: a bibliographic co-authorship information network from DBLP used in [17] that contains 14,376 papers and 45,583 edges (after obfuscation) classified into 4 classes; 2) Cora [34][11]: a research paper citation network which contains 2,708 machine learning papers and 8,365 edges (after obfuscation) classified into 7 classes.

Here, we consider uBayes^{+} [8][12] which is a Bayes-based classification algorithm customized for uncertain graphs.

For the embedding algorithms, after getting the 128 dimensional representations for each nodes, we use one-vs-rest k nearest neighbor (k-NN) classifiers with $k = 5$ to predict the labels of the test samples. We increase the training ratio (T_R) on both datasets from 20% to 80%. For each fixed T_R, we repeat the process 10 times and report the average Micro-F1 and Macro-F1 scores.

Results. Tables 4 and 5 show the results of classification on DBLP and Cora respectively. Notice that all our solutions (URGE$_{\texttt{EJS}}$ and URGE$_{\texttt{PRWR}}$) significantly outperform others in terms of Micro-F1 and Macro-F1 consistently under different train ratio. In particular, for the DBLP dataset (Table 4), URGE$_{\texttt{EJS}}$ achieves the best Micro-F1 improvement on 17.2% over the best performing baseline (node2vec$^{1}_{4}$) when T_R=80%. For the Cora dataset (Table 5), URGE$_{\texttt{PRWR}}$ can achieve the best Micro-F1 improvement on 28.7% over the best performing baseline (node2vec$^{1}_{4}$) when T_R=40%.

[9] http://www.micans.org/mcl

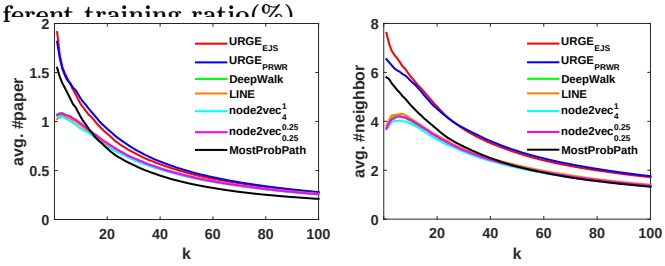
[10] http://boldi.di.unimi.it/obfuscation/

[11] http://linqs.cs.umd.edu/projects/projects/lbc/

[12] http://www.mi.parisdescartes.fr/~themisp/collectiveclassification/

Metric	Algorithm	20%	40%	60%	80%
Micro-F1(%)	DeepWalk	44.93	52.81	56.56	60.09
	LINE	33.41	38.92	43.04	47.14
	node2vec$^{1}_{4}$	44.50	52.48	56.29	59.62
	node2vec$^{0.25}_{0.25}$	44.15	51.60	56.39	60.19
	uBayes^{+}	36.49	44.70	52.46	58.15
	URGE$_{\texttt{EJS}}$	57.22	62.47	64.51	66.00
	URGE$_{\texttt{PRWR}}$	**62.87**	**67.56**	**69.61**	**72.03**
Macro-F1(%)	DeepWalk	40.05	48.91	53.08	56.72
	LINE	27.23	34.31	39.23	43.27
	node2vec$^{1}_{4}$	39.56	48.55	52.59	56.04
	node2vec$^{0.25}_{0.25}$	39.28	47.88	52.97	56.10
	uBayes^{+}	35.67	45.34	52.84	57.55
	URGE$_{\texttt{EJS}}$	54.07	60.48	61.96	63.25
	URGE$_{\texttt{PRWR}}$	**60.38**	**65.24**	**67.31**	**70.18**

Table 5: Results of classification on Cora under different training ratio(%).



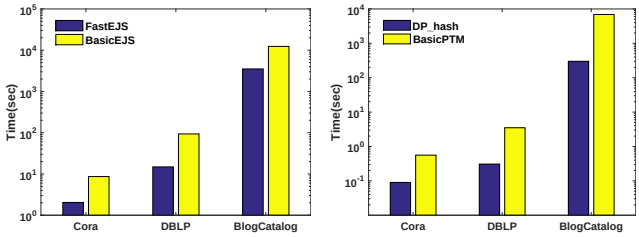
(a) Avg. #paper w.r.t. k **(b) Avg. #neighbor w.r.t. k**

Figure 3: Results of k-NN search on DBLP.

6.3 k-NN Search

We also evaluate the performance of our solutions on k-NN search tasks. Specifically, we conduct experiments on the DBLP dataset to compare the quality of k nearest authors in the embedded space. In DBLP, two well-studied metrics to measure the closeness between authors are: 1) the average number of co-author papers between authors; 2) the average number of common neighbors between authors. We evaluate the quality of k-NN results by measuring their closeness to the query node using the aforementioned metrics. For each node with degree larger than 10, we compute the average closeness between the query node and all its k-NN results by using the original DBLP dataset as the ground truth. We study MostProbPath [31] which is a k-NN search method for uncertain graphs.

Results. By varying k from 1 to 100, Fig. 3a and Fig. 3b show the results of k-NN search in terms of the average number of co-author papers and common neighbors respectively. We observe that our methods (URGE$_{\texttt{EJS}}$ and URGE$_{\texttt{PRWR}}$) perform much better than other competitors, since the nearest authors found by our methods are closer to the query nodes in terms of more co-author papers and common neighbors. Specifically, when $k = 1$, the average number of co-author papers (resp. common neighbors) is 1.9 (resp. 7.6) for URGE$_{\texttt{EJS}}$, while the corresponding value is 1.54 (resp. 5.8) for the best baseline (MostProbPath). In addition, URGE$_{\texttt{EJS}}$ and URGE$_{\texttt{PRWR}}$ achieve similar performance. The reason is that DBLP is a co-authorship network and low-order, short-distance relationship already reflects the graph structure well.



(a) EJS **(b) PTM**

Figure 4: Efficiency evaluation.

Proximity	Cora	DBLP	BlogCatalog
$\mathbf{S}^{\texttt{EJS}}$	2.04	14.85	3517.6
$\mathbf{S}^{\texttt{PRWR}}$	10.95	58.1	802.8

Table 6: Running time of computing different proximity matrices (in seconds).

6.4 Efficiency

Finally, we evaluate the efficiency of computing different kinds of proximity matrices on Cora and DBLP. We also experiment on a larger dataset, called BlogCatalog [37], which is a network of social relationships of the bloggers with 10,312 nodes and 665,018 edges (after obfuscation). For $\mathbf{S}^{\texttt{EJS}}$, we compare our algorithm (`FastEJS`, Alg. 1) with the basic solution (`BasicEJS`) which directly applies the algorithm proposed in [45]. For $\mathbf{S}^{\texttt{PRWR}}$, we first evaluate the time cost on computing the PTM and then report the running time of computing $\mathbf{S}^{\texttt{PRWR}}$. We compare our hash-based dynamic programming algorithm (`DP_hash`, Alg. 2) with the basic solution (`BasicPTM`) proposed in [10].

As shown in Fig. 4, both of our algorithms for EJS and PTM are significantly faster than the basic ones. For example, `FasterEJS` is 6.3 times faster than `BasicEJS` on DBLP, while `DP_hash` is 23 times faster than `BasicPTM` on BlogCatalog.

The execution time of computing different proximity matrices using our proposed algorithms is reported in Table 6. We can observe that all proximity matrices can be computed efficiently. In particular, computing $\mathbf{S}^{\texttt{EJS}}$ is faster than $\mathbf{S}^{\texttt{PRWR}}$ in smaller and sparser datasets (Cora and DBLP), while it is much slower in the larger and denser graph (BlogCatalog). In addition, using ASGD to train the model in our experiments is very efficient. For example it only takes an average of 47.4s on BlogCatalog.

To summarize, URGE outperforms existing embedding algorithms (DeepWalk, LINE and node2vec) and uncertain data mining algorithms (MCL, pKwikCluster, uBayes^{+} and MostProbPath), for clustering, classification and k-NN. Our novel algorithms for computing the proximity matrices are also faster than existing ones.

7 CONCLUSIONS

In this paper, we examine the problem of uncertain graph embedding and propose URGE, a proximity preserved embedding method for uncertain graphs. Specifically, we investigate two kinds of proximities (EJS and PRWR), as well as efficient algorithms to compute them. Our experiments on both real and synthetic datasets demonstrate the effectiveness of the

URGE for various tasks, e.g., clustering, classification and k-NN search, as well as the efficiency of our algorithms for the computation of proximity matrices.

ACKNOWLEDGMENTS

Jiafeng Hu, Reynold Cheng, Zhipeng Huang, Yixiang Fang and Siqiang Luo were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 17205115) and the University of Hong Kong (Projects 102009508 ,104004129, and 201611159247). We would like to thank the reviewers for their insightful comments.

REFERENCES

[1] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. 2013. Distributed Large-scale Natural Graph Factorization. In *WWW*. 37–48.

[2] Mikhail Belkin and Partha Niyogi. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, Vol. 14. 585–591.

[3] Smriti Bhagat, Graham Cormode, and S Muthukrishnan. 2011. Node classification in social networks. In *Social network data analytics*. Springer, 115–148.

[4] Sanjit Biswas and Robert Morris. 2005. ExOR: opportunistic multi-hop routing for wireless networks. *ACM SIGCOMM Computer Communication Review* 35, 4 (2005), 133–144.

[5] Paolo Boldi, Francesco Bonchi, Aristides Gionis, and Tamir Tassa. 2012. Injecting uncertainty in graphs for identity obfuscation. *VLDB* 5, 11 (2012), 1376–1387.

[6] Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. 2014. Core decomposition of uncertain graphs. In *KDD*. 1316–1325.

[7] Sean R Collins, Patrick Kemmeren, Xue-Chu Zhao, Jack F Greenblatt, Forrest Spencer, Frank CP Holstege, Jonathan S Weissman, and Nevan J Krogan. 2007. Toward a comprehensive atlas of the physical interactome of Saccharomyces cerevisiae. *Molecular & Cellular Proteomics* 6, 3 (2007), 439–450.

[8] Michele Dallachiesa, Charu Aggarwal, and Themis Palpanas. 2014. Node Classification in Uncertain Graphs. In *SSDBM*. Article 32, 4 pages.

[9] Lee R Dice. 1945. Measures of the amount of ecologic association between species. *Ecology* 26, 3 (1945), 297–302.

[10] Lingxia Du, Cuiping Li, Hong Chen, Liwen Tan, and Yinglong Zhang. 2015. Probabilistic SimRank computation over uncertain graphs. *Information Sciences* 295 (2015), 521–535.

[11] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics* 2, 3 (2005), 333–358.

[12] Anne-Claude Gavin, Patrick Aloy, Paola Grandi, Roland Krause, Markus Boesche, Martina Marzioch, Christina Rau, Lars Juhl Jensen, Sonja Bastuck, Birgit Dümpelfeld, and others. 2006. Proteome survey reveals modularity of the yeast cell machinery. *Nature* 440, 7084 (2006), 631–636.

[13] Aditya Grover and Jure Leskovec. 2016. Node2Vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.

[14] Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association* 58, 301 (1963), 13–30.

[15] Xin Huang, Wei Lu, and Laks V.S. Lakshmanan. 2016. Truss Decomposition of Probabilistic Graphs: Semantics and Algorithms. In *SIGMOD*. 77–90.

[16] Paul Jaccard. 1901. *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz.

[17] Ming Ji, Yizhou Sun, Marina Danilevsky, Jiawei Han, and Jing Gao. 2010. Graph regularized transductive classification on heterogeneous information networks. In *ECML PKDD*. Springer, 570–586.

[18] Ruoming Jin, Lin Liu, and Charu C Aggarwal. 2011. Discovering highly reliable subgraphs in uncertain graphs. In *KDD*. 992–1000.

[19] Vasileios Kassiano, Anastasios Gounaris, Apostolos N Papadopoulos, and Kostas Tsichlas. 2016. Mining Uncertain Graphs: An Overview. In *International Workshop of Algorithmic Aspects of Cloud Computing*. Springer, 87–116.

[20] David Kempe et al. 2003. Maximizing the spread of influence through a social network. In *KDD*. 137–146.

[21] George Kollios, Michalis Potamias, and Evimaria Terzi. 2013. Clustering large probabilistic graphs. *TKDE* 25, 2 (2013), 325–336.

[22] Nevan J Krogan, Gerard Cagney, Haiyuan Yu, Gouqing Zhong, Xinghua Guo, Alexandr Ignatchenko, Joyce Li, Shuye Pu, Nira Datta, Aaron P Tikuisis, and others. 2006. Global landscape of protein complexes in the yeast Saccharomyces cerevisiae. *Nature* 440, 7084 (2006), 637–643.

[23] Nan Li, Ziyu Guan, Lijie Ren, Jian Wu, Jiawei Han, and Xifeng Yan. 2013. giceberg: Towards iceberg analysis in large graphs. In *ICDE*. 1021–1032.

[24] Lin Liu, Ruoming Jin, Charu Aggarwal, and Yelong Shen. 2012. Reliable clustering on uncertain graphs. In *ICDM*. 459–468.

[25] Hans-Werner Mewes, C Amid, Roland Arnold, Dmitrij Frishman, Ulrich Güldener, Gertrud Mannhaupt, Martin Münsterkötter, Philipp Pagel, Normann Strack, Volker Stümpflen, and others. 2004. MIPS: analysis and annotation of proteins from whole genomes. *Nucleic acids research* 32, suppl 1 (2004), 41–44.

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

[27] A. P. Mukherjee, P. Xu, and S. Tirthapura. 2015. Mining maximal cliques from an uncertain graph. In *ICDE*. 243–254.

[28] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *KDD*. 1105–1114.

[29] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. 2004. Automatic multimedia cross-modal correlation discovery. In *KDD*. 653–658.

[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. 701–710.

[31] Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. 2010. K-nearest neighbors in uncertain graphs. *VLDB* 3, 1-2 (2010), 997–1008.

[32] Lior Rokach and Oded Maimon. 2005. Clustering methods. In *Data mining and knowledge discovery handbook*. Springer, 321–352.

[33] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[34] Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Magazine* 29, 3 (2008), 93–106.

[35] Alan Stuart and Keith Ord. 1998. *Kendall's Advanced Theory of Statistics*. Vol. 1. Wiley. 351–351 pages.

[36] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.

[37] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *KDD*. 817–826.

[38] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.

[39] Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, and Maosong Sun. 2016. Max-Margin DeepWalk: Discriminative Learning of Network Representation. In *IJCAI*. 3889–3895.

[40] Stijn Van Dongen. 2008. Graph clustering via a discrete uncoupling process. *SIAM J. Matrix Anal. Appl.* 30, 1 (2008), 121–141.

[41] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *KDD*. 1225–1234.

[42] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. 2015. Network Representation Learning with Rich Text Information. In *IJCAI*. 2111–2117.

[43] R. Zhu, Z. Zou, and J. Li. 2016. SimRank computation on uncertain graphs. In *ICDE*. 565–576.

[44] Zhaonian Zou, Hong Gao, and Jianzhong Li. 2010. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *KDD*. 633–642.

[45] Zhaonian Zou and Jianzhong Li. 2013. Structural-context similarities for uncertain graphs. In *ICDM*. 1325–1330.

[46] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. 2010. Mining frequent subgraph patterns from uncertain graph data. *TKDE* 22, 9 (2010), 1203–1218.