Are Meta-Paths Necessary? Revisiting Heterogeneous Graph Embeddings

Rana Hussein
eXascale Infolab, University of Fribourg, Switzerland

Dingqi Yang*
eXascale Infolab, University of Fribourg, Switzerland
{firstname.lastname}@unifr.ch

Philippe Cudré-Mauroux
eXascale Infolab, University of Fribourg, Switzerland

ABSTRACT

The graph embedding paradigm projects nodes of a graph into a vector space, which can facilitate various downstream graph analysis tasks such as node classification and clustering. To efficiently learn node embeddings from a graph, graph embedding techniques usually preserve the proximity between node pairs sampled from the graph using random walks. In the context of a heterogeneous graph, which contains nodes from different domains, classical random walks are biased towards highly visible domains where nodes are associated with a dominant number of paths. To overcome this bias, existing heterogeneous graph embedding techniques typically rely on meta-paths (i.e., fixed sequences of node types) to guide random walks. However, using these meta-paths either requires prior knowledge from domain experts for optimal meta-path selection, or requires extended computations to combine all meta-paths shorter than a predefined length. In this paper, we propose an alternative solution that does not involve any meta-path. Specifically, we propose JUST, a heterogeneous graph embedding technique using random walks with JUmp and STay strategies to overcome the aforementioned bias in an more efficient manner. JUST can not only gracefully balance between homogeneous and heterogeneous edges, it can also balance the node distribution over different domains (i.e., node types). By conducting a thorough empirical evaluation of our method on three heterogeneous graph datasets, we show the superiority of our proposed technique. In particular, compared to a state-of-the-art heterogeneous graph embedding technique Hin2vec, which tries to optimally combine all meta-paths shorter than a predefined length, our technique yields better results in most experiments, with a dramatically reduced embedding learning time (about 3x speedup).

CCS CONCEPTS

• **Computing methodologies** → **Knowledge representation and reasoning**; *Unsupervised learning*; • **Information systems** → *Social networks*;

KEYWORDS

Graph embedding, Heterogeneous graph, Random walk

ACM Reference Format:
Rana Hussein, Dingqi Yang, and Philippe Cudré-Mauroux. 2018. Are Meta-Paths Necessary? Revisiting Heterogeneous Graph Embeddings. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271777

*Corresponding author

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271777

1 INTRODUCTION

With the proliferation of Social Networks and graph data on the Web, graph embeddings (a.k.a. network embeddings or network representation learning) [2] have gained increasing popularity in recent years. The key idea is to represent nodes in a graph using a continuous low-dimensional vector space (i.e., node embeddings), while preserving key structural properties (e.g., node proximity) of the graph. These node embeddings can then be effectively used in various supervised/unsupervised graph analysis tasks, such as node classification or clustering.

Existing graph embedding techniques can be classified into two categories depending on the types of input graphs, i.e., homogeneous or heterogeneous graph embeddings [6]. On one hand, for a homogeneous graph containing nodes from a single domain only (e.g., a social network of users), graph embedding techniques try to preserve the proximity between node pairs *randomly* sampled from the graph (using random walks over the graph [9, 18], for example). On the other hand, a heterogeneous graph has a more complex structure containing multiple node domains, with both homogeneous edges linking nodes from the same domain and heterogeneous edges linking nodes across different domains [22, 23]. For example, a bibliographic graph with author (A), paper (P), topic (T) and venue (V) domains contains not only the homogeneous edges between papers (paper citation), but also heterogeneous edges linking authors and papers (authorship) or linking papers and venues (publication). Consequently, the corresponding graph embedding techniques usually give a particular consideration to the structure of an heterogeneous graph when sampling node pairs from it.

In the current literature, heterogeneous graph embedding techniques mostly rely on *meta-paths* [23] to sample node pairs for learning node embeddings. Specifically, a meta-path is defined as a sequence of node types encoding key composite relations among the involved node types [23]. Different meta-paths express different semantic meaning. Taking a bibliographic graph as an example, a meta-path "A-P-A" represents a co-authorship relationship on a paper between two authors, while "A-P-V-P-A" represents papers published by two authors in the same venue. To incorporate

such meta-paths in graph embeddings, existing techniques usually perform meta-path-guided random walks to sample node pairs for learning node embeddings [5, 7, 8, 11, 21]. However, despite the wide adoption of meta-paths in heterogeneous graph embeddings, how to select meta-paths from a given heterogeneous graph remains unclear. Even more critically, the existing literature has shown that the choice of meta-paths highly affect the quality of the learnt node embeddings, which makes them a double-edged sword. Taking the same example of a bibliographic graph, using the meta-path "A-P-T-P-A" results in 11% and 21% performance drop in node classification and clustering tasks, respectively, compared to the meta-path "A-P-V-P-A" (the optimal meta-path suggested by [7]). To overcome this issue, existing techniques either manually select a dataset-specific meta-path based on prior knowledge from domain experts [7], or propose (general or task-specific) strategies to combine a set of predefined meta-paths (e.g., meta-paths shorter than a predefined length) [5, 8, 11, 21]. However, as the number of possible meta-paths can be large (exponentially growing with their length [11]) even for a heterogeneous graph with a few domains [5], finding an optimal combination of shorter meta-paths is quite problematic in practice.

In this paper, by revisiting existing meta-path-based heterogeneous graph embedding methods, we tackle the following question: *"Are meta-paths really necessary for heterogeneous graph embeddings?"*

To answer this question, we start by reviewing the motivation for using meta-paths in heterogeneous graph embeddings. More precisely, the initial motivation of using meta-paths in heterogeneous graph embeddings lies in the fact that random walks on heterogeneous graphs are biased to highly visible types (domains) of nodes, where nodes are associated with a dominant number of paths [7]. In other words, the node distribution from the random walk sequences are skewed towards these highly visible domains. Subsequently, the learnt node embeddings are also biased to these domains in the sense that they mostly preserve the node proximity in these domains. In such a context, meta-paths are introduced to guide random walks to overcome this problem [7]. However, we argue that this issue can also be solved without using cumbersome meta-paths. Specifically, when performing random walks over a heterogeneous graph, we choose the next node either by *jumping* to one of the other data domains, or *staying* in the same data domain. The key idea of our solution is to probabilistically balance these two options, in order to avoid the above-mentioned bias.

Based on the above intuition, we propose in this paper JUST, a heterogeneous graph embedding technique using random walks with JUmp and STay strategies, rather than involving meta-paths. More precisely, when performing random walks in a heterogeneous graph, our method chooses the next node by probabilistically considering the following two steps: 1) jump or stay, 2) if jump, decide where to jump. First, by controlling the probability of staying using an exponential decay function, we are able to not only balance the number of heterogeneous and homogeneous edges in random walks, but to also effectively avoid staying for too long in one domain. Second, when choosing where to jump, our method memorizes the last m visited domains and tries to jump to a different one; by tuning the number of memorized domains m, we are then able to balance the node distribution over different domains.

Using three different heterogeneous graph datasets, we conduct a thorough empirical evaluation of our method on both node classification (supervised) and clustering (unsupervised) tasks. Empirical results show the superiority of our technique compared to state-of-the-art heterogeneous graph embedding methods. In particular, compared to a state-of-the-art heterogeneous graph embedding technique Hin2vec [8], which tries to optimally combine all meta-paths shorter than a predefined length, our technique yields higher results in most experiments, with a dramatically reduced embedding learning time (about 3x speedup).

2 RELATED WORK

Existing graph embedding methods can be classified into two broad categories according to the types of input graphs, i.e., homogeneous or heterogeneous graphs. On one hand, for a homogeneous graph where all nodes are from a single domain only (e.g., a social network of users), the corresponding graph embedding techniques try to either factorize a node proximity matrix obtained from the adjacency matrix of the input graph using singular vector decomposition [3, 17], or perform random walks [16] over the input graph which are then fed into a SkipGram model [14] to output the node embeddings [9, 18]. The SkipGram model learns node embeddings by maximizing the co-occurrence probability of nodes appearing within a certain context window in a random walk sequence. Due to the intrinsic scalability limitation of matrix factorization techniques, random walks combined with a SkipGram-like model are widely adopted in practice. On the other hand, heterogeneous graphs exhibit a more complex structure, with nodes coming from different domains and heterogeneous as well as homogeneous edges. Recent work [5, 7, 8, 11, 21] has shown that ignoring such structures usually results in low-quality node embeddings, leading to degraded performance in downstream graph analysis tasks. Therefore, considering the structure of a heterogeneous graph is suggested to learn the corresponding node embeddings.

To incorporate key structural properties of a heterogeneous graph into graph embeddings, existing techniques mostly rely on *meta-paths*. Specifically, these techniques first use meta-paths to guide random walks over an input heterogeneous graph, and then feed these guided random walks to a SkipGram-like model to output node embeddings. The meta-paths redefine the neighborhood of a node as a random sequence. To select appropriate meta-paths from an input graph, prior knowledge or heuristics are often required. For example, Metapath2vec [7] manually selects "A-P-A" (representing co-authorship relationship) and "A-P-V-P-A" (representing two papers published by two authors in the same venue) as meta-paths for a bibliographic graph, based on the suggestion from previous work using the same dataset. However, such a meta-path selection method is usually graph-specific (i.e., the selected meta-paths are only suitable for the bibliographic graph in that case) and highly rely on knowledge from domain experts (requiring empirical results from previous work on the same graph). More important, existing work has also shown that the choice of meta-paths highly affect the quality of the learnt node embeddings for different graph analysis tasks [21], which makes the meta-path selection process even more critical.

In order to improve the robustness of graph embedding methods, some techniques combine a set of predefined meta-paths to take advantages of different meta-paths. In the current literature, a set of predefined meta-paths are either task-specific [5, 20, 21], or selected according to a predefined criterion. First, task-specific meta-paths are often selected based on heuristics for performing a specific graph analysis task. For example, Shi et al. [21] proposed the heterogeneous graph embedding approach HERec for learning user/item embeddings for a recommendation task, where a set of predefined meta-paths are used to guide random walks to generate node embeddings, which are then merged together for item recommendation. Chen et al. [5] proposed a path-augmented heterogeneous graph embedding method for an author identification task under blind review settings; it relies on task guidance to select the meta-paths, and incrementally combines them in the learning process. However, these techniques still require prior knowledge on the targeted task to select meta-paths. Second, a set of predefined meta-paths can also be selected according to a certain criterion. Most of the techniques following this idea consider the length of meta-paths as a criterion and set a threshold to keep only short meta-paths. For example, HIN2Vec [8] exploits different types of relationships among nodes by combining meta-paths shorter than a certain length to guide random walks. HINE [11] learns node embeddings in a heterogeneous graph to preserve node proximity from random walks guided by meta-paths shorter than a certain length. Although these techniques are designed to optimally combine the advantages of different meta-paths, they still require setting a meta-path length as a parameter. The quality of the learnt node embeddings have been shown to be sensitive to this parameter [11]. In this paper, by revisiting existing meta-path based graph embedding techniques for heterogeneous graphs, we propose an alternative solution that does not use any meta-path. Our technique can not only balance between homogeneous edges (linking nodes in the same domain) and heterogeneous edges (linking nodes across different domain), it also balances the random walk node distribution over different domains.

Finally, we note that there are other techniques for heterogeneous graph embeddings that do not use meta-paths, such as techniques using deep architecture [4], hyperedges [10] or techniques designed for a specific heterogeneous graphs (e.g., coupled heterogeneous graphs [26]). These techniques do not use random walks to sample an input graph, which differs from the focus of this paper (we advocate random-walk based techniques due to their intrinsic scalability advantages). Embedding learning time of random walk based approaches linearly increases w.r.t. number of nodes in the input graph [9].

3 PRELIMINARIES

In this section, we introduce the key concepts and notations frequently used in this paper.

Definition 3.1. ***(Heterogeneous Graph)*** A heterogeneous graph is denoted as $G = (V, E)$, where V and E refer to the set of nodes and edges (either directed or undirected), respectively. Each node $v \in V$ is mapped to a specific data domain using a mapping function $\phi(\cdot)$, i.e., $\phi(v) = q$, where $q \in Q$ and Q denotes the set of domains in graph G. For a heterogeneous graph G, we always have $|Q| > 1$ as it contains more than one node domains.

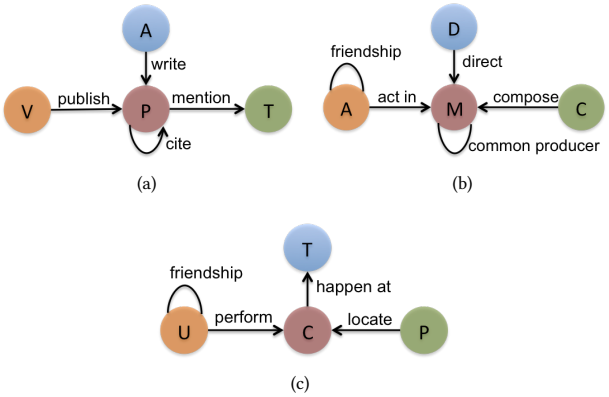

Figure 1: Structures of three heterogeneous graphs: a) a DBLP graph with four data domains: Author (A), Paper (P), Venue (V) and Topic (T); b) a movie graph with four data domains: Director (D), Movie (M), Actor (A) and Composer (C); c) a Foursquare graph with four data domains: User (U), Point of interest (P), Check-in (C) and Time slot (T).

Figure 1 shows the structures of three heterogeneous graphs, i.e., DBLP[1], Movie and Foursquare[2]. In the following, we further define both heterogeneous and homogeneous edges:

Definition 3.2. ***(Heterogeneous Edge)*** An edge $e_{he} = (v_s, v_t) \in E$ is a heterogeneous edge i.i.f. the two nodes it is connected to are from two different domains, i.e., $\phi(v_s) \neq \phi(v_t)$. We use E_{he} to denote the set of heterogeneous edges in G.

Definition 3.3. ***(Homogeneous Edge)*** An edge $e_{ho} = (v_s, v_t) \in E$ is a homogeneous edge i.i.f. the two nodes connected to it are from the same domain, i.e., $\phi(v_s) = \phi(v_t)$. We use E_{ho} to denote the set of homogeneous edges in G.

Taking the DBLP graph as an example, edges linking authors and papers (A-P) are heterogeneous edges, while edges linking pairs of papers (P-P) are homogeneous edges.

4 HETEROGENEOUS GRAPH EMBEDDING WITH JUMP & STAY

In this section, we introduce our method JUST, a heterogeneous graph embedding method using random walks with JUmp and STay on a heterogeneous graph rather than involving meta-paths. Specifically, we advocate the graph embedding paradigm combining random walks with a SkipGram-like model, which has been widely adopted by existing work and shows state-of-the-art performance on different tasks [5, 7, 8, 11, 21]. Our proposed method first performs a random walk over an input heterogeneous graph by probabilistically balancing the jump and stay options when picking its next node, and then feeds these random walks into a SkipGram model for outputting the node embeddings. In the following, we

[1] https://dblp.uni-trier.de/
[2] https://foursquare.com/

first present our proposed random walk strategy, followed by the node embedding learning process using SkipGram.

4.1 Random Walk with Jump & Stay

When performing a random walk on a homogeneous graph, the next node is selected by uniformly sampling a node from the neighbors of the current node. In contrast, for a heterogeneous graph, there are two options to select the next node; from the current node v_i, the next node v_{i+1} is selected by one of the following options:

- **Jump** to a target domain q: uniformly sampling one node from those in a target domain q connected to v_i via heterogeneous edges. The candidate set of nodes for jumping from v_i to the target domain q is denoted as $V^q_{jump}(v_i) = \{v|(v_i, v) \in E_{he} \vee (v, v_i) \in E_{he}, \phi(v) = q\}$.
- **Stay** in the current domain: uniformly sampling one node from those connected to v_i via homogeneous edges. The corresponding candidate set of nodes is denoted as $V_{stay}(v_i) = \{v|(v_i, v) \in E_{ho} \vee (v, v_i) \in E_{ho}\}$.

We highlight here that in heterogeneous graph embedding problems, the direction of edges is kept only for keeping their semantic meaning, but not used for constraining the random walks. In other words, a random walk can either follow the direction of a directed edge, or follow the edge in the opposite direction [5, 7, 8, 11, 21].

Based on these two options, we probabilistically control the random walk using two steps: 1) we decide whether to jump or stay; 2) in case of a jump decision, we then decide where to jump.

4.1.1 Jump or Stay? Based on the current node v_i, we choose to stay with the following probability (and we jump otherwise):

$$Pr_{stay}(v_i) = \begin{cases} 0, \text{ if } V_{stay}(v_i) = \emptyset \\ 1, \text{ if } \{V^q_{jump}(v_i)|q \in Q, q \neq \phi(v_i)\} = \emptyset \\ \alpha^l, \text{ otherwise} \end{cases} \quad (1)$$

where $\alpha \in [0, 1]$ is an initial stay probability, and l refers to the number of nodes consecutively visited in the same domain of the current node v_i. First, in case no homogeneous edge is connecting to v_i, i.e., $V_{stay}(v_i) = \emptyset$, we can only jump to another domain. Second, in case no heterogeneous edge is connecting to v_i, i.e., $\{V^q_{jump}(v_i)|q \in Q, q \neq \phi(v_i)\} = \emptyset$, we can only stay in the same domain. Finally, in case both heterogeneous and homogeneous edges are connecting to v_i, we probabilistically control jump/stay options by choosing to stay with a probability α^l, and jump otherwise. Here, we adopt an exponential decay function on this probability to penalize the cases were the walk stays in one domain for too long, as the stay probability Pr_{stay} decreases exponentially with l. Figure 2 shows an example on the DBLP graph where $l = 3$. Moreover, the initial stay probability α controls how fast Pr_{stay} decreases with l.

In summary, by tuning α, we are able to not only balance the number of heterogeneous and homogeneous edges we follow during random walks, but also to effectively avoid staying too long in one domain. A smaller value of α implies less homogeneous edges (staying less) and more heterogeneous edges (jumping more) in the resulting random walks.

4.1.2 Where to Jump? In case of a jump, given the current node v_i, a target domain q needs to be selected before performing the jump. Specifically, we try to select q by uniformly and randomly

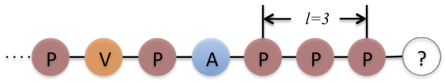

Figure 2: An example of a jump/stay probability on the DBLP graph. As the current node is from the P domain, the number of nodes consecutively visited in P domain is $l = 3$.

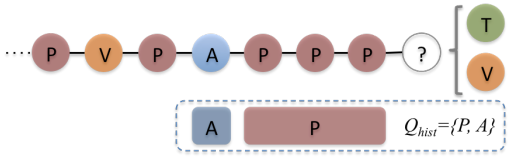

Figure 3: An example of selecting where to jump on the DBLP graph considering $m = 2$ previous visited domains. As $Q_{hist} = \{P, A\}$, we then randomly sample one domain from $\{T and V\}$ as a target domain, where the next node is sampled.

sampling one domain from those differing from the last m visited domains in a random walk, in order to balance the node distribution over different domains. To achieve this goal, we define a fixed-length queue Q_{hist} of size m to memorize up-to-m previously visited domains (including the current domain $\phi(v_i)$) in a random walk. Figure 3 illustrates an example with $m = 2$. When selecting a target domain to jump, we uniformly sample one domain from the following set:

$$Q_{Jump}(v_i) = \begin{cases} \{q|q \in Q \wedge q \notin Q_{hist}, V^q_{jump}(v_i) \neq \emptyset\}, \text{ if not empty} \\ \{q|q \in Q, q \neq \phi(v_i), V^q_{jump}(v_i) \neq \emptyset\}, \text{ otherwise} \end{cases} \quad (2)$$

First, we try to uniformly sample a target domain from a set of candidates satisfying the following three conditions: 1) $q \in Q$; 2) q is not one of the last m visited domains, i.e., $q \notin Q_{hist}$ and 3) there exists heterogeneous edges connecting nodes from q to v_i, i.e., $V^q_{jump}(v_i) \neq \emptyset$. If the candidate set is empty, we still need to choose one target domain to continue a random walk. Therefore, we relax the second condition by simply choosing a target domain which is different from the current domain, i.e., $q \neq \phi(v_i)$.

After selecting a target domain q, we continue the random walk with a jump, i.e, by uniformly sampling one node as v_{i+1} from $V^q_{jump}(v_i)$. Afterwards, Q_{hist} is updated to keep only the last m visited domains. Specifically, as a fixed-length queue, Q_{hist} is updated in a first-in-first-out manner by first adding q, and then dropping the oldest domain if $|Q_{hist}| > m$.

In summary, by tuning the number of memorized domains m, we are able to balance the node distribution over different domains. A small value of m introduces more "randomness" in choosing a target domain to jump into. Larger values of m, however, might lead to cases where we cannot find any candidate domain satisfying the aforementioned three conditions, such as we have to relax the second condition. For example, in the extreme case $m \geq |Q|$ where we never find any such candidate domain, we always sample a target domain differing from the current one, which is equivalent to the case $m = 1$. Therefore, m should be set to an integer in the range $[1, |Q| - 1]$.

Algorithm 1 Truncated Random Walk with Jump & Stay

Require: A heterogeneous graph $G = (V, E)$, initial stay probability α, number of memorized domains m, number of random walks per node r, maximum walk length L_{max}

```
Initialize an empty set of walks W = ∅
for i = 1 to r do
    for each v ∈ V do
        Initialize a random walk by adding v;
        Initialize Q_hist by adding φ(v);
        while |W| < L_max do
            Pick a Jump or Stay decision according to Eq. 1;
            if Stay then
                Continue W by staying;
            else if Jump then
                Sample a target domain q from Eq. 2;
                Continue W by jumping to domain q;
                Update Q_hist by keeping only last m domains;
            end if
        end while
        Add W to W
    end for
end for
return The set of random walks W
```

4.1.3 Truncated Random Walk with Jump & Stay. We perform truncated random walks with the above-mentioned jump and stay strategies to sample from an input heterogeneous graph. More precisely, for each node $v \in V$, we initialize a random walk W starting from v_i, until the maximum length L_{max} is reached. Similar to existing random walk based graph embedding techniques [7–9, 11, 18, 21], we generate a set of random walks $\mathcal{W}$ by performing r random walks rooted on each node $v \in V$. Algorithm 1 illustrates our random walk process to generate $\mathcal{W}$.

4.2 Node Embedding Learning with SkipGram

Based on the generated set of random walks $\mathcal{W}$, we adopt a SkipGram model to learn the node embeddings [14]. Specifically, the SkipGram model maximizes the co-occurrence probability of two nodes appearing within a context window of length k in a random walk. Formally, for a pair of nodes v_i and v_j appearing in the context window in the set of random walks $\mathcal{W}$, this co-occurrence probability is defined as:

$$Pr((v_i, v_j) \in \mathcal{W}) = \sigma(\vec{v}_i \cdot \vec{v}_j) \tag{3}$$

where $\sigma(\cdot)$ is the sigmoid function, and $\vec{v}_i$ and $\vec{v}_j$ refer to the embeddings (vectors) of v_i and v_j, respectively. Moreover, the SkipGram model usually adopts negative sampling techniques [15] to also maximize the probability of node v_i and a randomly sampled negative node v_N not appearing in the set of random walks $\mathcal{W}$:

$$Pr((v_i, v_N) \notin \mathcal{W}) = 1 - Pr((v_i, v_N) \in \mathcal{W}) = \sigma(-\vec{v}_i \cdot \vec{v}_N) \tag{4}$$

where negative samples (nodes) are often uniformly drawn according to the empirical node distribution in the random walks $\mathcal{W}$. In summary, for the pair of nodes (v_i, v_j), the SkipGram model maximize the following objective function:

$$\log \sigma(\vec{v}_i \cdot \vec{v}_j) + \gamma \cdot \mathbb{E}_{v_N}[\log \sigma(-\vec{v}_i \cdot \vec{v}_N)] \tag{5}$$

where $\gamma \in \mathbb{Z}^+$ refers to the number of negative samples. By iterating over all node pairs appearing within a context window of length k in each random walk in $\mathcal{W}$, the node embeddings can be efficiently learnt by using Asynchronous Stochastic Gradient Descent (ASGD) [19] in parallel.

5 EXPERIMENTAL EVALUATION

In this section, we conduct a thorough empirical evaluation of our method. We first present our experiment setup below, before evaluating our technique on both node classification (supervised) and clustering (unsupervised) tasks. Then, we investigate the impact of two key parameters (the initial stay probability α and the number of memorized domains m) on the quality of the learnt node embeddings, followed by a parameter sensitivity study on other parameters and a study on runtime performance.

5.1 Experiment Setup

We present our experiment setup including datasets, baseline methods and settings for our proposed method.

5.1.1 Datasets. We use three real world heterogeneous graphs for evaluation. Table 1 summarizes the main statistics of the three graphs.

- **DBLP**: We use a DBLP graph provided by [11]. It contains nodes in four domains, including 5,915 authors (A), 5,237 papers (P), 18 venues (V) and 4,479 topics (T), where heterogeneous edges (A-P, P-V and P-T) and homogeneous edges (P-P) are available. In addition, each author is associated with one of the following four labels "database", "data mining","machine learning" and "information retrieval". An illustration of DBLP's schema is available above in Figure 1(a).
- **Movie**: We use an augmented version of the MOVIE graph provided by [11]. It contains nodes in four domains, including 10,789 actors (A), 7,332 movies (M), 1,741 directors (D) and 1,483 composers (C), where heterogeneous edges (including A-M, D-M and C-M) and homogeneous edges (A-A and M-M) are available. As the original dataset does not contain homogeneous edges, we enrich this graph by adding homogeneous edges between actors. More precisely, by crawling the social relationships of the involved actors on Twitter, we add homogeneous edges (friendships) between actors if they follow each other on Twitter. Meanwhile, pairs of movies are also connected by homogeneous edges if they share the same producer. In addition, each movie is associated with one or more of the following five labels (genres) "action", "horror", "adventure", "scifi" and "crime". An illustration of this schema is given in Figure 1(b).
- **Foursquare**: We use a check-in graph collected from Foursquare in New York City provided by [27, 28], which has been widely used in enabling location based services [29, 30]. It contains nodes from four domains, including 2,449 users (U), 25,904 check-ins (C), 1,250 points of interest (P) and 168 time stamps (T), where heterogeneous edges (U-C, P-C and T-C) and homogeneous edges (U-U) are available. In addition, each point of interest is associated with one of ten categories (e.g., "bar").

Table 1: Characteristics of the experimental graphs

Dataset	DBLP	Movie	Foursquare
$\|V\|$	15,649	21,345	29,771
$\|E\|$	51,377	89,038	83,407
$\|E_{he}\|$	44,379	34,354	77,712
$\|E_{ho}\|$	6,998	54,684	5,695
#Labels	4	5	10

The schema of the Foursquare graph is previously shown in Figure 1(c).

5.1.2 Baselines. We compare our technique with the following state-of-the-art heterogeneous graph embedding methods.

- ***DeepWalk*** [18] learns node embeddings by first performing classical random walks on an input graph, and then feeding the generated random walks to a SkipGram model. We set the number of walks per node $r = 10$, the maximum walk length $L_{max} = 100$, and the window size for the SkipGram model $k = 10$.
- ***LINE*** [25] directly samples node pairs from an input graph to learn node embeddings by explicitly preserving the 1st- and 2nd-order node proximities only. To learn node embeddings of dimension d, it first separately learns two sets of $d/2$-dimension node embeddings according to 1st- and 2nd-order node proximity, respectively, and then concatenates them together.
- ***PTE*** [24] is a semi-supervised model for learning text embeddings using both labeled and unlabeled data. In our experiments, similar to [7, 8], we use PTE in an unsupervised way. Specifically, for a heterogeneous graph, we extract one bipartite graph between the domain of interest and each of the other domains, and then feed these graphs to PTE to output the node embeddings for the domain of interest. Subsequently, we construct three bipartite heterogeneous graphs for each dataset as follows: DBLP (A-P, A-V, A-T), Movie (M-A, M-D, M-C) and Foursquare (P-C, P-U, P-T).
- ***Metapath2vec*** [7] first generates meta-path guided random walks based on only one given meta-path, then feeds them to a SkipGram model. For each dataset, we conduct experiments with Metapath2vec using different meta-paths, and report the best performing one. For DBLP graph, we use the meta-paths suggested by the authors: "A-P-V-P-A" and "A-P-A". We also tried the following meta-paths: "A-P-P-V-P-P-A", "A-P-P-T-P-P-A" and "A-P-P-V-P-T-P-A" suggested by [13]. For MOVIE dataset, we use "A-M-D-M-A" and "A-M-C-M-A" which represent actors starring in movies with common directors and composers respectively. For Foursquare dataset, we use "U-C-P-C-U" which represents different users checking-in at the same point of interest, and "P-C-T-C-P" which represents different points of interest having check-ins at the same time stamp. We keep the same parameters r, L_{max}, k as for DeepWalk.
- ***Hin2vec*** [8] combines a set of meta-paths shorter than a certain length to perform meta-path guided random walks, in order to jointly learn both node embeddings and meta-path embeddings. The node embeddings are learnt such that they can be effectively used to predict the meta-paths which actually connects them. We tune the maximum length of meta-paths from 3 to 6 and report the best performing one. We keep other parameters r, L_{max}, k same as for DeepWalk.
- ***JUST_no_memory*** is a simplified version of our method, which does not memorize any of the previously visited nodes or domains. First, when choosing to jump or stay, we use a fixed stay probability Pr_{stay} (by setting $l = 1$), no matter how long a random walk stays in the same domain. Second, when choosing where to jump, we consider only the current domain (by setting $m = 1$), and uniformly sample a target domain from those differing from the current one. We keep the same parameters r, L_{max}, k as for DeepWalk.

Note that among these baselines, DeepWalk and LINE are originally designed for homogeneous graphs; they are applied to a heterogeneous graph by ignoring the schema of the graph and treating all nodes and edges equally as for a homogeneous graph.

For our method JUST, we configure it with three settings according to the number of memorized domains $m = 1, 2, 3$ (we have maximum four domains in our datasets), and tune the initial stay parameter α within $[0.1, 0.9]$ with a step of 0.1. By tuning our parameters m and *alpha*, we report the best performance on both classification and clustering tasks (we investigate the impact of these parameters later in Sections 5.4 and 5.5). We also keep the same random walk parameters r, L_{max}, k as for DeepWalk. The dimension of the node embeddings d is set to 128 and the number of negative samples γ is set to 10 for all methods in all experiments, if not specified otherwise. In all experiments, we tune the parameters for each method, to let it achieve its best performance.

5.2 Node Classification Task

The objective of node classification task is to predict the most probable label(s) for some nodes based on other labeled nodes. We focus on the author domain in the DBLP graph, the movie domain in the Movie graph, and the point of interest domain in the Foursquare graph. To implement this task, we use the same settings as in [7, 8, 18]. We randomly sample a set of nodes as labeled nodes for training, and use the rest for testing. Based on the embeddings of the training nodes, we train a one-vs-rest logistic regression classifier to predict the most probable labels for test nodes, and compare the prediction against their ground truth labels. We report the average Macro-F1 and Micro-F1 scores from 10 repeated trials.

Figure 4 shows our results on the node classification task. First, we clearly observe that more training data results in higher performance in general. Second, we see that our method outperforms baselines in most of cases. For the DBLP graph, JUST significantly outperforms all baselines. For the Movie graph, we observe that JUST/JUST_no_memory and Hin2vec show comparable performance. However, compared to Hin2vec which combines all meta-paths shorter than a certain length, JUST without involving meta-paths can learn node embeddings much more efficiently. For example, the end-to-end learning time is 442 seconds using JUST on the Movie graph, while it takes 1,301 seconds using Hin2vec (more discussion on this point in Section 5.7 below). For the Foursquare

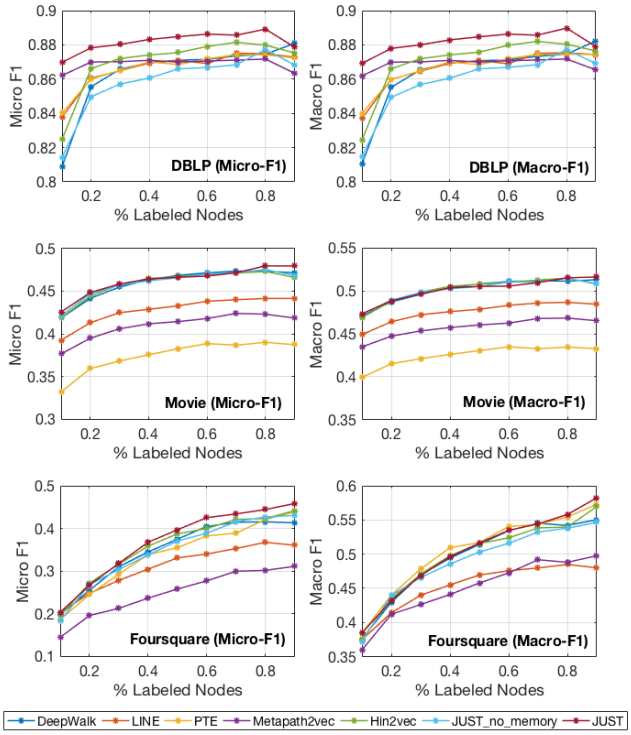

Figure 4: Performance on node classification task

graph, we find that although our method shows similar Macro-F1 score with PTE and Hin2vec, it consistently achieves the highest Micro-F1 compared to all baselines.

5.3 Node Clustering Task

The objective of community detection is to regroup similar nodes into clusters. Same as in the node classification task, we also focus on the domain with labeled nodes in each graph, and regard nodes with the same label as a ground truth community. To implement this task, we adopt similar settings as in [7, 11]. Specifically, we cluster nodes based on their embeddings using the k-means algorithm [1], and evaluate the clusters using Normalized Mutual Information (NMI). We note that while the output of k-means contains non-overlapping communities (i.e., each node belongs to one cluster only), the ground truth communities in our Movie graph are overlapping (i.e., one movie may belong to one or more genre). To evaluate the clusters in this case, we compute a generalized NMI using the method proposed by [12], which can measure NMI between partitions (i.e., non-overlapping communities) and covers (i.e., overlapping communities). As a large number of communities often results in very small NMI values (which makes it hard for comparing different methods), we thus select only the top two largest communities and the corresponding nodes in individual graphs in this experiment. We report the average NMI from 10 repeated trials.

Figure 5 shows the results of the node clustering task. We clearly observe that our proposed method JUST (even its simplified variation JUST_no_memory) consistently outperforms all other baselines across different datasets.

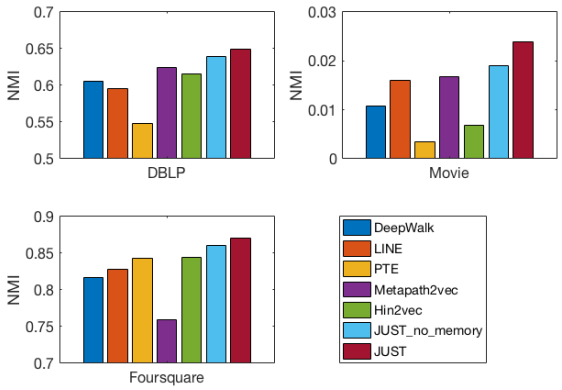

Figure 5: Performance on node clustering task

Interestingly, comparing the performance of Metapath2vec and Hin2vec, we observe very different results across three datasets. Specifically, Metapath2vec achieves slightly higher, much higher and much lower NMI than Hin2vec on the DBLP, Movie and Foursquare graphs, respectively. This observation shows the limitations of the meta-path based heterogeneous graph embedding techniques. First, for the techniques which require a predefined meta-path (like Metapath2vec), the selection of the meta-path is indeed complex, and the quality of the learnt node embeddings (in downstream graph analysis tasks) is sensitive to the selection of the meta-path. Second, the techniques combining different meta-paths (like Hin2vec) may not consistently outperform the techniques with a manually selected meta-path (Metapath2vec). In other words, the meta-path combination strategies are also complex, as they not only take advantages of different meta-paths, but also might introduce additional noise from suboptimal meta-paths.

5.4 Impact of the Initial StayProbability α

We investigate the impact of the initial stay probability α on the quality of the learnt node embedding. It balances the impact of heterogeneous and homogeneous edges on the learnt node embeddings. A smaller value of α leads to less homogeneous edges (staying less), and more heterogeneous edges in the resulting random walks. In this experiment, by keeping other parameters to their optimal settings, we tune α within $[0.1, 0.9]$ with a step of 0.1, and report the corresponding performance on both node classification and clustering.

Figure 6 shows the results. We clearly observe suboptimal results when considering either too many heterogeneous or homogeneous edges. In other words, balancing the number heterogeneous and homogeneous edges is important to learn high-quality node embeddings in a heterogeneous graph. We find that the optimal α mostly lies in the range $[0.2, 0.4]$ on all three datasets in both node classification and clustering tasks, which suggests that heterogeneous edges linking nodes across domains indeed play the primary role in learning node embeddings for heterogeneous graphs.

Moreover, to further verify the trade-off between heterogeneous and homogeneous edges, we investigate the percentage of heterogeneous/homogeneous edges in random walks. Figure 7 shows the results on all three datasets. We observe that a small value of α

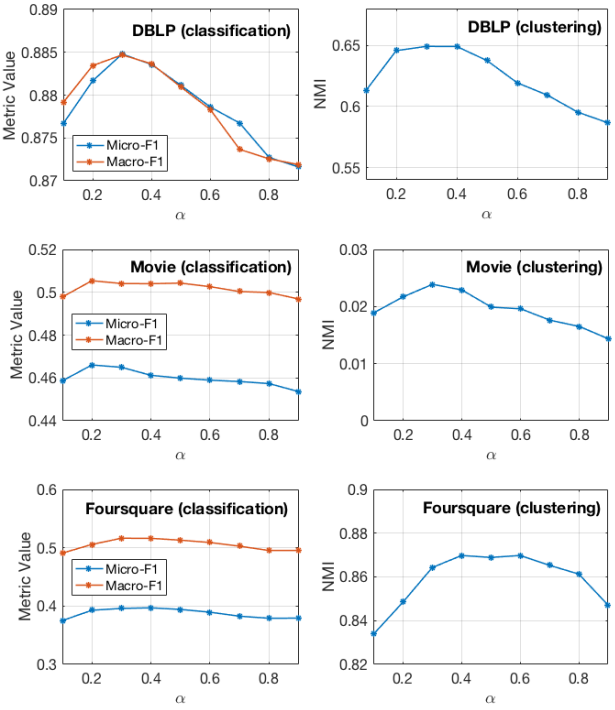

Figure 6: Impact of the initial stay probability α

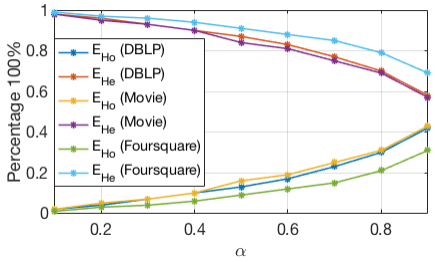

Figure 7: Trade-off between heterogeneous and homogeneous edges using α

leads to a large portion of heterogeneous edges in the resulting random walks. When increasing the initial stay probability α to "stay more", the portion of heterogeneous edges clearly decreases. We also note that our technique is robust to the absolute number of homogeneous/heterogeneous edges, as we actually control the stay/jump probability before choosing the next node in a random walk. For example, the portion of heterogeneous edges varies in a similar way for the DBLP and Movie datasets as shown in Figure 7, while we observe much more homogeneous edges in the Movie graph than in the DBLP graph as shown in Table 1.

5.5 Impact of the Number of Memorized Domains m

We investigate the impact of the number of memorized domains m on the quality of the learnt node embedding. It balances the node distribution over different domains. When tuning $m \in [1, |Q| - 1]$, a smaller value of m gives more "randomness" in choosing a target

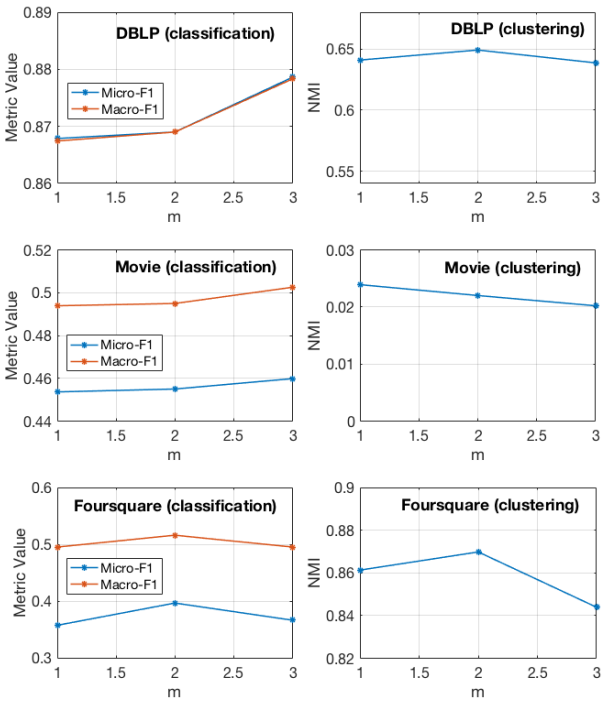

Figure 8: Impact of the number of memorized domains m

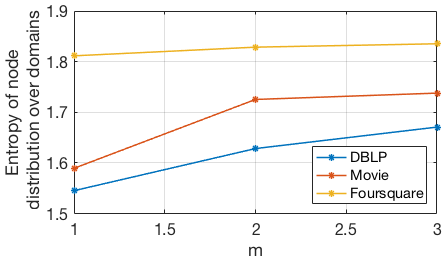

Figure 9: Balancing node distribution over different domains using m.

domain to jump to in general. In this experiment, by keeping other parameters to their optimal settings, we tune m within $\{1, 2, 3\}$, and report the corresponding performance on both node classification and clustering.

Figure 8 shows the results. First, we observe that compared to the initial stay probability α, the performances on both tasks are less sensitive to m. Second, we find that the optimal m varies across different datasets and tasks. Moreover, we are interested in how m balances the node distribution over different domains in random walks. We compute the entropy of the node distribution for different m. As shown in Figure 9, we observe that the entropy increases when increasing m. In other words, larger values of m indeed increase the evenness of the node distribution over domains. Note that we have four domains for our datasets[3], meaning the maximum entropy is 2.

[3]We coincidentally have four domains for all our datasets, which is not a design choice. In the future, we also plan to evaluate our technique on large heterogeneous graphs having vastly more complex schemas with more domains.

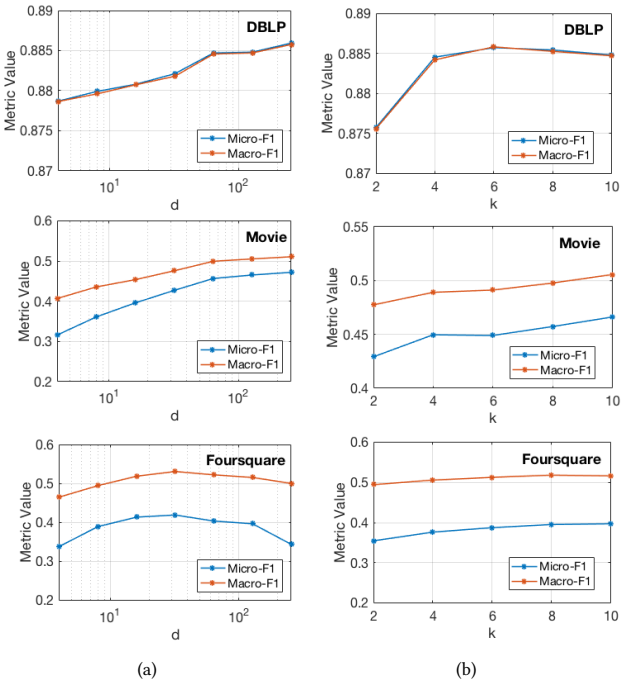

Figure 10: Impact of other parameters in node classification task: a) node embedding dimension d in the left column; and b) context window size k in the right column

5.6 Other Parameter Sensitivity Study

In this section, we investigate the impact of the node embedding size d and the context window size k on the quality of the learnt node embeddings.

5.6.1 Node embedding dimension d. By fixing other parameters to their optimal values, we study the impact of the node embedding dimension d by increasing it from 4 to 256. Figure 10(a) shows the results for node classification (similar results are observed for the clustering task). First, we observe that a small d yields low performance, as the corresponding node embeddings are not able to capture sufficient information from the graph. From there, the performance increases with increasing d. The performance further flattens out after a certain point, or even decreases (in the case of the Foursquare dataset) as the resulting node embeddings start to overfit the input graph. We select $d = 128$ in all previous experiments, as it shows good performance on all three datasets.

5.6.2 Context window size k. By fixing other parameters, we study the impact of the context window k by increasing it from 2 to 10 with a step of 2. Figure 10(b) shows the results for the node classification task (similar results are observed for the node clustering task). We observe that a larger context window size usually leads to better performance, as it is able to capture higher-order node proximity. We also observe that the performance flattens out after a certain point, and we empirically select $k = 10$ in all previous experiments.

Table 2: End-to-end node embedding learning time (in seconds)

	DBLP	Movie	Foursquare
DeepWalk	236	333	484
Metapath2vec (original)	965	19,200	2,248
Metapath2vec (ours)	290	408	550
Hin2vec	904	1,301	1,801
JUST	310	442	616

5.7 Runtime Performance

In order to evaluate the runtime performance of our proposed method, we investigate the end-to-end node embedding learning time (from an input graph to the final node embeddings) using our test PC[4], for all random-walk based methods from our baselines (including DeepWalk, Metapath2vec, Hin2vec) and our method JUST. In particular, we note that the original implementation of Metapath2vec released by the authors is highly customized to the DBLP graph (the only dataset used in the paper), and also takes long to run even on a small graph. Therefore, we implemented our own version of Metapath2vec (ours) using the efficient Gensim library[5] for the SkipGram training process.

Table 2 shows the results. First, we observe that DeepWalk shows the shortest learning time, as it simply combines classical random walks and a SkipGram model. Second, Metapath2vec (*ours*) requires a bit more time than DeepWalk, as it needs to follow a given meta-path to sample the next node in a random walk. In addition, we also find that our implementation of Metapath2vec is much faster than the original implementation. Third, our method JUST requires a bit more time than Metapath2vec (ours), as we probabilistically control random walks by balancing jumping and staying in a random walk. Here we note that the SkipGram training process is the same for DeepWalk, Metapath2vec (ours) and JUST; any runtime overhead is only caused by their random walk strategies. Finally, Hin2vec takes much longer than all other methods, as it tries to combine all meta-paths shorter than a certain length and also output the embeddings for the involved meta-paths.

In summary, compared to Metapath2vec (ours), our proposed technique without requiring prior knowledge on meta-paths is able to achieve higher performance on both node classification and clustering tasks, with a minor overhead on the learning time. Meanwhile, our technique outperforms also Hin2vec in most of the experiments, with a dramatically reduced embedding learning time (about 3x speedup).

6 DISCUSSION

Heterogeneous graphs with complex structures. In this study, we evaluate our proposed technique using several widely used heterogeneous graphs, which do not involve complex structures. However, real-world heterogeneous graphs often have complex structures, such as a large number of domains, or multi-relational edges (hyperedges simultaneously linking multiple nodes). For example, Knowledge Graphs often contain a large number of node domains (types). Therefore, it is interesting to investigate how our

[4]Intel Core i5-7440EQ@2.90 GHz, 16GB RAM, Mac OS X.
[5]https://radimrehurek.com/gensim/models/word2vec.html

proposed technique without meta-paths performs on those complex heterogeneous graphs.

Automatic discovery of relevant meta-paths. An alternative way without requiring meta-paths as the input of an heterogeneous graph embedding technique is to automatically discover/suggest high-quality meta-paths. For example, Meng et al. [13] suggest several meta-paths for DBLP graphs which are missed by domain experts, with a particular focus on the authorship prediction task. We note that we tried all their suggested meta-paths in our experiments, but none of them leads to good results in our evaluation tasks, as those meta-paths are suggested for a particular task only (i.e., authorship prediction). However, it is still interesting to investigate how to automatically discover general-purpose meta-paths for heterogeneous graph embedding problems.

7 CONCLUSION

Existing heterogeneous graph embedding techniques mostly resort to ad-hoc meta-paths to guide random walks on an input heterogeneous graph for learning node embeddings. However, using these meta-paths either requires prior knowledge from a domain expert for optimal meta-path selection, or requires much computation to combine all meta-paths shorter than a predefined length. In this paper, by revisiting these meta-path based techniques, we investigated the initial motivation of introducing meta-paths in heterogeneous graph embedding problems, and considered whether meta-paths were really necessary in this context. Specifically, as the meta-paths were originally introduced to overcome the bias of classical random walk to the highly visible domains, we proposed an alternative to this issue without involving any meta-path. More precisely, we proposed JUST, a heterogeneous graph embedding technique using random walks with jump and stay strategies to overcome the aforementioned bias in a more efficient manner. It can not only balance between homogeneous edges (linking nodes in the same domain) and heterogeneous edges (linking nodes across different domain), it also balances the node distribution over different domains. To validate our technique, we conducted a thorough empirical evaluation on three heterogeneous graph datasets on both node classification and clustering. Empirical results show the superiority of our technique compared to state-of-the-art methods. In particular, compared to Hin2vec which tries to optimally combine all meta-paths shorter than a predefined length, our technique yields higher performance in most experiments, while dramatically reducing the embedding learning time (3x speedup).

In the future, we plan to evaluate our technique on large heterogeneous graphs having vastly more complex schemas, such as RDF graphs or knowledge graphs.

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 683253/GraphInt).

REFERENCES

[1] David Arthur and Sergei Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In *SIAM*. Society for Industrial and Applied Mathematics, 1027–1035.
[2] Hongyun Cai, Vincent W Zheng, and Kevin Chang. 2018. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering* (2018).
[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*. ACM, 891–900.
[4] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *KDD*. ACM, 119–128.
[5] Ting Chen and Yizhou Sun. 2017. Task-guided and path-augmented heterogeneous network embedding for author identification. In *WSDM*. ACM, 295–304.
[6] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2017. A Survey on Network Embedding. *arXiv preprint arXiv:1711.08752* (2017).
[7] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*. ACM, 135–144.
[8] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning. In *KDD*. ACM, 1797–1806.
[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864.
[10] Huan Gui, Jialu Liu, Fangbo Tao, Meng Jiang, Brandon Norick, and Jiawei Han. 2016. Large-scale embedding learning in heterogeneous event data. In *ICDM*. IEEE, 907–912.
[11] Zhipeng Huang and Nikos Mamoulis. 2017. Heterogeneous Information Network Embedding for Meta Path based Proximity. *arXiv preprint arXiv:1701.05291* (2017).
[12] Andrea Lancichinetti, Santo Fortunato, and János Kertész. 2009. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics* 11, 3 (2009), 033015.
[13] Changping Meng, Reynold Cheng, Silviu Maniu, Pierre Senellart, and Wangda Zhang. 2015. Discovering meta-paths in large heterogeneous information networks. In *WWW*. International World Wide Web Conferences Steering Committee, 754–764.
[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[15] 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.
[16] Jae Dong Noh and Heiko Rieger. 2004. Random walks on complex networks. *Physical review letters* 92, 11 (2004), 118701.
[17] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *KDD*. ACM, 1105–1114.
[18] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. ACM, 701–710.
[19] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*. 693–701.
[20] Jingbo Shang, Meng Qu, Jialu Liu, Lance M Kaplan, Jiawei Han, and Jian Peng. 2016. Meta-Path Guided Embedding for Similarity Search in Large-Scale Heterogeneous Information Networks. *arXiv preprint arXiv:1610.09769* (2016).
[21] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and Philip S Yu. 2017. Heterogeneous Information Network Embedding for Recommendation. *arXiv preprint arXiv:1711.10730* (2017).
[22] Yizhou Sun and Jiawei Han. 2013. Mining heterogeneous information networks: a structural analysis approach. *Acm Sigkdd Explorations Newsletter* 14, 2 (2013), 20–28.
[23] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4, 11 (2011), 992–1003.
[24] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *KDD*. ACM, 1165–1174.
[25] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. International World Wide Web Conferences Steering Committee, 1067–1077.
[26] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S Yu. 2017. Embedding of embedding (eoe): Joint embedding for coupled heterogeneous networks. In *WSDM*. ACM, 741–749.
[27] Dingqi Yang, Daqing Zhang, Longbiao Chen, and Bingqing Qu. 2015. NationTelescope: Monitoring and visualizing large-scale collective behavior in LBSNs. *Journal of Network and Computer Applications* 55 (2015), 170–180.
[28] Dingqi Yang, Daqing Zhang, and Bingqing Qu. 2016. Participatory cultural mapping based on collective behavior data in location-based social networks. *ACM Transactions on Intelligent Systems and Technology (TIST)* 7, 3 (2016), 30.
[29] Dingqi Yang, Daqing Zhang, Zhiyong Yu, and Zhu Wang. 2013. A sentiment-enhanced personalized location recommendation system. In *HT*. ACM, 119–128.
[30] Dingqi Yang, Daqing Zhang, Vincent W Zheng, and Zhiyong Yu. 2015. Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45, 1 (2015), 129–142.