Interactive Paths Embedding for Semantic Proximity Search on Heterogeneous Graphs

Zemin Liu
Zhejiang University
China
liuzemin@zju.edu.cn

Vincent W. Zheng*
Advanced Digital Sciences Center
Singapore
vincent.zheng@adsc-create.edu.sg

Zhou Zhao
Zhejiang University
China
zhaozhou@zju.edu.cn

Zhao Li, Hongxia Yang
Alibaba Group
China
{lizhao.lz,yang.yhx}@alibaba-inc.com

Minghui Wu
Zhejiang University City College
China
mhwu@zucc.edu.cn

Jing Ying
Zhejiang University
China
yingj@zucc.edu.cn

ABSTRACT

Semantic proximity search on heterogeneous graph is an important task, and is useful for many applications. It aims to measure the proximity between two nodes on a heterogeneous graph w.r.t. some given semantic relation. Prior work often tries to measure the semantic proximity by paths connecting a query object and a target object. Despite the success of such path-based approaches, they often modeled the paths in a weakly coupled manner, which overlooked the rich interactions among paths. In this paper, we introduce a novel concept of interactive paths to model the interdependency among multiple paths between a query object and a target object. We then propose an Interactive Paths Embedding (IPE) model, which learns low-dimensional representations for the resulting interactive-paths structures for proximity estimation. We conduct experiments on seven relations with four different types of heterogeneous graphs, and show that our model outperforms the state-of-the-art baselines.

CCS CONCEPTS

• **Computing methodologies** → **Statistical relational learning**;

KEYWORDS

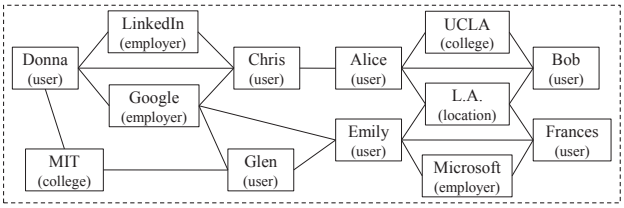
Semantic Proximity Search; Heterogeneous Graph; Interactive Paths Embedding

ACM Reference Format:
Zemin Liu, Vincent W. Zheng, Zhou Zhao, Zhao Li, Hongxia Yang, Minghui Wu, and Jing Ying. 2018. Interactive Paths Embedding for Semantic Proximity Search on Heterogeneous Graphs. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3219953

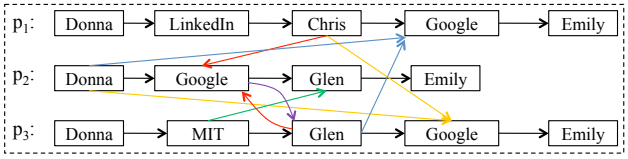
*Corresponding author.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3219953



(a) An example of heterogeneous graph extracted from Facebook.



(b) Interdependence among multiple object paths from Donna to Emily.

Figure 1: Semantic proximity search. Best view in color.

1 INTRODUCTION

Semantic proximity search is an important task on many real-world heterogeneous networks [9, 34]. It takes an object in the network as the query, and ranks the other objects according to a semantic relation. Consider the network in Fig. 1(a). The different ways that how two objects are connected imply different kinds of semantic relations. For example, *Alice* and *Bob* both attend *UCLA*, thus they are *schoolmates*; whereas *Donna* and *Chris* both work for *Google*, thus they are *colleagues*. Given this semantic difference, we can then take any user object (*e.g.*, *Alice*) as a query, and ask "who are likely to be her *schoolmates*?" As the answer, we shall have *Bob* rank higher than the other user objects. Such object ranking empowers many applications; *e.g.*, recommending connections in a professional network such as LinkedIn, categorizing friends in a social network such as Facebook, discovering advisors/advisees in a bibliography network such as DBLP, and linking user identities in an e-commerce network such as Taobao.

Semantic proximity search on heterogeneous graphs is a challenging task, because the semantic relations may not be always explicit (*e.g.*, advisor/advisee relations are not stated in DBLP) and the

heterogeneous network often has missing links among the objects (*e.g.*, users do not often update their affiliations in LinkedIn). Prior work often tries to measure the semantic proximity by **paths** connecting a query object and a target object [1, 14, 32]. Meta-Path Proximity (MPP) counts the number of *meta-path* (*e.g.*, "user-employer-user")[34] instances between the query and the target to measure the proximity w.r.t. certain semantic relation (*e.g.*, *colleague*). Path Ranking Algorithm (PRA) [17] first enumerates bounded-length (relation) path patterns; then it recursively defines a score for each path pattern; finally, for a target object, its proximity to the query object is computed as a linear combination of its corresponding path instances. Recent work starts to exploit learning representations for the paths between a query object and a target object, and then using them for proximity estimation. For example, ProxEmbed [22] first samples a number of paths from the query object to the target object, and then uses a recurrent neural network to embed each path as a vector; finally it aggregates multiple paths' embedding vectors for proximity estimation.

We find that, despite the success of the path-based approach, the paths are often *weakly coupled* in the modeling. That is, these paths are usually modeled separately, and only at the final stage their outputs (*e.g.*, PRA's path score, or ProxEmbed's path embedding) are aggregated for further computation. Such weak coupling hinders the paths from getting a more complete picture of the connecting structure between the query object and the target object. To help concretely understand why the interdependence among paths is useful, let us consider an example in Fig. 1(b). There are three object paths p_1, p_2 and p_3 sampled from the heterogeneous graph in Fig. 1(a). Each path alone captures *partial* semantics of the whole graph; *e.g.*, p_1 is unaware of *Donna* also working for *Google*, whereas p_2 is unaware of *Emily* working for *Google* either. If we are able to consider all the paths in a deeper manner, instead of modeling each independently, we have a better chance to infer that *Donna* and *Emily* are in fact *colleagues*.

In this paper, we introduce a novel concept of **interactive paths** to model the interdependence among multiple paths between a query object and a target object. With this concept, we wish to *strongly couple* the paths by adding dependencies among the objects in different paths. The interdependence among paths is the connections that lost in each single path, but could be extracted from the original graph structure to enrich the interactions among paths. The interactions facilitate the modeling for each independent path, because each path could obtain extra structure information from the interactions with other paths. Consider the example in Fig. 1(b). For *Google* in p_1, we wish to take into account its predecessors in p_2 and p_3, which are *Donna* and *Glen* respectively. Denote an object v in a path p as $(p : v)$. We add *directed* edges from $(p_2 : Donna)$ and from $(p_3 : Glen)$ to $(p_1 : Google)$. In this way, $(p_1 : Google)$ sees a bigger picture of how different users in Fig. 1(b) connect to it. Similarly, we can introduce dependencies for the other objects in the paths. As a result, we obtain a set of directed edges linking the paths in Fig. 1(b); for brevity, we skip the directed edges for *Emily*. We call these linked paths as "interactive paths". Our goal is to embed each of these interactive paths into a low-dimensional vector, such that all these vectors together can well capture the semantic relation between *Donna* and *Emily*.

Embedding interactive paths is non-trivial. Few methods model path interdependencies, and most of them are not designed for modeling more than two paths at a time. For example, Word-to-Word Attention LSTM [30] uses two LSTMs to recognize textual entailment between a premise and a hypothesis; each word in the premise depends on all the other words in the hypothesis, and vice versa. Coupled-LSTM [20] establishes a subsequence-to-subsequence interdependency between two paths, and employs a grid architecture for embedding. It is not clear how to generalize their models to handle multiple paths, since the generalization complexity scales quickly. We will elaborate this in Sect. 2. On the other hand, the interactive paths result in a complex directed graph with possible *cycles*, which presents a challenge of how to learn this structure and output embedding for each path therein. In general, cycles in a graph structure are not desired. For example in Fig. 1(b), there is a cycle between $(p_2 : Google)$ and $(p_3 : Glen)$. It makes the reachability from *Donna* to *Emily* become lower, thus weaker to explain their relation. Besides, cycles are not preferred in inference by probabilistic graphical models [16]. Therefore, we should try to avoid cycles in the interactive paths. But this does not mean that we can remove as many path interdependencies as possible; instead, we should also avoid losing much information about these interdependencies. Once having a cycle-free structure for the interactive paths, we also need to consider what unique task characteristics we can exploit to generate meaningful path embedding.

We propose a novel Interactive Paths Embedding (IPE) model to address the above challenges. In particular,

• We construct the interactive paths with an inference-friendly structure, by a **cycle-free shuffling** mechanism. We are inspired by DAG-LSTM [43] to first introduce all the possible interdependencies among paths, and then apply a highly efficient cycle removal algorithm on the interactive paths. To minimize information loss due to cycle removals, we shuffle the order of paths to remove cycles, so that every time we will remove different cycles. As a result, we generate multiple interactive-paths structures.

• We embed each interactive-paths structure, by an **interactive GRU** mechanism. We use a Gated Recurrent Unit (GRU) architecture [5] to model each path, and allows each GRU to model the interdependencies from the other GRUs. Specifically, we exploit three unique task characteristics for modeling the interdependencies: 1) *path heterogeneity*: for $(p_1 : Google)$, its own path predecessor $(p_1 : Chris)$ may contribute differently with its other path predecessors $(p_2 : Donna)$ and $(p_3 : Glen)$ to p_1's embedding, due to the path semantic differences; 2) *distance awareness*: for $(p_1 : Google)$, its predecessor $(p_2 : Donna)$ leads to a shorter walk from the query to it (*i.e.*, $Donna \rightarrow Google$) than another one $(p_3 : Glen)$ (*i.e.*, $Donna \rightarrow MIT \rightarrow Glen \rightarrow Google$). In practice, shorter paths are more expressive, hence $(p_2 : Donna)$ should contribute more than $(p_3 : Glen)$; 3) *node heterogeneity*: for $(p_1 : Emily)$, its predecessor $(p_1 : Google)$ indicates a working relation, whereas $(p_2 : Glen)$ indicates a friend relation. These two predecessors may contribute differently if the task is to find who is *Donna*'s *colleague*.

Finally, we aggregate all the interactive-paths structure embedding output as a single vector, and use it for proximity estimation w.r.t. a semantic relation. As we shall elaborate more in Sect. 2 and

Sect. 7, our concept of interactive paths and design of IPE are both novel and effective. We summarize our contributions as follows.

• We introduce a novel concept of interactive paths to strongly couple the paths for semantic proximity search.

• We propose a novel IPE model, which uses cycle-free shuffling mechanism to construct the interactive paths, and an interactive GRU mechanism to effectively embed the interactive paths.

• We evaluate IPE on seven semantic relations from four data sets, and show it outperforms the state-of-the-art baselines.

2 RELATED WORK

For semantic proximity search, earlier work such as Personalized PageRank [14] measures the proximity from a query node to a target node by random walk; but it only considers homogeneous graphs. Supervised Random Walk (SRW) [1] can be applied to heterogeneous graph; it tries to bias the random walk starting from the query node, such that relevant nodes w.r.t. a semantic relation are ranked higher than irrelevant ones. Both MPP [34] and PRA [17] count instances of certain path patterns between two objects to measure their proximity. PRep [32] considers path-based relevance from a probabilistic perspective. Recent work exploits graph embedding for semantic proximity search. Usually, graph embedding focuses on generating a low-dimensional vector for each node [3]. Examples include DeepWalk [28], node2vec [12], metapath2vec [7], struct2vec [29], GraphSAGE [13], EP [10] and more [4, 6, 21, 23, 26, 27, 40]. To use node embedding for semantic proximity search, one approach is to combine two node embedding vectors and turn the output into a proximity score. ProxEmbed [22] shows that such an approach is indirect; instead, it directly embeds each connecting path between two nodes into a vector, and aggregates multiple paths for a proximity score. Although the path-based approach is natural for modeling proximity between two possibly distant nodes, the above methods tend to weakly couple the paths in modeling.

Some pioneer work has explored path interdependencies. In addition to Word-to-Word Attention LSTM [30] and Coupled-LSTM [20], DF-LSTM [19] also models sentence-to-sentence level interdependency for text semantic matching. However, these methods are designed to handle two paths at a time. In our task, we have multiple interactive paths. If we apply these methods on every two possible paths, we still limit ourselves to only seeing two paths at a time in embedding. On the other hand, if we choose to directly extend their methods to more than two paths, we are likely to suffer from high generalization complexity. For example, Coupled-LSTM requires maintaining a 3D tensor of size $l_1 \times l_2 \times d$, where l_1 and l_2 are the lengths of two paths respectively, d is the embedding dimension. Extending this tensor to include more paths can substantially increase the complexity of computation and storage. Comparatively, in IPE, we carefully construct the inference-friendly interactive-paths structure based on the network topology, and enable multiple GRUs to run simultaneously and interactively. Despite some high-level similarity with multi-task sequence-to-sequence learning [24], our IPE does not require the paths to exist explicit many-to-one, one-to-many or many-to-many correspondence; besides, our IPE enforces object-level interdependencies, whereas the multi-task setting only considers sequence-level interdependencies.

Table 1: Notations used in this paper.

Notation	Description
G, V, E, C	Graph G, nodes V, edges E, node types C
$i(q, v)$	An interactive-paths structure between q and v
$\mathcal{I}(q, v)$	Set of interactive-paths structures between q and v
$\mathcal{D}$, m	Set of training tuples, # of training tuples
$\mathcal{P}$	Set of sampled paths on G
$\mathbf{g}(q, v)$	Embedding vector of a path between q and v within an $i(q, v)$
$\mathbf{o}(q, v)$	Embedding vector of an interactive-paths structure $i(q, v)$
$\mathbf{f}(q, v)$	Proximity embedding vector between q and v
$\pi(q, v)$	Proximity score between q and v
ℓ, ζ	# of sampled paths for each node, walk length ζ
d, d'	Embedding dimension, interactive GRU input feature dimension
α, β	Hyper-parameters for path heterogeneity & distance awareness
λ	Hyper-parameter for model regularization
R_j	Set of predecessors for node j in an interactive-paths structure

As single path has limited expressiveness, some recent work also exploits higher-order subgraph structures for semantic proximity search. For example, *Meta-graph Proximity* (MGP) [9] first mines the frequent subgraph patterns as meta-graphs (*e.g.*, "user–employer & location–user"); then it counts the instances of certain meta-graphs between two objects to measure their proximity. Although meta-graphs are powerful, frequent subgraph mining can be challenging [38]. In practice meta-graphs' sizes are small, which limits the performance too. Some other work considers embedding with trees [8, 35] or directed acyclic graphs (DAGs) [33, 43]. They are not designed for path embedding; besides, they are often not easily adaptable to our task, due to their highly customized design. Finally, it is worth mentioning some related, yet different concepts. Grid-LSTM [15] and Multidimensional-RNN [11] are designed for multidimensional data, instead of multiple paths. Graph kernels [6, 42] measure similarity between graphs via substructures, but they are not designed for embedding multiple paths. Knowledge graph embedding, such as TransE [2], TransH [41], ProjE [31] and TransNet [37], considers a special type of heterogeneous graphs. The knowledge graphs consist of (entity, relation, entity) triple facts. The typical goal of knowledge graph is to learn the embedding for entities and relations, such that for each tuple, adding the head entity's embedding and the relation's embedding in some way equals the tail entity's embedding. Such triple facts do not exist in our data, hence we cannot apply knowledge graph embedding.

3 PROBLEM FORMULATION

We first introduce terminologies and notations (listed in Table 1).

Definition 3.1. A **heterogeneous graph** is $G = (V, E, C, \tau)$, where V is a set of nodes, E is a set of edges, $C = \{c_1, ..., c_K\}$ is a set of distinct types, and $\tau : V \to C$ is a node type mapping function.

For example, in Fig. 1(a), we have C = {"user", "college", "location", "employer"}, and $\tau(Donna)$ = "user", $\tau(MIT)$ = "college", $\tau(Google)$ = "employer", and $\tau(L.A.)$ = "location".

Definition 3.2. A **path** on G is a sequence of nodes $v_1 \to v_2 \to ... \to v_t$, where each $v_i \in V$, and t is the path length.

For example, in Fig. 2, given a query node q and a target node v, we sample a set of paths from q to v, and also from v to q.

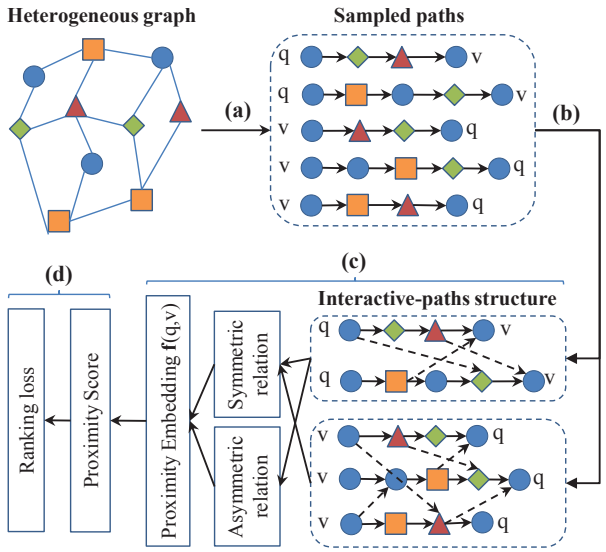


Figure 2: The overall framework of IPE: (a) sample paths $\mathcal{P}(q,v)$ from G; (b) use cycle-free shuffling to construct interactive-paths structures $\mathcal{I}(q,v)$ from $\mathcal{P}(q,v)$; (c) use interactive GRU to compute embedding $\mathbf{f}(q,v)$ for each (q,v); (d) compute ranking loss based on proximity score $\pi(q,v)$.

Definition 3.3. An **interactive-paths structure** $i(q,v)$ between two nodes $q,v \in V$ is a directed acyclic graph (DAG) consisting of multiple interdependent paths.

For example, after Fig. 2(b), we get two interactive-paths structures, which are DAGs consisting of multiple interdependent paths from q to v and from v to q respectively. We denote a set of interactive-paths structures between q and v as $\mathcal{I}(q,v)$.

Problem inputs and outputs. For *inputs* of our model, we have a heterogeneous graph G, and a set of training tuples $\mathcal{D} = \{(q_k, a_k, b_k) : k = 1, ..., m\}$, where for each query node q_k, node a_k is closer to q_k than node b_k. Generation of these training tuples is discussed in Sect. 7. Besides, we also *offline* sample some paths from G as inputs. We take a similar approach as DeepWalk [28] for path sampling. Specifically, starting from each node in G, we randomly sample ℓ paths, each of length ζ. As a result, we obtain a set of paths, denoted as $\mathcal{P}$. These paths will be indexed to support efficient training and testing. For each query node $q \in \{q_1, ..., q_m\}$ and a corresponding target node $v \in \{a_1, ..., a_m, b_1, ..., b_m\}$, we extract multiple subpaths from $\mathcal{P}$. We denote all the subpaths starting from q and ending at v in $\mathcal{P}$ as $\mathcal{P}(q,v)$, and those from v to q as $\mathcal{P}(v,q)$. For the paths in $\mathcal{P}(q,v)$, we use cycle-free shuffling mechanism to construct a set of interactive-paths structures $\mathcal{I}(q,v)$. Similarly, we construct $\mathcal{I}(v,q)$ from $\mathcal{P}(v,q)$. We introduce the details of construction in Sect. 4.

For *outputs* of our model, given a specific interactive-paths structure $i(q,v)$ constructed from the paths in $\mathcal{P}(q,v)$, we first compute a low-dimensional vector $\mathbf{g}(q,v) \in \mathbb{R}^d$ for each path. Here, $d > 0$ is the embedding dimension. As there are multiple paths in $i(q,v)$, we will aggregate multiple $\mathbf{g}(q,v)$'s into a interactive-paths structure embedding vector $\mathbf{o}(q,v) \in \mathbb{R}^d$. Finally, as there are multiple interactive-paths structures in $\mathcal{I}(q,v)$, we will again aggregate multiple $\mathbf{o}(q,v)$'s into a proximity embedding vector $\mathbf{f}(q,v) \in \mathbb{R}^d$. In this work, we consider both symmetric and asymmetric relations, where for symmetric relations $\mathbf{f}(q,v) = \mathbf{f}(v,q)$ and for asymmetric ones $\mathbf{f}(q,v) \neq \mathbf{f}(v,q)$. We discuss how to compute $\mathbf{g}(q,v)$, $\mathbf{o}(q,v)$ and $\mathbf{f}(q,v)$ by an interactive GRU model in Sect. 5. In the end, we use $\mathbf{f}(q,v)$ to estimate a proximity score between q and v as

$$\pi(q,v) = \boldsymbol{\theta}^T \mathbf{f}(q,v), \tag{1}$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ is a parameter vector.

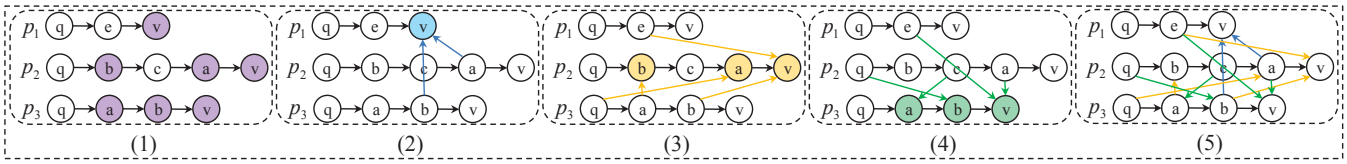
Our model has two types of parameters: 1) the interactive GRU parameters for getting $\mathbf{g}(q,v)$, $\mathbf{o}(q,v)$ and $\mathbf{f}(q,v)$; 2) the proximity estimation parameter $\boldsymbol{\theta}$. In training, we aim to learn these model parameters, such that $\pi(q_k, a_k) \geq \pi(q_k, b_k)$ for each $(q_k, a_k, b_k) \in \mathcal{D}$. We will introduce the details of training algorithm in Sect. 6. Note that in offline training, we only need to compute $\mathbf{g}(q,v)$, $\mathbf{o}(q,v)$ and $\mathbf{f}(q,v)$ for those (q_k, a_k) and (q_k, b_k) for $k = 1, ..., m$, instead of all the possible node pairs in G. In online testing, given a random query node q in G, we will leverage the offline indexing to efficiently extract from $\mathcal{P}$ a set of sample paths from q to each possible target node v in G. Then we construct a set of interactive-paths structures, and apply our model to compute each $\pi(q,v)$ for ranking.

4 INTERACTIVE PATHS CONSTRUCTION

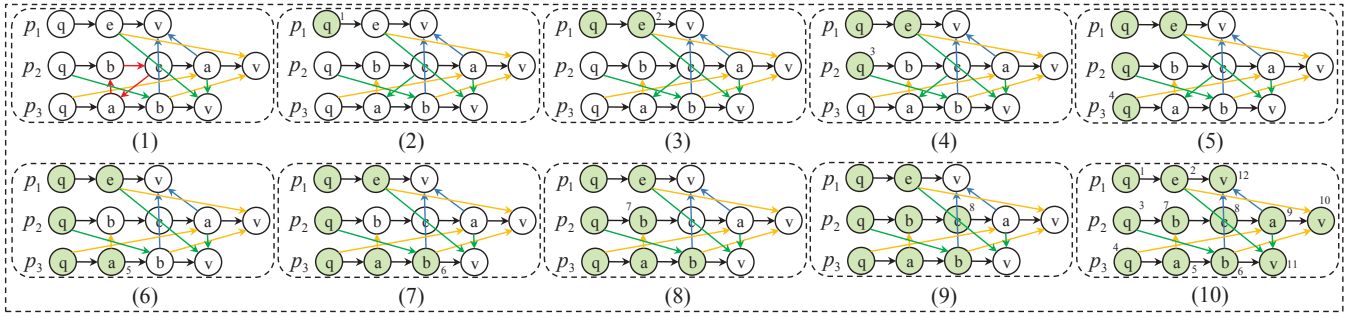
We introduce how to construct an interactive-paths structure by a cycle-free shuffling mechanism from multiple paths from a query node q to a target node v.

Overview of cycle-free shuffling. Given multiple paths, we choose to first add interdependencies with directed edges among the paths, which leads to a directed graph. Then we try to remove the possible cycles in the directed graph, by an efficient graph traversal algorithm. Because there are many possible ways to remove cycles from a directed graph, we use certain path ordering to decide which directed edge to remove first. As a result, we can shuffle the paths with different orderings, and thus generate multiple interactive-paths structures. In this way, we avoid the information loss due to the cycle removal. Next, we introduce two major components of this cycle-free shuffling mechanism, including "adding interdependencies" and "removing cycles", by examples.

Adding interdependencies. Consider Fig. 3(a)(1), where we have three paths from q to v, *i.e.*, $\mathcal{P}(q,v) = \{p_1, p_2, p_3\}$. We first scan all the paths, and identify all the nodes that have multiple predecessors. As a result, we find v, b and a (colored in purple). Then, we traverse each path $p \in \mathcal{P}(q,v)$. For each node u with multiple predecessors in p, we add u's predecessors from the other paths as the interdependencies. Take path p_1 as an example. As shown in Fig. 3(a)(2), we only have one node with multiple predecessors, *i.e.*, v (colored in blue). Then we add directed edges from $(p_2 : a)$ and $(p3 : b)$ to $(p_1 : v)$. After this, we move on to path p_2. As shown in Fig. 3(a)(3), we have three nodes with multiple predecessors, *i.e.*, b, a and v (colored in yellow). Then, for b we add directed edges from $(p_3 : a)$ to $(p_2 : b)$; for a, we add directed edges from $(p_3 : q)$ to $(p_2 : a)$; for v, we add directed edges from $(p_1 : e)$ and $(p_3 : b)$ to $(p_2 : v)$. Similarly,



(a) Add interdependencies among paths.



(b) Remove possible cycles to form an interactive-paths structure.

Figure 3: Interactive paths construction. Best view in color.

we can add directed edges for path p_3 in Fig. 3(a)(4). Finally, we obtain a directed graph with possible cycles in Fig. 3(a)(5).

Complexity analysis: suppose the average length of a path in $\mathcal{P}(q,v)$ is l. Traversing the paths to find nodes with multiple predecessors takes $O(|\mathcal{P}(q,v)|l)$. Adding interdependencies for all nodes takes $O(|\mathcal{P}(q,v)|^2 l)$. In total, the complexity to add interdependencies for $\mathcal{P}(q,v)$ is $O(|\mathcal{P}(q,v)|l + |\mathcal{P}(q,v)|^2 l) = O(|\mathcal{P}(q,v)|^2 l)$.

Removing cycles. As highlighted in Fig. 3(b)(1), the resulting directed graph of Fig. 3(a)(5) has a cycle among a, b and c (colored in red). To remove this cycle, we traverse each path according to a fixed ordering, *e.g.*, first p_1 then p_2 then p_3. We will only visit a node if its predecessors are already visited. We mark the already visited node as green color. Whenever we find no more node to continue visiting, yet still having unvisited nodes, we encounter a cycle. Then we will remove a directed edge to avoid the cycle. We use a variable $r = 0$ to record the consecutive times that we jump from one path to another. Let us start traversing from q in p_1. As shown in Fig. 3(b)(2), $(p_1 : q)$ has no predecessor, after visiting it we mark it as green and setting $r = 0$. We continue traversing on p_1, and similarly we mark $(p_1 : e)$ as green in Fig. 3(b)(3) and set $r = 0$. Then we reach $(p_1 : v)$, which depends on $(p_2 : a)$ and $(p_3 : b)$, both unvisited yet. In this case, we cannot finish visiting $(p_1 : v)$. Instead, we move on to traverse p_2 and set $r = r + 1 = 1$ to record we have a jump. Once visiting $(p_2 : q)$, we mark it as green in Fig. 3(b)(4), and set $r = 0$ because the jump continuity has been broken. Then we reach $(p_2 : b)$, which depends on $(p_3 : a)$. Since $(p_3 : a)$ is not visited yet, we jump to p_3 with setting $r = r + 1 = 1$, and visit $(p_3 : q)$ as shown in Fig. 3(b)(5). After visiting $(p_3 : q)$ with setting $r = 0$, we try to visit $(p_3 : a)$, but it has unvisited predecessor $(p_2 : c)$ then we set $r = r + 1 = 1$ and jump to p_1. It is the same way to visit the paths, until we jump to p_3 again with $r = 3$ and $r \geqslant |\mathcal{P}(q,v)|$, meaning we have jumped 3 times continuously without visiting any nodes. Now we know that we have no more nodes to continue visiting, although there are still many unvisited nodes. This is due to the cycle among a, b and c in p_2 and p_3. To avoid this cycle, we remove all the cross-path predecessors of the current node in traversal, which is $(p_3 : a)$ in this case. That is, we remove the directed edges from $(p_2 : c)$ to $(p_3 : a)$. After the cycle is removed, we are able to visit $(p_3 : a)$ in Fig. 3(b)(6). We can continue to visit the remaining nodes in Fig. 3(b)(7–10). In the end, we obtain a cycle-free interactive-paths structure. Note that, if the paths are ordered differently, we will get different interactive-paths structures. To minimize information loss, we shuffle $\mathcal{P}(q,v)$ for multiple times, and generate multiple interactive-paths structures $\mathcal{I}(q,v)$.

Complexity analysis: suppose the average number of predecessors for each node is s. To visit a node in a path, we need to check whether its s predecessors are already visited, which takes $O(s)$. The worst situation is that at each node we need to remove cycles, and we need to access the current nodes in all the paths until $r \geqslant |\mathcal{P}(q,v)|$. So for each node, the worst time complexity is $O(|\mathcal{P}(q,v)|s)$. In total, there are $|\mathcal{P}(q,v)|l$ nodes, the total complexity is $O(|\mathcal{P}(q,v)|^2 ls)$.

Interactive paths construction algorithm. We abstract the above approach of adding interdependencies and removing cycles in Alg. 1. Given (q,v), we run Alg. 1 for several times to generate multiple interactive-paths structures. In line 2, we shuffle the paths order. Lines 3-5 are to add interdependencies among paths and its time complexity is $O(|\mathcal{P}(q,v)|^2 l)$. Lines 6-21 are to remove cycles and its time complexity is $O(|\mathcal{P}(q,v)|^2 ls)$. In total, the complexity for Alg. 1 is $O(|\mathcal{P}(q,v)|^2 l + |\mathcal{P}(q,v)|^2 ls) = O(|\mathcal{P}(q,v)|^2 ls)$.

Algorithm 1 Interactive-Paths Structure Construction

Require: graph $G = (V, E, C, \tau)$, a set of paths $\mathcal{P}(q, v)$,
Ensure: interactive-paths structure $i(q, v)$.

```
initialize r = 0, finished paths set S = ∅, i(q,v) = P(q,v);
Shuffle(P(q,v));
for all path p ∈ P(q,v) do
    append(i(q,v), interdependencies from the other paths to p);
end for
while |P(q,v)| ≠ |S| do
    for all path p ∈ {P(q,v) − S} do
        if r ⩾ |P(q,v)| − |S| then
            remove(i(q,v), p.curNode.dep);
        end if
        if p.curNode.dep = ∅ or p.curNode.dep traversed then
            traverse p.curNode; r = 0;
            p.curNode = p.curNode.next;
        else
            r = r + 1; continue;
        end if
        if p.curNode = null then
            append(S, p);
        end if
    end for
end while
return i(q,v).
```

5 INTERACTIVE PATHS EMBEDDING

We introduce how to model multiple interactive-paths structures $\mathcal{I}(q, v)$ for embedding. To model the recurrent nature of the interactive paths and meanwhile deal with the interdependencies among the paths, we develop an interactive GRU mechanism.

Interactive GRU. We use Fig. 3(b)(10) as an example to overview the architecture of our interactive GRU. As shown in the left hand side of Fig. 4, we do recurrent embedding for each node along path p_2 with both its own features and its predecessors. We denote a node v's features as $\mathbf{x}_v \in \mathbb{R}^{d'}$, and we define it as a concatenation of: 1) *node type*, which is a $|C|$-dimension vector, with only the dimension that corresponds to v's type as one and the others as zeros; 2) *node degree*, which is the v's degree in G; 3) *neighbors' type distribution*, which is a K-dimension vector, with each dimension as the logarithm number of v's neighbors of one type; 4) *neighbors' type entropy*, which is the entropy of the type distribution for v's neighbors. For example, to embed node v in p_2, we need to consider its input features $\mathbf{x}_v$ and the embedding vectors of its same-path predecessor a, as well as different-path predecessors e and b.

Next, we introduce each computation units in interactive GRU.

• ***Hidden state for input***: for a node j, it takes the hidden states of its multiple predecessors as input. Denote the set of predecessors of node j as R_j; *e.g.*, in Fig. 3(b)(10), $R_{(p_2:v)} = \{(p_2 : a), (p_1 : e), (p_3 : b)\}$. For different predecessors, we consider them as contributing differently according to three unique task characteristics.

1) *Path heterogeneity*: depending on whether a predecessor comes from the same path as node j, we assign different weights. Denote R_j^{same} as the set of j's predecessors from the same path as j, whereas

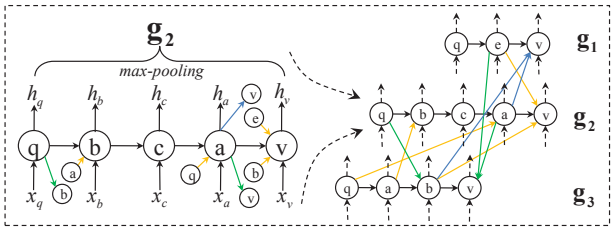


Figure 4: The interactive GRU architecture.

R_j^{diff} as the set of predecessors from different paths as j's; *e.g.*, in Fig. 3(b)(10), $R_{(p_2:v)}^{\text{same}} = \{(p_2 : a)\}$ and $R_{(p_2:v)}^{\text{diff}} = \{(p_1 : e), (p_3 : b)\}$.

2) *Distance awareness*: we differentiate the predecessors based on their distances to the start node in their own paths. For example, in Fig. 3(b)(10), $(p_2 : a)$'s distance to the start node in p_2 is 3, whereas $(p_1 : e)$'s distance to the start node in p_1 is 1. Both of them are predecessors of node $(p_2 : v)$. Generally, the larger the distance is, the weaker the connection is.

3) *Node heterogeneity*: we also differentiate the predecessors based on their node types, as different types imply different semantics.

Denote the embedding of a node k as $\mathbf{h}_k \in \mathbb{R}^d$. For path heterogeneity, we introduce a hyper-parameter $\alpha \in [0, 1]$ as the weight for a same-path predecessor, and use $(1 - \alpha)$ as the weight for a different-path predecessor. For distance awareness, we denote node k's distance to the start node in its own path as $k.dist$. For node heterogeneity, we denote $\mathbf{t}_k \in \{0, 1\}^{|C|}$ as one-hot type indicator vector for node k. Finally, we aggregate the predecessor embedding from R_j by max pooling, as the input for node j:

$$\mathbf{h}_{R_j} = maxPool(\{\hat{\mathbf{h}}_k \cdot e^{-\beta \times k.dist} \cdot \sigma(\boldsymbol{\eta}^T \mathbf{t}_k) : k \in R_j\}), \quad (2)$$

where each path discounted predecessor embedding

$$\hat{\mathbf{h}}_k = \begin{cases} \alpha \mathbf{h}_k, & \text{for } k \in R_j^{\text{same}}, \\ (1-\alpha)\mathbf{h}_k, & \text{for } k \in R_j^{\text{diff}}, \end{cases} \quad (3)$$

$\beta > 0$ is a distance discount hyper-parameter, $\sigma(x) = 1/(1 + e^{-x})$ is a sigmoid function, and $\boldsymbol{\eta} \in \mathbb{R}^{|C|}$ is a node type hyper-parameter vector.

• ***Reset gate***: for node j, the reset gate is defined as

$$\mathbf{r}_j = \sigma(W_r \mathbf{x}_j + U_r \mathbf{h}_{R_j} + \mathbf{b}_r), \quad (4)$$

where $W_r \in \mathbb{R}^{d \times d'}$, $U_r \in \mathbb{R}^{d \times d}$ and $\mathbf{b}_r \in \mathbb{R}^d$ are the parameters.

• ***Update gate***: for node j, the update gate is defined as

$$\mathbf{z}_j = \sigma(W_z \mathbf{x}_j + U_z \mathbf{h}_{R_j} + \mathbf{b}_z), \quad (5)$$

where $W_z \in \mathbb{R}^{d \times d'}$, $U_z \in \mathbb{R}^{d \times d}$ and $\mathbf{b}_z \in \mathbb{R}^d$ are the parameters.

• ***Temporary hidden state***: it is defined as

$$\tilde{\mathbf{h}}_j = tanh(W_h \mathbf{x}_j + U_h[\mathbf{r}_j \odot \mathbf{h}_{R_j}] + \mathbf{b}_h), \quad (6)$$

where $W_h \in \mathbb{R}^{d \times d'}$, $U_h \in \mathbb{R}^{d \times d}$ and $\mathbf{b}_h \in \mathbb{R}^d$ are the parameters, $\odot$ is an element-wise multiplication.

• ***Output hidden state***: we compute the embedding for node j as

$$\mathbf{h}_j = (1 - \mathbf{z}_j) \odot \mathbf{h}_{R_j} + \mathbf{z}_j \odot \tilde{\mathbf{h}}_j, \quad (7)$$

where $\odot$ is an element-wise multiplication.

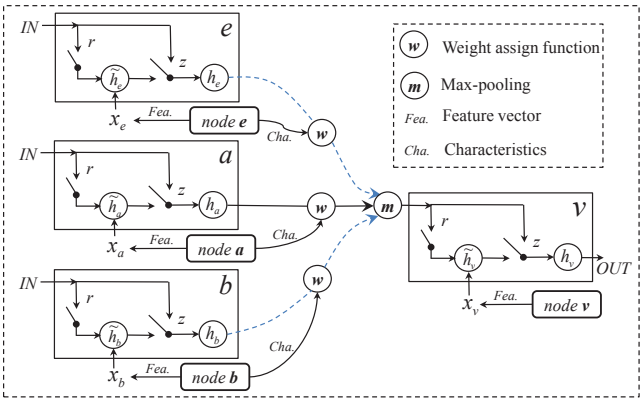


Figure 5: Interactive GRU units to aggregate predecessors $\{e, a, b\}$ with weights for a target node v's embedding.

In all, we use node $(p_2 : v)$ in Fig. 3(b)(10) as an example, to visualize the above interactive GRU units in Fig. 5. Once obtaining the embedding (*i.e.*, hidden state) for each node j in $i(q, v)$, we compute the path embedding for each path $p \in i(q, v)$ as

$$\mathbf{g}_p = maxPool(\{\mathbf{h}_k : k \in p\}), \tag{8}$$

where k is a node in path p. After that, we calculate the interactive-paths structure embedding for $i(q, v)$ as

$$\mathbf{o}_{i(q,v)} = maxPool(\{\mathbf{g}_p \cdot e^{-\beta \times p.length} : p \in i(q, v)\}), \tag{9}$$

where $p.length$ is the length of path p. Generally, the longer a path is, the weaker relation it describes. Finally, we aggregate multiple interactive-paths structures $\mathcal{I}(q, v)$ as

$$\hat{\mathbf{o}}_{\mathcal{I}(q,v)} = maxPool(\{\mathbf{o}_{i(q,v)} : i(q, v) \in \mathcal{I}(q, v)\}). \tag{10}$$

Denote $\Upsilon(q, v)$ as all the interactive-paths structures for (q, v), where for asymmetric we have $\Upsilon(q, v) = \mathcal{I}(q, v)$, for symmetric relation we have $\Upsilon(q, v) = \mathcal{I}(q, v) \cup \mathcal{I}(v, q)$. Therefore, we compute the proximity embedding for (q, v) as

$$\mathbf{f}(q, v) = maxPool(\{\hat{\mathbf{o}}_{\mathcal{I}(q,v)} : \mathcal{I} \in \Upsilon(q, v)\}). \tag{11}$$

6 END-TO-END TRAINING

For each training tuple (q_k, a_k, b_k), we obtain a loss function as

$$\ell(q_k, a_k, b_k) = -\sigma[\pi(q_k, a_k) - \pi(q_k, b_k)], \tag{12}$$

where $\pi(q, v)$ is defined in Eq. 1. Denote the set of IPE parameters as $\Theta = \{\boldsymbol{\theta}, W_r, U_r, \mathbf{b}_r, W_z, U_z, \mathbf{b}_z, W_h, U_h, \mathbf{b}_h\}$. We learn Θ by minimizing

$$L(\Theta) = \sum_{i=k}^{m} \ell(q_k, a_k, b_k) + \lambda\Omega(\Theta) \tag{13}$$

where Ω is a regularization function (*e.g.*, the sum of each IPE parameter's l_2-norm), and $\lambda > 0$ is a hyper-parameter.

We summarize IPE in Alg. 2. Line 1 is to sample paths by random walk. Line 2 constructs interactive-paths structures from these sampled paths. Line 3 splits all the training tuples into batches. Lines 4–13 optimize Θ with the training tuples via end-to-end training, where lines 7–8 computes the proximity embedding for each (q, v), line 9 accumulates the loss and line 12 optimizes Θ.

Complexity analysis. At line 1 of Alg. 2, we sample ℓ paths with length ζ for each node, hence the complexity is $O(|V|\ell\zeta)$. At line 2,

Algorithm 2 Interactive-Paths Structure Embedding

Require: heterogeneous graph $G = (V, E, C, \tau)$, training tuples $\mathcal{D} = \{(q_k, a_k, b_k)\}$, set of interactive-paths structures Υ, embedding dimension d, path discount α, distance discount β, # of walks per node ℓ, walk length ζ.
Ensure: model parameters Θ.
1: $\mathcal{P} \leftarrow$ SamplePath(G, ℓ, ζ);
2: $\Upsilon \leftarrow$ GenerateInteractivePaths$(\mathcal{P})$ by Alg.1;
3: $\mathcal{B} \leftarrow$ GenerateBatches$(\mathcal{D})$;
4: **for all** batch in $\mathcal{B}$ **do**
5: Initialize loss as $L = 0$;
6: **for all** each (q_k, a_k, b_k) in this batch **do**
7: $\mathbf{f}(q_k, a_k) \leftarrow$ Embed$(\Upsilon, q_k, a_k, d, \alpha, \beta)$ by Eq. 11;
8: $\mathbf{f}(q_k, b_k) \leftarrow$ Embed$(\Upsilon, q_k, b_k, d, \alpha, \beta)$ by Eq. 11;
9: $L = L + \ell(q_k, a_k, b_k)$, based on Eq. 12;
10: **end for**
11: $L = L + \lambda\Omega(\Theta)$;
12: Update Θ based on L by stochastic gradient descent.
13: **end for**
14: **return** Θ.

constructing the interactive-paths structures for each (q, v) takes $O(|\mathcal{P}(q, v)|^2 ls)$ as analyzed in Alg. 1. Here, l is the average length of a path in $\mathcal{P}(q, v)$ and s is the average number of predecessors for each node therein. Given m training tuples $\{(q_i, a_i, b_i)\}$, the total complexity for line 2 is $O(m|\mathcal{P}(q, v)|^2 ls)$. At lines 4–13, the complexity to compute embedding for one interactive-paths structure is $O(|\mathcal{P}(q, v)|ls)$, hence the total complexity is $O(m|\mathcal{P}(q, v)|ls)$. In all, the complexity of Alg. 2 is $O(|V|\ell\zeta) + O(m|\mathcal{P}(q, v)|^2 ls) + O(m|\mathcal{P}(q, v)|ls) = O(|V|\ell\zeta + m|\mathcal{P}(q, v)|^2 ls)$. This complexity is linear to both the node size $|V|$ and the number of training tuples m; it is quadratic to the average number of paths between q and v.

7 EXPERIMENTS

7.1 Datasets and Settings

We used four different types of heterogeneous networks to construct our datasets. Next we introduce each network, and summarize the resulting datasets statistics in Table 2.

• **Professional Network**: we use one public dataset *LinkedIn* [18]. It was collected from LinkedIn, which is an online service for members (both workers and employers) to create profiles and connections with each other. The LinkedIn network contains *user*, *college*, *employer* and *location*.

• **Social Network**: we use one public dataset *Facebook* [25]. It was collected from Facebook, which is an online service for members to share information and connect with people they know. The Facebook network contains *user*, *work-project*, *hometown*, *work-location*, *school*, *degree*, *last-name*, *employer*, *location* and *concentration*.

• **Bibliography Network**: we use one public dataset *DBLP* [39]. It was collected from DBLP, which is an online reference for bibliographic information on major computer science publications. The DBLP network contains *user*, *keyword*, *year*, *conference* and *paper*.

• **E-Commerce Network**: we use one private dataset *Taobao*. It was collected and provided by Taobao, a world-leading e-commerce

Table 2: Statistics of our four datasets.

	$\|V\|$	$\|E\|$	$\|C\|$	Semantic relations	Symmetry	Queries	Results per query
LinkedIn	65,925	220,812	4	*school.*	yes	172	16.2
				collea.	yes	173	12.8
Facebook	5,025	100,356	10	*family.*	yes	340	4.0
				classmate	yes	904	6.5
DBLP	165,728	928,513	5	*advisor*	no	2,439	1.3
				advisee	no	1,024	2.6
Taobao	6,689,132	47,478,942	6	*oneID*	yes	27,039	2.4

site. It records a week of user activity logs on mobile or PC in a city of China. The Taobao network contains *MID*, *PID*, *auction*, *shop* and *keyword*. Here, an MID is the identity of a mobile device used by one Taobao user, whereas a PID is the identity of a PC device used by one Taobao user. A user (*i.e.*, MID or PID) can interact with IP, auction, shop and keyword.

Ground truth. In LinkedIn, the user relationships were already labeled into different types [18]. We consider two major semantic relations: *schoolmate* and *colleague*. In Facebook, the user relationships were defined by [9] as: *family* and *classmate*. In DBLP, the *adivsor* and *advisee* in a co-author pair were identified by [39], based on homepages of some faculty members, Mathematics Genealogy and AI Genealogy projects. All unidentified co-author pairs were assumed to be negative. In Taobao, we consider on semantic relation of *oneID*. We labeled a pair of (MID, PID) as positive, if they refer to one real world Taobao user; otherwise, we label it as negative. To generate the candidate pairs of (MID, PID), we employed a set of practical rules that are currently used in production at Taobao; *e.g.*, a PID and an MID must co-occur with the same IP(s), the same router(s), and so on.

As summarized in Table 2, the *advisor* and *advisee* classes are asymmetric, whereas the others are all symmetric.

Training and testing. On each graph, a user q can be used as a query node if there exists at least another user v such that q and v have the desired semantic relation in our ground truth. The number of query users for each network and each type of semantic relations, and the average number of results per query are shown in Table 2. We randomly split these queries into two subsets: 20% reserved as training and the rest as testing. We repeated such splitting for 10 times, and averaged any result over these 10 splits. In each split, based on the training queries, we further generated training examples (q, v, u) such that q and v belong to the desired semantic relation whereas q and u do not. For testing, we constructed an ideal ranking for each test query user and each desired semantic relation. We compared this ideal ranking against the ranking generated by various semantic user search algorithms. We adopted NDCG and MAP [9] to evaluate the quality of ranking at the top 10 results.

Parameters and environment. We set $\lambda = 0.0001$, and tuned the other hyper-parameters d, ℓ, ζ, α and β on each dataset. By default, $d = 100$ in Taobao to handle its larger graph size, whereas in LinkedIn, Facebook and DBLP we set smaller d for their smaller graph size, *e.g.*for *schoolmate* we set $d = 16$. For fair comparison, we sample paths from the heterogeneous graphs with the same parameters setting in ProxEmbed[22]. Specifically, we set $\ell = 20$, $\zeta = 20$ on LinkedIn and Taobao, whereas on Facebook we set $\ell = 40$, $\zeta = 80$ for *classmate* and $\ell = 20$, $\zeta = 80$ for *family*. On DBLP, we set $\ell = 20$, $\zeta = 80$ for *advisor* and $\ell = 20$, $\zeta = 40$ for *advisee*. α was set in the range of $[0.25, 0.75]$, whereas β was set in the range of $[0.1, 1.0]$. We ran our experiments on Linux machines with 8-core 2.27GHz Intel Xeon(R) CPUs and 32GB memory. We use java-1.8 and Theano [36] for coding development.

Our implementation code for IPE is available[1].

7.2 Performance Study

We compare our IPE with the following state-of-the-art baselines.

• *SRW*: Supervised Random Walk [1] learns edge weights to bias random walk for consistent ranking results with the ground truth. We define edge features as a vector based on its nodes' types.

• *DeepWalk*: we applied [28] to embed the graph, and used a Hadamard product over two nodes' embedding for proximity estimation.

• *ProxEmbed*: it [22] first samples multiple paths between two nodes, then feeds these paths into LSTM to get the proximity embedding.

• *Metapath2vec*: it [7] samples paths w.r.t. meta-path patterns for node embedding with heterogeneous skip-gram. We used a Hadamard product over two nodes' embedding for proximity estimation. For all the datasets, we used meta-path patterns of "user-item-user", "user-item-item-user" and "user-item-user-item-user"; *e.g.*, in Taobao, "user" is *MID* or *PID*, "item" is *IP*, *auction*, *shop* or *keyword*.

• *DAG-LSTM*: it [33] models each interactive-paths structure $i(q, v)$ as a DAG, and outputs an embedding vector for each end node v in $p \in i(q, v)$. We then max-pooled these outputs as an embedding vector for $i(q, v)$, and used it for proximity estimation.

• *IPE-α*: we designed a degenerated version of IPE without path heterogeneity. We aggregated predecessors with equal weights.

• *IPE-α-β*: we designed a degenerated version of IPE without path heterogeneity and distance awareness. We further set $\beta = 0$.

• *IPE-α-β-η*: we designed a degenerated version of IPE without path heterogeneity, distance awareness and node heterogeneity. We further ignored the node types in predecessor aggregation.

For fair comparison, we feed the same sampled paths to all the baselines (*e.g.*, DeepWalk, ProxEmbed, DAG-LSTM, and all the degenerated versions of IPE). For the parameters in all the baselines, we referred to their corresponding papers and further fine tuned them on each dataset. For SRW, we set its regularization parameter $\lambda = 10$, random walk teleportation parameter $\alpha = 0.2$ and loss parameter $b = 0.1$. For DeepWalk and metapath2vec, we set its embedding dimension as 128, which gave good results in both their papers [28] and our datasets. For DAG-LSTM, we set its embedding dimention as 32 in all the datasets, as it gave the best results.

Comparison with baselines. In Table 3, we report the results under different amounts of training tuples. We observe that, IPE and its degenerated versions generally outperform the baselines. Next, we analyze each baseline and compare it with IPE.

SRW generally has relatively lower performance than the other methods, except on the *advisee* relation in DBLP. Compared with other graph embedding methods, SRW seems not very good at identifying useful graph patterns to help proximity search. Besides,

[1] https://github.com/shuaiokshuai/IPE

Table 3: Result comparison under different amounts of labels. Highlighted results are significant.

	Methods	LinkedIn						Facebook						DBLP						Taobao		
		schoolmate			*colleague*			*classmate*			*family*			*advisor*			*advisee*			*oneID*		
		10	100	1000	10	100	1000	10	100	1000	10	100	1000	10	100	1000	10	100	1000	1K	10K	100K
NDCG@10	SRW	0.520	0.515	0.515	0.513	0.502	0.503	0.396	0.389	0.394	0.396	0.389	0.394	0.689	0.689	0.689	**0.402**	0.402	0.402	-	-	-
	DeepWalk	0.515	0.518	0.530	0.493	0.506	0.504	0.692	0.689	0.699	0.585	0.575	0.583	0.741	0.743	0.739	0.382	0.379	0.374	0.437	0.496	0.509
	ProxEmbed	0.646	0.652	0.670	0.561	0.606	0.616	0.796	0.851	0.852	0.584	0.743	0.761	0.753	0.765	0.771	0.374	0.405	0.411	0.646	0.701	0.708
	Metapath2vec	0.513	0.524	0.528	0.507	0.509	0.523	0.688	0.702	0.709	0.563	0.560	0.574	**0.775**	0.769	0.735	0.378	0.403	0.395	0.532	0.595	0.600
	DAG-LSTM	0.612	0.638	0.633	0.480	0.576	0.584	0.396	0.656	0.743	0.481	0.663	0.701	0.704	0.758	0.787	0.366	0.385	0.408	0.682	0.704	0.710
	IPE-α	0.657	0.674	0.674	**0.636**	0.626	0.641	**0.823**	0.831	0.816	**0.739**	0.758	0.761	0.726	0.703	**0.792**	0.378	0.403	0.423	**0.699**	0.718	0.732
	IPE-α-β	0.642	0.647	0.666	0.607	0.596	0.621	0.481	0.616	0.794	0.463	0.731	**0.768**	0.683	0.649	0.743	0.362	0.363	0.402	0.685	0.713	0.725
	IPE-α-β-η	0.640	0.641	0.655	0.588	0.604	0.556	0.443	0.535	0.625	0.494	0.663	0.762	0.681	0.679	0.717	0.369	0.370	0.399	0.692	0.706	0.715
	IPE	**0.667**	**0.683**	**0.684**	0.607	**0.641**	**0.645**	0.593	**0.856**	**0.857**	0.673	**0.767**	**0.768**	0.704	**0.782**	0.788	0.368	**0.419**	**0.425**	0.696	**0.726**	**0.736**
MAP@10	SRW	0.275	0.272	0.272	0.294	0.289	0.291	0.312	0.324	0.384	0.259	0.255	0.259	0.630	0.630	0.630	**0.294**	**0.294**	0.294	-	-	-
	DeepWalk	0.393	0.391	0.398	0.363	0.369	0.368	0.578	0.575	0.587	0.467	0.455	0.463	0.663	0.659	0.656	0.263	0.264	0.257	0.337	0.395	0.407
	ProxEmbed	0.535	0.541	0.562	0.443	0.492	0.503	0.735	0.801	0.803	0.498	0.678	0.711	0.672	0.690	0.701	0.251	0.283	0.297	0.522	0.593	0.601
	Metapath2vec	0.394	0.395	0.396	0.375	0.378	0.390	0.572	0.589	0.595	0.435	0.436	0.441	**0.691**	0.683	0.649	0.259	0.281	0.275	0.431	0.497	0.502
	DAG-LSTM	0.498	0.522	0.521	0.352	0.450	0.457	0.275	0.537	0.648	0.340	0.581	0.614	0.573	0.652	0.702	0.246	0.260	0.277	0.570	0.591	0.594
	IPE-α	0.551	0.566	0.568	**0.519**	0.511	0.522	**0.765**	0.764	0.749	**0.662**	0.701	0.695	0.616	0.582	**0.708**	0.251	0.272	0.283	0.590	0.614	0.633
	IPE-α-β	0.526	0.531	0.556	0.482	0.472	0.501	0.359	0.493	0.715	0.311	0.635	0.706	0.541	0.504	0.621	0.232	0.242	0.273	0.573	0.609	0.624
	IPE-α-β-η	0.524	0.525	0.547	0.457	0.479	0.431	0.321	0.418	0.511	0.350	0.538	0.713	0.539	0.540	0.588	0.237	0.245	0.269	0.579	0.601	0.612
	IPE	**0.560**	**0.573**	**0.574**	0.490	**0.523**	**0.525**	0.491	**0.806**	**0.808**	0.622	**0.719**	**0.721**	0.585	**0.692**	0.705	0.245	0.288	**0.302**	**0.593**	**0.628**	**0.639**

SRW tends to have a higher complexity, since in each step of its gradient update, it does random walk over the whole graph. Hence, we were unable to produce results within a reasonable time (*e.g.*, a week) on the Taobao dataset, which has millions of nodes.

DeepWalk has lower performance than ProxEmbed, DAG-LSTM and IPE. As suggested in [22], using DeepWalk for proximity search is an indirect approach. Comparatively, ProxEmbed, DAG-LSTM and IPE all try to directly learn proximity embedding from the connecting structure between two nodes.

Metapath2vec has a comparable performance with DeepWalk. Surprisingly, it seems not to benefit much from its meta-path patterns. This is possibly because the meta-path patterns are limited and they have to be engineered, which are unlikely to be optimal.

The results of ProxEmbed show the usefulness of modeling paths by deep learning for proximity search. As there is no interactions among paths, only modeling these independent paths makes its performance worse than IPE. Surprisingly, for DAG-LSTM, its DAG structure seems not to help the prediction much. This is possibly because the DAG structure does not take into account those distance awareness and node heterogeneity issues into consideration.

Generally, IPE outperforms not only the competing baselines, but also its degenerated version. It validates the effectiveness of our modeling the interactive-paths structure. It also validates the need to consider each task characteristics, including path heterogeneity, distance awareness and node heterogeneity. It is worth noting that, distance awareness appears to be critical for the performance, as both IPE-α-β and IPE-α-β-η are significantly worse than IPE-α. This is understandable that, in semantic proximity search, a longer distance indicates a weak relation, thus likely to be noise in prediction.

Parameter sensitivity. In Fig. 6, we report the IPE performances w.r.t. its hyper-parameters, using 10K training tuples on Taobao and 100 training tuples on the other datasets. Firstly, embedding dimension $d = 32$ seems to achieve good results over LinkedIn, Facebook and DBLP, whereas $d = 128$ gives better results on Taobao. This difference is due to the fact that, Taobao network is much larger than the other three networks, hence we need a higher dimension for the embedding to capture enough information. Secondly, path discount α tends to achieve the best results at different values on different datasets. Empirically, we found that $\alpha \in [0.25, 0.75]$ often gave good results. Finally, distance discount β is sensitive to the performance, which shows the importance of distance discount in the IPE model. It tends to achieve the best results when $\beta \in [0.1, 1.0]$. When $\beta > 1$, it over-penalizes the node distance and the path length in interactive GRU. When $\beta < 0.01$, it underutilizes the distance awareness.

8 CONCLUSION

In this paper, we studied the semantic proximity search task on heterogeneous graphs. The state-of-the-art usually took a path-based approach, and measure the proximity between two nodes by their connecting paths. Despite the success, the prior work often modeled the paths separately, which overlooked the path interactions. We proposed a novel concept of interactive paths to model the path interdependencies, and designed a cycle-free shuffling mechanism to efficiently construct interactive-paths structures from the offline sampled paths. We further developed a novel interactive paths embedding (IPE) framework, which used an effective interactive GRU mechanism to learn a relation representation between two nodes for proximity search. We tested IPE on four different types of heterogeneous graphs, ranging from professional network to e-commerce network. Our extensive results on seven different semantic relations showed that, our IPE generally outperformed the state-of-the-art baselines.

In the future, we plan to extend IPE to handle attributes and dynamics in heterogeneous networks for semantic proximity search.

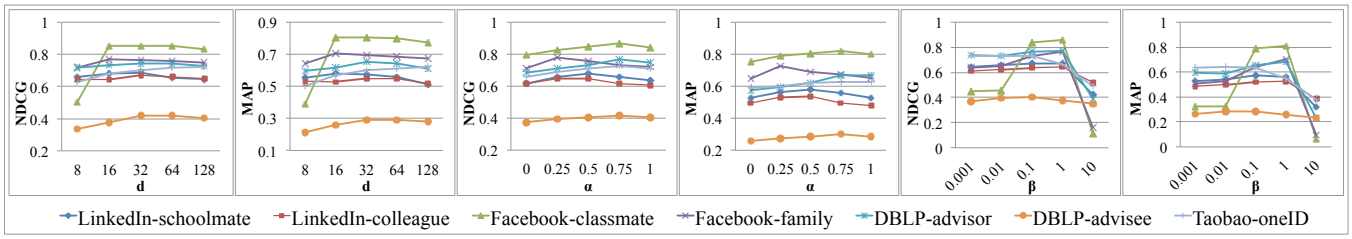


Figure 6: Impact of hyper-parameters: embedding dimension$d$, path discount α, distance discount β.

ACKNOWLEDGMENTS

We thank the support from: Zhejiang Science and Technology Plan Project (No. 2015C01027); National Natural Science Foundation of China (No. 61602405); National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme; and Alibaba Innovative Research program.

REFERENCES

[1] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*. 635–644.
[2] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*. 2787–2795.
[3] 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).
[4] Sandro Cavallari, Vincent W Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2017. Learning community embedding with community detection and node embedding on graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 377–386.
[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).
[6] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *ICML*. 2702–2711.
[7] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD*. 135–144.
[8] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent Neural Network Grammars. In *NAACL HLT*. 199–209.
[9] Yuan Fang, Wenqing Lin, Vincent W Zheng, Min Wu, Kevin Chen-Chuan Chang, and Xiao-Li Li. 2016. Semantic proximity search on graphs with metagraph-based learning. In *ICDE*. 277–288.
[10] Alberto García-Durán and Mathias Niepert. 2017. Learning Graph Representations with Embedding Propagation. In *NIPS*. 5125–5136.
[11] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2007. Multi-Dimensional Recurrent Neural Networks. *CoRR* abs/0705.2011 (2007).
[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. 855–864.
[13] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1025–1035.
[14] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *WWW*. 271–279.
[15] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. 2016. Grid Long Short-Term Memory. In *ICLR*.
[16] Daphne Koller and Nir Friedman. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.
[17] Ni Lao and William W. Cohen. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine Learning* 81, 1 (2010), 53–67.
[18] Rui Li, Chi Wang, and Kevin Chen-Chuan Chang. 2014. User profiling in an ego network: co-profiling attributes and relationships. In *WWW*. 819–830.
[19] Pengfei Liu, Xipeng Qiu, Jifan Chen, and Xuanjing Huang. 2016. Deep Fusion LSTMs for Text Semantic Matching.. In *ACL (1)*.
[20] Pengfei Liu, Xipeng Qiu, Yaqian Zhou, Jifan Chen, and Xuanjing Huang. 2016. Modelling Interaction of Sentence Pair with Coupled-LSTMs. In *EMNLP*. 1703–1712.
[21] Zemin Liu, Vincent W Zheng, Zhou Zhao, Hongxia Yang, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. 2018. Subgraph-augmented Path Embedding for Semantic User Search on Heterogeneous Social Network. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1613–1622.
[22] Zemin Liu, Vincent W Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. 2017. Semantic Proximity Search on Heterogeneous Graph by Proximity Embedding. In *AAAI*. 154–160.
[23] Zemin Liu, Vincent W Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. 2018. Distance-aware dag embedding for proximity search on heterogeneous graphs. AAAI.
[24] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2016. Multi-task Sequence to Sequence Learning. In *ICLR*.
[25] Julian J. McAuley and Jure Leskovec. 2012. Learning to Discover Social Circles in Ego Networks. In *NIPS*. 548–556.
[26] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*. 2014–2023.
[27] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding.. In *KDD*. 1105–1114.
[28] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. 701–710.
[29] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. 2017. Struc2Vec: Learning Node Representations from Structural Identity. In *KDD*. 385–394.
[30] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. 2015. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664* (2015).
[31] Baoxu Shi and Tim Weninger. 2017. ProjE: Embedding Projection for Knowledge Graph Completion. In *AAAI*. 1236–1242.
[32] Yu Shi, Po-Wei Chan, Honglei Zhuang, Huan Gui, and Jiawei Han. 2017. PReP: Path-Based Relevance from a Probabilistic Perspective in Heterogeneous Information Networks. In *KDD*. 425–434.
[33] Bing Shuai, Zhen Zuo, Bing Wang, and Gang Wang. 2017. Scene Segmentation with DAG-Recurrent Neural Networks. *T-PAMI* (2017).
[34] 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. *PVLDB* 4, 11 (2011), 992–1003.
[35] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *ACL*. 1556–1566.
[36] Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *CoRR* abs/1605.02688 (may 2016).
[37] Cunchao Tu, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2017. TransNet: Translation-Based Network Representation Learning for Social Relation Extraction. In *IJCAI*. 2864–2870.
[38] Julian R Ullmann. 1976. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)* 23, 1 (1976), 31–42.
[39] Chi Wang, Jiawei Han, Yuntao Jia, Jie Tang, Duo Zhang, Yintao Yu, and Jingyi Guo. 2010. Mining advisor-advisee relationships from research publication networks. In *KDD*. 203–212.
[40] Jia Wang, Vincent W Zheng, Zemin Liu, and Kevin Chen-Chuan Chang. 2017. Topological recurrent neural network for diffusion prediction. *Proceedings of The IEEE International Conference on Data Mining (ICDM)* (2017).
[41] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAAI*. 1112–1119.
[42] Muhan Zhang and Yixin Chen. 2017. Weisfeiler-Lehman Neural Machine for Link Prediction. In *KDD*. 575–583.
[43] Xiao-Dan Zhu, Parinaz Sobhani, and Hongyu Guo. 2016. DAG-Structured Long Short-Term Memory for Semantic Compositionality.. In *HLT-NAACL*. 917–926.