# Subgraph-augmented Path Embedding for Semantic User Search on Heterogeneous Social Network

Zemin Liu
Zhejiang University
China

Vincent W. Zheng*
Advanced Digital Sciences Center
Singapore

Zhou Zhao
Zhejiang University
China

Hongxia Yang
Alibaba Group
China

Kevin Chen-Chuan Chang
University of Illinois at Urbana-Champaign
USA

Minghui Wu
Zhejiang University City College
China

Jing Ying
Zhejiang University
China

## ABSTRACT

Semantic user search is an important task on heterogeneous social networks. Its core problem is to measure the proximity between two user objects in the network w.r.t. certain semantic user relation. State-of-the-art solutions often take a ~~path-based approach~~, which uses the sequences of objects connecting a query user and a target user to measure their proximity. Despite their success, we assert that ~~path~~ as a low-order structure ~~is insufficient to capture the rich semantics~~ between two users. Therefore, in this paper we introduce a new concept of *subgraph-augmented path* for semantic user search. Specifically, we consider sampling a set of object paths from a query user to a target user; then in each object path, we replace the linear object sequence between its every two neighboring users with their shared subgraph instances. Such subgraph-augmented paths are expected to leverage both path's distance awareness and subgraph's high-order structure. As it is non-trivial to model such subgraph-augmented paths, we develop a Subgraph-augmented Path Embedding (SPE) framework to accomplish the task. We evaluate our solution on six semantic user relations in three real-world public data sets, and show that it outperforms the baselines.

## CCS CONCEPTS

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

## KEYWORDS

Heterogeneous Network; Subgraph-augmented Path Embedding

**ACM Reference Format:**
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 Web Conference (WWW 2018).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178876.3186073

*Corresponding author.


*WWW 2018, April 23–27, 2018, Lyon, France*

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186073

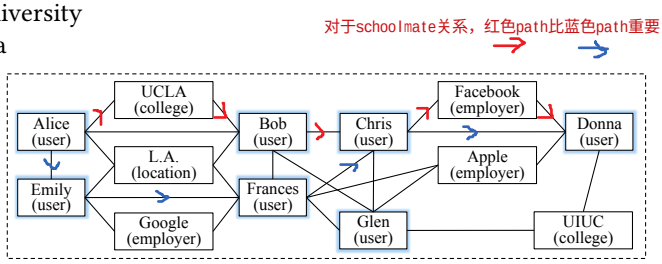


**Figure 1: Semantic user search on a heterogeneous social network with rich user interactions with different objects.**

## 1 INTRODUCTION

Heterogeneous social networks are prevalent nowadays [37]. Because social networks are human-centric, it is common to observe that users interact with many other types of objects. For example, as shown in Fig. 1, on a social network, user interact with not only other users, but also college, location and employer. These different types of interactions suggest different semantics of the user-user relationships. For example, Alice and Bob both attend UCLA, thus they are *schoolmates*; whereas Chris and Donna both work for Facebook, thus they are *colleagues*. Therefore, it gives us a unique opportunity to do *semantic user search*. In general, semantic user search is a task that given a query user (*e.g.*, Alice) on a heterogeneous social network and a semantic relation (*e.g.*, *schoolmates*), we want to find the other users (*e.g.*, Bob) that meet that relation with the query user. Such semantic user search is very useful [19, 22]. For example, we can use it to find colleagues, schoolmates and families on social networks such as Facebook and LinkedIn, or find advisors and advisees on academic networks such as DBLP.

Traditionally, *path-based* approach is used for solving semantic user search. This is because in semantic user search, the target user is often not immediately linked to the query user. A plausible choice is then to consider paths from the query user to the target user, and see whether they match the desired semantic relation. For example, Meta-Path Proximity (MPP) [30] first relies on domain

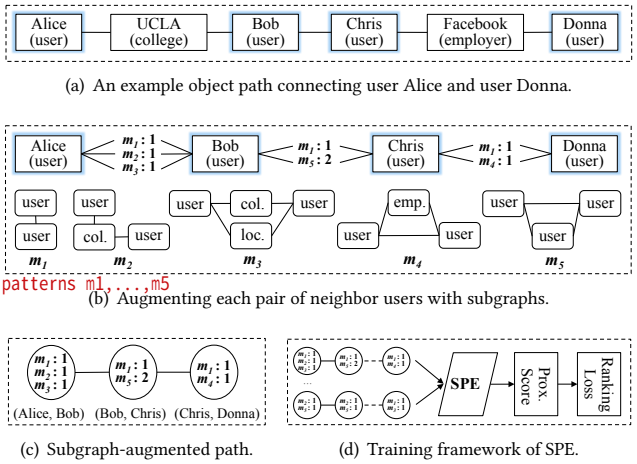


**Figure 2: Combining path's distance awareness and subgraph's higher-order structure for semantic user search.**

experts to specify a few path patterns (*i.e.*, metapaths) that indicate the desired semantic relation, and then enumerates the number of metapath instances between the query user and a target user. Path Ranking Algorithm (PRA) [18] first enumerates bounded-length (relation) path patterns; then it recursively defines a score for each path pattern; finally, for a target node, its proximity to the query node is computed as a linear combination of its corresponding path instances. Recently deep learning starts to exploit learning representations for the paths between a query node and a target node, and then using them for proximity estimation. For example, ProxEmbed [20] first samples a number of paths from the query node to the target node, and then uses a recurrent neural network to embed each path as a vector; finally it aggregates multiple path embedding vectors for proximity estimation.

Despite the success of such a path-based approach, we assert that ~~path as a low-order structure~~ is ~~insufficient~~ to capture the rich semantics between two users. ~~Consider~~ an object path connecting a query user Alice and a target user Donna in ~~Fig. 2(a).~~ In fact, Alice and Bob not only attended the same college UCLA, but also live in the same city L.A.. Such information is missing in the path, but it is possible to be captured by some higher-order subgraph structure. We are inspired by the state-of-the-art work on exploting subgraph patterns to organize complex networks [3]. Suppose we already have some offline mined subgraph patterns in Fig. 2(b), such as user-user ($m_1$), user-college-user ($m_2$), user-college & location-user ($m_3$) and so on. Then we can replace the linear object sequence between Alice and Bob with richer subgraph instances for $m_1$, $m_2$ and $m_3$. In this way, we have a more complete picture of the semantic relation between Alice and Bob. We envision that, once we better understand the semantic relation between every two neighboring users in a path, we can better estimate the proximity between the query user and the target user. Note that, we focus on augmenting the neighoring user objects only. There are two reasons of avoiding augmenting any two neighboring objects regardless of their types. Firstly, in semantic user search, we wish to directly model the semantic relation between users. Secondly, by constraining the subgraph patterns to involve two users, we can significantly reduce the number of subgraph patterns and thus greatly improve the efficiency in offline subgraph indexing, as suggested in [11].

In this paper, we introduce a new concept of *subgraph-augmented path* for semantic user search. Specifically, we consider sampling a set of object paths from a query user to a target user; then in each object path, we replace the linear object sequence between its every two neighboring users with their shared subgraph instances. Such subgraph-augmented paths are expected to leverage both path's *distance awareness* (*i.e.*, able to model multi-hop connections between a query user and a target user) and subgraph's *high-order structure* (*i.e.*, able to use more complex structures than linear sequences). Given these subgraph-augmented paths as new inputs, we aim to embed them into low-dimensional vectors and then aggregate them for proximity estimation. In this work, we assume the subgraph patterns and subgraph instances are given as inputs. Such an assumption is mild in practice, because frequent subgraphs are useful, and often offline mined as basic graph indexing to support many useful applications [10]. For example, frequent subgraphs are used for fraud detection in Alibaba[1], and user/content recommendation in Twitter [13]. There also exist efficient algorithms to mine frequent subgraph patterns and match subgraph instances [9, 31].

However, embedding subgraph-augmented path (or, *s-path* for abbreviation) is not trivial. A straightforward approach is to apply ProxEmbed [20]. For each s-path, we represent each of its node as a key-value pair (Def. 3.5), where the key is the end user pair, and the value denotes the number of each subgraph instances shared by these two users. Then we apply a recurrent neural network to encode each node in the s-path, and finally we pool all the output vectors of the nodes as one. For multiple s-paths, we use distance discounted pooling to de-emphasize those long paths. Yet, such a straightforward approach overlooks two challenges. First of all, subgraphs are structural and noisy. To represent a node in an s-path, we have to take into account the structure of each subgraph, as well as the fact that not all the subgraphs are useful for a particular semantic user relation (*e.g.*, $m_5$ is less indicative than $m_2$ for *schoolmates*). Secondly, s-paths are noisy in and among themselves. In each s-path, its nodes are not equally useful for a semantic relation; *e.g.*, if Alice and Donna are truly *schoolmates*, then node (Alice, Bob) in the s-path, which implies a *schoolmates* relation, is more important than the other nodes in the same s-path. Similarly, not all the s-paths are equally useful either; *e.g.*, an s-path constructed from Alice–Emily–Frances–Donna in Fig. 1 is less indicative than the one in Fig. 2(c) for the *schoolmates* relation, since it has no clear signal for that relation.

To model s-paths for semantic user search, we develop a novel *Subgraph-augmented Path Embedding* (SPE) framework. In SPE, we first represent an object in an s-path with an aggregation of its subgraphs' embedding vectors. Specifically, we construct a structural similarity matrix among the subgraphs, and based on this matrix we learn an embedding vector for each subgraph to preserve the structural similarity. Then, we introduce the *attention* mechanism [40] to automatically weigh the subgraphs in aggregation to represent each s-path's node. To deal with the noise in and among s-paths, we also introduce the attention mechanism to automatically weigh

[1] https://www.alibabacloud.com/forum/read-492

each node in an s-path and each s-path between two users. In all, we have a three-layer attention architecture on subgraphs, s-path's nodes and s-paths. Finally, we embed all the s-paths together into a vector, based on which we compute the proximity score and thus the ranking loss for model training.

We summarize our contributions as follows.

• We introduce a new concept of subgraph-augmented path, which for the first time systematically combine path's distance awareness and subgraph's high-order structure to solve semantic user search.

• We develop a novel SPE framework to embed these subgraph-augmented paths for user proximity estimation.

• We evaluate SPE on six semantic user relations in three public data sets, and show it outperforms the state-of-the-art baselines.

## 2 RELATED WORK

Earlier graph semantic search work such as Personalized PageRank [17] and SimRank [16] often consider homogeneous networks as input, and they do not differentiate semantic classes. Recent work starts to consider the rich network structure in heterogeneous networks. For example, Supervised Random Walk (SRW) [1] tries to bias a random walk on the network, so as to ensure the resulting ranking result on the network to be consistent with the ground truth. MPP [30] and PRA [18] try to match the paths between a query node and a target node with some supervision (*i.e.*, either metapath patterns or ground truth labels) to see whether certain semantic relation holds. Meta-Graph Proximity (MGP) [11] considers more general subgraph patterns than metapaths. It first identifies a few frequent subgraph patterns as metagraphs; then it leverages the supervision to automatically learn which metagraph is indicative for a desired semantic relation; finally it counts the number of indicative metagraphs between two users to measure their proximity. Thanks to the higher-order structure of subgraph, MGP improves the path-based methods such as SRW and MPP. But MGP lacks distance awareness; *i.e.*, if a query user and a target user are multi-hop away and no metagraph is shared by them, then their proximity becomes (close to) zero. Both MPP and MGP can be seen as exploiting explicit graph features for proximity estimation.

With the development of neural networks, some recent studies start to consider learning "implicit" graph features for proximity estimation. For example, in graph embedding, DeepWalk [27], LINE [32], node2vec [12], and many more [23, 25, 26] all try to learn an embedding vector for each node in the graph, which can preserve the graph structure. In particular, metapath2vec [8] extends DeepWalk by using metapath to guide the random walk for better node embedding. Struc2vec [28] exploits additional structural equivalence of two nodes for node embedding. A comprehensive survey of graph embedding is recently available [5]. One possible approach to make use of such a node-level embedding for semantic search is to aggregate two nodes' embedding vectors (*e.g.*, first applying a Hadamard product and then multiplying it with a parameter vector) for estimating their proximity. However, such an approach is considered as "indirect", as suggested by ProxEmbed [20], since it does not directly encode the network structure between two possibly distant nodes. In contrast, ProxEmbed expresses the network structure between two objects by a set of paths connecting them, and directly encodes these paths into a proximity embedding vector. However, since ProxEmbed takes object paths as input, it is unable to leverage the readily available subgraphs' high-order structure. Similarly, although D2AGE [21] manages to model multiple object paths as one directed acyclic graph for proximity embedding, it is also unable to leverage the offline mined subgraphs.

In the line of graph embedding, there exist several related, yet different concepts. First of all, some recent work exploits the concept of "high-order proximity" in graph embedding [6, 38]. They use higher-order reachability to construct adjacency matrix of a graph, and then run node embedding on this adjacency matrix. There are two major differences with our method: 1) they do not exploit the high-order structure of subgraph patterns; 2) they consider node embedding instead of path embedding. Secondly, some other work models high-order structure by graph convolution [14, 24]. These methods are powerful to capture local graph patterns, but it is not clear how to incorporate the path's distance awareness for the task of semantic user search. Besides, they cannot leverage the readily available subgraph patterns. Thirdly, graph kernel methods [7], especially Weisfeiler-Lehman graph kernels [29, 41], try to measure the similarity between two (small) graphs w.r.t. their structures. They often exploit some predefined subgraph structures, such as edges, subtrees and shortest paths. But their goal of measuring similarity between graphs is very different from ours of measuring proximity between nodes. Besides, it is also not clear how to adapt their methods with path distance awareness for our task. Finally, in the field of knowledge base, recent work such as TransE [4], TransH [39] and TransNet [34] has greatly advanced the study of knowledge embedding. However, these methods are not directly applicable to our task, due to the different problem settings. They often require the edges to have explicit descriptions, and aim to generate node/edge embedding instead of path embedding. Besides, it is also not clear how to extend these methods with the subgraph's high-order structure and the path's distance awareness for our task.

## 3 PROBLEM FORMULATION

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

*Definition 3.1.* A **heterogeneous network** is $G = (V, E, C, \tau)$, where $V$ is a set of objects, $E$ is a set of edges between the objects in $V$, $C = \{c_1, ..., c_K\}$ is a set of distinct object types, and $\tau : V \to C$ is an object type mapping function.

For example, in Fig. 1, we have $C = \{\text{user}, \text{college}, \text{location}, \text{employer}\}$. For an object of Alice, $\tau(\text{Alice}) = \text{user}$.

*Definition 3.2.* A **subgraph pattern** is $m = (C_m, E_m)$, where $C_m$ is a set of object types, $E_m$ is a set of edges between object types in $C_m$.

For example, Fig. 2(b) lists five subgraph patterns $m_1, ..., m_5$. We denote the set of possible subgraph patterns on $G$ as $M$. As discussed in Sect. 1, we consider $\mathcal{M}$ as the frequent subgraph patterns offline mined from $G$, and readily available as the input. Note that unlike $C$ in $G$, the object types in $C_m$ may be nondistinct. E.g., for subgraph pattern $m_1$ in Fig. 2(b), $C_{m_1} = \{\text{user}, \text{user}\}$.

*Definition 3.3.* An object subgraph $g = (V_g, E_g)$ is a **subgraph instance** of $m = (C_m, E_m)$, if there exists a bijection between the node set of $g$ and $m$, $\phi : V_g \to C_m$, such that

**Table 1: Notations used in this paper.**

| Notation | Description |
|---|---|
| $G, V, E, C$ | Network $G$, objects $V$, edges $E$, object types $C$ |
| $\mathcal{M}$ | Set of frequent subgraph patterns from $G$ |
| $\mathcal{I}$ | Set of subgraph instances for $\mathcal{M}$ on $G$ |
| $\mathcal{D}$ | Set of training tuples |
| $\mathcal{P}$ | Set of sampled object paths on $G$ |
| $\hat{\mathcal{P}}$ | Set of subgraph-augmented paths constructed from $\mathcal{P}$ |
| $r$ | A subgraph-augmented node |
| $\mathbf{x}$ | Embedding vector for a subgraph |
| $\mathbf{z}$ | Embedding vector for a subgraph-augmented path |
| $\mathbf{f}$ | Proximity embedding vector between two users |
| $\pi$ | Proximity score |
| $S$ | Structural similarity matrix of subgraph patterns |
| $\gamma, \ell$ | Number of paths per object $\gamma$, walk length $\ell$ |
| $d, d'$ | Embedding dimensions |
| $\zeta$ | Average number of subgraph instances for a user |

- $\forall v \in V_g$, we have $\tau(v) = \phi(v)$;
- $\forall v, u \in V_g$, we have $(v,u) \in E_g$ iff $(\phi(v), \phi(u)) \in E_m$.

For example, Alice-UCLA-Bob is an instance of $m_2$ in Fig. 2(b). We denote the set of possible subgraph instances for $\mathcal{M}$ on $G$ as $\mathcal{I}$. For each $m \in \mathcal{M}$, there may be multiple subgraph instances on $G$.

*Definition 3.4.* An **object path** on $G$ is a sequence of objects $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_t$, where each $v_i \in V$, and $t$ is the path length.

For example, the sequence in Fig. 2(a) is an object path.

*Definition 3.5.* A **subgraph-augmented node** (or "s-node" for abbreviation) $r$ for two user objects $u, v \in V$ is a key-value pair, whose key is $(u,v)$ and value is a set of tuples $\{m_1 : e_1, ..., m_l : e_l\}$. $\forall m_i \in M$, $e_i$ is the number of $m_i$'s instances between $u$ and $v$.

For example, in Fig. 2(b), the s-node for Alice and Bob is defined as $r.key = (\text{Alice}, \text{Bob})$, $r.value = \{m_1 : 1, m_2 : 1, m_3 : 1\}$. We skip the subgraphs with zero instance in $r.value$. As discussed in Sect. 1, we choose only augmenting two user objects with their shared subgraphs to both focus on user-user semantic relation and improve subgraph indexing efficiency. We leave as future work augmenting any two objects regardless of their types for semantic search.

*Definition 3.6.* An **subgraph-augmented path** (or "s-path" for abbreviation) is a sequence of s-nodes $r_1 \rightarrow r_2 \rightarrow ... \rightarrow r_t$.

For example, the sequence in Fig. 2(c) is an s-path.

**Problem inputs and outputs**. For *inputs* of our model, we have a heterogeneous network $G$, a set of readily available frequent subgraph patterns $\mathcal{M}$ and their subgraph instances $\mathcal{I}$ on $G$, and finally a set of training tuples $\mathcal{D} = \{(q_i, v_i, u_i) : i = 1, ..., n\}$, where for each query user object $q_i$, user $v_i$ is closer to $q_i$ than user $u_i$. Besides, we also offline sample some object paths from $G$ as inputs. We take a similar approach as DeepWalk [27] for path sampling. Specifically, starting from each object in $G$, we randomly sample $\gamma$ object paths, each of length $\ell$. As a result, we obtain a set of object paths, denoted as $\mathcal{P}$. These object paths are indexed to support efficient training and testing. For each query object $q \in \{q_1, ..., q_n\}$ and a corresponding target object $v \in \{v_1, ..., v_n, u_1, ..., u_n\}$, 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 each object path from $q$ to $v$, we use the subgraph patterns $\mathcal{M}$ and their subgraph instances $\mathcal{I}$ to construct a subgraph-augmented path. We will introduce the details of s-path construction in Sect. 4.

For *outputs* of our model, we generate a subgraph-augmented path embedding vector $\mathbf{z}(q,v) \in \mathbb{R}^d$ for each s-path between $q$ and $v$, where $d > 0$ is the embedding dimension. Since there are multiple s-paths between $q$ and $v$, we will reasonably aggregate multiple $\mathbf{z}(q,v)$'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 will discuss how to compute $\mathbf{z}(q,v)$ and $\mathbf{f}(q,v)$ by some hierarchical neural network model in Sect. 5. Finally, 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 hierarchical neural network parameters for getting $\mathbf{z}(q,v)$ and $\mathbf{f}(q,v)$; 2) the proximity estimation parameter $\boldsymbol{\theta}$. In training, we aim to learn these model parameters, such that $\pi(q_i, v_i) \geq \pi(q_i, u_i)$ for each $(q_i, v_i, u_i) \in \mathcal{D}$. We will introduce the details of training algorithm in Sect. 6. Note that in offline training, we only need to compute subgraph-augmented path embedding for those $(q_i, v_i)$ and $(q_i, u_i)$ for $i = 1, ..., n$, instead of all the possible object pairs in $G$. In online testing, given a random query user $q$ in $G$, we will quickly extract from $\mathcal{P}$ a set of sample object paths from $q$ to each possible target user $v$ in $G$. Then we construct the s-paths with $M$, and apply our model to compute the $\pi(q,v)$ for each target $v$ for ranking.

## 4 S-PATH CONSTRUCTION

We introduce how to construct subgraph-augmented paths for a query user $q$ and a target user $v$, based on: 1) a set of already sampled object paths from $q$ to $v$ on $G$; 2) a set of readily available frequent subgraph patterns $\mathcal{M}$ and their subgraph instances $\mathcal{I}$ on $G$.

**Running example**: Take Fig. 2(b) as an example. For an object path of Alice–UCLA–Bob–Chris–Facebook–Donna, we first extract every pair of neighboring users by collapsing the non-user objects in the path. As a result, we get (Alice, Bob), (Bob, Chris) and (Chris, Donna). For each pair of neighboring users, we try to get each user's involved subgraph instances. In Fig. 2(b), we have listed five possible subgraph patterns $m_1, ..., m_5$; due to space limit, we skip listing their subgraph instances on the heterogeneous network in Fig. 1. Take (Alice, Bob) as an example. For Alice, she has involved four subgraph instances w.r.t. $m_1$, $m_2$ and $m_3$. Specificaly, for $m_1$, Alice has two subgraph instances in Fig. 1: Alice–Bob and Alice–Emily. For $m_2$, Alice has one subgraph instances in Fig. 1: Alice–UCLA–Bob. For $m_3$, Alice has one subgraph instance in Fig. 1: Alice–UCLA & L.A.–Bob. To replace the linear object path Alice–UCLA–Bob with subgraphs, we want to find all the subgraph instances shared by Alice and Bob. An easy way to find such shared subgraph instances is to scan all the subgraph instances of Alice and see whether they contain Bob. As we can see, Alice and Bob share one instance of $m_1$, one instance of $m_2$ and one instance of $m_3$. Then we construct a subgraph-augmented node (s-node) $r_1$, with $r_1.key \leftarrow (\text{Alice}, \text{Bob})$

**Algorithm 1** SPathConstruct

**Require:** a set of object paths $\mathcal{P}$, a query user $q$, a target user $v$, a set of subgraph patterns $M$ and its instances $\mathcal{I}$.
**Ensure:** a set of subgraph-augmented paths $p$'s for $(q, v)$.

```
1: Q(q,v) ← GetSubpaths(P,q,v);
2: for each object path o_i ∈ Q(q,v) do
3:    p_i ← ∅;
4:    {(u,u')} ← GetNeighborUserPairs(o);
5:    for each (u,u') do
6:       I_{u,u'} ← GetSharedSubgraphInstance(I,u,u');
7:       r.key ← (u,u');
8:       r.value ← CountSubgraphInstance(I_{u,u'},M);
9:       p_i.append(r);
10: return {p_i}.
```

and $r_1.value \leftarrow [m_1:1, m_2:1, m_3:1]$. Similarly, we can construct an s-node $r_2$ for (Bob, Chris) and an s-node $r_3$ for (Chris, Donna). In the end, we obtain an subgraph-augmented path (s-path) of $p: r_1 \rightarrow r_2 \rightarrow r_3$.

We abstract the above running example, and summarize the s-path construction algorithm in Alg. 1. In line 1, we first extract all the object subpaths from $q$ to $v$ from $\mathcal{P}$. Here we overload the function "GetSubpaths" to get different subpaths for symmetric and asymmetric semantic relations. In line 4, for each resulting object path, we get all the neighboring user pairs. In line 6, for each pair of neighboring users, we identify their shared subgraph instances from the pre-indexed subgraph instance set $\mathcal{I}$. After that, we construct an s-node in lines 7 and 8, and further append it to the previous s-node sequence to form an s-path in line 9.

**Complexity analysis**: as each object path's length is bounded by $\ell$, getting the neighboring user pairs in line 4 takes $O(\ell)$. The straightforward approach to get shared subgraph instances between two users has to scan through all the subgraph instances of one user. Denote the average number of subgraph instance for a user on $G$ as $\zeta$. Because in practice the subgraph patterns have limited sizes (*e.g.*, less than six in our experiments), the above straightforward approach to run line 6 takes $O(\zeta)$. This complexity can be further reduced; as suggested by [11], if in the subgraph indexing stage only those subgraph patterns that involved at least two users were considered, then both the number of subgraph patterns and the number of subgraph instances can be significantly reduced. By a sophisticated indexing of which subgraph instance matching which two users, we may reduce line 6's complexity to a constant. In all, constructing an s-path from an object path takes $O(\ell + \zeta)$.

## 5 S-PATH EMBEDDING

We first introduce how to embed each subgraph-augmented path to a vector $\mathbf{z}(q, v)$, and later aggregate multiple such vectors into a single one $\mathbf{f}(q, v)$. We design a hierarchical neural network for subgraph-augmented path embedding (SPE) as shown in Fig. 3. As motivated in Sect. 1, we will take multiple factors into the design. First of all, we embed each subgraph with structural information. Then, we aggregate the subgraph embedding in each subgraph-augmented node (s-node) with attention to obtain an s-node embedding. To model the sequential information of each s-path, we

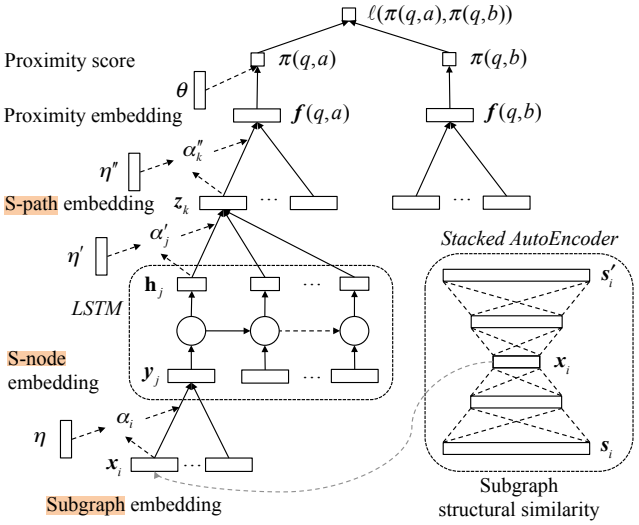


**Figure 3: Subgraph-augmented path embedding.**

also employ a recurrent neural network architecture to learn an s-path's embedding. Finally, we aggregate all the s-paths' embedding with attention to obtain an overall proximity embedding vector for $(q, v)$. Next we introduce each step of embedding in Fig. 3.

**Subgraph Embedding.** To take the subgraph structure into account, we are inspired by the structural deep network embedding [38] to consider embedding the subgraphs from a *subgraph structural similarity matrix*. In general, two subgraphs are similar if they share some common structures. Therefore, we adopt the widely used Maximum Common Subgraph (MCS) approach [35] to measure the similarity between two subgraphs. Given two subgraphs $m_i$ and $m_j$, we denote $m^*$ as their MCS. Then the structural similarity between two subgraphs is defined as

$$S(m_i, m_j) = \frac{(|C_{m^*}| + |E_{m^*}|)^2}{(|C_{m_i}| + |E_{m_i}|) \times (|C_{m_j}| + |E_{m_j}|)}. \quad (2)$$

If $m^*$ is bigger, $S(m_i, m_j)$ is bigger. We use stacked AutoEncoder [2] to learn an embedding $\mathbf{x}_i \in \mathbb{R}^d$ for each subgraph $m_i$. Denote $\mathbf{s}_i = [S(m_i, m_1), ..., S(m_i, m_{|M|})]^T$. For simplicity, we use a three-layer AutoEncoder to illustrate how we construct $\mathbf{x}_i$ from its $\mathbf{s}_i$. In particular, we define the subgraph embedding vector $\mathbf{x}_i$ for $m_i$ as

$$\mathbf{x}_i = \sigma(W^{(a)}\mathbf{s}_i + \mathbf{b}^{(a)}), \quad (3)$$

where $W^{(a)} \in \mathbb{R}^{d \times |\mathcal{M}|}$ and $\mathbf{b}^{(a)} \in \mathbb{R}^d$ are parameters; $\sigma(\cdot)$ is a sigmoid function. We reconstruct $\mathbf{s}_i$ from $\mathbf{x}_i$ by

$$\hat{\mathbf{s}}_i = \sigma(W^{(b)}\mathbf{x}_i + \mathbf{b}^{(b)}), \quad (4)$$

where $W^{(b)} \in \mathbb{R}^{|\mathcal{M}| \times d}$ and $\mathbf{b}^{(b)} \in \mathbb{R}^{|\mathcal{M}|}$ are also parameters. Finally, we minimize the reconstruction error:

$$\sum_{i=1}^{|M|} \|\hat{\mathbf{s}}_i - \mathbf{s}_i\|_2. \quad (5)$$

Although it is possible to optimize $\mathbf{x}_i$'s together with the proximity embedding later, in this paper we choose to optimize $\mathbf{x}_i$'s from $S$ separately, so as to keep the model simple.

**S-Node Embedding.** In general, there are multiple subgraphs involved in a subgraph-augmented node (s-node). To differentiate their contributions, we introduce an attention mechanism to automatically learn the weight for each subgraph in s-node embedding. For an s-node $r_j.value = \{m_1 : e_1, ..., m_l : e_l\}$, we compute the attention score $\alpha_i$ for each subgraph $m_i$ in $r_j$ as

$$\alpha_i = \frac{\exp(\beta_i)}{\sum_{j=1}^{l} \exp(\beta_j)}, \tag{6}$$

$$\beta_i = e_i \left[\boldsymbol{\eta} \cdot \sigma(Q\mathbf{x}_i + \mathbf{b})\right], \tag{7}$$

where $\boldsymbol{\eta} \in \mathbb{R}^{d'}$, $Q \in \mathbb{R}^{d' \times d}$ and $\mathbf{b} \in \mathbb{R}^{d'}$ are parameters. As a result, we compute the s-node embedding for $r_j$ as

$$\mathbf{y}_j = \sum_{i=1}^{l} \alpha_i \mathbf{x}_i. \tag{8}$$

**S-Path Embedding.** Once having the s-node embeddings in an s-path $p_k : r_1 \rightarrow ... \rightarrow r_t$, we learn the s-path embedding by LSTM (Long Short Term Memory) [15]. Formally, for each input s-node embedding $\mathbf{y}_j$, we output a vector $\mathbf{h}_j \in \mathbb{R}^{d'}$, by computing a series of neuron activations for an input gate $\mathbf{g}_j^{(i)}$, an forget gate $\mathbf{g}_j^{(f)}$, a memory cell state $\mathbf{g}_j^{(c)}$ and an output gate $\mathbf{g}_j^{(o)}$:

$$\mathbf{g}_j^{(i)} = \sigma(W^{(i)}\mathbf{y}_j + U^{(i)}\mathbf{h}_{j-1} + \mathbf{b}^{(i)}), \tag{9}$$

$$\mathbf{g}_j^{(f)} = \sigma(W^{(f)}\mathbf{y}_j + U^{(f)}\mathbf{h}_{j-1} + \mathbf{b}^{(f)}), \tag{10}$$

$$\mathbf{g}_j^{(c)} = \mathbf{g}_j^{(i)} \odot \tilde{\mathbf{g}}_j^{(c)} + \mathbf{g}_j^{(f)} \odot \mathbf{g}_{j-1}^{(c)}, \tag{11}$$

$$\mathbf{g}_j^{(o)} = \sigma(W^{(o)}\mathbf{y}_j + U^{(o)}\mathbf{h}_{j-1} + \mathbf{b}^{(o)}), \tag{12}$$

$$\mathbf{h}_j = \mathbf{g}_j^{(o)} \odot tanh(\mathbf{g}_j^{(c)}), \tag{13}$$

where $\tilde{\mathbf{g}}_{t,j}^{(c)} = tanh(W^{(c)}\mathbf{y}_j + U^{(c)}\mathbf{h}_{j-1} + \mathbf{b}^{(c)})$ and $\odot$ is an element-wise product. We denote the set of LSTM parameters as $\Theta^{(lstm)} = \{W^{(i)} \in \mathbb{R}^{d' \times d}, U^{(i)} \in \mathbb{R}^{d' \times d'}, \mathbf{b}^{(i)} \in \mathbb{R}^{d'}, W^{(f)} \in \mathbb{R}^{d' \times d}, U^{(f)} \in \mathbb{R}^{d' \times d'}, \mathbf{b}^{(f)} \in \mathbb{R}^{d'}, W^{(c)} \in \mathbb{R}^{d' \times d}, U^{(c)} \in \mathbb{R}^{d' \times d'}, \mathbf{b}^{(c)} \in \mathbb{R}^{d'}, W^{(o)} \in \mathbb{R}^{d' \times d}, U^{(o)} \in \mathbb{R}^{d' \times d'}, \mathbf{b}^{(o)} \in \mathbb{R}^{d'}\}$.

To differentiate the contributions of s-nodes, we also compute the attention score $\alpha'_j$ for each s-node $r_j$ as

$$\alpha'_j = \frac{\exp(\beta'_j)}{\sum_{k=1}^{t} \exp(\beta'_k)}, \tag{14}$$

$$\beta'_j = \boldsymbol{\eta}' \cdot \sigma(Q'\mathbf{h}_j + \mathbf{b}'), \tag{15}$$

where $\boldsymbol{\eta}' \in \mathbb{R}^{d'}$, $Q' \in \mathbb{R}^{d' \times d'}$ and $\mathbf{b}' \in \mathbb{R}^{d'}$ are parameters. As a result, we compute the s-path embedding for $p_k$ as

$$\mathbf{z}_k = \sum_{j=1}^{t} \alpha'_j \mathbf{h}_j. \tag{16}$$

**Proximity Embedding.** Once having the s-path embedding for each of the s-paths between $q$ and $v$, we can compute their proximity embedding. For a unified notation, we introduce $\hat{\mathcal{P}}(q,v)$ as the s-path set between $q$ and $v$. For asymmetric relations, we define $\hat{\mathcal{P}}(q,v)$ as the s-paths from $q$ to $v$. For symmetric relations, we define $\hat{\mathcal{P}}(q,v)$ as the s-paths both from $q$ to $v$ and from $v$ to $q$. To differentiate the contributions of s-paths, we compute the attention score $\alpha''_k$ for each s-path $p_k$ as

$$\alpha''_k = \frac{\exp(\beta''_k)}{\sum_{p_i \in \hat{\mathcal{P}}(q,v)} \exp(\beta''_i)}, \tag{17}$$

$$\beta''_k = \boldsymbol{\eta}'' \cdot \sigma(Q'' \cdot \mathbf{z}_k + \mathbf{b}''), \tag{18}$$

where $\boldsymbol{\eta}'' \in \mathbb{R}^{d'}$, $Q'' \in \mathbb{R}^{d' \times d'}$ and $\mathbf{b}'' \in \mathbb{R}^{d'}$ are parameters. Finally, we compute the proximity embedding for $(q,v)$ as

$$\mathbf{f}(q,v) = \sum_{p_k \in \hat{\mathcal{P}}(q,v)} \alpha''_k \mathbf{z}_k. \tag{19}$$

This $\mathbf{f}(q,v)$ encodes the information of all the subgraph-augmented paths between $q$ and $v$. It will be later used to estimate the proximity score of $q$ and $v$ by Eq. 1.

**Algorithm 2** SPETrain

**Require:** heterogeneous network $G$, a set of subgraph patterns $M$ and its instances $\mathcal{I}$, training tuples $\mathcal{D}$, number of paths per object $\gamma$, walk length $\ell$, embedding dimensions $d$ and $d'$, trade-off parameters $\{\lambda, \mu\}$.
**Ensure:** SPE model parameters $\Theta$.

```
1: Initialize an object path set P = ∅ and an s-path set P̂ = ∅;
2: for all v ∈ V do
3:   for i = 1 : γ do
4:     P ← P ∪ SamplePath(G, v, ℓ);
5: for all each (q, v, u) ∈ D do
6:   P̂(q, v) ← SPathConstruct(P, q, v, M, I);
7:   P̂(q, u) ← SPathConstruct(P, q, u, M, I);
8: B ← GenerateBatches(D);
9: for all batch b ∈ B do
10:  Initialize loss for batch b as L_b = 0;
11:  for all each (q, v, u) ∈ b do
12:    Compute f(q, v) with P̂(q, v), d and d' by Eq. 19;
13:    Compute f(q, u) with P̂(q, u), d and d' by Eq. 19;
14:    L_b = L_b + ℓ(π(q, v), π(q, u)), according to Eq. 20;
15:  L_b = L_b + μΩ(Θ);
16:  Update Θ based on L_b by stochastic gradient descent.
17: return Θ.
```

## 6 END-TO-END TRAINING

In training, for each tuple $(q_i, v_i, u_i)$, $\forall i = 1, ..., n$, we define a ranking loss based on the proximity scores $\pi(q_i, v_i)$ and $\pi(q_i, u_i)$. We define the ranking loss function as

$$\ell(\pi(q_i, v_i), \pi(q_i, u_i)) = -\log \sigma_\lambda(\pi(q_i, v_i) - \pi(q_i, u_i)), \tag{20}$$

where $\sigma_\lambda(x) = 1/(1+e^{-\lambda x})$ and $\lambda > 0$ is a parameter. We denote the parameter set of our three-layer attentions for subgraphs, s-nodes and s-paths as $\Theta^{(att)} = \{\boldsymbol{\eta}, Q, \mathbf{b}, \boldsymbol{\eta}', Q', \mathbf{b}', \boldsymbol{\eta}'', Q'', \mathbf{b}''\}$. In total, our model parameters are $\Theta = \{\Theta^{(lstm)}, \Theta^{(att)}, \boldsymbol{\theta}\}$. Our ultimate goal in training is to minimize

$$L(\Theta) = \sum_{i=1}^{n} \ell(\pi(q_i, v_i), \pi(q_i, u_i)) + \mu\, \Omega(\Theta), \tag{21}$$

where $\mu > 0$ is a trade-off parameter, $\Omega(\cdot)$ is a regularization function (*e.g.*, the sum of $l_2$-norm for each parameter in $\Theta$).

**Training algorithm**: we summarize the SPE training algorithm in Alg. 2. In lines 2–4, we sample object paths on $G$. In lines 6 and 7,

**Table 2: Summary of data sets and subgraphs.**

| Network | Objects | Edges | Types | Average degree | Subgraph patterns | Subgraph instances |
|---|---|---|---|---|---|---|
| LinkedIn | 65,925 | 220,812 | 4 | 6.7 | 173 | 604,848,383 |
| Facebook | 5,025 | 100,356 | 10 | 39.9 | 981 | 2,398,306,414 |
| DBLP | 165,728 | 928,513 | 5 | 11.2 | 88 | 740,682,735 |

we construct subgraph-augmented paths based on the object paths for each $(q, v)$ and $(q, u)$ in the training tuples. In line 8, we split the training tuples into batches, and then do batch stochastic gradient descent. In lines 12 and 13, we compute the proximity embedding vectors $\mathbf{f}(q, v)$ and $\mathbf{f}(q, u)$. In line 14, we compute the ranking loss $\ell(\pi(q, v), \pi(q, u))$. In lines 15 and 16, we accumulate the loss for batch $b$ and do stochastic gradient descent.

**Complexity analysis**: we analyze the time complexity for Alg. 2. In lines 2–4, we sample $\gamma$ object paths of length $\ell$ starting from each object in $G$, hence it takes $O(|V|\gamma\ell)$. In lines 5–7, we in total construct $|\hat{\mathcal{P}}|$ s-paths. According to Alg. 1, contructing an s-path takes $O(\ell + \zeta)$. Thus the complexity of lines 7-10 is $O(|\hat{\mathcal{P}}|(\ell + \zeta))$. In line 8, generating the batches takes $O(n)$. In line 12, computing proximity embedding $\mathbf{f}(q, v)$ requires three steps: 1) embedding an s-node, which takes $O(|\mathcal{M}|(d'd+d'))$ given at most $|\mathcal{M}|$ subgraphs; 2) embedding an s-path, which takes $O(\ell(d'd + d'^2 + d'))$ given an s-path's length of at most $\ell$; 3) proximity embedding, which takes $O(|\hat{\mathcal{P}}(q, v)|(d'^2 + d'))$ given $|\hat{\mathcal{P}}(q, v)|$ s-paths between $q$ and $v$. In lines 9–16, we essentially compute $\mathbf{f}(q, v)$'s and $\mathbf{f}(q, u)$'s for all the $(q, v, u) \in \mathcal{D}$, which in total takes $O(|\hat{\mathcal{P}}|((d'^2 + d') + \ell(d'd + d'^2 + d') + \ell(|\mathcal{M}|(d'd+d')))) = O(|\hat{\mathcal{P}}|(\ell d'^2+\ell|\mathcal{M}|d'd+\ell d'))$. Computing the loss in line 14 over all the training tuples takes $O(nd')$. Updating $\Theta$ in line 16 for $|\mathcal{B}|$ batches takes $O(|\mathcal{B}|(d'd + d'^2))$. Note that we compute the subgraph embedding offline once. It takes $O(|\mathcal{M}|^2)$ to construct the structural similarity matrix, and $O(|\mathcal{M}|d + |\mathcal{M}|)$ to learn the subgraph embedding, which in total is $O(|\mathcal{M}|^2 + |\mathcal{M}|d)$. In summary, the total complexity of Alg. 2 is $O(|\hat{\mathcal{P}}|\zeta + |\mathcal{M}|^2 + |\mathcal{M}|d + |\hat{\mathcal{P}}|\ell(d'^2 + |\mathcal{M}|d'd + d'))$. Since $|\hat{\mathcal{P}}| \leq |V|\gamma\ell$, the complexity of Alg. 2 becomes $O(|V|\gamma\ell(\zeta + \ell(d'^2 + |\mathcal{M}|d'd + d')) + |\mathcal{M}|^2)$.

## 7 EXPERIMENTS

**Heterogeneous social networks.** We conducted extensive experiments on three real-world data sets collected by previous studies, namely LinkedIn [19], Facebook [22] and DBLP [36]. Each data set contains objects of various types. In particular, LinkedIn includes the types of user, employer, location and college; Facebook includes user, concentration, degree, school, hometown, last-name, location, employer, work-location and work-project (other types are ignored due to their sparsity or irrelevance); DBLP includes paper, author, year, conference and keyword. We organized them into heterogeneous social networks, as summarized in Table 2.

**Ground truth.** On LinkedIn, the user relationships are already labeled into different semantic classes. We tested two major classes: *schoolmate* and *colleague*. On Facebook, the user relationships are defined by [11] with two classes: *family* and *classmate*. On DBLP, the *adivsor* and *advisee* in a co-author pair are identified based on the website of some faculty members as well as the Mathematics Genealogy and AI Genealogy projects [36]. All unidentified co-author pairs are assumed to be negative. As summarized in Table 3, the *advisor* and *advisee* classes are asymmetric, whereas the others are all symmetric.

**Table 3: Summary of semantic user relations and queries.**

| Network | Semantic relations | Symmetry | Queries | Results per query |
|---|---|---|---|---|
| LinkedIn | Schoolmate | Yes | 172 | 16.2 |
| | Colleague | Yes | 173 | 12.8 |
| Facebook | Family | Yes | 340 | 4.0 |
| | Classmate | Yes | 904 | 6.5 |
| DBLP | Advisor | No | 2,439 | 1.3 |
| | Advisee | No | 1,204 | 2.6 |

**Table 4: Running time of offline subgraph indexing.**

| | LinkedIn | Facebook | DBLP |
|---|---|---|---|
| Time (hour) | 1.77 | 3.34 | 4.43 |

**Training and testing.** On each graph, a user $q$ can be used as a query node, if there exists 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 semantic relation, and the average number of results per query are shown in Table 3. 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 [11] to evaluate the quality of the algorithmic rankings at the top 10 results.

**Subgraphs.** We repeated the subgraph pattern mining and subgraph instance matching algorithms in [11] to obtain the set of frequent subgraph patterns $\mathcal{M}$ and its instances $\mathcal{I}$ on each network. Specifically, we first applied GRAMI [9] on each graph to mine the set of frequent subgraph patterns. Then we filtered out those clearly non-viable subgraphs: 1) since our ground truth is designed for semantic user search, a viable subgraph must have at least two user objects; 2) a subgraph must contain at least two different types for capturing richer semantics; 3) to further constrain the number of subgraphs, we restricted them to have at most five nodes on LinkedIn and Facebook or six nodes on DBLP, which are found to be adequate in expressing the interactions between two users. The resulting number of subgraph patterns are shown in Table 2.

As shown in Table 4, the subgraph indexing (including mining the frequent set of subgraph patterns, and matching each subgraph pattern with its possible instances on the graph) can be done in a reasonable time (*i.e.*, a few hours). Since subgraph indexing is not the focus of this paper, we consider subgraph indexing as already done and the resulting subgraph patterns/instances are readily available for our subgraph-augmented path embedding algorithm.

**Table 5: Result comparison under different amounts of labels (*i.e.*, 10, 100, and 1000).**

| | Methods | LinkedIn | | | | | | Facebook | | | | | | DBLP | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *Schoolmate* | | | *Colleague* | | | *Classmate* | | | *Family* | | | *Advisor* | | | *Advisee* | | |
| | | 10 | 100 | 1000 | 10 | 100 | 1000 | 10 | 100 | 1000 | 10 | 100 | 1000 | 10 | 100 | 1000 | 10 | 100 | 1000 |
| **NDCG@10** | 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 |
| | MGP | 0.546 | 0.568 | 0.574 | 0.527 | 0.546 | 0.552 | **0.797** | 0.843 | 0.849 | **0.627** | 0.732 | 0.753 | 0.618 | 0.718 | 0.745 | 0.385 | 0.403 | 0.405 |
| | MPP | 0.503 | 0.504 | 0.504 | 0.497 | 0.510 | 0.508 | 0.707 | 0.803 | 0.796 | 0.575 | 0.647 | 0.640 | 0.581 | 0.662 | 0.674 | 0.345 | 0.380 | 0.390 |
| | 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 |
| | DWR | 0.515 | 0.518 | 0.530 | 0.493 | 0.506 | 0.504 | 0.692 | 0.689 | 0.699 | 0.585 | 0.575 | 0.583 | **0.764** | 0.758 | 0.661 | 0.401 | 0.395 | 0.392 |
| | PES | 0.650 | 0.676 | 0.687 | 0.552 | 0.647 | 0.668 | 0.720 | 0.851 | 0.862 | 0.482 | 0.748 | 0.752 | 0.741 | 0.768 | 0.773 | 0.374 | 0.406 | 0.416 |
| | SPE-A (ours) | 0.541 | 0.603 | 0.640 | 0.545 | 0.597 | 0.630 | 0.587 | 0.565 | 0.547 | 0.424 | 0.451 | 0.411 | 0.436 | 0.451 | 0.449 | 0.325 | 0.353 | 0.356 |
| | SPE (ours) | **0.660** | **0.690** | **0.694** | **0.644** | **0.686** | **0.704** | 0.764 | **0.866** | **0.869** | 0.467 | **0.761** | **0.774** | 0.763 | **0.782** | **0.789** | 0.373 | **0.413** | **0.417** |
| **MAP@10** | 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 |
| | MGP | 0.288 | 0.305 | 0.313 | 0.317 | 0.333 | 0.338 | 0.673 | 0.729 | 0.738 | **0.520** | 0.651 | 0.681 | 0.548 | 0.663 | 0.696 | 0.278 | 0.280 | 0.295 |
| | MPP | 0.260 | 0.260 | 0.259 | 0.294 | 0.305 | 0.304 | 0.576 | 0.681 | 0.675 | 0.467 | 0.543 | 0.534 | 0.499 | 0.597 | 0.611 | 0.245 | 0.273 | 0.281 |
| | 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 |
| | DWR | 0.393 | 0.391 | 0.398 | 0.363 | 0.369 | 0.368 | 0.578 | 0.575 | 0.587 | 0.467 | 0.455 | 0.463 | **0.683** | 0.675 | 0.541 | **0.295** | 0.281 | 0.276 |
| | PES | **0.547** | 0.567 | 0.581 | 0.432 | 0.535 | 0.559 | 0.617 | 0.797 | 0.815 | 0.354 | 0.685 | 0.691 | 0.667 | 0.693 | 0.699 | 0.267 | 0.282 | 0.293 |
| | SPE-A (ours) | 0.447 | 0.508 | 0.545 | 0.439 | 0.490 | 0.525 | 0.457 | 0.436 | 0.418 | 0.294 | 0.326 | 0.277 | 0.287 | 0.297 | 0.305 | 0.231 | 0.254 | 0.243 |
| | SPE (ours) | **0.547** | **0.581** | **0.587** | **0.529** | **0.575** | **0.602** | 0.682 | **0.822** | **0.826** | 0.368 | **0.704** | **0.723** | 0.679 | **0.700** | **0.703** | 0.249 | **0.298** | **0.299** |

**Parameters and environment**. For the fair comparison, we use the same object path sampling design and parameters as ProxEmbed [20]. Specifically, on LinkedIn, for both *schoolmate* and *colleague* we set $\gamma = 20, \ell = 20$. On Facebook, we set $\gamma = 40, \ell = 80$ for *classmate* and $\gamma = 20, \ell = 80$ for *family*. On DBLP, we set $\gamma = 20, \ell = 80$ for *advisor* and $\gamma = 20, \ell = 40$ for *advisee*. By default, we set the number of dimension $d' = 12$ for the parameters in $\Theta^{(att)}$, and $\mu = 10^{-4}$ in Eq. 21. We tune different dimensions $d$ of subgraph embedding and $\lambda$ for different semantic relations. We run these experiments on Linux servers with 32GB memory, and use Theano [33] for SPE implementation and Java jdk-1.8 for path sampling and s-path construction.

**Baselines**. We compare our SPE with the following state-of-the-art semantic search baselines.

• ProxEmbed [20]: ProxEmbed uses object paths to describe the relations between two objects and measure their proximity.

• MGP [11]: Meta-Graph Proximity uses the number of meta-graph instances between two objects as features to measure proximity.

• MPP [30]: Meta-Path Proximity uses the number of meta-path instances between two objects as features to measure proximity.

• SRW [1]: Supervised Random Walk learns edge weights to bias a random walk for generating consistent ranking results with the ground truth. We define each edge's feature as a binary vector based on the types of its two objects.

• DWR: DeepWalk Ranking was introduced in [20]. It first learns object embedding by DeepWalk [27], then outputs a Hadamard product over two objects' embedding as the proximity embedding.

• PES: ProxEmbed with s-paths is a straightforward solution to model s-paths. It directly feeds s-paths into ProxEmbed without subgraph embedding, s-node embedding and s-path embedding.

• SPE-A: SPE without Attention is a baseline for validating the need to modeling attention. It replaces the attention mechanisms for subgraph embedding, s-node embedding and s-path embedding in SPE with mean pooling, max pooling and max pooling respectively.

**Table 6: Relative improvement of SPE over the best baselines in each relation when using 100 training tuples.**

Normalized Discounted Cumulative Gain — Mean Average Precision

| | NDCG | MAP |
|---|---|---|
| LinkedIn-*schoolmate* | 5.8% ($p < 0.01$) | 7.4% ($p < 0.01$) |
| LinkedIn-*colleague* | 13.2% ($p < 0.01$) | 16.9% ($p < 0.01$) |
| Facebook-*family* | 2.4% ($p < 0.05$) | 3.8% ($p < 0.01$) |
| Facebook-*classmate* | 1.8% ($p < 0.1$) | 2.6% ($p < 0.01$) |
| DBLP-*advisor* | 2.2% ($p < 0.05$) | 1.4% ($p < 0.1$) |
| DBLP-*advisee* | 2.0% ($p < 0.05$) | 1.4% ($p < 0.1$) |

For MGP and MPP, we use the same parameter setting as [11]. For SRW, we set its regularization parameter $\lambda = 10$, random walk teleportation parameter $\alpha = 0.2$ and loss parameter $b = 0.1$. We set the dimension of DWR as 128, the same as [27]. For SPE-A, we use the same parameter values as our SPE. We input the same object paths to ProxEmbed, DWR, PES, SPE-A and SPE.

**Data and code availability.** All the three data sets are publicly available online from their corresponding references, as mentioned earlier. We have made our code available online[2].

## 7.1 Comparison with Baselines

We compare our proposed SPE with the seven state-of-the-art semantic search baselines introduced above. We test all the methods on the six semantic relations under different amount of training tuples, *i.e.*, 10, 100 and 1000. From the results reported in Table 5, we make the following observations.

First of all, our SPE generally outperforms the baselines across all the six semantic relations in terms of both NDCG and MAP. The only exception is when training with 10 tuples, SPE does not generate the best performance among all the baselines. This is because SPE has more parameters to learn; when the number of

[2]https://github.com/vwz/SPE

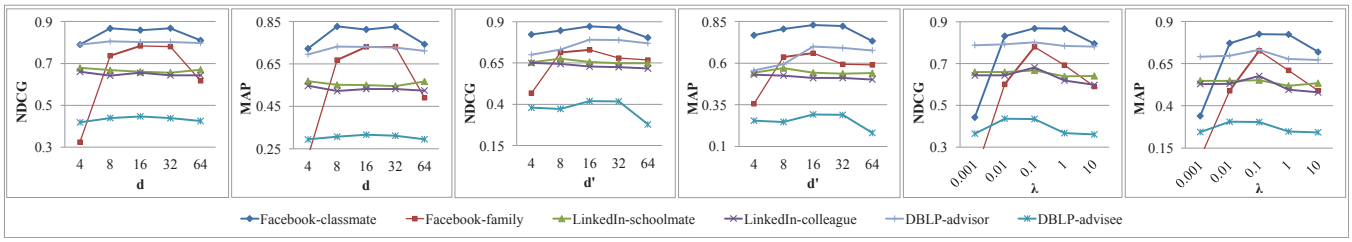


**Figure 4: Impact of parameters: subgraph embedding dimension $d$, attention parameter dimension $d'$, ranking loss discount $\lambda$.**

training tuples is small, it does not perform well. As the number of training tuples increases, SPE consistently outperforms others.

Secondly, SPE is better than ProxEmbed since s-paths carry more semantics than simple o-paths used in ProxEmbed. Moreover, SPE is also better than directly applying ProxEmbed on s-paths (*i.e.*, PES). This is because PES is unable to deal with the subgraph structures and noises. Such results validate that, modeling s-paths for proximity embedding is not trivial.

Thirdly, SPE is better than MGP and MPP, showing that feature learning is more effective than feature engineering in proximity learning. Although SPE uses the same subgraph inputs as MGP, SPE further learns subgraph embedding, s-node embedding and s-path embedding from the sampled o-paths. SPE clearly benefits from the constructed subgraph-augmented paths, which leverage both path's distance awareness and subgraph's high-order structure.

Fourthly, SPE outperforms SRW. which uses the biased random walk to guide semantic ranking. SRW seems insensitive to the number of training tuples. Besides, SPE is better than DWR, which uses the Hadamard product over two objects' embedding as the proximity embedding. This observation shows that, the node embedding method, as solving the semantic search in an indirect manner, is less effective for proximity search.

Finally, SPE outperforms SPE-A, which validates the need of modeling attentions. As can be seen in Table 5, SPE-A consistently generates inferior performance than SPE. This implies that well handling the noise in subgraph and s-paths is importantant.

We summarize the performance improvement of SPE over the best baselines with paired t-test in Table 6. The largest improvement is observed in LinkedIn-*colleague*, where SPE improves the best baseline (PES) by relatively 13.2% in terms of NDGG and 16.9% in terms of MAP, with t-test $p$-values less than 0.01.

## 7.2 Parameter Sensitivity

We also test the parameter sensitivity of SPE using 100 training tuples. We vary the subgraph embedding's dimension $d$ (Eq. 3), the attention parameters' dimension $d'$ (Eq. 7, Eq. 15 and Eq. 18) and the loss discount $\lambda$ (Eq. 20).

As shown in Fig. 4, Facebook data set is much more sensitive to the parameters setting than the other two data sets (especially LinkedIn). The reason is that the total number of object types in Facebook is much larger than in LinkedIn and DBLP as shown in Table 2. When searching for users that meet a particular semantic relation type, the irrelevant types may bring in noises to the semantic user search process. The more types there are, the more noises may exist. Therefore, for the data sets with more types, the parameters need to be carefully tuned so that the embedded subgraph, the learned attentions and loss discount help to filter useful information out of the noises. On the other hand, for the data sets with less types like LinkedIn, the performance is relatively robust because the noises introduced by irrelevant object types are limited.

Based on both NDCG and MAP over all the six semantic relations on three data sets, SPE tends to generate the best performance at $d = 16$. Especially, for *classmate* and *family*, when $d$ is too small, the resulting embeddings are unable to capture the rich semantics. When $d$ is too big, it may bring in more noises and increase the number of parameters to learn. The attention dimension $d'$ also tends to suffer from the similar performance decrease when $d'$ is too small or too big. $d' = 16$ is the best setting. For the ranking loss discount parameter $\lambda$, we see that $\lambda = 0.1$ usually gives the best results, suggesting the need to discount the ranking loss in Eq. 20.

## 8 CONCLUSION

In this paper, we study the problem of semantic user search in heterogeneous social networks. We exploit the opportunity of integrating the path's distance awareness and the subgraph's high-order structure for learning a better representation of the proximity between two users. We propose a novel Subgraph-augmented Path Embedding (SPE) model. It takes object paths as input, and enriches them into subgraph-augmented paths. Then it addresses the challenges of incorporating the subgraph structure, the subgraph noise and the subgraph-augmented path noise. Finally, it embeds the subgraph-augmented paths between two users into a proximity embedding vector. With such a proximity embedding vector, we can easily measure the proximity between two users for semantic user search. We test SPE with six semantic relations in three public data sets and it improves the state of the art by at least 1.8%–13.2% (NDCG) and 1.4%–16.9% (MAP) with 100 training samples.

In the future, we would like to explore the heterogeneous social networks with rich edge features and graph dynamics.

## 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 National Science Foundation under Grant No. IIS 16-19302. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.

## REFERENCES


[1] Lars Backstrom and Jure Leskovec. 2011. Supervised Random Walks: Predicting and Recommending Links in Social Networks. In *WSDM*. 635–644.

[2] Yoshua Bengio. 2009. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning* 2, 1 (2009), 1–127.

[3] Austin R. Benson, David F. Gleich, and Jure Leskovec. 2016. Higher-order Organization of Complex Networks. *Science* 353, 6295 (2016), 163–166.

[4] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*. 2787–2795.

[5] Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. 2018. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. *TKDE* (2018).

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations. In *AAAI*. 1145–1152.

[7] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *ICML*. 2702–2711.

[8] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD*. 135–144.

[9] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. 2014. GRAMI: Frequent Subgraph and Pattern Mining in a Single Large Graph. *PVLDB* 7, 7 (2014), 517–528.

[10] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. 2015. Efficient Densest Subgraph Computation in Evolving Graphs. In *WWW*. 300–310.

[11] Yuan Fang, Wenqing Lin, Vincent W. Zheng, Min Wu, Kevin Chen-Chuan Chang, and Xiaoli Li. 2016. Semantic proximity search on graphs with metagraph-based learning. In *ICDE*. 277–288.

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*.

[13] Pankaj Gupta, Venu Satuluri, Ajeet Grewal, Siva Gurumurthy, Volodymyr Zhabiuk, Quannan Li, and Jimmy J. Lin. 2014. Real-Time Twitter Recommendation: Online Motif Detection in Large Dynamic Graphs. *PVLDB* 7, 13 (2014), 1379–1380.

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780.

[16] Glen Jeh and Jennifer Widom. 2002. SimRank: A Measure of Structural-context Similarity. In *KDD*. 538–543.

[17] Glen Jeh and Jennifer Widom. 2003. Scaling Personalized Web Search. In *WWW*. 271–279.

[18] Ni Lao and William W. Cohen. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine Learning* 81, 1 (2010), 53–67.

[19] 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.

[20] 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*.

[21] 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. In *AAAI*.

[22] Julian J. McAuley and Jure Leskovec. 2012. Learning to Discover Social Circles in Ego Networks. In *NIPS*. 548–556.

[23] Feiping Nie, Wei Zhu, and Xuelong Li. 2017. Unsupervised Large Graph Embedding. In *AAAI*.

[24] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *ICML*. 2014–2023.

[25] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. 2017. Matching Node Embeddings for Graph Similarity. In *AAAI*.

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

[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *KDD*. 701–710.

[28] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. 2017. Struc2Vec: Learning Node Representations from Structural Identity. In *KDD*. 385–394.

[29] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research* 12 (2011), 2539–2561.

[30] 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).

[31] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. 2012. Efficient Subgraph Matching on Billion Node Graphs. *PVLDB* 5, 9 (2012), 788–799.

[32] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW*. 1067–1077.

[33] Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *CoRR* abs/1605.02688 (may 2016).

[34] Cunchao Tu, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2017. TransNet: Translation-Based Network Representation Learning for Social Relation Extraction. In *IJCAI*. 2864–2870.

[35] Rogier J. P. van Berlo, Wynand Winterbach, Marco J. L. de Groot, Andreas Bender, Peter J. T. Verheijen, Marcel J. T. Reinders, and Dick de Ridder. 2013. Efficient calculation of compound similarity based on maximum common subgraphs and its application to prediction of gene transcript levels. *IJBRA* 9, 4 (2013), 407–432.

[36] 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*. ACM, 203–212.

[37] Chi Wang, Rajat Raina, David Fong, Ding Zhou, Jiawei Han, and Greg Badros. 2011. Learning Relevance from Heterogeneous Social Network and Its Application in Online Targeting. In *SIGIR*. 655–664.

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

[39] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAAI*. 1112–1119.

[40] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *ICML*. 2048–2057.

[41] Muhan Zhang and Yixin Chen. 2017. Weisfeiler-Lehman Neural Machine for Link Prediction. In *KDD*. 575–583.