Metagraph-based Learning on Heterogeneous Graphs

Yuan Fang, Wenqing Lin, Vincent W. Zheng, Min Wu, Jiaqi Shi, Kevin Chen-Chuan Chang, Xiao-Li Li

Abstract—Data in the form of graphs are prevalent, ranging from biological and social networks to citation graphs and the Web. In particular, most real-world graphs are heterogeneous, containing objects of multiple types, which present new opportunities for many problems on graphs. Consider a typical proximity search problem on graphs, which boils down to measuring the proximity between two given nodes. Most earlier studies on homogeneous or bipartite graphs only measure a generic form of proximity, without accounting for different "semantic classes"—for instance, on a social network two users can be close for different reasons, such as being classmates or family members, which represent two distinct semantic classes. Learning these semantic classes are made possible on heterogeneous graphs through the concept of *metagraphs*. In this study, we identify metagraphs as a novel and effective means to characterize the common structures for a desired class of proximity. Subsequently, we propose a family of metagraph-based proximity, and employ a learning-to-rank technique that automatically learns the right parameters to suit the desired semantic class. In terms of efficiency, we develop a symmetry-based matching algorithm to speed up the computation of metagraph instances. Empirically, extensive experiments reveal that our metagraph-based proximity substantially outperforms the best competitor by more than 10%, and our matching algorithm can reduce matching time by more than half. As a further generalization, we aim to derive a general node and edge representation for heterogeneous graphs, in order to support arbitrary machine learning tasks beyond proximity search. In particular, we propose the finer-grained *anchored metagraph*, which is capable of discriminating the roles of nodes within the same metagraph. Finally, further experiments on the general representation show that we can outperform the state of the art significantly and consistently across various machine learning tasks.

Index Terms—Semantic Proximity Search, Meta-structures, Graph Mining, Heterogeneous Graph Representation.

1 Introduction

The proliferation of the Internet has availed an increasingly rich collection of data objects. Typically these objects can be organized into a graph $G = (V, E)$, where the nodes V model the objects and the edges E model their interactions. These graphs are often *heterogeneous* [1] containing different types of objects. Consider the graph in Figure 1 based on a toy social network, which interconnects various users and their attributes. We treat each user and attribute value as a node, and each node is further associated with a type like `user` and `school`. More generally, the edges may also belong to different types. In our toy graph, the edge types can be understood as a bijection from the pairs of node types.

Unlike traditional homogeneous graphs which only carry inter-node *structural* information, heterogeneous graphs further encompass *semantic* information that explain the nodes and their interactions. Such semantic information present new opportunities to many data-driven problems on graphs. In particular, we introduce a motivating problem next, as the main subject to address in this paper.

- *Y. Fang and J. Shi are with Singapore Management University, Singapore. E-mail: {yfang,jqshi}@smu.edu.sg*
- *M. Wu and X.-L. Li are with Institute for Infocomm Research, Singapore. E-mail: {wumin,xlli}@i2r.a-star.edu.sg*
- *W. Lin is with Tencent, China. E-mail: edwlin@tencent.com*
- *V. W. Zheng (corresponding author) is with WeBank, China. E-mail: vincent.zheng@adsc-create.edu.sg*
- *K. C.-C. Chang is with University of Illinois at Urbana-Champaign, United States. E-mail: kcchang@illinois.edu*

Manuscript received xxx; revised xxx.

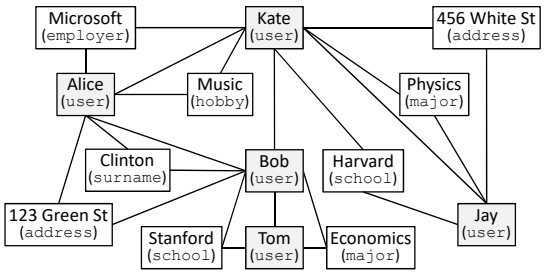


Fig. 1: Toy heterogeneous graph (node type in parenthesis).

1.1 Motivating Problem: Semantic Proximity Search

One important problem on graphs is *proximity search*. Given a query node $q \in V$, how do we measure the proximity of other nodes to q, so that we can return the nodes closest to q? Most earlier studies, including Personalized PageRank [2] and SimRank [3], fail to capitalize on the rich semantics carried by a heterogeneous graph. Specifically, with various types of interconnected objects, different *semantic classes of proximity* arise from different underlying reasons, as illustrated in Table 1. For the same query node (*e.g.*, Bob), there could be multiple classes of proximity with different result nodes (*e.g.*, Alice as family, and Tom as classmate). Thus, it falls short to only measure a "generic" form of proximity without differentiating the various semantic classes.

We call the task of searching for a desired semantic class of proximity w.r.t. a query node as *semantic proximity search* [4]. It is a new problem in the sense that previous studies on proximity search [2], [3], [5], [6], [7] neither

TABLE 1: Toy semantic classes of proximity

Query	Semantic class	Answer(s)	Reason
Kate	close friends	Alice	same `employer` and `hobby`
		Jay	same `address`
Kate	classmates	Jay	same `school` and `major`
Bob	family	Alice	same `surname` and `address`
Bob	classmates	Tom	same `school` and `major`

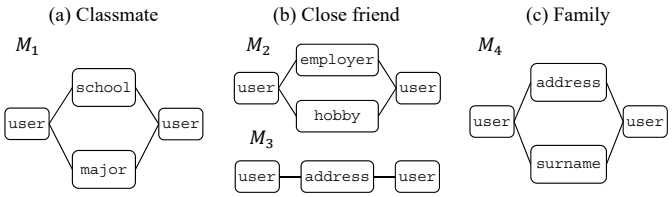


Fig. 2: Possible toy structures for each class of proximity.

intend to explicitly differentiate the semantic classes, nor can effectively accomplish so. Beyond proximity search, the closest problems to ours are social circle learning [8] and relationship profiling [9] on graphs. In terms of *semantic proximity*, their circles or relationships are also semantic-oriented, but they only find latent clusters and thus do not target specific classes of interest. In terms of *search*, they need to optimize for the best global configuration, and thus cannot process ad-hoc queries in real time.

1.2 Insights and Challenges

To differentiate various semantic classes, it comes to us a natural question: what kinds of representation or structure can characterize a class. Ideally, we require one that is not only universal in capturing different semantics, but also systematically enumerable for efficient processing.

We hinge on the novel insight that different semantic classes can often be characterized by different tell-tale common structures. For instance, in Figure 1 and Table 1, the proximity between Kate and Jay (classmate) can be attributed to their common `school` and `major`, as illustrated by the structure (M_1) in Figure 2(a). Likewise, Figure 2(b) and (c) showcase some possible structures which can characterize, to different extents, close friend (M_2 and M_3) and family (M_4), respectively. We call such common structures *metagraphs* as they abstract objects into types. That is, each node in a metagraph (denoted by a rounded rectangle) describes the type of an object, rather than an object itself. Intuitively, two nodes "sharing" more characteristic metagraphs of a class are more likely to satisfy that class of proximity. Apart from capturing the semantic classes, metagraphs also enable online proximity search. By computing and indexing metagraphs offline, we can efficiently support any query on-the-fly by looking up the precomputed metagraphs.

Note that a less general concept known as *metapath* has been proposed [6], which only considers common path structures between two nodes. In fact, metagraph M_3 in Figure 2(b) is also a metapath, which only captures the common address between two users. In contrast, metagraphs can *jointly* model multiple common attributes. Consider two metapaths `user`–`employer`–`user` and `user`–`hobby`–`user`. Each of them cannot characterize the proximity of close friends on their own. However, by taking them jointly we obtain metagraph M_2, which can better characterize close friends. In other words, each metagraph is a nonlinear combination of metapaths, and is thus more expressive. Given the increased complexity of metagraphs compared to metapaths, it is also more challenging to utilize and process metagraphs, as we will discuss next.

Utilizing metagraphs. The characteristic metagraphs for an arbitrary class of proximity are often unknown. Furthermore, a class may be characterized by multiple metagraphs to varying extents. For instance, close friends could be colleagues with the same hobby or simply roommates, corresponding to M_2 and M_3 in Figure 2(b), respectively. One may also say that M_2 is more likely to indicate close friends than M_3. While domain experts can lend some guidance on certain special classes, it is impractical to rely on them alone for the general case. To better cope with different graphs and semantic classes in proximity search, we propose a machine learning approach that can automatically identify the characteristic metagraphs (*e.g.*, M_2 and M_3) based on some example query and answer nodes (*e.g.*, Kate as query, Alice and Jay as answers). In practice, we learn a weight for each metagraph to quantify how well it can characterize the desired class. These weights can be applied to answer future queries for the same class of proximity.

Processing metagraphs. It is necessary to compute the instances of each metagraph in order to know what metagraphs are "shared" by any two given nodes. However, computing the instances of a metagraph (also called *matching* a metagraph) is highly costly. It is equivalent to solving the NP-hard subgraph matching problem [10]. Furthermore, the number of instances of a metagraph on an input graph does not follow the property of *downward closure*, which excludes the techniques for frequent subgraph mining [11], [12], [13]. To compute the instances of a metagraph more efficiently, we observe that many useful metagraphs are symmetric, such as M_1–M_4 in Figure 2 where two objects sharing one or more common attributes. However, existing methods spend a large amount of redundant computation on symmetric substructures within a metagraph. Thus, we propose a novel *symmetry-based matching* algorithm which re-uses "symmetric" computation. As a result, we can avoid redundancy and substantially improve the efficiency of metagraph matching.

1.3 Generalization

Given that metagraphs are able to capture rich semantics on heterogeneous graphs, they can be leveraged to address many other machine learning tasks on graphs beyond semantic proximity search.

However, there is a major drawback with metagraphs of the form shown in Figure 2. While they can effectively model the interactions between nodes, they lack the granularity to discriminate the role of individual nodes within the interaction. Although this is not an issue for many metagraphs, such as M_1 where the two `user` nodes play balanced roles as classmates of each other, in general a relationship may involve imbalanced or distinct roles, such as the roles of advisor and advisee, physician and patient, or vendor and client. To differentiate these roles within the same metagraph, we propose the more granular *anchored*

metagraphs—the nodes in a metagraph are "anchored" (*i.e.*, differentiated) as the head or tail nodes. The differentiation enables us to better capture the semantics of individual nodes and their interactions.

Based on anchored metagraphs, we build general node and edge representations to support arbitrary machine learning tasks on heterogeneous graphs. The tasks can be node-centric like node classification and clustering, or edge-centric like link prediction and proximity search. Anchored metagraph-based representations are generally more effective due to the extra granule, and could become particularly advantageous when nodes have distinct roles or edges are directed.

1.4 Contributions

To summarize, we make the following contributions.

- *Concept.* We propose the novel concept of metagraphs to capture rich semantics on heterogeneous graphs. In particular, metagraphs are well suited to address our motivating problem of semantic proximity search. (Section 2)
- *Learning.* Towards semantic proximity search, we design a family of metagraph-based proximity measures, whose parameters are learnable to model different semantic classes. (Section 3)
- *Matching.* We devise an efficient metagraph matching algorithm, which exploits the symmetric components in a metagraph to avoid redundant computation. (Section 4)
- *Generalization.* We generalize metagraphs to anchored metagraphs in order to model nodes of distinct roles. Based on anchored metagraphs, we develop universal node and edge representations to support arbitrary machine learning tasks. (Section 5)
- *Experiments.* Our extensive experiments demonstrate the superiority of our metagraph-based approach for semantic proximity search. Further experiments show that the general representation based on anchored metagraphs perform consistently well across various machine learning tasks. (Sections 6 and 7)

2 Preliminaries

In this section, we first formalize the problem, and present the metagraph concept as well as the overall framework.

2.1 Problem statement

We formalize the notion of object graph, and introduce the task of semantic proximity search on such graphs.

Object graph. An *object graph* can be represented as $G = (V, E, \tau)$, where V denotes the set of objects and E denotes the set of edges between objects. Given objects of heterogeneous types T, there is a *type* function for objects, $\tau : V \to T$. On the toy graph in Figure 1, we would have types $T = \{\texttt{user}, \texttt{school}, \texttt{hobby}, \ldots\}$, and for instance, $\tau(\text{"Alice"}) = \texttt{user}$ and $\tau(\text{"123 Green St"}) = \texttt{address}$. Furthermore, a graph $S = (V_S, E_S, \tau)$ is a *subgraph* of G iff $V_S \subseteq V$ and $E_S \subseteq E$.

Semantic proximity search. On a graph $G = (V, E, \tau)$, given a query node $q \in V$ and a desired class of proximity, the task is to produce a ranking over V in descending proximity to q w.r.t. the desired class. We cast this as a machine learning problem, where a set of training examples Ω are available for learning the desired class of proximity. In particular, we adopt a learning-to-rank framework [14], where each training example is a triple (q, v, u) such that node v is ranked *before* node u for the query node q. That is, v's proximity to q should be greater than u's. Thus, the task boils down to defining a family of proximity measures between nodes that can abstract arbitrary classes of proximity, with a set of parameters that can be optimized by learning to rank.

2.2 Metagraph and related concepts

We propose the notion of metagraph to measure the proximity between nodes.

Metagraph. There are many distinct objects of the same type, *e.g.*, both "123 Green St" and "456 White St" are addresses. In order to identify and summarize common structures on the object graph, it becomes necessary to consider a type-level description, which we call a *metagraph*. Formally, a metagraph can be represented as $M = (V_M, E_M, \tau_M)$, where V_M is the set of nodes to denote the types, and E_M is the set of edges between V_M. That is, $\forall v \in V_M$, we have $\tau_M(v) \in T$ where τ_M is a type function for the metagraph. Note that a node on the object graph has both an intrinsic value (*e.g.*, "Alice" or "Microsoft") and a type, whereas the value of a node on the metagraph is *immaterial* and only the type matters.

Metagraph instances. In order to know whether two nodes on the object graph "share" a characteristic metagraph for the desired class of proximity, it is crucial to identify subgraphs on G that are instances of any given metagraph M. Informally, a subgraph S is an instance of M if they have the same structure and their nodes have matching types. For instance, in Figure 1 the following subgraphs

- S_1: "Alice"–"123 Green St"–"Bob" and
- S_2: "Kate"–"456 White St"–"Jay"

both match metagraph M_3 in Figure 2, and thus they are the instances of M_3. We present a formal definition next.

Definition 1 (Metagraph instance). Consider a subgraph $S = (V_S, E_S)$ and a metagraph $M = (V_M, E_M, \tau_M)$. S is an *instance* of M if there exists a bijection between the node sets of S and M, $\phi : V_S \to V_M$, such that

- $\forall v \in V_S$, $\tau(v) = \tau_M(\phi(v))$, and
- $\forall v, u \in V_S$, $\langle v, u \rangle \in E_S$ iff $\langle \phi(v), \phi(u) \rangle \in E_M$. □

Metagraph indices. Subsequently, we can quantify how two nodes v and u share any given metagraph, *i.e.*, how v and u occur in the instances of the metagraph. Let $\mathcal{M} = \{M_1, M_2, \ldots\}$ be a set of metagraphs, and $\mathcal{I}(M_i)$ be the set of instances of M_i. The co-occurrences of v and u can be encoded by a vector $\mathbf{m}_{vu}$ with $|\mathcal{M}|$ elements. Its i-th element, $\mathbf{m}_{vu}[i]$, is the number of instances of M_i containing both v and u. Likewise, let $\mathbf{m}_v[i]$ be the number of instances of M_i containing v. That is,

$$\mathbf{m}_{vu}[i] \triangleq |\{S \in \mathcal{I}(M_i) : (v, u) \in V_S^2\}|. \quad (1)$$

$$\mathbf{m}_v[i] \triangleq |\{S \in \mathcal{I}(M_i) : v \in V_s\}|. \quad (2)$$

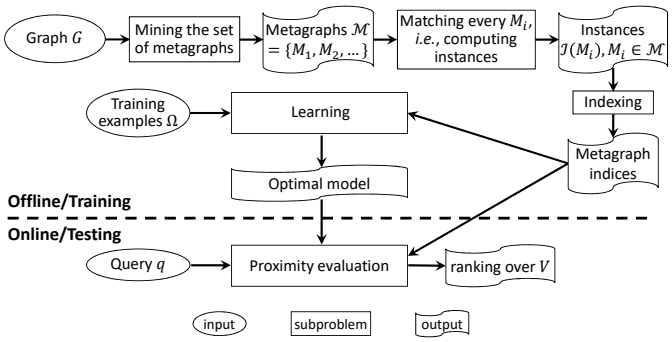


Fig. 3: Overall framework for metagraph-based learning, using semantic proximity search as an example task.

More generally, we can further transform these vectors, such as applying logarithm to the raw counts.

We call $\mathbf{m}_{vu}$ and $\mathbf{m}_v$ *metagraph indices*, which form the basis of our proximity measure as we shall discuss in Section 3. They can be precomputed offline by scanning through the metagraph instances, and can be loaded on-demand during training and testing.

2.3 Overall Framework

To summarize our approach, we present the overall framework in Figure 3. It consists of online and offline phases.

Offline phase. It consists of three main subproblems.

Initially, given a graph G, we enumerate the set of metagraphs $\mathcal{M} = \{M_1, M_2, \ldots\}$. Abundant literature [12], [13] exists on this subproblem, and its time cost typically accounts for only a few percent of the entire offline phase. Therefore, we directly apply an existing state-of-the-art approach GRAMI [13].

Next, for each metagraph $M_i \in \mathcal{M}$ from the output of the previous subproblem, we compute the set of instances $\mathcal{I}(M_i)$. We also call the process of computing the instances of M_i as *matching* M_i. We study the subproblem of efficient metagraph matching in Section 4. Subsequently, we further compute the metagraph indices, which form part of the input to learning and evaluating the proximity.

Lastly, with a set of training examples for the desired semantic class, we need to learn the optimal model. We address this subproblem in Section 3. In particular, we need to develop a family of metagraph-based proximity measures to accommodate arbitrary classes, and a supervised method to learn the best parameters.

Online phase. Given a query node $q \in V$, the precomputed metagraph indices, as well as the optimal model for the desired class, we can evaluate the proximity between q and other nodes $v \in V$. Subsequently, we rank the nodes in V in descending order of their proximity to q.

3 Learning Semantic Proximity

We propose to learn the optimal proximity based on metagraphs. We start with defining a family of proximity measures which can flexibly cater to different semantic classes. Next, given some training examples, we develop a learning-to-rank approach to optimize the parameters within the proximity family.

3.1 Metagraph-based Proximity

Given a class with certain characteristic metagraphs, a good proximity measure must account for two aspects. First, if v and u share many characteristic metagraphs, v and u are more likely to satisfy the desired class. Second, if v (or u) indiscriminately occurs with many metagraphs, v and u may simply appear by chance to share many characteristic metagraphs. Incorporating both aspects, we propose a metagraph-based measure below.

Definition 2 (Semantic proximity). The *semantic proximity* between any two nodes v and u is

$$\pi(v, u; \mathbf{w}) \triangleq \frac{2\, \mathbf{m}_{vu}^\top \mathbf{w}}{\mathbf{m}_v^\top \mathbf{w} + \mathbf{m}_u^\top \mathbf{w}}, \tag{3}$$

for some non-negative vector $\mathbf{w}$ of $|\mathcal{M}|$ elements. □

The measure π entails a family of proximity with parameters $\mathbf{w}$. We interpret $\mathbf{w}$ as the *characteristic weights* (or simply weights) of the metagraphs, which can be varied to fit different classes of proximity. The weights shall be non-negative, indicating the importance of the metagraphs towards the desired semantic class. Consider the toy example in Figure 2 with $\mathcal{M} = \{M_1, \ldots, M_4\}$. A reasonable $\mathbf{w}$ could be $(0.9, 0, 0, 0)^T$ for classmate, $(0, 0.6, 0.1, 0)^T$ for close friends, and $(0, 0, 0, 0.8)^T$ for family, where the i-th dimension of $\mathbf{w}$ encodes the importance of metagraph M_i. Thus, within the family of proximity, the optimal model for a class is completely specified by its optimal weights $\mathbf{w}^*$, which we aim to learn automatically.

Interestingly, the proposed measure exhibits a few desirable properties, as described in Theorem 1. In particular, partial transitivity implies that if a node v is close to both nodes u and z, u and z tends to be close to each other too. This is a common phenomenon on social networks, where friends of friends are more likely to be friends than a random person.

Theorem 1 (Properties). Given any three nodes x, y, z and weights $\mathbf{w}$, the following hold.

- *Symmetry*. $\pi(v, u; \mathbf{w}) = \pi(u, v; \mathbf{w})$.
- *Boundedness*. $0 \leq \pi(v, u; \mathbf{w}) \leq 1$.
- *Scale-invariance*. $\pi(v, u; \mathbf{w}) = \pi(v, u; c\mathbf{w})$ for any $c > 0$.
- *Partial transitivity*. There exists some $\delta > 0$, such that for any $\epsilon \in [0, 1]$, if $\pi(v, u; \mathbf{w}) \geq \frac{1+\epsilon\delta}{1+\delta}$ and $\pi(v, z; \mathbf{w}) \geq \frac{1+\epsilon\delta}{1+\delta}$, then $\pi(u, z; \mathbf{w}) \geq \epsilon$. □

3.2 Learning to Rank

For a desired class of proximity, we assume some training examples Ω as supervision. Each example is a triple (q, v, u), where node v is ranked before node u for the query node q, *i.e.*, v's proximity to q should be greater than u's. These examples can often be gathered by user studies [8], [9], while some platforms like Facebook also allow users to label their connections directly.

Given the training data Ω, we can find the optimal weights $\mathbf{w}^*$ by maximizing the log-likelihood. Intuitively, it becomes more likely to observe an example (q, v, u) when v's proximity to q is increasingly larger than u's. In other words, the probability of the example, $P(q, v, u; \mathbf{w})$, tends

to increase with the difference in v and u's proximity to q, $\pi(q, v; \mathbf{w}) - \pi(q, u; \mathbf{w})$. In particular, we define the probability using a sigmoid function in Eq. 4. Here $\mu \in (0, \infty)$ is a scaling variable to control the shape of the distribution; we set $\mu = 5$, which is found to be robust in our experiments.

$$P(q, v, u; \mathbf{w}) \triangleq \frac{1}{1 + e^{-\mu(\pi(q,v;\mathbf{w}) - \pi(q,u;\mathbf{w}))}} \tag{4}$$

Subsequently, given all the examples, we aim to maximize the following log-likelihood function L, to ultimately find the optimal weights $\mathbf{w}^* = \arg\max_{\mathbf{w}} L(\mathbf{w}|\Omega)$. Since L is differentiable, it can be optimized by employing the gradient descent algorithm.

$$L(\mathbf{w}|\Omega) = \sum_{(q,v,u)\in\Omega} \log P(q, v, u; \mathbf{w}) \tag{5}$$

3.3 Dual-Stage Training

As illustrated in Section 1, there exist a huge number of metagraphs even with just a few types of object. To reduce the overall matching time, we propose a novel process of dual-stage training. As the key insight, while there are many metagraphs in $\mathcal{M}$, the vast majority of them are irrelevant. Only a small number of metagraphs among $\mathcal{M}$ can characterize the desired class of proximity. In other words, the optimal weights $\mathbf{w}^*$ are sparse with many zero or nearly zero entries. Therefore, it is ideal to only focus on a small subset of *candidate* metagraphs, $\mathcal{K} \subset \mathcal{M}$, which show promise to characterize the desired class. Subsequently, we only match the candidates and compute the proximity based on the instances of these candidates.

Seed metagraphs. Note that, without computing the instances of any metagraph, there is no clue at all to locate the promising candidates. Instead, we first identify a small number of *seed* metagraphs $\mathcal{K}_0$ as the initial candidates and compute their instances $\mathcal{I}(M), \forall M \in \mathcal{K}_0$, which can lead us to more candidates. The seeds must meet the following criteria. First, *easy identification*: we can easily recognize the seeds without computing any instance. Second, *fast matching*: the seeds can be matched very fast. Third, *candidate heuristic*: the seeds must enable some heuristic for selecting more candidates without computing more instance.

To select the seeds, we observe that metapaths (*i.e.*, metagraphs that are paths such as M_3 in Figure 2) are less complex than general metagraphs. As a result, there are far fewer metapaths than metagraphs. Matching a metapath also tends to be much faster due to the simpler structure.

Candidate heuristic. We need to further develop a heuristic using the seeds in order to identify additional metagraphs from $\mathcal{M} \setminus \mathcal{K}_0$, without computing more instances. Again, we can only rely on the structural information of the metagraphs. On the one hand, two metagraphs are structurally similar if they share some common pattern, such as their maximum common subgraph (MCS) [15]. The larger MCS shared by two metagraphs, the more structurally similar they are. Letting M be the MCS of M_i and M_j, their structural similarity can be defined as

$$\text{SSim}(M_i, M_j) = \frac{(|V_M| + |E_M|)^2}{(|V_{M_i}| + |E_{M_i}|) \times (|V_{M_j}| + |E_{M_j}|)}. \tag{6}$$

On the other hand, the function of a metagraph refers to its contribution to (or its weight in) the proximity measure.

Algorithm 1: Dual-Stage Training

Input: graph G; set of metagraphs $\mathcal{M}$; number of candidates $|\mathcal{K}|$; training examples Ω
Output: optimal weights $\mathbf{w}^*$

```
// seed stage
```
$\mathcal{K}_0 \leftarrow \{M \in \mathcal{M} | M \text{ is a path}\}$
$\mathcal{I}_0 \leftarrow \{\mathcal{I}(M) | M \in \mathcal{K}_0\}$
$\mathbf{w}_0 \leftarrow \text{Train}(\Omega, \mathcal{K}_0, \mathcal{I}_0)$
```
// candidate stage
```
$\mathcal{K} \leftarrow$ Top $|\mathcal{K}|$ metagraphs by H scores based on $\mathcal{K}_0, \mathbf{w}_0$
$\mathcal{I} \leftarrow \{\mathcal{I}(M) | M \in \mathcal{K}\}$
$\mathbf{w}^* \leftarrow \text{Train}(\Omega, \mathcal{K}_0 \cup \mathcal{K}, \mathcal{I}_0 \cup \mathcal{I})$
return $\mathbf{w}^*$.

That is, two metagraphs are functionally similar if their corresponding weights are also similar.

Intuitively, metagraphs that are structurally similar tend to be functionally similar too. Given the seeds $\mathcal{K}_0$ and their instances, we can learn the function of the seeds, *i.e.*, their corresponding weights in $\mathbf{w}_0$. Supposing that a metagraph $M_j \in \mathcal{M} \setminus \mathcal{K}_0$ is structurally similar to a seed $M_i \in \mathcal{K}_0$, M_j and M_i will also be functionally similar. That is, if M_i has a large weight (*i.e.*, $\mathbf{w}_0[i]$ is large), M_j is also likely to have a large weight (*i.e.*, M_j is a promising candidate). Thus, we select candidates with the largest *candidate heuristic score* H, which maximizes their structural similarity to any seed metagraph with a large weight:

$$H(M_j) \triangleq \max_{M_i \in \mathcal{K}_0} \{\mathbf{w}_0[i] \cdot \text{SSim}(M_i, M_j)\}. \tag{7}$$

Dual-stage algorithm. We outline the above heuristic in Alg. 1, consisting of two stages. In the *seed* stage, we compute the instances of the seeds $\mathcal{K}_0$, and train their weights $\mathbf{w}_0$. In the *candidate* stage, based on $\mathcal{K}_0$ and $\mathbf{w}_0$, we further identify candidates $\mathcal{K}$ using the candidate heuristic, and train new weights $\mathbf{w}^*$ for $\mathcal{K}_0 \cup \mathcal{K}$.

4 Efficient Metagraph Matching

In this section, we address the subproblem of metagraph matching, which dominates the offline phase. We first summarize existing algorithms, and further present a new solution to address their drawback.

4.1 Subgraph Matching Revisited

Consider a metagraph $M = (V_M, E_M, \tau_M)$ on a graph $G = (V, E, \tau)$. To compute the instances of M on G, there are a number of existing approaches [16], [17], [18], [19] based on the backtracking method, summarized as follows.

Given an ordering of nodes in V_M, let $u_i \in V_M$ be the i-th node in the ordering where $1 \le i \le |V_M|$. Denote D_k as the set of k nodes in V that match $\{u_1, u_2, \ldots, u_k\}$, and $C(u_{k+1}|D_k)$ as the set of nodes each of which can match u_{k+1} given the existing matching D_k.

Initially, we have $D_0 = \emptyset$. The backtracking method first identifies the set $C(u_1|D_0)$ of nodes in V such that the type of each node $v \in C(u_1|D_0)$ equals the type of the first node u_1 in V_M, *i.e.*, $\tau(v) = \tau_M(u_1)$. For each node $v \in C(u_1|D_0)$, we match v to u_1, *i.e.*, $D_1 = \{v\}$. Given D_1, we further identify the set $C(u_2|D_1)$ for u_2 such that each node $v' \in C(u_2|D_1)$ can match u_2 and the graph induced

(a) Metagraph M_5. (b) Simplified metagraph M_5^+.

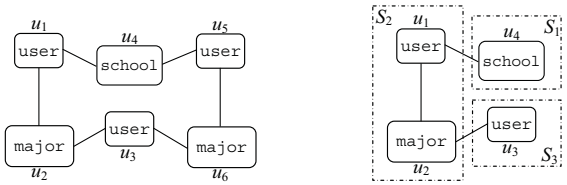


Fig. 4: A metagraph M_5 and its simplified metagraph M_5^+.

on $D_1 \cup \{v'\}$ is an instance of the metagraph induced on u_1 and u_2 (Def. 1). In other words, $v' \neq v$, $\tau(v') = \tau_M(u_2)$, and $\langle u_2, u_1 \rangle \in V_M$ if and only if $\langle v', v \rangle \in V$. If $C(u_2|D_1)$ is empty, we stop searching further and immediately backtrack to another node in $C(u_1|D_0)$. Otherwise, $C(u_2|D_1)$ is not empty, and for each node in $v' \in C(u_2|D_1)$ we have $D_2 = D_1 \cup \{v'\}$, from which we can recursively compute $D_3, D_4, \ldots, D_{|V_M|}$. We then report the subgraph induced by $D_{|V_M|}$ as an instance of M on G, and backtrack to compute other instances.

4.2 Metagraph Symmetry

The above approaches all deal with general metagraphs. However, we observe that symmetric metagraphs like M_1–M_4 in Figure 2 are very common, forming the vast majority of all metagraphs containing two `user` nodes. Thus, how to efficiently handle symmetric metagraphs become crucial. To begin with, we present a formal definition of metagraph symmetry below.

Definition 3 (Metagraph symmetry). Consider a metagraph $M = (V_M, E_M, \tau_M)$. M is a *symmetric* metagraph if there exists a non-empty set Ψ_M containing pairs of distinct nodes of the same type in V_M, such that the edge set E_M remains unchanged even if, for each pair $(u, u') \in \Psi_M$, we exchange u and u' in all edges incident to u or u'. We also say that such u and u' are symmetric to each other in M. □

For example, the metagraph M_5 in Figure 4 is symmetric, since there exists a set $\{(u_1, u_5), (u_2, u_6)\}$, such that if we exchange u_1 and u_5 (resp. u_2 and u_6) in all edges incident to u_1 or u_5 (resp. u_2 or u_6), the set of edges in M_5 remain the same.

Previous approaches [16], [17], [18], [19] often incur a large amount of redundant computation on symmetric metagraphs. To illustrate, after matching $u_1, u_2, \ldots, u_4$ in M_5, previous approaches need to compute the matchings $C(u_5|D_4)$ and $C(u_6|D_5)$ from scratch, even though u_5 (resp. u_6) is symmetric to u_1 (resp. u_2). Take u_6 as an example, they have to examine every node in V if its type is the same as u_6, and if it appropriately connects to the graph induced by D_5. Since u_2 is symmetric to u_6, potentially we do not need to examine every node in V, but rather only those matched by u_2.

4.3 Symmetry-based Matching

To leverage the symmetry of metagraphs, we propose a novel approach to compute the matchings of a node u from its symmetric node u' in M. For example, in Figure 4, the instances of u_5 and u_6 can be computed from the instances of u_1 and u_2, since u_5 (resp. u_6) is symmetric to u_1 (resp. u_2). However, we cannot treat each pair of symmetric nodes independently. For example, in Figure 4, the matchings of u_2 cannot be re-used by u_6 without considering u_1 and u_5 in conjunction, since u_2 is adjacent to u_1 but not u_5 (which u_6 is adjacent to).

Algorithm 2: Compute Instances of Metagraph

Input: a graph G; a metagraph M
Output: the set $\mathcal{I}(M)$ of instances of M

1 decompose M into a set $\mathcal{B}$ of components based on symmetry
2 simplify M as M^+ using $\mathcal{B}$
3 compute a matching order o for components of M^+
4 $\mathcal{I}(M) \leftarrow$ MatchingByComponent(G, M, o, 1, $\emptyset$)
5 **return** $\mathcal{I}(M)$.

To cope with the above issue, we decompose the node set V_M into disjoint connected components, so that each component can be handled independently. In particular, if a node u is not symmetric to any other node on M, u forms a singleton component S, *i.e.*, $S = \{u\}$. Otherwise, we partition the symmetric nodes into several connected components, such that for each component S, we have (i) each node $u \in S$ has the same number of symmetric nodes on M, (ii) each node $u \in S$ is not symmetric to any other node $u' \in S$, and (iii) S is the largest such set. For example, we can decompose M_5 in Figure 4 into 4 components, namely $S_1 = \{u_4\}$, $S_2 = \{u_1, u_2\}$, $S_3 = \{u_3\}$ and $S_4 = \{u_5, u_6\}$. We say that a component S is *symmetric* to another component S', if for each node u in S, there exists a node u' in S' such that u is symmetric to u' on M. For instance, the components S_2 and S_4 described above are symmetric to each other.

The above decomposition ensures that each component is independently symmetric to some other component. Thus, if a component S is matched prior to its symmetric component S', we can save the cost for S' by re-using the instances of S. Alg. 2 outlines our proposed approach by utilizing symmetric components. We still follow the backtracking framework, but instead of trying one node at time, we match one component at a time. Thus, we first need to decompose a metagraph M into several components. Next, we simplify M into a smaller graph M^+, to avoid redundant computation on symmetric components. Finally, we design a matching ordering over the components in M^+. In what follows, we elaborate on each step.

Metagraph decomposition. To decompose M into components, we first construct a component for each node u which is not symmetric to any node in M, and remove u from M. Then, in the residual graph M', we construct the symmetric components. In particular, we process the nodes in M' iteratively. In each iteration, we randomly choose a node u and construct a component S initially containing only u, as well as a component S' for each u' that is symmetric to u. Then, we iteratively add more nodes into S (resp. each S') such that the rules of components specified earlier are not violated. When no more nodes can be added into S, we remove S from M' and continue to construct components in the residual graph M'' until M'' is empty.

Metagraph simplification. Now we simplify the metagraph by representing it with its components. Specifically, we

replace the nodes in M by the components containing them, and add an edge between components S and S' if there exists nodes $u \in S$ and $u' \in S'$ such that u and u' are adjacent to each other on M. To further simplify, among each set of symmetric components, we only retain one of them and remove the rest. Denote M^+ as the resulting simplified metagraph. In Figure 4, M_5 is converted into M_5^+ with three components S_1–S_3, where S_2 (retained) is symmetric to $S_4 = \{u_5, u_6\}$ (removed).

Matching order. To reduce the search space, the matching order of nodes is important. Previous approaches [16], [20] select the next node such that the number of intermediate instances can be minimized. For example, starting from a metagraph $M^{(1)}$ containing only one edge $\langle u_1, u_2 \rangle$ from M, we can extend $M^{(1)}$ by adding an edge $\langle u_2, u_3 \rangle$ from M, resulting in a larger intermediate metagraph $M^{(2)}$. We can thus estimate the number of instances of $M^{(2)}$ as $f(M^{(2)}) = |\mathcal{I}(M^{(1)})| \cdot \frac{|\mathcal{I}(\langle u_2, u_3 \rangle)|}{|\mathcal{I}(u_2)|}$. In general, $M^{(i+1)}$ can be obtained by adding an edge $\langle u, u' \rangle$ from M to $M^{(i)}$, and the number of its instances can be estimated as $f(M^{(i+1)}) = f(M^{(i)}) \cdot \frac{|\mathcal{I}(\langle u, u' \rangle)|}{|\mathcal{I}(u)|}$. Thus, in each step, we pick the next node to minimize the number of estimated instances of the intermediate metagraph. We can generalize this approach to order the components of M^+: when a node of a component is chosen, we select that component as the next to match.

Matching simplified metagraphs. The matching algorithm for a simplified metagraph follows the backtracking framework in Alg. 3. Compared with previous methods that match a node at a time, our approach matches one component at a time. Given the set D of already matched nodes, the matchings of a component S are the matchings of its constituent nodes, denoted as $C(S|D)$. We can save significant computation when S is a symmetric component. Let $\mathcal{B}$ be the set containing S and the symmetric components of S. Subsequently, we can compute the matchings for all components in $\mathcal{B}$, denoted by $C(\mathcal{B}|D)$, based on $C(S|D)$. That is, we do not need to compute $C(S'|D)$ for any $S' \neq S$ and $S' \in \mathcal{B}$. We simply choose $|\mathcal{B}|$ number of distinct matchings from $C(S|D)$. For each choice of $|\mathcal{B}|$ matchings, we inspect whether the connectivity between components satisfies Def. 1. If so, we add the choice to $C(\mathcal{B}|D)$.

Complexity analysis. Consider a metagraph $M = (V_M, E_M, \tau_M)$ and a graph $G = (V, E, \tau)$. In Alg. 2, the decomposition and computation of matching order (lines 1–3) require at most one scan of V_M for each node, leading to the time complexity of $O(|V_M|^2)$, where $|V_M|$ is often very small. Next, we consider the cost of matching (line 4). Assuming a general M without symmetry, Alg. 3 starts from the instance set $\mathcal{I}(M^{(1)})$, and then repeatedly inspects the neighbors of a chosen node for a depth of at most $|V_M| - 2$. Since $\mathcal{I}(M^{(1)}) \leq |E|$, the time complexity of matching is upper bounded by $O(|E| \cdot d^{|V_M|-2})$, where d is the maximal node degree in G. Again, $|V_M|$ is typically very small. Furthermore, for a symmetric M, the depth of inspection is even smaller given a smaller simplified metagraph.

Adaptation to dynamic graphs. Real-world graphs often evolve over time. When a large graph changes, it is infeasible to re-compute the instances from scratch. Instead, the instances can be updated incrementally by refining the affected ones [21], which are often of a small number in practice. In particular, when edges are deleted from the graph, we only need to remove the instances that contain those edges. On the other hand, when adding edges to the graph, for each new edge we enumerate the instances containing the new edge, within a distance of not more than the maximum size of metagraphs.

Algorithm 3: MatchingByComponent

Input: a graph G; a metagraph M; a matching order o; the index of matching component k; the set D of matched nodes;
Output: the set $\mathcal{I}'(M)$ of instances with D

```
if |D| = |V_M| then
    return the instance induced by D.
end
S ← k-th component in the matching order o
B ← the set including S and its symmetric components, if any
compute the set C(B|D)
I'(M) ← ∅
for each S' ∈ B do
    D' ← the merge of D and the matching of S' from C(B|D)
    I* ← MatchingByComponent(G, M, o, k + 1, D')
    add I* into I'(M)
end
return I'(M).
```

5 General Representation for Learning

While metagraphs are initially motivated by the problem of semantic proximity search, in this section, we investigate their generalization to other machine learning problems on heterogeneous graphs. We first introduce the concept of *anchored metagraphs*, a further differentiation of metagraphs to better describe the nodes and their interactions. Next, we examine a general node and edge representation based on anchored metagraphs, towards solving various machine learning problems including semantic proximity search.

5.1 Anchored Metagraphs

Metagraphs form the building blocks of our proximity measure. As discussed in Section 2, they can effectively relate two nodes on a heterogeneous graph and work well for many semantic classes of proximity. However, they are not fine-grained enough to discriminate the role of individual nodes within the same metagraph.

Consider a graph that connects Peter and Steven in Figure 5(a). Although both of them are represented by `author` nodes, Peter and Steven likely play two distinct (latent) roles. In particular, Peter is likely a professor, as he not only co-authors papers with Steven, but also manages a grant and serves on the admission committee. On the other hand, Steven is likely a graduate student. Additionally, Peter and Steven possibly form the advisor–advisee relationship. Unfortunately, if we only know that both co-occur in the same metagraph as shown in Figure 5(b), we cannot tell apart the professor and student as there is no differentiation between the two *author* nodes in the metagraph.

In other words, the metagraph definition in Section 2 is unable to capture the semantics of two nodes of the same type when their roles are not commensurate with each other. Apart from advisor–advisee, many other examples

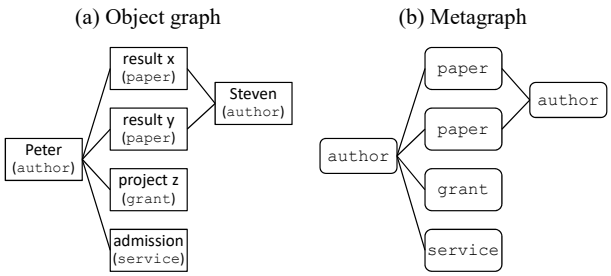


Fig. 5: Object graph and metagraph for co-authors.

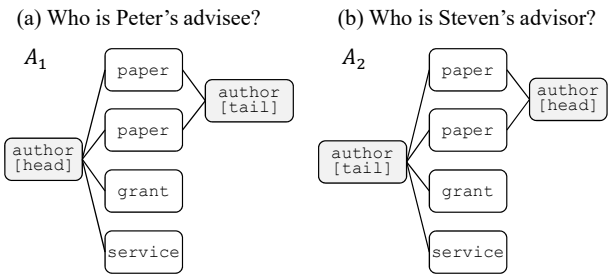


Fig. 6: Anchored metagraphs for co-authors.

with distinct roles exist, such as landlord–tenant, physician–patient, and client–vendor. More generally, it is inadequate to describe individual nodes or their interactions with the frequency of occurring or co-occurring in a given metagraph. The underlying weakness is the lack of granularity to differentiate nodes within a metagraph.

Towards finer-granularity, we propose *anchored metagraphs*, where different nodes are explicitly "anchored" within a metagraph. We use the problem of semantic proximity search as an example. As illustrated in Figure 6, to search for Peter's advisees, the left `author` node specifies or "anchors" the *head* node to abstract the query Peter, whereas the right one anchors the *tail* node to abstract the answer Steven. In contrast, to search for Steven's advisor, the opposite happens in Figure 6(b). Thus, the same metagraph can be anchored in different ways to capture different semantics and directions of the search. In particular, the head and tail anchors provide the necessary granularity to discriminate nodes within a metagraph. Note that, for simplicity and practicality, we only discuss the case with two anchors or roles. However, we emphasize that it is also straightforward to anchor multiple nodes with multiple latent roles. A formal definition is presented below.

Definition 4 (Anchored metagraph). An *anchored metagraph* is represented as a triple $A = (M, h, t)$, where $M = (V_M, E_M, \tau_M)$ is a metagraph, and $h, t \in V_M$ respectively define the head and tail nodes in M for some $h \neq t$. □

The above definition implies that the same metagraph can generate more than one anchored metagraphs. For instance, the metagraph in Figure 5(b) corresponds to two different anchored metagraphs A_1 and A_2, shown in Figure 6(a) and (b). In other words, anchored metagraphs are more granular and expressive than metagraphs.

We further note that anchored metagraphs can subsume metagraphs as a special case, when two nodes play commensurate or balanced roles within a metagraph. Using the metagraph `user–employer–user` as an example, there is only one corresponding anchored metagraph, because $\texttt{user}_{\texttt{[head]}}\texttt{–employer–}\texttt{user}_{\texttt{[tail]}}$ and $\texttt{user}_{\texttt{[tail]}}\texttt{–employer–}\texttt{user}_{\texttt{[head]}}$ are isomorphic, meaning that they are in fact identical to each other. Therefore, anchored metagraphs can be adopted universally, regardless of whether there exist different latent roles.

5.2 Node and Edge Representations

The frequency of a node's occurrences in an anchored metagraphs at a specific anchor can be treated as a feature for this node. Likewise, the frequency of two nodes' co-occurrences in an anchored metagraph at specific anchors is a feature for the (potential) edge between the two nodes.

Consider a collection $\mathcal{A} = \{A_1, A_2, \ldots\}$, where each $A_i = (M_i, h_i, t_i)$ is an anchored metagraph. Note that it is possible to have $M_i = M_j$ for some $i \neq j$ since the same metagraph can generate multiple anchored metagraphs. We can subsequently encode any node and any pair of nodes using these anchored metagraphs.

Node representation. We can represent any node v on the graph using two vectors $\mathbf{a}_v^h$ and $\mathbf{a}_v^t$, each with $|\mathcal{A}|$ elements. The i-th elements, $\mathbf{a}_v^h[i]$ and $\mathbf{a}_v^t[i]$, record the number of instances of M_i containing v at its head and tail anchors in $\mathcal{A}_i$, respectively. That is,

$$\mathbf{a}_v^h[i] \triangleq |\{S \in \mathcal{I}(M_i) : v \in V_S, \phi(v) = h_i\}|, \quad (8)$$

$$\mathbf{a}_v^t[i] \triangleq |\{S \in \mathcal{I}(M_i) : v \in V_S, \phi(v) = t_i\}|. \quad (9)$$

Recall that ϕ is the bijection between the nodes in an instance and the nodes in a metagraph, as introduced in Definition 1.

The two vectors are mirror images of each other: $\mathbf{a}_v^h[i] = \mathbf{a}_v^t[j]$ if $M_i = M_j$ and $h_i = t_j$, a result immediately follows from their definitions. As an example, consider $\mathcal{A} = \{A_1, A_2\}$ shown in Figure 6. Since they are derived from the same metagraph, $M_1 = M_2$ and $h_1 = t_2$. Assuming the only matching instance in Figure 5(a), we have $\mathbf{a}_{\text{Peter}}^h = (1, 0)$ and $\mathbf{a}_{\text{Peter}}^t = (0, 1)$. The implication of the mirror image is that, if we only study each node in isolation, such as a node classification task where a label needs to be assigned to each node, without loss of generality, it is equivalent to use either $\mathbf{a}_v^h$ or $\mathbf{a}_v^t$ as long as we adhere to the same form for all the nodes consistently. However, when we study two nodes in conjunction, such as a proximity search task where there is a distinction of query and answer nodes, both forms are necessary depending on the node. We will further elaborate on this use case in Section 5.3.

Edge representation. We can encode the edge representation between any two nodes v and u on the graph. Note that it is irrelevant whether an actual edge exists between v and u, as the goal here is to derive an effective representation for differentiating various states between v and u, including the absence and presence of edges, as well as different types of edges if present. Consider a vector $\mathbf{a}_{vu}$ of $|\mathcal{A}|$ elements. Its i-th elements, $\mathbf{a}_{vu}[i]$, captures the number of instances of M_i containing both v and u such that v is the head and u is the tail anchor. That is,

$$\mathbf{a}_{vu}[i] \triangleq |\{S \in \mathcal{I}(M_i) : (v, u) \in V_S^2, \phi(v) = h_i, \phi(u) = t_i\}|. \quad (10)$$

In general, $\mathbf{a}_{vu} \neq \mathbf{a}_{uv}$, due to the requirement to align the head and tail anchors. Therefore, the proposed edge representation based on anchored metagraphs is also able to model the *direction* of the edges between nodes. In contrast, metagraph-based representation can only model *undirected* edges since $\mathbf{m}_{vu} \equiv \mathbf{m}_{uv}$. In particular, $\mathbf{a}_{vu}$ degenerates into $\mathbf{m}_{vu}$ if the head and tail anchors are symmetric to each other in every metagraph.

Matching. As evident in Eq. 8, 9 and 10, the node and edge representations can be derived from the instances of the original metagraph $\mathcal{I}(M_i)$, as well as the head h_i and tail t_i which can be understood as indices that point to the head and tail nodes. Thus, for any matched instance, the head and tail nodes can be easily located by indexing into the instance with h_i or t_i in $O(1)$ time, without requiring any additional matching. In other words, the matching algorithm in Section 4 can still be applied as is.

Remark. Our node and edge representations are derived solely based on matching instances on the graph, without requiring any task-specific supervision. Essentially, the same representations can be used universally in different downstream tasks, including supervised tasks such as semantic proximity search, node classification or relationship prediction, as well as unsupervised tasks such as node clustering.

5.3 Use Case: General Semantic Proximity

The proposed node representation can be immediately used for node-centric machine learning tasks, such as node classification and clustering. Likewise, the edge representation can be used for edge-centric tasks such as link prediction. In either scenario, standard supervised and unsupervised learning algorithms can be applied.

Apart from standard learning tasks, as one of the main use case in this paper, we introduce a more general form of semantic proximity based on the proposed node and edge representations. Similar to Definition 2, two nodes are more likely to satisfy the desired class of proximity if they "share" or co-occur in many characteristic anchored metagraphs. Furthermore, to ensure that such co-occurrences are not by chance, each of the two nodes should appear in fewer anchored metagraphs individually. However, in the general case, we must also differentiate the head and tail anchors in order to support potentially distinct roles played by query and answer nodes.

Definition 5 (General semantic proximity). The *general semantic proximity* between a query node q and a candidate answer node v is

$$\Pi(q, v; \mathbf{w}) \triangleq \frac{2\, \mathbf{a}_{qv}^{\top}\mathbf{w}}{\mathbf{a}_q^{h\,\top}\mathbf{w} + \mathbf{a}_v^{t\,\top}\mathbf{w}}, \tag{11}$$

for some non-negative vector $\mathbf{w}$ of $|\mathcal{A}|$ elements. □

This definition notably does not satisfy the symmetry property in Theorem 1 in the general case, which is an expected behavior when query and answer nodes play different roles. Nonetheless, as a special case, Π subsumes π and thus satisfy symmetry if the head and tail anchors are symmetric to each other in every metagraph.

TABLE 2: Summary of datasets.

Graph	# Nodes	# Edges	# Types	# Metagraphs
LinkedIn	65 925	220 812	4	153
Facebook	5 025	100 356	10	934
DBLP	172 136	968 822	5	74

6 Experiments

The goal of our experiments is twofold. First, the proposed metagraph-based approach can effectively model semantic proximity. Second, the proposed metagraph matching algorithm is efficient.

6.1 Experimental Setup

Datasets. We conducted extensive experiments on two real-world datasets collected by previous studies, namely LinkedIn [9] and Facebook [8], as summarized in Table 2 and elaborated below.

- *LinkedIn.* The graph contains objects of four types: `user`, `employer`, `location` and `college`. The relationships between some `user` pairs are labeled into different semantic classes. We chose two major classes, namely, College friend and Coworker[1].
- *Facebook.* The graph includes the following types: `user`, `concentration`, `degree`, `school`, `hometown`, `last-name`, `location`, `employer`, `work-location` and `work-project`.[2] Given no explicit labels, we generated the ground truth based on rules mimicking natural classes of proximity. Specifically, we considered two classes: (i) Family, two users sharing the same `last-name` as well as the same `location` or `hometown`; (ii) Classmate, two users sharing the same `school` as well as same `degree` or `concentration`. Notwithstanding the rules, we dictated a 5% chance to assign a random class as noises.

Training and testing. A user q can be used as a query node if there exists at least another user v such that the relationship between q and v belongs to the desired class of proximity in our ground truth. 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 report the results averaged over the 10 splits.

In each split, based on the training queries, we further generated training examples (q, x, y) such that q and x belong to the desired class while q and y do not. For testing, we constructed an ideal ranking for each test query node and desired class, which is compared against the ranking generated by various proximity algorithms. In particular, we adopted NDCG and MAP [14] to evaluate the quality of the rankings at top 10 nodes.

Metagraphs. As discussed in Section 2, we applied GRAMI [13] to mine the set of metagraphs. Preprocessing was done to prune less viable metagraphs. First, a viable metagraph must have at least two "core" nodes, which are `user` nodes in our case, since our ground truth is designed for the proximity between such node pairs. Additionally, the core nodes should be symmetric to each other in their containing

1. Including those labeled as "colleague" and "excolleague".
2. Other types are not used due to their sparsity or irrelevance.

metagraph, since our target proximity such as coworker and classmate are symmetric. In general, the symmetry-based pruning is not always necessary, as we shall see in Section 7.1 where asymmetric relationships are being dealt with. Second, a metagraph must contain at least two different types in order to capture the heterogeneity. Third, we removed metagraphs with "dangling" nodes, which are non-core nodes with degree one, as such nodes often do not explain the interactions between core nodes. Finally, we restricted metagraphs to have at most 5 nodes, which are found to be adequate in expressing complex interactions.

We apply logarithms on the metagraph indices (Eq. 1 and 2). That is, a raw count f would transform into $\log(f+1)$, as raw counts often generate sublinear returns.

6.2 Empirical Results on Semantic Proximity Search

Comparison to baselines. We first evaluate our proposed method against baseline methods, as follows.

- **MGP**: The proposed metagraph-based proximity.
- **MPP**: Metapath-based proximity, by restricting the set of metagraphs to paths only.
- **MGP-U**: Metagraph-based proximity with uniform weights. That is, we do not differentiate the importance of metagraphs to any semantic class.
- **MGP-B**: Proximity based on the single best metagraph.
- **SRW**: Supervised random walks [7], a supervised variant of personalized PageRank [2]. The general principle is to learn different weights for edges, so that the transition matrix is biased to make certain nodes more likely to be visited in accordance with the training data.

In our experiments, we varied the number of training examples from 10 to 1000. Note that this has no effect on MGP-U, as it simply uses a uniform weighting independent of the training data. Moreover, for our methods, no dual-stage training is employed, which will be investigated separately next.

We report the NDCG and MAP of the rankings produced by these algorithms in Figure 7 and 8, respectively. The first key finding is that, MGP performs consistently better than all other algorithms, by more than 10% in many cases. As our second finding, we observe a steady increase in the performance of MGP when the number of training examples grows, indicating that our learning is effective.

Evaluation of dual-stage training. We further investigate the efficacy of dual-stage training. Treating the ranking accuracy (NDCG and MAP) and time of using only the seed metagraphs $\mathcal{K}_0$ as 0%, and those of using all metagraphs $\mathcal{M}$ as 100%, we compute the relative percentage in accuracy and time when we vary the number of candidates $|\mathcal{K}|$.

We present the outcomes in Fig. 9. As we increase the number of candidates, we expect an increase in both accuracy and time. In particular, the rate of increase in accuracy is much faster than in time. Using only 50 and 150 candidates on LinkedIn and Facebook, respectively, the increase in accuracy is approaching 100% (*i.e.*, as good as using all metagraphs). Meanwhile, the time cost is far from reaching 100% (*i.e.*, requiring much less time than using all metagraphs). Comparing with using all the metagraphs, we sacrifice the ranking accuracy by only 1% in absolute values, on average. In contrast, we can reduce the overall matching time by an average of 83%. In summary, our dual-stage training can significantly reduce overall matching time with only a minuscule impact on the ranking accuracy.

6.3 Empirical Results on Metagraph Matching

We further examine the efficiency of metagraph matching. In particular, we compare the proposed symmetry-based algorithm, denoted by SymISO, with three state-of-the-art baselines: BoostISO [19], TurboISO [18] and QuickSI [16]. To further illustrate the importance of node matching orders in SymISO, we also compare to a weaker scheme of SymISO that uses a random matching order, dubbed SymISO-R.

We illustrate the average running time per metagraph for all the algorithms in Figure 10, where the size of metagraphs varies. Observe that SymISO consistently outperforms the best baseline, namely BoostISO, by 52% on average among metagraphs of all sizes. When the number of nodes in a metagraph increases, the performance margins between SymISO and the baselines become larger, since more redundant computation can be avoided due to larger symmetric components. In particular, on metagraphs with merely three nodes, SymISO is better by only about 10%. Furthermore, SymISO is also faster than SymISO-R in all cases, suggesting the usefulness of our matching order.

7 Further Experiments on Representation

We present additional empirical results on general representations based on anchored metagraphs. Specifically, the proposed representations perform consistently well across common learning tasks and classifiers.

7.1 Experiment Setup

Dataset. We used DBLP [22], which naturally contains not only balanced roles such as colleagues, but also imbalanced roles such as advisors and advisees. The graph includes the types of `paper`, `author`, `year`, `venue` and `keyword`. Some of the `author` pairs are labeled as "adivsor–advisee" or "colleague". We leveraged these labels as the ground truth for three learning tasks, to be further elaborated later. We only retained `paper` nodes connected to at least one `author` node appearing in the ground truth. A summary of the graph is shown in Table 2.

Metagraphs. Metagraphs were filtered and processed in the same way as in Section 6.1, except that we kept asymmetric metagraphs and restricted the metagraph size to 6. Generally, a metagraph of size 5 or 6 is complex enough to capture rich semantics, as our experiments in Section 7.3 will further demonstrate. In total 74 metagraphs remained, from which we enumerated all possible combinations of head and tail anchors on each metagraph, and obtained 101 anchored metagraphs. Note that the number of combinations of head and tail is relatively small since they can only be assigned to core nodes, and some combinations result in isomorphic anchored metagraphs.

Downstream tasks. We considered four machine learning tasks, as follows.

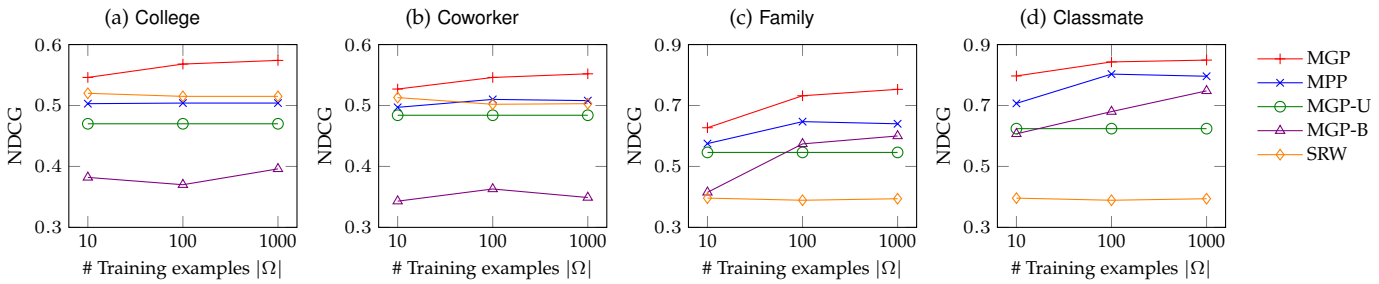


Fig. 7: Evaluation of metagraph-based proximity and baselines with NDCG.

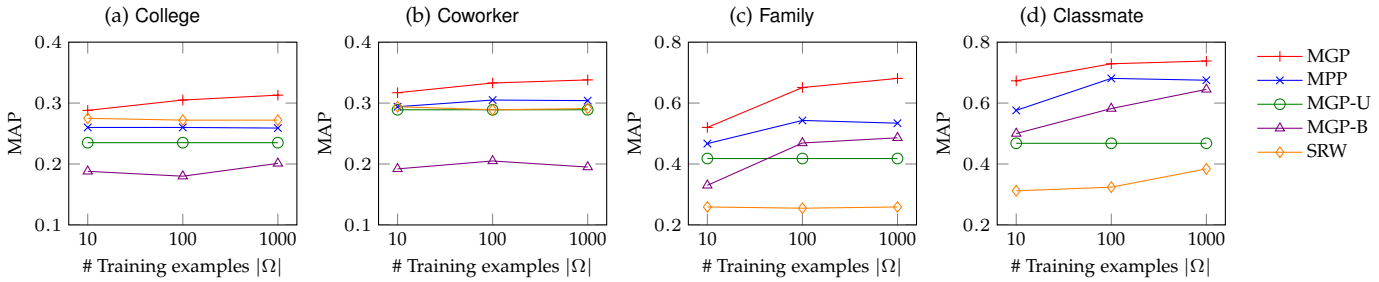


Fig. 8: Evaluation of metagraph-based proximity and baselines with MAP.

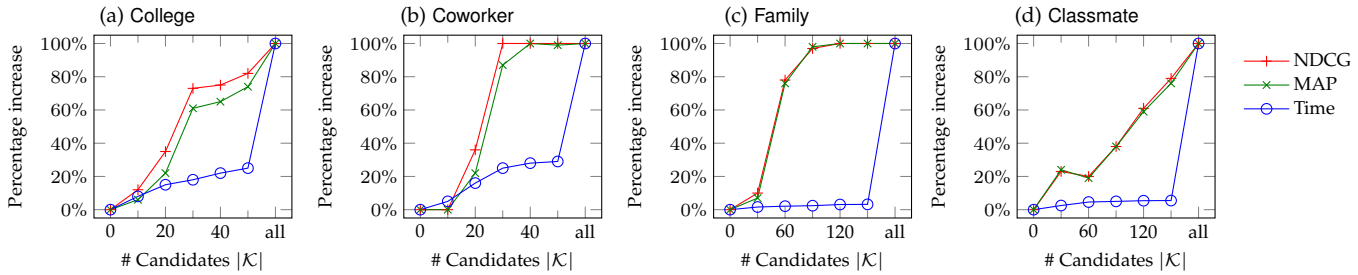


Fig. 9: Impact of dual-stage training (special values of $|\mathcal{K}|$: 0 if only use seed metagraphs; "all" if use all metagraphs).

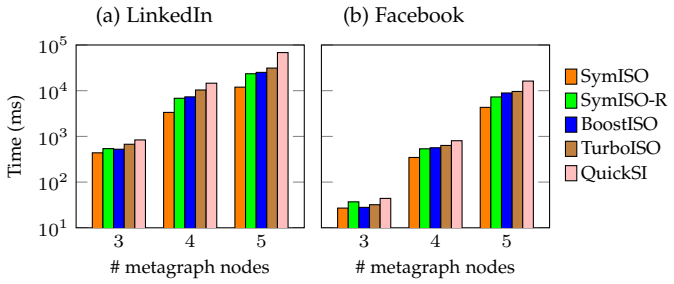


Fig. 10: Average matching time per metagraph.

- *Semantic proximity search.* We search for the advisor(s) of a given `author` query node. Note that this is an asymmetric search given the the imbalanced roles of advisor and advisee. Training, testing and evaluation followed the set up in Section 6.1, using 1 000 training examples.
- *Binary node classification.* We classify a given `author` node as an advisor (Yes) or otherwise (No). An `author` node belongs to the Yes class if and only if it has at least one advisee. We split the nodes in the ground truth into 80% training and 20% testing data, repeated for 50 times, and report their average AUC (under the ROC curve) and F-score on the test sets.
- *Node clustering.* Similar to the node classification task above, we aim to separate advisors and non-advisors. However, we assume an unsupervised setting, where the nodes are clustered into two groups without any training data. We repeat the clustering with 100 random initializations and report their average results in terms of NMI and Rand index.
- *Multi-class relationship prediction.* We predict the relationship between two given `author` nodes, among three possibilities: IsAdvisor, IsAdvisee and IsColleague. Note that the IsColleague relationships are undirected, whereas the other two are directed. We split the node pairs in the ground truth into 80% training and 20% testing data, repeated for 10 times, and report their average micro and macro-averaged F-scores on the test sets.

7.2 Empirical Comparisons with Baselines

Baselines. We compared the following representations on the four tasks.

- **MG+**: The anchored metagraph-based representation, which is our proposed solution.

- **DeepWalk** [23]: an pioneering work on distributed representations for (homogeneous) graphs.
- **node2vec** [24]: a more advanced variant of DeepWalk that samples a mixture of depth and breadth-first walks. The mixture can be adjusted with parameters p and q, and we ran a grid search over $p, q \in \{0.5, 1, 2\}^2$ to choose the best setting. When $p = q = 1$, it reduces to DeepWalk.
- **GraphSAGE** [25]: A state-of-the-art, general graph representation learning framework that supports various neighborhood aggregators. While we tried multiple aggregators including mean, LSTM, pooling and GCN-based aggregation, we only report the mean aggregator due to its superior performance on most tasks. Finally, we adopted its unsupervised variant to align with MG+.
- **metapath2vec** [26]: A state-of-the-art heterogeneous graph embedding algorithm. Some domain knowledge is required to select a metapath to guide random walks. We tried two common metapaths used on bibliographic graphs, `author-paper-author` and `author-paper-venue-paper-author`, and only report the results of the latter given its superior performance.
- **hin2vec** [27]: A state-of-the-art heterogeneous graph embedding algorithm, which does not require the selection of a metapath.

Apart from various graph representations, we further compared with a classic graph algorithm called supervised random walk (**SRW**) [7], as already explained in Section 6.1. Note that SRW is a supervised method and thus cannot be applied to the unsupervised node clustering task.

For graph embedding baselines, we derive the edge representation from the concatenation of the two nodes' embeddings. Other common choices include taking their sum or Hadamard product, which cannot model directed edges and show inferior empirical results. For all representation-based methods including MG+, we applied the proposed learning to rank approach (Section 3.2) for semantic proximity search, k-means algorithm for node clustering, and logistic regression with elastic net for node classification and relationship prediction. We select hyperparameters via five-fold cross validation on the training sets.

Empirical results. We report the evaluation of MG+ and various baselines on the four tasks in Table 3. We observe that MG+ significantly outperforms all the baselines in all four tasks. Among the baselines, on the one hand, DeepWalk and node2vec tend to give weak results as they cannot leverage the rich semantics in heterogeneous graphs. On the other hand, metapath2vec and hin2vec, which accounts for the heterogeneity, generally outperform DeepWalk and node2vec. Moreover, the performances of GraphSAGE and SRW often falls in-between of the aforementioned two groups. Note that while MG+ is the consistent winner, no single baseline emerges as the consistent runner-up.

7.3 Analysis of Metagraphs

We further investigate the impact of varying meta-structures on the four tasks, in two aspects.

Nature of the structures. We compare MG+ to two reduced schemes: **MG** and **MP**, denoting non-anchored representations based on metagraphs and metapaths, respectively.

TABLE 3: Evaluation on the four tasks. Best results are bolded and marked with */† if significantly different from the runner-up at the 0.01/0.05 level under t-test.

	(a) Semantic proximity search		(b) Node classification	
	NDCG	MAP	AUC	F-score
DeepWalk	0.532	0.358	0.781	0.489
node2vec	0.535	0.352	0.788	0.494
GraphSAGE	0.902	0.766	0.858	0.636
metapath2vec	0.651	0.470	0.778	0.463
hin2vec	0.922	0.792	0.893	0.685
SRW	0.868	0.747	0.796	0.526
MG+	**0.949***	**0.875***	**0.909***	**0.699**†
	(c) Node clustering		**(d) Relationship prediction**	
	Rand	NMI	Micro-F	Macro-F
DeepWalk	0.031	0.006	0.540	0.420
node2vec	0.025	0.005	0.569	0.452
GraphSAGE	0.152	0.134	0.870	0.808
metapath2vec	0.002	0.001	0.836	0.694
hin2vec	0.020	0.124	0.547	0.447
SRW	-	-	0.367	0.306
MG+	**0.176***	**0.145***	**0.891***	**0.864***

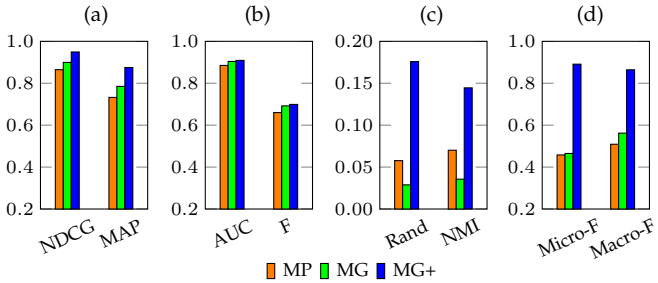


Fig. 11: Performance comparison of various metagraph schemes. (a) Semantic proximity search. (b) Node classification. (c) Node clustering. (d) Relationshihp prediction.

Without anchors, MG cannot distinguish different roles of nodes, whereas MP is a further simplification that only utilizes metapaths. Not surprisingly, MG+ consistently outperforms both MG and MP+, as reported in Fig. 11. Between MG and MP, MG often works better as metagraphs are more expressive and thus able to capture richer semantics.

We further examine a case study on the task of relationship prediction through confusion matrices. In both Table 4(a) and (b), the majority of the classification errors happen between IsAdvisor and IsAdvisee classes, which means both MP and MG are unable to differentiate the two directed relationships. While there is some improvement in MG over MP, the reduction of errors happens with the undirected IsColleague class, due to the more expressive metagraphs compared to metapaths. Nevertheless, the increased expressive power of metagraphs does not help with modeling directed relationships.

Size of structures. Next, we impose a size limit on the metagraphs to study the impact. In Fig. 12, we varied the size limit of metagraphs from 4 to 6. In most tasks, the performance becomes stable when we increase the size limit to 6, *i.e.*, the performance lift from 5 to 6 becomes much smaller than the lift from 4 to 5. In semantic proximity

TABLE 4: Confusion matrices of relationship prediction using MP, MG, MG+. A = IsAdvisor, E = IsAdvisee, C = IsColleague.

(a) MP

		Predicted		
		A	E	C
True	A	113	289	38
	E	126	288	33
	C	22	30	128

(b) MG

		Predicted		
		A	E	C
True	A	114	308	17
	E	134	297	15
	C	14	17	149

(c) MG+

		Predicted		
		A	E	C
True	A	398	30	11
	E	36	402	9
	C	21	23	136

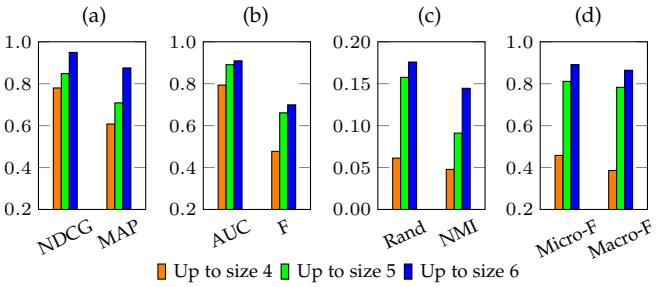


Fig. 12: Performance comparison of various metagraph sizes using MG+. (a) Semantic proximity search. (b) Node classification. (c) Node clustering. (d) Relationshihp prediction.

search, although the lift from size 5 to 6 is still substantial, the absolute performance is approaching the ceiling of 1.0. Thus, further increasing the size limit in this task would have marginal benefits too. Therefore, metagraphs of up to size 5 or 6 are often expressive enough to capture most complex semantics between nodes. It is worth noting that an existing study [28] on meta-structures also reaches a similar conclusion, where metagraphs of larger size may bring in remotely connected nodes with weaker semantic ties.

7.4 Empirical Results on Metagraph Matching

Finally, we compare our proposed matching algorithm SymISO with two state-of-the-art baselines CFLMatch [29] and BoostISO [19]. Note that the experiments in Section 6.3 are focused on symmetric metagraphs only since the semantic classes involved therein are all symmetric. In this section, we aim to evaluate both symmetric and asymmetric metagraphs, since our tasks involve asymmetric semantics. We illustrate the average running time per metagraph in Fig. 13, where we examine the size of metagraphs in Fig. 13(a), and the symmetry in Fig. 13(b). In all cases, our proposed

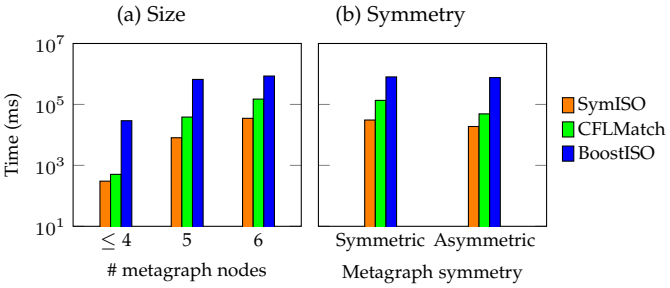


Fig. 13: Average matching time per metagraph.

SymISO outperforms the baselines. In particular, compared to CFLMatch, our performance edge on symmetric metagraphs is indeed larger than asymmetric ones.

8 Related Work

Meta-structures. While the less general concept of metapath has been proposed [6], metagraphs are more expressive and effective than metapaths in capturing interactions between nodes. Given the increased complexity and variety of metagraphs, we cannot handle metagraphs in the same way as metapaths. First, the metapath-based PathSim [6] relies on manually selecting the useful metapaths. It becomes difficult given the much larger number of metagraphs and arbitrary classes of proximity. Thus, we propose a supervised learning approach. While another work [30] also employs learning for metapaths, it is only designed for a different task of clustering. Second, metagraphs are much more difficult to match than metapaths. Thus, we develop a symmetry-based matching algorithm to improve efficiency. Finally, a recent work [28] proposes a directed acyclic graph as a meta-structure. However, they do not handle the general case where roles of nodes need to be differentiated. Similar structures have also been employed for other applications lately such as social influence analysis [31].

Proximity search. Most earlier research [2], [3], [5], [6] only measures a "generic" form of proximity on graphs. Different senses of proximity have also emerged, such as hub and authority [32], probabilistic precision and recall [33], [34], as well as importance and specificity [35], [36]. However, these senses are only formed due to specific patterns in the link structures (*i.e.*, non-semantic). Although there exist semantic-oriented studies on graphs, such as social circle learning [8] and relationship profiling [9], they do not support online query processing and thus cannot be easily adapted for proximity search. There also exist several random walk approaches [7], [37], [38], which learn from example ranking preferences [14] to bias transition probabilities between nodes of different types or features. However, they are equivalent to adjusting a linear combination of path probabilities only [2].

Subgraph matching. A plethora of techniques [10], [16], [17], [18], [19], [29] have been proposed for subgraph matching, which follow the backtracking framework as discussed in Section 4. Their major issue is the extremely huge search space on a large graph. To prune the search space, Shang et al. [16] have proposed a special ordering of nodes for matching instead of a random ordering. Subsequently, Han et al. [18] and Ren et al. [19] have introduced more improvements to further reduce and reuse redundant computation. However, they do not account for graph symmetry, which leads to substantial redundant computations.

Graph representation. Inspired by the success of word embedding approaches, advances in learning representations in an unsupervised fashion have been extended to graph data [39]. Earlier work such as DeepWalk [23], node2vec [24] and LINE [40] only deal with homogeneous graphs, without accounting for the complex semantics carried by a heterogeneous graph. A few studies on heterogeneous graphs exist, including state-of-the-art metapath2vec [26]

and hin2vec [27]. Both works utilize the guidance of metapaths, which are less expressive structures than metagraphs. More recently, end-to-end graph neural networks, such as GCN [41], GraphSAGE [25] and GAT [42], have emerged to learn graph representations in a supervised fashion. Comparing to these existing studies, our proposed representations have a few advantages. First, anchored metagraphs are able to capture direction-aware semantics through head and tail anchors that can model different roles. Second, our method tends to be more universal. Previous unsupervised methods [23], [24], [26] depend on skip-gram models to optimize node co-occurrences, and end-to-end methods [25], [41], [42] optimize task-specific goals. In contrast, anchored metagraphs capture a wide variety of fine-grained semantics and interations between nodes, instead of depending on only co-occurrences or task-specific supervisions. Third, our method is often more interpretable, since each dimension corresponds to one anchored metagraph that carries semantics and can be easily visualized.

9 Conclusion

In this paper, we proposed metagraph-based learning on heterogeneous graphs. Motivated by the problem of semantic proximity search, we identified and employed *metagraphs* to characterize arbitrary semantic classes, which can be learned in a supervised manner. We further generalized *metagraphs* to *anchored metagraphs*, in order to model nodes with distinct roles within the same metagraph. In particular, anchored metagraphs can be used to construct universal node and edge representations, to support various machine learning tasks such as semantic proximity search, node classification and link prediction. Finally, we also improved the efficiency of metagraph matching by eliminating redundant computations on symmetric components, which are present in the majority of our metagraphs. Empirical results on three real-world graphs consistently demonstrated the superior performance of metagraph-based learning.

Acknowledgement

This research was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant (Approval No. 18-C220-SMU-006).

References

[1] Y. Sun, "Mining heterogeneous information networks," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2012.
[2] G. Jeh and J. Widom, "Scaling personalized web search," in *WWW*, 2003, pp. 271–279.
[3] ——, "SimRank: a measure of structural-context similarity." in *KDD*, 2002, pp. 538–543.
[4] Y. Fang, W. Lin, V. W. Zheng, M. Wu, K. C. Chang, and X. Li, "Semantic proximity search on graphs with metagraph-based learning," in *ICDE*, 2016, pp. 277–288.
[5] Y. Koren, S. C. North, and C. Volinsky, "Measuring and extracting proximity in networks," in *KDD*, 2006, pp. 245–255.
[6] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "PathSim: Meta path-based top-k similarity search in heterogeneous information networks," *PVLDB*, vol. 4, no. 11, 2011.
[7] L. Backstrom and J. Leskovec, "Supervised random walks: Predicting and recommending links in social networks," in *WSDM*, 2011, pp. 635–644.
[8] J. J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks," in *NIPS*, 2012, pp. 548–556.
[9] R. Li, C. Wang, and K. C. Chang, "User profiling in an ego network: co-profiling attributes and relationships," in *WWW*, 2014, pp. 819–830.
[10] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, 1976.
[11] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *ICDM*, 2002, pp. 721–724.
[12] M. Kuramochi and G. Karypis, "Finding frequent patterns in a large sparse graph," in *ICDM*, 2004, pp. 345–356.
[13] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "GRAMI: frequent subgraph and pattern mining in a single large graph," *PVLDB*, vol. 7, no. 7, pp. 517–528, 2014.
[14] T.-Y. Liu, *Learning to rank for information retrieval*. Springer, 2011.
[15] R. J. P. van Berlo, W. Winterbach, M. J. L. de Groot, A. Bender, P. J. T. Verheijen, M. J. T. Reinders, and D. de Ridder, "Efficient calculation of compound similarity based on maximum common subgraphs and its application to prediction of gene transcript levels," *IJBRA*, vol. 9, no. 4, pp. 407–432, 2013.
[16] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: an efficient algorithm for testing subgraph isomorphism," *PVLDB*, vol. 1, no. 1, pp. 364–375, 2008.
[17] J. Lee, W. Han, R. Kasperovics, and J. Lee, "An in-depth comparison of subgraph isomorphism algorithms in graph databases," *PVLDB*, vol. 6, no. 2, pp. 133–144, 2012.
[18] W. Han, J. Lee, and J. Lee, "Turbo$_{\text{iso}}$: towards ultrafast and robust subgraph isomorphism search in large graph databases," in *SIGMOD*, 2013, pp. 337–348.
[19] X. Ren and J. Wang, "Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs," *PVLDB*, vol. 8, no. 5, pp. 617–628, 2015.
[20] W. Lin, X. Xiao, J. Cheng, and S. S. Bhowmick, "Efficient algorithms for generalized subgraph query processing," in *CIKM*, 2012, pp. 325–334.
[21] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu, "Incremental graph pattern matching," in *SIGMOD*, 2011, pp. 925–936.
[22] C. Wang, J. Han, Y. Jia, J. Tang, D. Zhang, Y. Yu, and J. Guo, "Mining advisor-advisee relationships from research publication networks," in *KDD*, 2010, pp. 203–212.
[23] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: online learning of social representations," in *KDD*, 2014, pp. 701–710.
[24] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.
[25] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1024–1034.
[26] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *KDD*, 2017, pp. 135–144.
[27] T. Fu, W. Lee, and Z. Lei, "Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning," in *CIKM*, 2017, pp. 1797–1806.
[28] Z. Huang, Y. Zheng, R. Cheng, Y. Sun, N. Mamoulis, and X. Li, "Meta structure: Computing relevance in large heterogeneous information networks," in *KDD*, 2016, pp. 1595–1604.
[29] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient subgraph matching by postponing cartesian products," in *SIGMOD*, 2016, pp. 1199–1214.
[30] Y. Sun, B. Norick, J. Han, X. Yan, P. S. Yu, and X. Yu, "Integrating meta-path selection with user-guided object clustering in heterogeneous information networks," in *KDD*, 2012, pp. 1348–1356.
[31] J. Zhang, J. Tang, Y. Zhong, Y. Mo, J. Li, G. Song, W. Hall, and J. Sun, "Structinf: Mining structural influence from social streams," in *AAAI*, 2017, pp. 73–80.
[32] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, no. 5, pp. 604–632, 1999.
[33] G. Agarwal, G. Kabra, and K. C. Chang, "Towards rich query interpretation: walking back and forth for mining query templates," in *WWW*, 2010, pp. 1–10.
[34] Y. Fang and K. C. Chang, "Searching patterns for relation extraction over the Web: rediscovering the pattern-relation duality," in *WSDM*, 2011, pp. 825–834.
[35] V. Hristidis, H. Hwang, and Y. Papakonstantinou, "Authority-based keyword search in databases," *TODS*, vol. 33, no. 1, 2008.
[36] Y. Fang, K. C. Chang, and H. W. Lauw, "RoundTripRank: Graph-based proximity with importance and specificity," in *ICDE*, 2013, pp. 613–624.

[37] S. Chakrabarti and A. Agarwal, "Learning parameters in entity relationship graphs from ranking preferences," in *PKDD*, 2006, pp. 91–102.

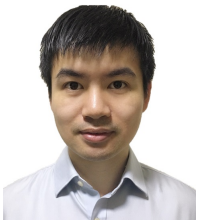
[38] A. Agarwal, S. Chakrabarti, and S. Aggarwal, "Learning to rank networked entities," in *KDD*, 2006, pp. 14–23.

[39] H. Cai, V. W. Zheng, and K. C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE TKDE*, vol. 30, no. 9, pp. 1616–1637, 2018.

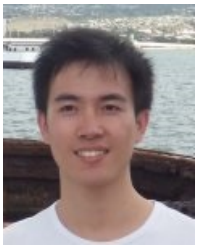
[40] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[41] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[42] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

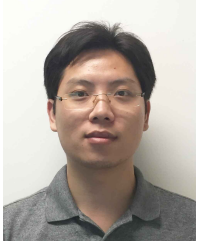


Yuan Fang received his Ph.D. degree in Computer Science from University of Illinois at Urbana-Champaign in 2014. He is currently an Assistant Professor in the School of Information Systems, Singapore Management University. His research focuses on graph-based machine learning and data mining, as well as their applications for the Web and social media.

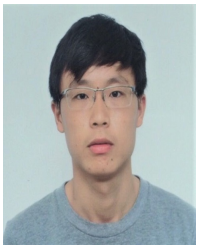


Wenqing Lin received his Ph.D. degree in Computer Science from Nanyang Technological University in 2015. Currently he is a Senior Researcher at Tencent in Shenzhen, China. His research interests include graph databases and data mining.

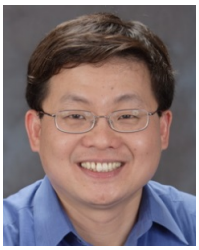
Vincent W. Zheng received his Ph.D. degree in Computer Science from Hong Kong University of Science and Technology in 2011. He is a senior tech lead in WeBank, China. Previously, he was a senior research scientist at the Advanced Digital Sciences Center, Singapore, and a research affiliate at the University of Illinois at Urbana-Champaign. He research interests include graph mining, information extraction, ubiquitous computing and machine learning.



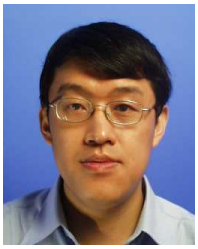
Min Wu received his Ph.D. degree in Computer Science from Nanyang Technological University in 2011. He is currently a Senior Scientist at the Data Analytics Department, Institute for Infocomm Research, Singapore. His research interests include graph mining, machine learning, as well as bioinformatics.



Jiaqi Shi received his Master's degree in Data Sciences from ESSEC Business School in 2019. He is currently a Research Engineer in the School of Information Systems, Singapore Management University. His research interests include graph mining, machine learning and information systems.



Kevin Chen-Chuan Chang is a Professor in the Department of Computer Science, University of Illinois at Urbana-Champaign. His research addresses large-scale information access, for search, mining, and integration across structured and unstructured big data including Web data and social media. He also co-founded Cazoodle for deepening vertical data-aware search over the Web.



Xiao-Li Li is the Department Head and a Principal Scientist of the Data Analytics Department, Institute for Infocomm Research, Singapore. He also holds adjunct associate professorship at Nanyang Technological University. His research interests include data mining, machine learning, bioinformatics and information retrieval.