Semantic Proximity Search on Graphs with Metagraph-based Learning*

Yuan Fang †, Wenqing Lin †, Vincent W. Zheng ‡, Min Wu †, Kevin Chen-Chuan Chang #‡, Xiao-Li Li †

† Institute for Infocomm Research, Singapore {yfang,linw,wumin,xlli}@i2r.a-star.edu.sg
‡ Advanced Digital Sciences Center, Singpaore vincent.zheng@adsc.com.sg
# University of Illinois at Urbana-Champaign, USA kcchang@illinois.edu

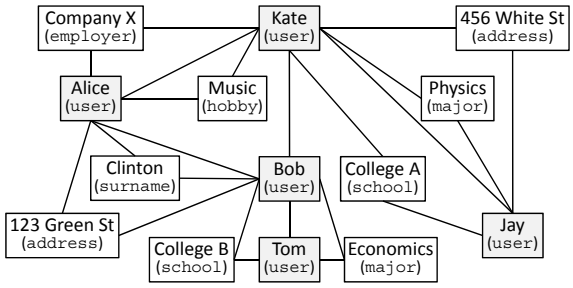
***Abstract*—Given ubiquitous graph data such as the Web and social networks, proximity search on graphs has been an active research topic. The task boils down to measuring the proximity between two nodes on a graph. Although most earlier studies deal with homogeneous or bipartite graphs only, many real-world graphs are heterogeneous with objects of various types, giving rise to different semantic classes of proximity. 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 classes of proximity. Thus, it becomes inadequate to only measure a "generic" form of proximity as previous works have focused on. In this paper, 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 supervised technique to automatically learn the right form of proximity within its family to suit the desired class. As it is expensive to match (*i.e.*, find the instances of) a metagraph, we propose the novel approaches of *dual-stage training* and *symmetry-based matching* to speed up. Finally, our experiments reveal that our approach is significantly more accurate and efficient. For accuracy, we outperform the baselines by 11% and 16% in NDCG and MAP, respectively. For efficiency, dual-stage training reduces the overall matching cost by 83%, and symmetry-based matching further decreases the cost of individual metagraphs by 52%.**

I. Introduction

The proliferation of the Web and social media has availed an increasingly rich collection of data objects. 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*, containing different types of objects. Consider the graph in Fig. 1(a) based on a toy social network, which interconnects various users and their attributes. Note that we treat each user and attribute value as a node, and each node is further associated with a type like `user`, `school` or other attribute names. We call this graph *typed object graph*, to be formally defined in Sect. II.

One important problem on graphs is *proximity search*. Given a query node $q \in V$, how do we measure the proximity between q and other nodes $v \in V$, so that we can return the nodes closest to q? However, most earlier studies, including Personalized PageRank [14] and SimRank [13], fail to capitalize on the rich semantics embedded in a heterogeneous graph. Specifically, with various types of interconnected objects, there exist different *semantic classes of proximity* arising from different underlying reasons, as illustrated in Fig. 1(b). 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. Such differentiation enables endless possibilities, as the two scenarios below envision.

(a) Toy graph (node type in parenthesis)



(b) Various semantic classes of proximity

Query	Semantic class	Result (with explanation)
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. 1: Example graph and semantic proximity search.

- *Circle-based friend suggestion.* On a social network (*e.g.*, Fig. 1), suggesting friends by circles greatly enhances user experience. For instance, who share the same passion as I do for Lakers? Who were my classmates? Each case is a circle or semantic class, and we can differentiate them based on user attributes such as `hobby` and `school`.
- *Context-aware citation search.* Consider a citation graph connecting papers, authors, journals and keywords. Automatically filtering papers by contexts can improve productivity. For instance, given a paper as the query, which citations addressed the same core problem? Which are simply background citations? Each case is a context or semantic class, and we can tell them apart based on paper attributes such as `keywords` and `journal`.

* This material is based upon work partially supported by the research grant for the Human-centered Cyber-physical Systems Programme at the Advanced Digital Sciences Center from Singapore's Agency for Science, Technology and Research. 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.

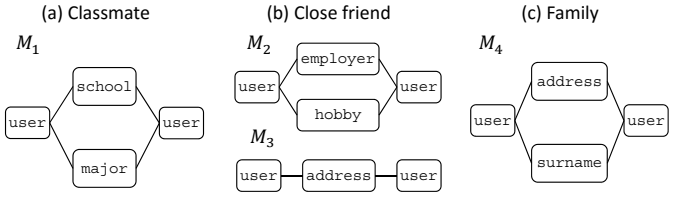


Fig. 2: Possible toy structures for each class of proximity.

We call the task of searching for a desired semantic class of proximity w.r.t. a query node as *semantic proximity search*. It is a new problem in the sense that previous studies on proximity search [13], [14], [16], [26], [4] neither intend to explicitly differentiate the semantic classes, nor can effectively accomplish so. Beyond proximity search, the closest problems to ours are social circle learning [22] and relationship profiling [19] on graphs. In terms of *semantic proximity*, their circles or relationships are also semantic-oriented, but they only find latent clusters in an unsupervised way, and thus do not target specific classes of interest. In terms of *search*, they need lengthy optimization to obtain a global configuration, and thus cannot process ad-hoc queries in real time.

A. 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 efficient to process online for enabling real-time search.

We hinge on the novel insight that different semantic classes can often be characterized by different tell-tale common "structures." For instance, in Fig. 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 Fig. 2(a). Likewise, Fig. 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 name such common structures *metagraphs* (to be formally defined in Sect. II), 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 [26], which only considers common path structures between two nodes. In fact, metagraph M_3 in Fig. 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 discuss next.

Proximity learning. First, the characteristic metagraphs for an arbitrary class of proximity are often unknown. While domain experts can lend some guidance on certain special classes, it is impractical to rely on human wisdom alone. 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 Fig. 2(b), respectively. One may also say that M_2 is more likely to indicate close friends than M_3.

To better generalize to different graphs and semantic classes, we propose a *supervised approach* that automatically identifies 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.

To realize proximity learning, a further challenge arises from the enormous number of metagraphs [17], [7]. Even for our toy example in Fig. 1 and 2, there exist many different forms of metagraphs beyond M_1–M_4, such as two users connected to three common attributes (of types `address`, `school` and `major`), or a longer path (`user`–`hobby`–`user`–`address`–`user`). As a result, processing the numerous metagraphs becomes prohibitively expensive especially on a large graph, let alone learning based on these metagraphs.

Fortunately, we observe that not all metagraphs are useful for a given class of proximity. That is, many metagraphs do not characterize and are thus irrelevant to the desired class (*e.g.*, M_2–M_4 and many not shown are useless for classmates). Therefore, it is inefficient to spend time and computing resources on these irrelevant metagraphs. We propose a novel paradigm of *dual-stage training*, which allows us to identify a small number of candidate metagraphs that are the most promising. As a result, we can discard all the other metagraphs and save a significant amount of time.

Metagraph matching. Second, it is necessary to compute the instances of each metagraph in order to know what metagraphs are "shared" by any two given nodes. (Instances and related concepts will be defined in Sect. II.) 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 [28], [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 [32], [17], [7].

To compute the instances of a metagraph more efficiently, we propose a novel subgraph matching algorithm that exploits the symmetry of metagraphs. We observe that many useful metagraphs are symmetric, such as M_1–M_4 in Fig. 2. (We will further explain in Sect. II.) However, existing methods spend a large amount of redundant computation on symmetric substructures within a metagraph. Thus, we propose a *symmetry-based matching* algorithm which re-uses "symmetric" computation. As a result, we can avoid redundancy and substantially improve the efficiency of metagraph matching.

Notation	Description
$G = (V, E)$	an object graph
$M = (V_M, E_M)$	a metagraph of G
$\mathcal{M}$	the set of metagraphs on G
T	the set of object types
τ, τ_M	type mapping for graph and metagraph, resp.
$\mathcal{I}(M)$	the set of instances of M on G
$\mathbf{m}_x$, $\mathbf{m}_{xy}$	metagraph vector for node x and node pair x, y, resp.
Ω	the set of training examples
$\mathbf{w}$	characteristic weight vector
$\mathcal{K}_0, \mathcal{K}$	the set of seed and candidate metagraphs, resp.

Fig. 3: Frequently used notations.

B. Contributions

Hinged on the above insights, we adopt a supervised approach to realize semantic proximity search with metagraphs. To summarize, we make the following contributions.

- *Problem.* We identify a new problem of semantic proximity search, and propose the novel concept of metagraphs to represent different semantic classes. (Sect. II)
- *Learning model.* We first propose a family of metagraph-based proximity, which can quantify different semantic classes. We further develop a supervised model to learn the characteristic metagraphs to obtain the right "form" of proximity within the family. Lastly, we introduce a novel paradigm of dual-stage training to avoid handling the vast majority of unpromising metagraphs. (Sect. III)
- *Matching algorithm.* We devise an efficient symmetry-based metagraph matching algorithm, which exploits the symmetry of metagraphs to avoid redundant computation. (Sect. IV)
- *Experiments.* Our experiments on two real-world graphs showcase the superiority of our approach. For accuracy, we significantly outperform the baselines by 11% in NDCG and 16% in MAP. For efficiency, dual-stage training can reduce the overall cost of handling metagraphs by 83%, and symmetry-based matching can shorten the matching time for individual metagraphs by 52%. (Sect. V)

II. Preliminaries and Overall Framework

Towards semantic proximity search, we first present some preliminaries, and then outline the overall framework.

A. Preliminaries

We formalize the notions of object graph and metagraph, as well as the task of semantic proximity search. Major notations are summarized in Fig. 3.

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

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 *metagraph*. Formally, a metagraph can be represented as $M = (V_M, E_M)$, 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 mapping function for metagraphs. Note that a node on the object graph has both an intrinsic value (*e.g.*, "Alice" or "Company X") and type, whereas the value of a node on the metagraph is immaterial and only the type matters.

Since metagraphs describe common structures on the object graph, they can capture various interactions between nodes on the graph. One common and useful interaction between two nodes is the co-owning of one or more attribute values, which can be modeled by *symmetric* metagraphs like M_1–M_4 in Fig. 2. More generally, symmetric metagraphs can also capture two nodes connecting to some attributes indirectly, as in "A"–"B"–"Company X"–"C"–"D" where "A" and "D" eventually converge to "Company X." Certainly, there also exist scenarios where asymmetric metagraphs would be useful. However, for the purpose of this paper, which is to illustrate the key concept of metagraphs, we only consider symmetric ones. The formal definition of symmetry will be introduced in Sect. IV when we present our symmetry-based matching algorithm. Note that our proposed learning framework in Sect. III can be applied to both symmetric and asymmetric metagraphs alike.

In order to know whether two nodes "share" a characteristic metagraph for the desired class of proximity, it is crucial to identify subgraphs on G that are instances of any 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 Fig. 1, the subgraphs "Alice"–"123 Green St"–"Bob" and "Kate"–"456 White St"–"Jay" can both match metagraph M_3 in Fig. 2, and thus they are the instances of M_3. We present a formal definition below.

DEFINITION 1 (INSTANCE OF METAGRAPH). Assume a subgraph $S = (V_S, E_S)$ and metagraph $M = (V_M, E_M)$. 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

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

Subsequently, we can quantify how two nodes x and y on G "share" any given metagraph, *i.e.*, how x and y co-occur in the instances of the metagraph. On graph G, let $\mathcal{M} = \{M_1, M_2, \ldots, M_{|\mathcal{M}|}\}$ be a set of metagraphs, and $\mathcal{I}(M_i)$ be the set of instances of $M_i \in \mathcal{M}$. The co-occurrences of x and y can be captured by a column vector with $|\mathcal{M}|$ elements, such that its i-th element, $\mathbf{m}_{xy}[i]$, is the number of instances of M_i containing both x and y. Likewise, let $\mathbf{m}_x[i]$ be the number of instances of M_i containing x. That is,

$$\mathbf{m}_{xy}[i] \triangleq |\{S \in \mathcal{I}(M_i) | x \in V_S \wedge y \in V_S\}|, \quad (1)$$

$$\mathbf{m}_x[i] \triangleq |\{S \in \mathcal{I}(M_i) | x \in V_S\}|. \quad (2)$$

We call $\mathbf{m}_{xy}$ and $\mathbf{m}_x$ *metagraph vectors*. As we shall see later, these vectors form the basis of our proximity measure.

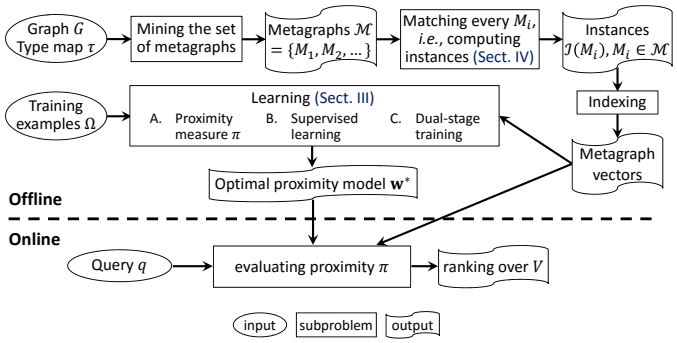


Fig. 4: Overall framework for semantic proximity search.

Semantic proximity search. On a graph $G = (V, E)$, 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 given class. As we adopt a supervised approach, the desired class of proximity can be substituted by a set of training examples Ω, which will be elaborated later. Thus, the task of semantic proximity search boils down to the core problem of defining and learning a family of proximity measures that can abstract arbitrary classes of proximity.

B. Overall Framework

To summarize our approach, we present the overall framework in Fig. 4. It consists of online and offline phases, for two reasons. First, we can mine the set of metagraphs and compute their instances independently of proximity search, which only has to be done once to cater to any class of proximity or query node. Second, we only need to learn the optimal model once for each class of proximity, in order to answer any query for the same class. We describe the two phases below.

Offline phase. It consists of three main subproblems.

1) Initially, given a graph G and a type mapping function τ, we enumerate the set of metagraphs $\mathcal{M}$. There is abundant literature [17], [7] on this subproblem, and its time cost is insignificant compared to that of the entire offline phase. Therefore, we simply apply an existing state-of-the-art approach GRAMI [7].

2) For each metagraph $M_i \in \mathcal{M}$ from the output of subproblem 1, we compute the set of instances $\mathcal{I}(M_i)$, so that we can further derive the metagraph vectors. We also call the process of computing the instances $\mathcal{I}(M_i)$ as *matching* metagraph M_i. We study this subproblem of *efficient metagraph matching* in Sect. IV. (We can further index the instances, which is to precompute the metagraph vectors based on Eq. 1–2. The metagraph vectors are part of the input to the learning subproblem as well as online query processing.)

3) Lastly, given the metagraph vectors and a set of training examples for the desired class of proximity, we need to learn the optimal proximity model $\mathbf{w}^*$. We address this subproblem of *learning with metagraphs* in Sect. III. In particular, we need to develop a family of metagraph-based proximity π to accommodate arbitrary classes, a supervised method to learn the best form of π within the family from the training examples Ω, as well as a dual-stage training approach to reduce the number of metagraph needed.

Online phase. In the online phase, given a query node $q \in V$, the precomputed metagraph vectors, as well as an optimal model $\mathbf{w}^*$ for the desired class, we can compute the proximity π between q and other nodes $v \in V$. Subsequently, we rank the nodes in V in descending order of π.

III. Learning Proximity with Metagraphs

In this section, 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 of the desired class, we develop a supervised approach to choose the optimal model within the proximity family. Lastly, we propose a novel paradigm of dual-stage training, which do not require us to compute the instances of all metagraphs.

A. Metagraph-based Proximity

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

Definition 2 (MGP). The *metagraph-based proximity* (abbreviated as MGP) between any two nodes x and y is

$$\pi(x, y; \mathbf{w}) \triangleq \frac{2\ \mathbf{m}_{xy} \cdot \mathbf{w}}{\mathbf{m}_x \cdot \mathbf{w} + \mathbf{m}_y \cdot \mathbf{w}}, \tag{3}$$

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

MGP entails a family of proximity measures with parameter $\mathbf{w}$. We interpret $\mathbf{w}$ as a *characteristic weight vector* of the metagraphs, which can be varied to cater to different classes of proximity. Consider the toy example in Fig. 2 with $\mathcal{M} = \{M_1, \dots, M_4\}$. A good $\mathbf{w}$ could be $(0.9, 0, 0, 0)^T$ for classmate, $(0, 0.6, 0.4, 0)^T$ for close friends, and $(0, 0, 0, 0.8)^T$ for family. Thus, within the family of proximity, the optimal model for a class is completely specified by its optimal weight vector $\mathbf{w}^*$, which we aim to learn automatically.

Interestingly, MGP satisfies a few desirable properties, as summarized in Theorem 1. Among them, symmetry, self-maximum and scale-invariance can be easily derived from Eq. 3. Partial transitivity implies that if one node x is close to both y and z, y 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. We omit its proof due to space constraint.

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

- *Symmetry*. $\pi(x, y; \mathbf{w}) = \pi(y, x; \mathbf{w})$.
- *Self-maximum*. $\pi(x, y; \mathbf{w}) \in [0, 1]$ and $\pi(x, x; \mathbf{w}) = 1$.
- *Scale-invariance*. $\pi(x, y; \mathbf{w}) = \pi(y, x; c\mathbf{w})$ for any $c > 0$.
- *Partial transitivity*. There exists some $\delta > 0$, such that for any $\epsilon \in [0, 0.5]$, if $\pi(x, y; \mathbf{w}) \geq \frac{1+2\epsilon\delta}{1+\delta}$ and $\pi(x, z; \mathbf{w}) \geq \frac{1+2\epsilon\delta}{1+\delta}$, then $\pi(y, z; \mathbf{w}) \geq 2\epsilon$. □

B. Supervised Learning

For a desired class of proximity, we assume some training examples Ω as supervision. Just as pairwise learning to rank [21], each example is a triplet (q, x, y) such that node x is ranked *before* node y w.r.t. query node q. That is, x's proximity to q should be greater than y's. These examples for social networks can often be gathered by user studies [22], [19]. Some platforms like Facebook and Google+ also allow users to label the classes of their connections directly.

Objective function. Given the training examples Ω, we can find the optimal weights $\mathbf{w}^*$ by maximizing the log-likelihood. Intuitively, it becomes more likely to observe an example (q, x, y) when x's proximity to q is increasingly larger than y's. In other words, the probability of the example, $P(q, x, y; \mathbf{w})$, tends to increase with the difference in x and y's proximity to q, $\pi(q, x; \mathbf{w}) - \pi(q, y; \mathbf{w})$. In particular, we define the probability in Eq. 4. We adopt the sigmoid function to transform the difference in proximity into a probability value, which is a common practice [21]. Note that $\mu \in (0, \infty)$ is a scaling variable to control the shape of the distribution.

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

Subsequently, we aim to maximize the following log-likelihood function L given all the examples, to ultimately find the optimal weights $\mathbf{w}^* = \arg\max_{\mathbf{w}} L(\mathbf{w}; \Omega)$.

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

Optimization. We employ the standard gradient ascent algorithm to maximize L. First, the gradient of L is denoted as

$$\nabla L = \left(\frac{\partial L}{\partial \mathbf{w}[1]}, \frac{\partial L}{\partial \mathbf{w}[2]}, \ldots, \frac{\partial L}{\partial \mathbf{w}[|\mathcal{M}|]} \right)^T.$$

Then, starting from some initial random weights $\mathbf{w}^{(0)}$, the iterative updating in Eq. 6 is applied for $k \in \{0, 1, \ldots\}$ until convergence. That is, the optimal weights $\mathbf{w}^*$ is given by $\mathbf{w}^{(k+1)}$ for some large enough k. Note that $\gamma \in (0, \infty)$ is a parameter to control the learning rate of gradient ascent.

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \gamma \nabla L \left(\mathbf{w}^{(k)} \right) \tag{6}$$

Thus, the optimization boils down to solving the gradient ∇L or the partial derivatives for $i \in \{1, \ldots, |\mathcal{M}|\}$,

$$\frac{\partial L}{\partial \mathbf{w}[i]} = \sum_{\Omega} \left\{ \mu(1 - P(q, x, y; \mathbf{w})) \left(\frac{\partial \pi(q,x;\mathbf{w})}{\partial \mathbf{w}[i]} - \frac{\partial \pi(q,y;\mathbf{w})}{\partial \mathbf{w}[i]} \right) \right\},$$

where $\frac{\partial \pi(v,u;\mathbf{w})}{\partial \mathbf{w}[i]} = \frac{2(\mathbf{m}_v \cdot \mathbf{w} + \mathbf{m}_u \cdot \mathbf{w})\mathbf{m}_{vu}[i] - 2(\mathbf{m}_{vu} \cdot \mathbf{w})(\mathbf{m}_v[i] + \mathbf{m}_u[i])}{(\mathbf{m}_v \cdot \mathbf{w} + \mathbf{m}_u \cdot \mathbf{w})^2}$.

Due to the scale-invariance property (Theorem 1), the absolute weights $\mathbf{w}[i]$ do not matter, and only their relative ratios are important. Thus, to make the weights easier to interpret, we can further constrain them to fall in $[0, 1]$.

C. Dual-Stage Training

As illustrated in Sect. I, there exist a huge number of metagraphs $\mathcal{M}$ even with just a few types of object. While it is already non-trivial to compute the instances of a single metagraph (which will be addressed in Sect. IV), it becomes prohibitive to compute the instances of all metagraphs, *i.e.*, $\mathcal{I}(M_i), \forall M_i \in \mathcal{M}$. To reduce the overall matching time, we propose a novel paradigm of dual-stage training. Unlike previous cost-aware learning paradigms [5], [23], [31], [30] which only aim to reduce feature extraction cost during *online testing*, our goal is to reduce feature extraction (*i.e.*, metagraph matching) cost for *offline training*. As we will see in the experiments, metagraph matching dominates the entire offline phase, whereas online testing can be extremely fast if metagraphs are pre-matched offline.

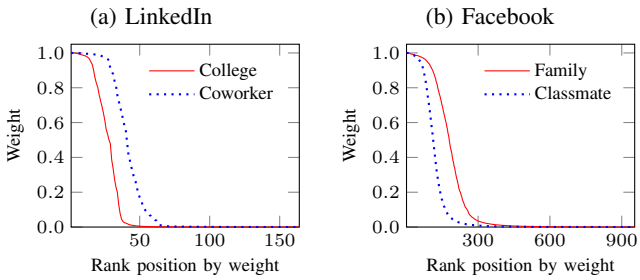


Fig. 5: Sparsity of optimal characteristic weights.

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. As illustrated in our toy example (Fig. 1–2), each class of proximity only has one or two characteristic metagraphs, among many other metagraphs (including many beyond M_1–M_4 not shown in the example). In other words, we expect that the optimal characteristic weight vector $\mathbf{w}^*$ is sparse, with many zero or nearly zero entries. Consider the two real-world datasets used in our experiments (details in Sect. V), LinkedIn (with proximity classes of college and coworker) and Facebook (with classes of family and classmate). We can indeed verify that the optimal $\mathbf{w}^*$ for each class of proximity is sparse, as shown in Fig. 5 where the individual weights $\mathbf{w}[i], \forall i \in \{1, \ldots, |\mathcal{M}|\}$ are ranked in descending order. As it turns out, the distribution of the optimal weights is long-tailed, with only 12% are high (above 0.9), while a overwhelming majority of 75% are close to zero (below 0.1).

Although these irrelevant metagraphs in the right tails contribute little to the desired class of proximity, computing their instances requires immense time and computing power. Thus, it is ideal to discard the irrelevant metagraphs, and 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. However, without computing the instances of any metagraph, there is no clue at all to locate the promising candidates. Thus, it is infeasible to select all candidates $\mathcal{K}$ from $\mathcal{M}$ in one go. 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.

- *Easy identification.* We can easily recognize the seeds without computing any instance (or it defeats the purpose).
- *Fast matching.* The seeds can be matched very fast. That is, $|\mathcal{K}_0| \ll |\mathcal{M}|$ and $\mathcal{I}(M), \forall M \in \mathcal{K}_0$ are fast to compute.

- *Candidate heuristic.* The seeds and their instances must enable some heuristic for selecting more candidates without computing any further instance.

To select the seeds, we observe that metapaths (*i.e.*, metagraphs that are paths such as M_3 in Fig. 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. Our experiments indeed show that only 2–3% metagraphs are metapaths, and matching a metapath can be 2–5 times faster than matching a non-path metagraph. Thus, if we choose all the metapaths as the seeds, the first two criteria are immediately met.

Candidate heuristic. For the third criteria, we need to develop a heuristic using the seeds in order to identify additional metagraphs from $\mathcal{M} \setminus \mathcal{K}_0$, without computing any further instance. Again, we can only rely on the structural information of the metagraphs. Can we infer the *function* of a metagraph from its *structure*? To answer the question, we first explain the structure and function of a metagraph, as follows.

On the one hand, the structure of a metagraph refers to its physical representation. Two metagraphs are structurally similar if they share some common representation. One such representation is their maximum common subgraph (MCS) [29]. The larger MCS shared by two metagraphs, the more structurally similar they are. Letting M_i and M_j be two metagraphs with MCS M, their structural similarity can be defined as $SS(M_i, M_j) = \frac{(|V_M|+|E_M|)^2}{(|V_{M_i}|+|E_{M_i}|)\times(|V_{M_j}|+|E_{M_j}|)}$.

On the other hand, the function of a metagraph refers to the role it plays in the proximity measure. Two metagraphs are functionally similar if they can characterize the same class of proximity. Assuming optimal characteristic weights $\mathbf{w}^*$, functional similarity of two metagraphs is reflected in their weights—the smaller difference between their weights, the more functionally similar they are. Thus, we define functional similarity as $FS(M_i, M_j) = 1 - |\mathbf{w}^*[i] - \mathbf{w}^*[j]|$.

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

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

Note that selecting metagraphs which are structurally similar to the seeds would potentially limit the diversity of candidates. Nonetheless, it is not an issue in our case, as there exist a diverse range of candidates (non-path metagraphs) similar to each seed (metapath), and even the most similar candidates are still quite different from the seeds.

Dual-stage approach. We outline the above heuristic in Alg. 1, which consists of two stages in training. In the *seed* stage, we compute the instances of the seeds $\mathcal{K}_0$ (*i.e.*, metapaths), 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 a new weight vector $\mathbf{w}^*$ for $\mathcal{K}_0 \cup \mathcal{K}$.

Algorithm 1: Dual-Stage Training

Input: graph G; set of metagraphs $\mathcal{M}$; number of candidates $|\mathcal{K}|$ (such that $|\mathcal{K}| \ll |\mathcal{M}|$); 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
```
$\mathcal{R} \leftarrow$ Order $\mathcal{M} \setminus \mathcal{K}_0$ by candidate heuristic H using $\mathcal{K}_0$ and $\mathbf{w}_0$
$\mathcal{K} \leftarrow$ Top $|\mathcal{K}|$ metagraphs in $\mathcal{R}$
$\mathcal{I} \leftarrow \{\mathcal{I}(M) | M \in \mathcal{K}\}$
$\mathbf{w}^* \leftarrow \text{Train}(\Omega, \mathcal{K}_0 \cup \mathcal{K}, \mathcal{I}_0 \cup \mathcal{I})$
return $\mathbf{w}^*$.

More generally, we can extend this approach to a multi-stage process, such that the candidates $\mathcal{K}$ are identified not all in one stage, but progressively in multiple stages. In each stage, we identify a small batch of candidates $\mathcal{K}_i$, treating K_0 and previously identified candidates $K_1, \ldots, K_{i-1}$ as the new seeds. Essentially, we gradually add more candidates, and stop once the training accuracy becomes acceptable.

IV. Efficient Metagraph Matching

In this section, we address the subproblem of metagraph matching, which dominates the offline phase in time cost. That is, for any metagraph M of graph $G = (V, E)$, we develop an efficient algorithm to compute $\mathcal{I}(M)$. We first summarize existing matching algorithms and highlight their drawbacks that render them inefficient to process the symmetric metagraphs. To address these drawbacks, we present a new algorithm to reduce the redundant computation.

A. Subgraph Matching Revisited

Consider a metagraph $M = (V_M, E_M)$ on a graph $G = (V, E)$. To compute the instances of M on G, there are a number of existing approaches [25], [18], [11], [24] 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 $1 \leq i \leq |V_M|$. 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 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 $D_2 = D_1 \cup \{v'\}$,

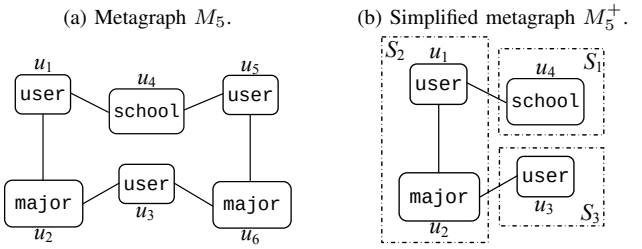


Fig. 6: A metagraph M_5 and its simplified metagraph M_5^+.

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.

B. Graph Symmetry

All above approaches focus on matching general metagraphs. As discussed in Sect. II, useful interactions between nodes are often be captured by symmetric metagraphs. In practice, among all metagraphs with two user nodes (suppose we are measuring the proximity between user nodes), symmetric metagraphs also form a significant portion. Thus, we devise a matching algorithm specifically catered to symmetric metagraphs. While we only consider symmetric metagraphs in this paper, we can still fall back to existing algorithms for asymmetric metagraphs whenever needed. In any case, our learning framework remains unchanged.

First, we formally define graph symmetry as follows.

DEFINITION 3 (METAGRAPH SYMMETRY). Consider a metagraph $M = (V_M, E_M)$. M is a symmetric metagraph if there exists a non-empty set Ψ containing pairs of distinct nodes in V_M, such that the edge set E_M remains unchanged even if, for each pair $(u, u') \in \Psi$, we exchange u and u' in all edges incident to u or u'. □

For example, consider the metagraph M_5 in Fig. 6. M_5 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 remains unchanged. Due to such symmetry, previous approaches [25], [18], [11], [24] could incur a large amount of redundant computation. To illustrate, consider M_5 and a matching order $u_1, u_2, \ldots, u_6$. After matching $u_1, u_2, \ldots, u_4$, 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.

C. Symmetry-based Approach

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 Fig. 6, 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 Fig. 6, 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 Fig. 6 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 of computation 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 an ordering over the components in M^+, and match each component in that order using the backtracking method. 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. Note that symmetric nodes can be identified by GRAMI [7] in the mining phase.

Simplifying metagraph. 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

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
$S \leftarrow k$-th component in the matching order o
$\mathcal{B} \leftarrow$ the set including S and its symmetric components, if any
compute the set $C(\mathcal{B}|D)$
$\mathcal{I}'(M) \leftarrow \emptyset$
for each $S' \in \mathcal{B}$ **do**
  $D' \leftarrow$ the merge of D and the matching of S' from $C(\mathcal{B}|D)$
  $\mathcal{I}^* \leftarrow$ MatchingByComponent(G, M, o, $k+1$, D')
  add $\mathcal{I}^*$ into $\mathcal{I}'(M)$
end
return $\mathcal{I}'(M)$.

$u \in S$ 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 Fig. 6, 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 [25], [20] select the next node to match from M such that the number of intermediate instances (of subgraphs of M) can be minimized. For instance, starting from a metagraph $M^{(1)}$ containing only one edge $\langle u_1, u_2 \rangle$ (suppose it is a subgraph of 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 utilize the above approach to order the components of M^+: when a node of a component S is chosen, we select S as the next component to match.

Matching simplified metagraphs. The matching algorithm for a simplified metagraph follows the backtracking framework, as outlined in Alg. 3. Compared with the traditional 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 $C(S|D)$. In particular, 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 $C(\mathcal{B}|D)$.

V. Experiments

We conduct extensive experiments to demonstrate three goals. First, our supervised approach using metagraphs can effectively model different classes of proximity. Second, our dual-stage training can greatly reduce overall metagraph matching time with negligible impact on accuracy. Third, our symmetry-based matching algorithm is efficient.

	#Nodes	#Edges	#Types	#Metagraphs	#Queries
LinkedIn	65 925	220 812	4	164	172 (college), 173 (coworker)
Facebook	5 025	100 356	10	954	340 (family), 904 (classmate)

Fig. 7: Description of datasets.

A. Experimental Setup

Graphs. We conducted extensive experiments on two real-world datasets collected by previous studies, namely LinkedIn [19] and Facebook [22]. Both datasets contain objects of various types. In particular, LinkedIn included the types of `user`, `employer`, `location` and `college`. Facebook included `user`, `major`, `degree`, `school`, `hometown`, `surname`, `location`, `employer`, `work-location`, `work-project` and others. (We ignored the other types on Facebook due to their sparsity or uselessness). We organized them into two heterogeneous graphs, as summarized in Fig. 7.

Metagraphs. As discussed in Sect. II, we applied GRAMI [7] on each graph to mine the set of metagraphs, and only retained symmetric ones. Each metagraph must contain at least two user nodes, since our ground truth (see next) is designed for the proximity between users. We stress that our framework itself is capable of modeling the proximity between any two nodes, not necessarily between users only. Finally, to reduce the number of metagraphs, we restricted them to have at most 5 nodes, which are found to be adequate in expressing various interactions between users. The resulting number of metagraphs on each graph is shown in Fig. 7.

Ground truth. On LinkedIn, some of the relationships between two users are already labeled into different classes of proximity. We tested two major classes: *college* friend and *coworker*. On Facebook, given no explicit labels, we generated our own ground truth based on some rules to mimic natural classes of proximity. More specifically, we consider the class of *family* such that two users must share the same `surname` as well as same `location` or `hometown`, and also the class of *classmate* such that two users must share the same `school` as well as same `degree` or `major`. Of course, these rules are not known to the proximity algorithms.

Training and testing. On each graph, a user q can be used as a query node if there exists at least another user v such that the relationship between q and v belongs to the desired class of proximity in our ground truth. The number of query nodes for each graph and class is shown in Fig. 7. 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, 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, such that other user nodes with the desired class label w.r.t. the query node are ranked higher than those with a different or unknown label. Subsequently, we could compare the ranking generated

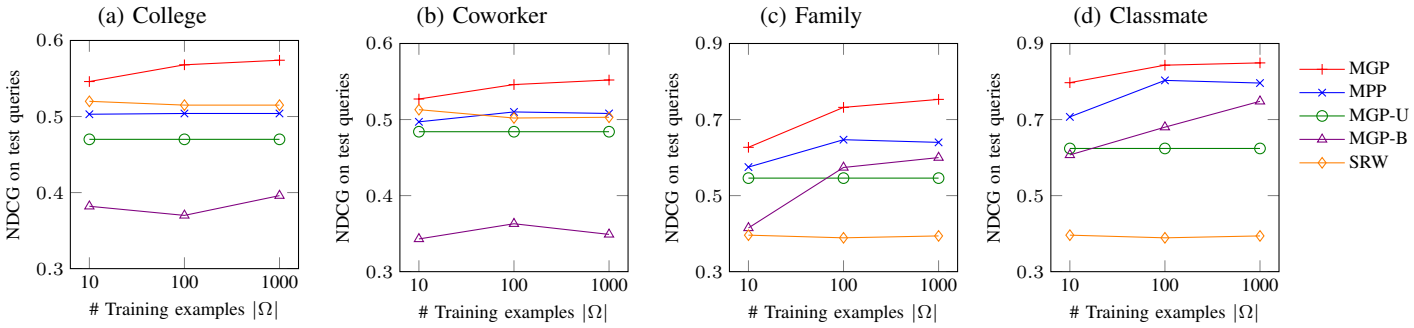


Fig. 8: Accuracy comparison of MGP and baselines (NDCG).

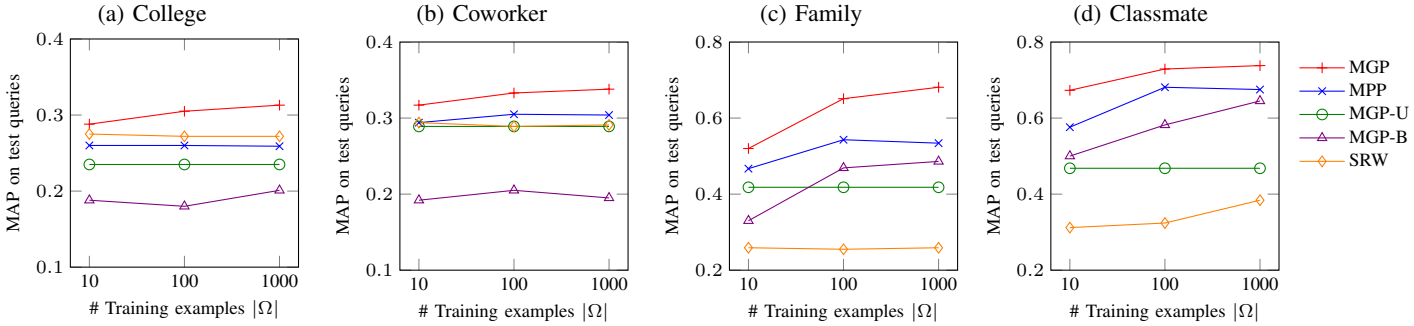


Fig. 9: Accuracy comparison of MGP and baselines (MAP).

by some proximity algorithm against the ideal ranking. In particular, we adopted NDCG and MAP [21] to evaluate the quality of the algorithmic rankings at top 10 nodes.

Environment. The matching algorithm (Sect. IV) and learning model (Sect. III) were implemented in C++ and Java, respectively. Time-sensitive experiments were run on a machine with 3.7GHz CPU and 64GB RAM, using only one thread.

B. Accuracy of Metagraph-based Proximity

We first study the accuracy of our approach without employing dual-stage training. Specifically, we compare the following algorithms.

- MGP: Metagraph-based proximity with our supervised approach (Sect. III-B). To compute the probability of a training example (Eq. 4), we set $\sigma = 5$ which was shown to be generally robust in our experiments. For gradient ascent, we used an initial learning rate of $\gamma = 10$, and gradually reduced it by 5% every 100 iterations. Convergence was deemed to happen once the log-likelihood changed less than 0.001% from the previous iteration. Lastly, as gradient ascent can stuck at local maximums, we repeated with 5 different random initializations and picked the best one.
- MPP: Metapath-based proximity with our supervised approach. We adapted the metapaths employed in an earlier work [26] to also use our supervised approach, by restricting the set of metagraphs to paths only. The earlier work only relied on manually selected metapaths to measure the proximity between two nodes.
- MGP-U: Metagraph-based proximity with uniform weights. That is, we did not differentiate the importance of metagraphs to any class of proximity.
- MGP-B: Metagraph-based proximity with the "single best" metagraph. Specifically, we found out the best performing metagraph on training data, and used this single best one to evaluate the test queries.
- SRW: Supervised random walks. We adopted a state-of-the-art random walk approach [4], which is a supervised variant of personalized PageRank [14]. The general principle is to assign different strengths to different edges, so that the transition matrix is biased to make certain nodes more likely to be visited in accordance with the training data. Specifically, for each edge, we used the types of its nodes to generate its features, and learnt feature weights in a supervised manner. As edge strength is a function of the features and their weights, different edges can end up with different strengths. In our experiments, we varied the parameters of SRW and chose their respective optimal values.

For each algorithm except MGP-U, we varied the number of training examples from 10 to 1000, as they all rely on the training data to learn their models. For MGP-U, it simply uses a uniform weighting independent of the training data.

We report the NDCG and MAP of the rankings produced by these algorithms in Fig. 8 and 9, respectively. On the one hand, MGP is consistently better than all other algorithms. In particular, using 1000 training examples and averaging across all four classes, MGP significantly outperforms the second best algorithm by 11% in NDCG and 16% in MAP. On the other hand, as the number of training examples grows, we can observe a steady increase in the performance of MGP as well. However, in some other supervised approaches, the accuracy either only improves when the number of training examples increases from 10 to 100 (*e.g.*, MPP), or does not improve at all (*e.g.*, SRW).

	Offline phase			Online phase
	Mining (GRAMI)	Matching (our algorithm)	Training w/ 1000 ex.	Testing per query
LinkedIn	247.6	9 870.3	11.6	8.2×10^{-5}
Facebook	213.2	10 021.6	142.8	2.8×10^{-4}

Fig. 10: Time costs without dual-stage training (seconds).

The results imply that our metagraph-based representation can effectively model the learning task, whereas other representations adopted by MPP and SRW are inadequate. In fact, it has been established [14] that the personalized PageRank score (which is the underlying model of SRW) of a node v w.r.t. query node q is equivalent to a linear combination of path probabilities over all paths starting from q and ending with v. That is, both MPP and SRW are linear combinations of path representations, whereas MGP's metagraph representation can capture nonlinear aggregations over paths.

C. Impact of Dual-Stage Training

To highlight the need of reducing the cost of metagraph matching, in Fig. 10 we report the time spent by different subproblems, without employing dual-stage training to reduce matching time for now. The results reveal that metagraph matching is the most expensive subproblem, surpassing the other subproblems by at least one order of magnitude and dominating the entire offline phase. Thus, it bolsters our design choice in Sect. II that we focus on improving the efficiency of metagraph matching over mining.

In particular, in this subsection, we study the impact of dual-stage training, which aims to reduce the *overall* matching time by only considering a small subset of candidate metagraphs. At the same time, accuracy can be adversely affected since some promising metagraphs may be mistakenly left out. Ideally, dual-stage training should achieve significant time reduction, and simultaneously, result in minimal loss of accuracy, as we will demonstrate in the following.

Treating the accuracy (NDCG and MAP) of using only seed metagraphs $\mathcal{K}_0$ as 0%, and that of using all metagraphs $\mathcal{M}$ as 100%, we compute the relative percentage increase in accuracy when we vary the number of candidates $|\mathcal{K}|$. (As a remark, in practice, to set the optimal $|\mathcal{K}|$, we can start from a small $|\mathcal{K}|$ first and gradually increase it until the accuracy is satisfactory, as discussed in Sect. III-C.) Likewise, treating the time cost of matching only seed metagraphs as 0% and that of matching all metagraphs as 100%, we also compute the relative percentage increase in the time cost.

We present the outcomes in Fig. 11. As we increase the number of candidates, naturally we can expect an increase in both accuracy and time. However, the rate of increase in accuracy is much faster than time. In particular, 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 the absolute accuracy metrics of MGP (which uses all metagraphs) shown in Fig. 8 and 9, we sacrifice both NDCG and MAP by only 1% on average. In contrast, comparing with the absolute time of matching

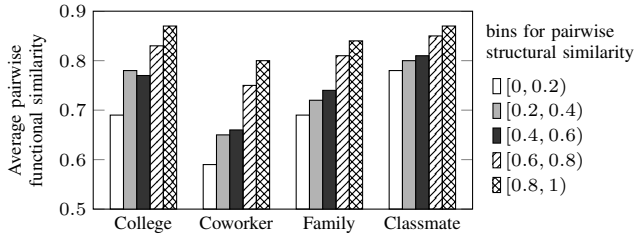


Fig. 12: Correlation of structural and functional similarities.

all metagraphs shown in Fig. 10, we can on average reduce the overall matching time by a massive 83%. Note that the time reduction on Facebook is more significant than that on LinkedIn, since the former has much more metagraphs due to more types of objects (Fig. 7). In summary, our dual-stage training can significantly reduce overall matching time with only a minuscule impact on accuracy.

Next, we validate in two ways that the proposed candidate heuristic can identify promising candidates.

First, as the basis of our candidate heuristic (Sect. III), structural similarity implies functional similarity. Using supervised learning on all metagraphs without dual-stage training, we can learn the optimal weight for each metagraph. Subsequently, we can compute structural and functional similarities for each pair of metagraphs, and plot them in Fig. 12. Our datasets indeed reveal a general correlation between the two kinds of similarities, supporting the very foundation of our proposed candidate heuristic.

Second, the candidate heuristic score H (Eq. 7) can induce a meaningful order on the promise (or usefulness) of the metagraphs. We compare our candidate heuristic (CH) with reverse candidate heuristic (RCH). RCH simply reverses the order induced by CH. If the order by CH is meaningful, we expect RCH to give worse accuracy than CH, since least promising metagraphs are chosen as candidates. This expectation can be verified by Fig. 13, which clearly shows that CH can identify better candidates than RCH.

D. Efficiency of Metagraph Matching

While dual-stage training can already reduce the overall matching time significantly, we further examine the efficiency of our matching algorithm for individual metagraphs. In particular, we compare our symmetry-based algorithm, denoted by SymISO, with three state-of-the-art baselines: BoostISO [24], TurboISO [11] and QuickSI [25]. All these baselines build upon the backtracking framework (Sect. IV-A), but do not utilize graph symmetry to avoid redundant computation. We obtained the codes of BoostISO and TurboISO from their respective authors, and used the implementation of QuickSI in the benchmark paper [18]. Lastly, to illustrate the importance of node matching orders in SymISO, we also compare to a weaker scheme of SymISO using a random matching order, dubbed as SymISO-R.

We illustrate the average running time per metagraph for all the algorithms in Fig. 14, where the size of metagraphs (number of nodes) varies between 3 and 5. Observe that SymISO consistently outperforms the best baseline (BoostISO), by 52% on average. When the number of nodes in

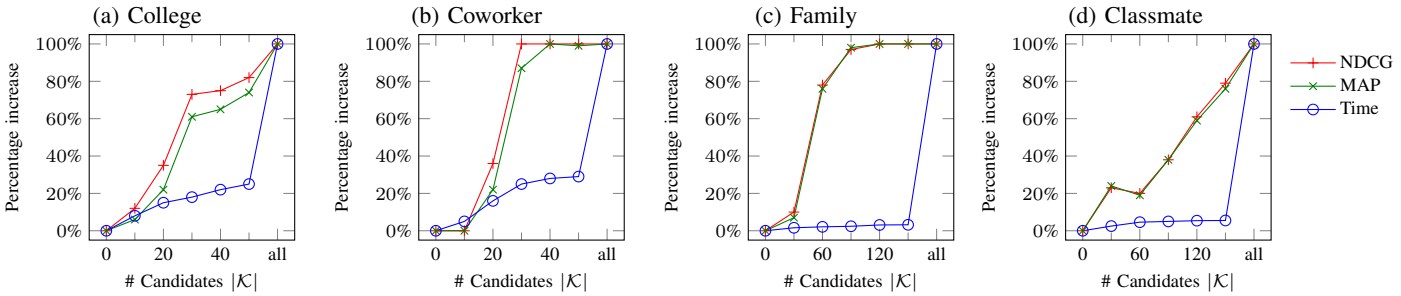


Fig. 11: Impact of dual-stage training (special values of $|\mathcal{K}|$: 0 if only use seed metagraphs; "all" if use all metagraphs).

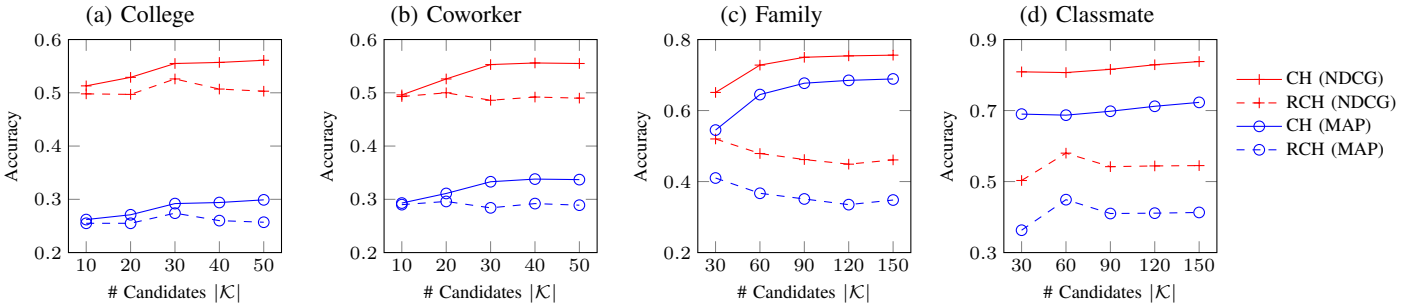


Fig. 13: Accuracy comparison of candidate heuristic (CH) and reverse candidate heuristic (RCH) in dual-stage training.

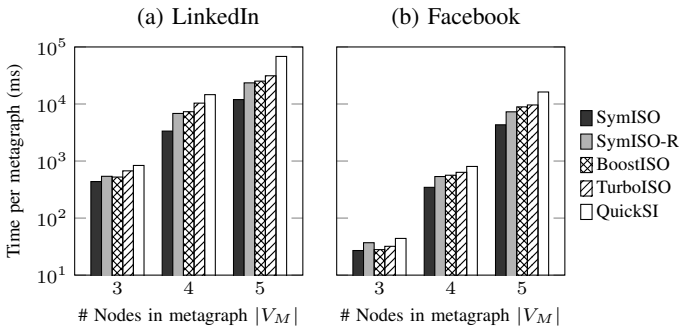


Fig. 14: Comparison of average matching time per metagraph.

a metagraph increases, the performance margin of SymISO becomes larger, since more redundant computation can be avoided due to larger symmetric components. Furthermore, SymISO is about 45% faster than SymISO-R, which indicates the usefulness of our matching order.

VI. Related Work

Meta-structures. While the less general concept of metapath has been proposed [26], metagraphs are more expressive and effective than metapaths in capturing interactions between nodes, as explained in Sect. I. Given the increased complexity and variety of metagraphs, we cannot handle metagraphs in the same way as metapaths. First, the metapath-based PathSim [26] 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 [27] 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 dual-stage training and symmetry-based matching to improve efficiency.

Proximity search. Most earlier research [13], [14], [1], [16], [26] only measures a "generic" form of proximity on graphs. Different roles or senses of proximity have also emerged, such as hub and authority [15], probabilistic precision and recall [3], [8], as well as importance and specificity [12], [9]. However, these roles or senses are only formed due to specific patterns in the link structures (*i.e.*, non-semantic), whereas our classes of proximity aim to capture different kinds of semantic proximity between nodes. Although there exist semantic-oriented studies on graphs, such as social circle learning [22] and relationship profiling [19], they do not support online query processing and thus cannot be easily adapted for proximity search.

There also exist several random walk approaches [6], [2], [4], which learn from example ranking preferences [21] to bias transition probabilities between nodes of different types or features. However, these studies do not recognize various semantic classes of proximity, and only intend to learn a generic measure best suited to the given graph. Although they can be adapted to learn for the desired class of proximity, their way of adjusting the transition probabilities is indirect towards differentiating the classes. Furthermore, random walks fundamentally reduce to a linear aggregation of individual path probabilities [14], and paths lack the expressiveness of metagraphs. Not surprisingly, the state-of-the-art method [4] turns out to be ineffective in our experiments.

Feature cost in training. We believe that reducing feature extraction cost for *training* has not been studied. To learn a desired class of proximity, our dual-stage training approach only tries to match a subset of metagraphs in training. (A

metagraph is analogous to a feature in traditional settings). In contrast, existing studies [5], [23], [31], [30] focus on minimizing feature extraction cost for *testing*. That is, there is a budget constraint on the total cost of feature extraction for testing examples. If a testing example is easier, then fewer features are needed for classification; otherwise, more features are needed. To realize such cost savings in testing, a training process to examine all features are required, which is exactly what we aim to avoid. In our setting, reducing feature cost for training is made possible because each metagraph has structural information, which enables us to infer its function even without knowing its instances.

Subgraph matching. A plethora of techniques [28], [25], [18], [11], [24] have been proposed for subgraph matching, which follow the backtracking framework as discussed in Section IV. Their major issue is the extremely huge search space on a large graph. To prune the search space, Shang et al. [25] have proposed a special ordering of nodes for matching instead of just a random ordering. Subsequently, Han et al. [11] and Ren et al. [24] have introduced more improvements to further reduce and reuse redundant computation. However, they do not account for graph symmetry, and are thus inefficient for symmetric metagraphs. Moreover, significant research has been devoted to mining frequent subgraphs among a large set of input graphs, based on the property of downward closure [32], [17], [7]. However, our problem is to compute the instances of a metagraph on a single input graph, which does not obey downward closure.

VII. Conclusion

In this paper, we proposed and addressed the problem of semantic proximity search on graphs. To differentiate various semantic classes of proximity, we identified and employed *metagraphs* to characterize any arbitrary semantic class, and automatically learnt the characteristic metagraphs in a supervised manner. While metagraph-based proximity is effective in modeling different classes of proximity, we also improved its efficiency in two ways. First, we developed a *dual-stage training* approach to only consider a small subset of promising metagraphs. Second, we also devised a *symmetry-based matching* algorithm to speed up the matching of individual metagraphs. Empirical results on two real-world graphs demonstrated that our proposed approach is not only accurate but also efficient.

References

[1] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.

[2] A. Agarwal, S. Chakrabarti, and S. Aggarwal. Learning to rank networked entities. In *SIGKDD*, pages 14–23, 2006.

[3] G. Agarwal, G. Kabra, and K. C. Chang. Towards rich query interpretation: walking back and forth for mining query templates. In *WWW*, pages 1–10, 2010.

[4] L. Backstrom and J. Leskovec. Supervised random walks: Predicting and recommending links in social networks. In *WSDM*, pages 635–644, 2011.

[5] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. In *WSDM*, pages 411–420, 2010.

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

[7] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. GRAMI: frequent subgraph and pattern mining in a single large graph. *PVLDB*, 7(7):517–528, 2014.

[8] Y. Fang and K. C. Chang. Searching patterns for relation extraction over the Web: rediscovering the pattern-relation duality. In *WSDM*, pages 825–834, 2011.

[9] Y. Fang, K. C. Chang, and H. W. Lauw. RoundTripRank: Graph-based proximity with importance and specificity. In *ICDE*, pages 613–624, 2013.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[11] W. Han, J. Lee, and J. Lee. $\text{Turbo}_{\text{iso}}$: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *SIGMOD*, pages 337–348, 2013.

[12] V. Hristidis, H. Hwang, and Y. Papakonstantinou. Authority-based keyword search in databases. *TODS*, 33(1), 2008.

[13] G. Jeh and J. Widom. SimRank: a measure of structural-context similarity. In *KDD*, pages 538–543, 2002.

[14] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.

[15] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[16] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *SIGKDD*, pages 245–255, 2006.

[17] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. In *ICDM*, pages 345–356, 2004.

[18] J. Lee, W. Han, R. Kasperovics, and J. Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *PVLDB*, 6(2):133–144, 2012.

[19] R. Li, C. Wang, and K. C. Chang. User profiling in an ego network: co-profiling attributes and relationships. In *WWW*, pages 819–830, 2014.

[20] W. Lin, X. Xiao, J. Cheng, and S. S. Bhowmick. Efficient algorithms for generalized subgraph query processing. In *CIKM*, pages 325–334, 2012.

[21] T.-Y. Liu. *Learning to rank for information retrieval*. Springer, 2011.

[22] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *NIPS*, pages 548–556, 2012.

[23] V. C. Raykar, B. Krishnapuram, and S. Yu. Designing efficient cascaded classifiers: Tradeoff between accuracy and cost. In *KDD*, pages 853–860, 2010.

[24] X. Ren and J. Wang. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. *PVLDB*, 8(5):617–628, 2015.

[25] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *PVLDB*, 1(1):364–375, 2008.

[26] 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*, 4(11), 2011.

[27] 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 *SIGKDD*, pages 1348–1356, 2012.

[28] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[29] 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*, 9(4):407–432, 2013.

[30] D. J. Weiss and B. Taskar. Learning adaptive value of information for structured prediction. In *NIPS*, pages 953–961. 2013.

[31] Z. E. Xu, K. Q. Weinberger, and O. Chapelle. The greedy miser: Learning under test-time budgets. In *ICML*, 2012.

[32] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.