Neighborhood Based Fast Graph Search in Large Networks

Arijit Khan
Dept. of Computer Science
University of California
Santa Barbara, CA 93106
arijitkhan@cs.ucsb.edu

Nan Li
Dept. of Computer Science
University of California
Santa Barbara, CA 93106
nanli@cs.ucsb.edu

Xifeng Yan
Dept. of Computer Science
University of California
Santa Barbara, CA 93106
xyan@cs.ucsb.edu

Ziyu Guan
Dept. of Computer Science
University of California
Santa Barbara, CA 93106
ziyuguan@cs.ucsb.edu

Supriyo Chakraborty
Dept. of Electrical Engineering
University of California
Los Angeles, CA 90095
supriyo@ee.ucla.edu

Shu Tao
IBM T. J. Watson
19 Skyline Drive
Hawthorne, NY 10532
shutao@us.ibm.com

ABSTRACT

Complex social and information network search becomes important with a variety of applications. In the core of these applications, lies a common and critical problem: Given a labeled network and a query graph, how to efficiently search the query graph in the target network. The presence of noise and the incomplete knowledge about the structure and content of the target network make it unrealistic to find an exact match. Rather, it is more appealing to find the top-k approximate matches.

In this paper, we propose a neighborhood-based similarity measure that could avoid costly graph isomorphism and edit distance computation. Under this new measure, we prove that subgraph similarity search is NP hard, while graph similarity match is polynomial. By studying the principles behind this measure, we found an information propagation model that is able to convert a large network into a set of multidimensional vectors, where sophisticated indexing and similarity search algorithms are available. The proposed method, called **Ness** (**Ne**ighborhood Based **S**imilarity **S**earch), is appropriate for graphs with low automorphism and high noise, which are common in many social and information networks. Ness is not only efficient, but also robust against structural noise and information loss. Empirical results show that it can quickly and accurately find high-quality matches in large networks, with negligible cost.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process; I.2.8 [**Problem Solving, Control Methods, and Search**]: Graph and tree search strategies

General Terms

Algorithms, Performance

SIGMOD'11, June 12–16, 2011, Athens, Greece.
Copyright 2011 ACM 978-1-4503-0661-4/11/06 ...$10.00.

Keywords

Graph Query, Graph Search, Graph Alignment, RDF

1. INTRODUCTION

Recent advances in social and information science have shown that linked data pervade our society and the natural world around us [36]. Graphs become increasingly important to represent complicated structures and schema-less data such as wikipedia, freebase [15] and various social networks. Given an attributed network and a small query graph, how to efficiently search the query graph in the target network is a critical task for many graph applications. It has been extensively studied in chemi-informatics, bioinformatics, XML and Semantic Web. SPARQL [27] is the state-of-the RDF query language for Semantic Web. SPARQL requires accurate knowledge about the graph structure to write a query and also it performs an exact graph pattern matching. However, due to the noise and the incomplete information (structure and content) in many networks, it is not realistic to find exact matches for a given query. It is more appealing to find the top-k approximate matches.

Unfortunately, graph similarity measures such as subgraph isomorphism, maximum common subgraphs, graph edit distance, missing edges that are appropriate for chemical structures and biological networks, are not suitable for entity-relationship graphs and social networks. There are two challenging issues for these graph theoretic measures. First, entity-relationship graphs and social networks have quite different characteristics from physical networks. They are not governed by physical laws and often full of noise, thus making strict topological similarity examination nearly impossible. How the entities are connected in these networks are not as important as how closely these entities are connected. Second, these graphs are very large and complex with a lot of attributes associated. If accuracy is to be ensured, the algorithms developed for edit distance and missing edges are not scalable. These two issues motivate us to invent new graph similarity measures that are less sensitive to structure changes, and have scalable indexing and search solutions.

Figure 1(a) shows a graph query to "*Find the athlete who is from 'Romania' and won 'gold' in '3000m' and 'bronze' in '1500m' both in '1984' olympics.*[1]." Compare this query against a possible match in FreeBase (Olympics) shown in Figure 1(b), it is observed that these two graphs are by no means similar under traditional graph similarity definitions. Graph edit distance between

[1] http://en.wikipedia.org/wiki/Maricica_Puică

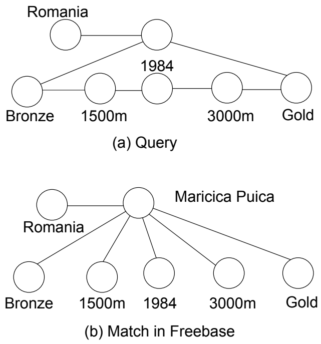


Figure 1: Top-1 Match for Query (a) in FreeBase

these two graphs is 7. The size of their maximum common graph is 3. The number of maximum missing edges for the query graph is 4. However, "Maricica Purca" in Figure 1(b) is a good match for the query shown in Figure 1(a), because she has all these attributes quite close to her in Figure 1(b). In practice, it is hard to come up with a query that exactly conforms with the graph structures in the target network due to the lack of schemas in linked data. However, it is easy to write a query like Figure 1(a), where a user connects entities with possible links. As long as the the proximity between these entities is approximately maintained in a query graph, the system shall be able to deliver matches like Figure 1(b).

The above approximate query form can serve as a primitive for many advanced graph operators such as RDF query answering, network alignment, subgraph similarity search, name disambiguation and database schema matching. For example, based on partial information related to one person, e.g. his friends, one can align his physical social circle with his cyber social network on Facebook. In many cases, nodes in social or information networks have incomplete information or even anonymized information. Nevertheless, the partial neighborhood information available from a query graph will be helpful to identify entities in the target network.

Clearly, there is a need to adopt approximate similarity search techniques to solve the above problem. In bioinformatics, approximate graph alignment has been extensively studied, e.g. PathBlast [21], Saga [33]. These studies resort to strict approximation definition such as graph edit distance, whose optimal solution is expensive to compute. Since they are targeting a relatively small biological networks with less than 10k nodes, it is difficult to apply them in social and information networks with thousands or even millions of nodes. As illustrated in NetAlign [23], in order to handle large graphs with 10k nodes, one has to sacrifice accuracy to achieve better query response time. Recently there have been other studies on approximate matching with large graphs, i.e., TALE [34], SIGMA [24] and G-Ray [35]. However, both TALE and SIGMA consider the number of missing edges as the qualitative measure of approximate matching and hence, the techniques cannot capture the notion of proximity among labels, as shown in Figure 1. G-Ray, on the other hand, tries to maintain the shape of the query by allowing some approximation in the match. Unfortunately, shape is not an important factor in entity-relationship graphs.

In this paper, we introduce a novel neighborhood-based similarity measure by vectorizing nodes according to the label distribution of their neighbors. We further extend the similarity notion to graph by finding the embeddings in the target graph that maximize the sum of node matches. This graph matching technique avoids complicated subgraph isomorphism and graph edit distance calculation, which becomes infeasible for large graphs. It is observed that social/information networks usually have more diversified node labels and therefore less auto-isomorphic structure, but may contain more noise. Our objective function can provide better similarity semantics for graphs with various random noise. It simplifies the procedure of graph matching, leading to the development of an efficient graph search framework, called **Ness** (**Ne**ighborhood Based **S**imilarity **S**earch). With the introduction of scalable indices built on vectorized nodes and an intelligent query optimization technique, Ness can quickly and accurately find high-quality matches in large networks, with negligible time cost.

Our contributions. We propose a novel similarity search problem in graphs, **neighborhood-based similarity search**, which combines the topological structure and content information together during the search process. The similarity definition proposed in this work is able to avoid expensive isomorphism testing as much as possible. The principles to derive appropriate functions to fit this definition are carefully examined. We found that the information propagation model satisfies these principles, where each node propagates a certain fraction of its labels to its neighbors, and thereby we could convert each node into a multidimensional vector, where sophisticated indexing and similarity search algorithms are available. That is, we successfully turn a graph search problem into a high-dimension index problem.

We first identify a set of rules to define approximate matches of nodes based on their neighborhood structure and labels. These rules are important since the query may not always have complete information about the exact neighborhood structure in the target graph. The approximate node match concept is further extended to subgraph similarity search, i.e. multiple node alignment for a given query graph. We prove that under this measure, subgraph similarity search is NP hard. However, in comparison with graph isomorphism, which is neither known to be solvable in polynomial time nor NP-hard, graph similarity match is proved to be polynomial. We demonstrate that, without performing subgraph isomorphism testing, it is possible to prune unpromising nodes by iteratively propagating node information among a shrinking candidate set, which significantly reduces query execution time.

We further analyze how to index the vector structure as well as optimize query processing to speed up similarity search. The information propagation model and the neighborhood vectorization approach keep the index structure much simpler than the graph itself, thus making it easy to be updated dynamically for graph changes arising from node/edge insertion and deletion.

In summary, we propose a completely new graph similarity search framework, Ness, to define and determine approximate matches in massive graphs. As tested in real and synthetic networks, Ness is able to find high-quality matches efficiently in large scale networks.

2. PRELIMINARIES

A labeled graph $G = (V_G, E_G, L_G)$ has a label set L_G and each node $u \in V_G$ is attached with a set of labels. The label set of a node u in G is denoted by $L(u) \subseteq L_G$. For the sake of simplicity, we assume there are no labels and weights on the edges. Nevertheless, the proposed techniques could be extended for graphs with labeled or weighted edges. Given two labeled graphs G and G', G is called subgraph isomorphic to G', if there exists a subgraph H of G', such that G is isomorphic to H. Formally, we define subgraph isomorphism as follow.

DEFINITION 1 (SUBGRAPH ISOMORPHISM). *A subgraph isomorphism is an injective function* $f : V_G \rightarrow V_{G'}$, *s.t.,* $(1)\ \forall u \in$

V_G, $L(u) \subseteq L(f(u))$, and (2) $\forall (u,v) \in E_G$, $(f(u), f(v)) \in E_{G'}$.

DEFINITION 2 (EMBEDDING). *Given a graph G and a query graph Q, an embedding of Q is an (injective) function $f : V_Q \rightarrow V_G$, such that, $\forall v \in V_Q, L(v) \subseteq L(f(v))$, where $f(v) \in V(G)$.*

In this work, we only studied the one-to-one node matching for a query graph Q and the node labels are preserved in the embedding. However, our cost function and algorithms can be extended to include other matching and node label similarity scenarios. Given two graphs G and Q, there might be many possible embeddings. Certainly, the quality of an embedding depends on whether it preserves the connections and labels in the query graph or not. Subgraph isomorphism actually defines an *exact* embedding, written as f_e. The "quality" of an embedding can be defined in various ways; i.e., for a given label-preserved embedding f, we can count the number of edge mismatches, $\mathcal{C}_e = |\{(u,v) \in E_Q : (f(u), f(v)) \notin E_G\}|$, as the embedding's quality. In general, for a cost function $\mathcal{C} : f \rightarrow \mathbb{R}$, we define the top-k graph similarity search problem as below.

PROBLEM STATEMENT 1. *Given a graph G and a query graph Q, find the top-k embeddings with respect to a cost function $\mathcal{C}$.*

The edge mismatch cost function $\mathcal{C}_e$ has been studied in [38, 34, 24]. Unfortunately, it cannot differentiate the case where two nodes are close to each other but there is no direct edge between them.

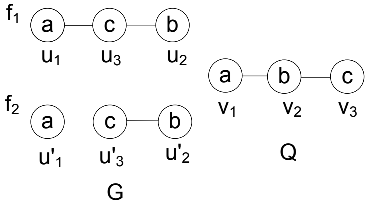


Figure 2: Problem with Edge Mismatch Cost Function

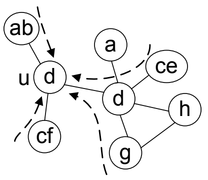


Figure 3: Information Propagation Model

Figure 2 shows one example. There are two label-preserved embeddings f_1 and f_2 of the query graph Q in a target graph G. In f_1 and f_2, there is no edge connecting a and b. Thus, $\mathcal{C}_e$ will assign equal cost to both embeddings. On the other hand, the graph edit distance between f_1 and Q is 2, whereas it is only 1 between f_2 and Q. Although, intuitively it is observed that f_1 is a better match than f_2, because the nodes with labels a and b are only 2-hops away in f_1, whereas they are disconnected in f_2. This observation inspires us to develop a neighborhood-based similarity measure that *discounts how nodes are exactly connected, but focuses on the proximity among the labels carried by these nodes.* It needs to achieve the following two objectives: (1) The cost function should identify approximate embeddings, and (2) it must be easy to compute. In the next section, we will define the neighborhood-based similarity cost function and the complexity analysis of that function.

3. NEIGHBORHOOD-BASED GRAPH SIMILARITY

In order to solve the problem raised by the edge mismatch cost function, we define a novel neighborhood-based similarity measure by comparing the h-hop neighbors of a node, defined as follows.

DEFINITION 3 (h-HOP NEIGHBORS). *Given a graph G and a node $u \in V(G)$, the h-hop neighborhood of u is the set of nodes v whose distance from u is less than or equal to h.*

To compare the neighborhoods of two nodes, we resort to an information propagation model [22] that is able to transform neighborhoods into vectors in a multidimensional space, where sophisticated indexing and fast similarity search algorithms are available.

3.1 Information Propagation Model

Figure 3 shows the information propagation model to characterize the neighborhood information around node u. The label information encoded in u's neighbors is propagated to u through different paths and accumulated at u. One could use the accumulated information and its strength as a vector to describe the neighborhood of u. The neighborhood vector of u is denoted by $R(u)$, which consists of a set of tuples, $R(u) = \{\langle l, A(u,l)\rangle\}$, where l is a label present in the neighborhood of u and $A(u,l)$ represents the strength of label l at node u in a graph.

There are many different mechanisms to propagate information. However, not every one is valid for graph similarity search. Any valid one must comply with the following principle,

PROPERTY 1 (COST FUNCTION). *For a graph similarity cost function $\mathcal{C}$, given an exact embedding f_e, $\mathcal{C}(f_e)$ must be equal to 0.*

Here, we consider a simple but effective information propagation model so that the derived neighborhood-based similarity measure satisfies the above principle. It propagates information along the shortest paths between two nodes with exponential decay to the length. Eq. 1 describes the formula of $A(u,l)$ in $R(u) = \{\langle l, A(u,l)\rangle\}$ that represents the h-hop neighborhood of node u in a graph.

$$A(u,l) = \sum_{i=1}^{h} \alpha^i \sum_{d(u,v)=i} I(l \in L(v)), \tag{1}$$

where $I(l \in L(v))$ is an indicator function which takes value one when l is in the label set of v and zero otherwise. $d(u,v)$ is the distance between u and v. α is a constant called the *propagation factor*. It is between 0 and 1, whose optimum value will be discussed later. Eq. 2 confines Eq. 1 to an embedding f in G by only considering the vertices and the shortest paths in f.

$$A_f(u,l) = \sum_{i=1}^{h} \alpha^i \sum_{v \in V_f, d(u,v)=i} I(l \in L(v)). \tag{2}$$

Using this information propagation model, we shall formulate the neighborhood-based cost function.

3.2 Neighborhood-based Cost Function

Given a query graph Q and its embedding f in the target graph G, we can apply the information propagation model to propagate labels in Q and f. Since vertices in f might not be directly connected, we will consider all of the shortest paths connecting these vertices during propagation. To derive the neighborhood-based cost function $\mathcal{C}_N(f)$, we first compute the difference between the neighborhood vectors $R_f(u)$ and $R_Q(v)$, representing the neighborhoods u and v in the embedding and the query graph, respectively.

$$\mathcal{C}_N(v,u) = \sum_{l \in R_Q(v)} M(A_Q(v,l), A_f(u,l)), \tag{3}$$

where $M(x,y)$ is a positive difference function as given below.

$$M(x,y) = \begin{cases} x - y, & \text{if } x > y; \\ 0, & \text{otherwise.} \end{cases}$$

The reason to adapt a positive difference function is that if the embedding f in G carries more labels than Q, we shall not penalize it. Only when there are labels and edges missed in f, $\mathcal{C}_N(v,u)$ will return a positive value. Note that, the summation in Equation 3 is considered over all labels l present in $R_Q(v)$, i.e. $\{l : A_Q(v,l) > 0\}$. For brevity, we simply denote this by $l \in R_Q(v)$ in Equation 3, and the same notation will be used in the remaining of the paper.

Given an embedding f, we aggregate the differences for all pairs (v,u), where $u = f(v)$. The neighborhood based graph similarity cost $\mathcal{C}_N(f)$ is given as follows.

$$\mathcal{C}_N(f) = \sum_{v \in V_Q} \mathcal{C}_N(v, f(v)) \qquad (4)$$

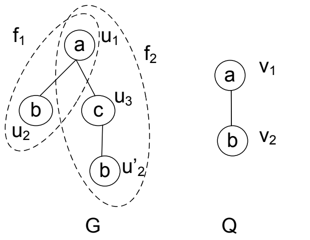


Figure 4: Neighborhood Based Similarity Cost

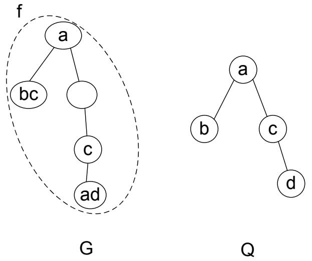


Figure 5: Example of False Positive

Figure 4 provides an example of neighborhood based graph matching cost. In graph G, label b is propagated to node u_1 from node u_2 and u_2', via the corresponding shortest paths respectively. Assume $\alpha = 0.5$ and $h = 2$, we have $A_G(u_1, b) = 0.5 + 0.25 = 0.75$. We can derive the neighborhood vectors for other nodes in G: $R_G(u_1) = \{\langle b, 0.75\rangle, \langle c, 0.5\rangle\}$, $R_G(u_2) = \{\langle a, 0.5\rangle, \langle c, 0.25\rangle\}$, $R_G(u_3) = \{\langle a, 0.5\rangle, \langle b, 0.75\rangle\}$ and $R_G(u_2') = \{\langle c, 0.5\rangle, \langle a, 0.25\rangle\}$. Similarly, $R_Q(v_1) = \{\langle b, 0.5\rangle\}$ and $R_Q(v_2) = \{\langle a, 0.5\rangle\}$.

In Figure 4, we have two possible embeddings f_1 and f_2. $R_{f_1}(u_1)$ = $\{\langle b, 0.5\rangle\}$ and $R_{f_1}(u_2) = \{\langle a, 0.5\rangle\}$. Hence, $\mathcal{C}_N(f_1) = (0.5 - 0.5) + (0.5 - 0.5) = 0$. For f_2, we match v_1 to u_1 and v_2 to u_2'. We have $R_{f_2}(u_1) = \{\langle b, 0.25\rangle\}$ and $R_{f_2}(u_2') = \{\langle a, 0.25\rangle\}$. Therefore, $\mathcal{C}_N(f_2) = (0.5 - 0.25) + (0.5 - 0.25) = 0.5$. Note that, for the embedding f_2, node u_3 will not contribute any labels to R_{f_2} since it does not participate in the matching. However, it is on the shortest path from u_2' to u_1, thus propagating labels between u_2' and u_1.

We must mention that the vectorization of the neighborhoods and the comparison among these vectors can be done in various ways. However, the final cost function must satisfy the basic property of $\mathcal{C}$ (Property 1) to avoid false negatives for exact embeddings. The following theorem shows that $\mathcal{C}_N$ follows this property.

THEOREM 1. *For an exact embedding f_e, $\mathcal{C}_N(f_e) = 0$.*

PROOF. For an exact embedding f_e, if $(v_1, v_2) \in E_Q$, then $(f_e(v_1), f_e(v_2)) \in E_G$. Thus, the shortest distance between the node pairs $f_e(v_1), f(v_2)$ in f_e cannot be higher than the shortest distance between the node pairs v_1, v_2 in Q. Hence, it follows from Eq. 1 that $\forall l, v, A_f(f_e(v), l) \geq A_Q(v,l)$. Therefore, based on Eq. 3 and Eq. 4, $\mathcal{C}_N(f_e) = 0$. □

Theorem 1 ensures that there is no false negatives for exact embeddings. However, there might be some false positives as shown in Figure 5. In this example, if $h = 1$, $\mathcal{C}_N(f) = 0$, although f is not an exact embeddings of Q. Fortunately, if we increase h to 2, $\mathcal{C}_N(f) > 0$. In real-life graphs that have low automorphism and more distinct labels in nodes, false positives can mostly be avoided, as shown in our experiments and in the following Lemma.

LEMMA 1. *Given a graph G and a query graph Q, if each of their nodes has a distinct label, for any inexact embedding f, $\forall h > 0, \alpha > 0$, $\mathcal{C}_N(f) > 0$.*

PROOF. Omitted. □

Our definition of neighborhood-based cost function is robust against structural differences and other forms of noises. As long as two close labels in a query graph are close enough in the target graph, we consider it as a potential match. We can also rank the embeddings based on the proximity of their labels in the target graph compared to that in the query graph. Thus, even if there exists no exact embedding of the query graph, the cost function can identify the closely approximate matches and rank them based on their structural differences.

We formally define our problem statement as follows.

PROBLEM STATEMENT 2. ***[Neighborhood-Based Top-k Similarity Search]*** *Given a target graph G and a query graph Q, find the top-k embeddings with respect to the cost function $\mathcal{C}_N$.*

In the following discussion, we show that the above problem is NP-hard by reducing the clique problem to it.

LEMMA 2. *Given a graph G and a query graph Q, $\forall u \in V_G, v \in V_Q$, $|L(u)| = 1, |L(v)| = 1$, if Q is a complete graph, then for all inexact embeddings f, $\mathcal{C}_N(f) > 0$.*

PROOF. Since $\forall u \in V_G, v \in V_Q$, $|L(u)| = 1$, $|L(v)| = 1$, for any inexact embedding f, each node $u = f(v)$ has only one label, which is same as the label of node v in Q. Since, Q is a complete graph, there exists at least one node $f(v)$ in f and a label l such that the number of 1-hop neighbors of v in Q that has label l is more than the number of 1-hop neighbors of $f(v)$ in f with label l. Hence, $A_Q(v,l) > A_f(f(v), l)$. Therefore, it follows from the definition of $\mathcal{C}_N$ that, $\mathcal{C}_N(f) > 0$. □

THEOREM 2. *Neighborhood-Based Top-k Similarity Search is NP-hard.*

PROOF. Let us consider the case where $|L(u)| = 1, |L(v)| = 1, \forall u \in V_G, v \in V_Q$, and Q is a complete graph. Suppose the top-1 match f can be identified in polynomial time. Given f, it can also be verified in polynomial time, whether $\mathcal{C}_N(f) = 0$. Now, if $\mathcal{C}_N(f) = 0$, by Lemma 2, there exists a clique of size of Q in the target graph G. So, it is possible to solve the clique problem in polynomial time. However, we know that, the clique decision problem is NP-hard [10], therefore we have a contradiction. Hence, the similarity search problem is NP-hard. □

The graph isomorphism problem is neither known to be solvable in polynomial time nor NP-complete. However, given two graphs Q and G of same size, it is possible to determine in polynomial time, if G itself is an embedding of Q with cost $\mathcal{C}_N(f) = 0$. We call this problem as the *Graph Similarity Match* problem. Thus, we suspect that neighborhood-based similarity search might have lower time complexity than graph theoretic measures such as graph isomorphism and edit distance.

THEOREM 3. *Graph Similarity Match is polynomial in n, where $n = |V_Q|$.*

PROOF. Since G itself is an embedding f of Q, we can determine the individual node matching costs $\mathcal{C}_N(v, u)$ in polynomial time, for all $v \in V_Q$, $u \in V_G$. Next, we construct a flow network and determine the minimum cost of maximum flow in that network (see Figure 6). From the source node s, add a directed edge to each node v in Q. The capacity of each of these edges is 1 and the cost is 0. Similarly, from each node u in G, add a directed edge to the sink node t. The capacity and cost of each of these edges are 1 and 0 respectively. From each node v in Q, add a directed edge to each node u in G, if $L(v) \subseteq L(u)$. The capacity and cost of this edge are 1 and $\mathcal{C}_N(v, u)$ respectively. Due to the capacity constraints, each node in Q can be matched with at most one node in G, and also only one node of Q can be matched with same node in G. Clearly, if the maximum flow in this network is n and the minimum cost of the maximum flow is 0, then G is an embedding of Q with cost $\mathcal{C}_N(f) = 0$. However, this flow problem can be solved using the Ford and Fulkerson algorithm [11] in $O(n^3)$ time. Therefore, given two graphs Q and G of the same size, it is possible to determine in polynomial time, if G itself is an embedding f of Q with cost $\mathcal{C}_N(f) = 0$. □

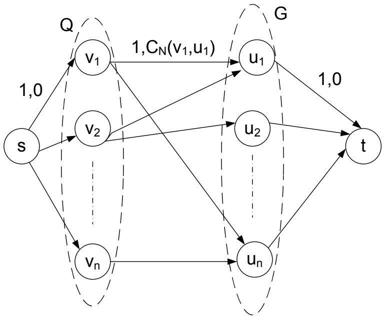


Figure 6: Flow Network to Solve Graph Similarity Match

3.3 Propagation Factor: α

In the information propagation model described in Eq. 1, the propagation factor, α, should be less than 1 in order to reflect the relation that the strength $A(u, l)$ of label l at node u decreases with the increase of distance. However, we find the top-k embeddings by repeatedly matching the individual nodes from G and Q that satisfies a cost threshold ϵ (The detailed procedure will be discussed in the next section). Now, if α is large, each node will propagate a high fraction of labels to its neighbors and this can increase the number of false positives at the initial node matching stage, thus slowing down the overall search process. In Figure 7, for $\alpha = 0.5$ and $h = 2$, we get $R_G(u) = \{\langle a, 0.25 + 0.25\rangle\}$ = $\{\langle a, 0.5\rangle\}$ and $R_Q(v) = \{\langle a, 0.5\rangle\}$. Thus, node $u \in G$ will be reported as a match of node $v \in Q$ even for cost threshold $\epsilon = 0$. Clearly, this is a false positive.

To solve this problem, we do not employ a uniform propagation factor for different labels. Instead, for each label l, we select an optimum $\alpha(l)$. For a given label l, let us assume that, the maximum number of one-hop neighbors with label l, of any node in G is $n(l)$. To consider the worst case, let us assume that, some node u in G has no one-hop neighbor with label l; but it has $n^2(l)$ two-hop neighbors with label l, $n^3(l)$ three-hop neighbors with label l and so on. Therefore, the strength of label l at node u in G will be as follow,

$$\begin{aligned} A_G(u, l) &= \sum_{i=2}^{h} n^i(l)\alpha^i(l) \\ &< \frac{n^2(l)\alpha^2(l)}{1 - n(l)\alpha(l)} \end{aligned} \tag{5}$$

To avoid false positive, we want $A_G(u, l) < A_Q(v, l) = \alpha(l)$ as shown in Figure 7. Hence, $\alpha(l) < \frac{1}{n(l)+n^2(l)}$.

In the next section, we will introduce an iterative method to find the top-k embeddings in a large graph.

4. SEARCH ALGORITHM

In this section, we introduce a scalable iterative approach to find the top-k graph embeddings. Our goal is not to enumerate all the possible embeddings f in G for a given query graph, whose cost is prohibitive. Instead of enumerating f, we directly use $A_G(u, l)$ to bound $A_f(u, l)$ since $A_G(u, l) \geq A_f(u, l)$.

LEMMA 3. *Given a query graph Q and its embedding f in G, $\forall l, u \in V_f$, $A_G(u, l) \geq A_f(u, l)$.*

PROOF. Omitted. □

Lemma 3 shows that $A_G(u, l)$ in the neighborhood vector $R_G(u)$ cannot be lower than $A_f(u, l)$ of the same label l in the neighborhood vector $R_f(u)$, where f is a subgraph of G.

THEOREM 4. *Given a query graph Q and its embedding f in G,*

$$\sum_{v \in V_Q} \sum_{l \in R_Q(v)} M(A_Q(v, l), A_G(f(v), l)) \leq C_N(f)$$

PROOF. It follows from Lemma 3 so that $M(A_Q(v, l), A_f(u, l)) \geq M(A_Q(v, l), A_G(u, l))$. □

Theorem 4 shows that without enumerating embeddings of Q in the target graph G, we can derive the lower bound: $M(A_Q(v, l), A_G(u, l))$, where u is a possible match of v in G.

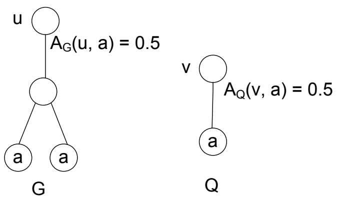


Figure 7: False Positive for High α

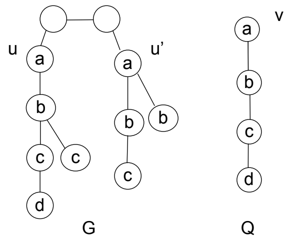


Figure 8: Node Matching Example

Our algorithm works by iteratively pruning unpromising nodes in the target graph.

1. Match the individual nodes of the query graph with some nodes in the target graph, which satisfies a predefined cost threshold ϵ (See Eq. 7).
2. Discard the labels of the unmatched nodes in the target graph.
3. Propagate the labels only among the matched nodes from the previous step. Recompute the neighborhood vectors $R_G(u)$ only for the matched nodes. Repeat Step 1 until convergence.

During each iteration, we remove the labels of the unmatched nodes in the target graph G and then recompute the neighborhood vectors only for the matched nodes. Since the modified target graph has more unlabeled nodes compared to the previous iteration, it will decrease $A_G(u, l)$. With this new and reduced set of neighborhood vectors and using the same cost threshold ϵ, we determine the individual node matches with the nodes of the query graph. Therefore, some additional nodes in G will be unmatched at each iteration. The iteration continues until there is no unmatched nodes found. For real life graphs, with less automorphism and more distinct labels, we can unlabel most of the unpromising nodes using this technique. Thus finding the top-k embeddings from the set of remaining matched nodes of G becomes almost trivial.

To determine the runtime complexity of our iterative search algorithm, let us denote the number of promising nodes present before i-th iteration as n_i and the number of unpromising nodes discovered at i-th iteration as k_i; where $i \geq 1$. Clearly, $n_1 = n$ and $n_{i+1} = n_i - k_i$. If there are total r iterations, $\sum_{i=1}^{r} k_i = O(n)$. Let the complexity of iteration i be T_i. In the first iteration, for each node, it needs to propagate its labels at h hops. Thus, $T_1 = O(nld^h)$, where l is the average number of labels, d^h is the average number of h-hop neighbors for each node in G. However, for each of the subsequent iterations, it is not necessary to perform such propagation for all the nodes in the graph. Rather, the number of unpromising nodes at iteration $i + 1$, for $i \geq 1$, can be determined by either propagating the remaining n_{i+1} nodes' labels, or by subtracting the effect of k_i unpromising nodes from previous iteration. Hence, $T_{i+1} = O(\min\{n_{i+1}, k_i\}ld^h)$, for $i \geq 1$. Therefore, the overall runtime complexity of our search algorithm is given as follow.

$$\begin{aligned} T_1 + \sum_{i=2}^{r} T_i &= O(nld^h) + \sum_{i=1}^{r-1} O(\min\{n_{i+1}, k_i\}ld^h) \\ &= O(nld^h) + \sum_{i=1}^{r-1} O(k_i ld^h) \\ &= O(nld^h) \end{aligned} \tag{6}$$

In practice, it converges much faster. Next, we shall discuss the details of the iterative algorithm and the algorithm to find the top-k embeddings from the nodes filtered by the iterative algorithm.

4.1 Node Match

Given the target graph G and the query graph Q, we compute the vectors $R_G(u)$ and $R_Q(v)$ for all nodes $u \in V_G, v \in V_Q$, considering their h-hop neighborhoods. For each node pair $u \in V_G, v \in V_Q$, s.t. $L(v) \subseteq L(u)$, we calculate the node matching cost, $cost(u, v)$ as the difference of their neighborhood vectors,

$$cost(u, v) = \sum_{l \in R(v)} M(A_Q(v, l), A_G(u, l)). \tag{7}$$

Figure 8 shows an example. Assume $\alpha = 0.5$ and $h = 2$. We get $R_G(u) = \{\langle b, 0.5\rangle, \langle c, 0.25 \times 2\rangle\} = \{\langle b, 0.5\rangle, \langle c, 0.5\rangle\}$, and similarly, $R_G(u') = \{\langle b, 1\rangle, \langle c, 0.25\rangle\}$. Meanwhile, for the query graph Q, we have $R_Q(v) = \{\langle b, 0.5\rangle, \langle c, 0.25\rangle\}$. Hence, $cost(u, v) = 0$ and also $cost(u', v) = 0$ following the above equation.

Now, for each node $v \in V_G$, we maintain a list of nodes $u \in V_G$, such that $L(v) \subseteq L(u)$ and $cost(u, v) \leq \epsilon$. Here, ϵ is a predefined cost threshold. The value of ϵ will be discussed shortly.

4.2 Top-k Search

In order to find the top-k graph embedding, we initialize the cost threshold ϵ to a small value $\epsilon_0 \geq 0$ and perform the above mentioned iterative procedure until it terminates. Given the matched nodes, if we cannot find at least k embeddings from them, with cost $\mathcal{C}_N(f) \leq \epsilon|V_Q|$ each; then the threshold cost ϵ is doubled and we repeat the above procedure, until the k embeddings are found. Otherwise, we find the top-k embeddings among the matched nodes. Note that, at this point, any embedding formed by all unmatched nodes will have a cost $\mathcal{C}_N(f) > \epsilon|V_Q|$. However, it is possible to have some embedding with a few matched and unmatched nodes, and the cost of such embeddings might also be $C_N(f) \leq \epsilon|V_Q|$. The problem is eliminated as follow. We set ϵ equal to the highest cost of the discovered top-k embeddings and then run the algorithm again (this step will find top-k embeddings whose node cost might be higher than ϵ). In this case, any embedding formed by at least one of the unmatched node will have a cost more than that of any of the top-k embeddings found earlier. Hence, the top-k embeddings identified only using the matched nodes will be the best top-k embeddings. The complete algorithm is given below.

Algorithm 1 Top-k Search

```
Input: Target graph $G$, query graph $Q$, a positive integer $k$.
Output: Top-$k$ matches $f$ based on the cost metric $\mathcal{C}_N$.
procedure
 1: $\epsilon \leftarrow \epsilon_0$, compute $R_G(v), \forall v \in V_Q$
 2: $list_0(v) = \{u : u \in V_G \wedge L(v) \subseteq L(u)\}$
 3: $i \leftarrow 1$, start with original graph $G$ and compute $R_G(u), \forall u \in V_G$
 4: for all $v \in V_Q$ do
 5:     $list_i(v) = \{u : u \in V_G \wedge L(v) \subseteq L(u) \wedge cost(u, v) \leq \epsilon\}$
 6: end for
 7: $(list, i)$ = Iterative Unlabel$(list, i, G, Q)$
 8: if $k$ matches of cost $C_N(f) \leq \epsilon|V_Q|$ can be found in $\{u : u \in list_i(v) \forall v \in V_Q\}$ then
 9:     report top-$k$ matches and stop
10: else
11:     $\epsilon \leftarrow 2\epsilon$
12:     go back to step 2
13: end if
```

Algorithm 2 Iterative Unlabel $(list, i, G', Q)$

```
procedure
 1: if $|list_i(v)| < |list_{i-1}(v)|$ for some $v \in V_{G'}$ then
 2:     for all $u \in V_{G'}$ do
 3:         if $u \notin list_i(v) \forall v \in V_Q$ then
 4:             unlabel $u$
 5:         end if
 6:     end for
 7:     recompute $R(u) \forall u \in V_{G'}$
 8:     $(list, i)$ = Iterative Unlabel$(list, i + 1, G', Q)$
 9: else
10:     return $(list, i)$
11: end if
```

From the final list of matched nodes for each node in V_Q, how can we find embeddings with cost $\mathcal{C}_N(f) \leq \epsilon|V_Q|$ each (line 8 of Algorithm 1)? One simple technique is to consider all possible combinations from the lists and verify their costs. When the number of matched nodes in each of the final lists is small, it is not time consuming to check. However, when the lists are long, we can do better than brute force enumeration using dynamic programming.

After a final list of matched nodes $list(v)$ for each $v \in V_Q$ is generated, we perform the propagation once more among the matched nodes; however this time we propagate the node id's instead of labels. After this propagation, each matched node u in G will have its neighboring nodes (denoted as $neighbor(u)$) within h hops who have influence on the cost (Eq. 1).

The final embeddings can be formed as follows. We select a node $u \in list(v)$ for some $v \in V_Q$ and initialize a set $Possible\_Match = neighbor(u)$. We have two situations: (1) within h hops of u, there is no $f(v')\ \forall v' \neq v$ in Q. (2) $\forall v' \neq v$ of Q, we try to identify a match u' inside $Possible\_Match$ and extend this set by adding $neighbor(u')$ and also eliminating the node u' from $Possible\_Match$. For the first situation, we could derive the cost for node u, $\sum_{l \in L(v)} A_Q(v, l)$. We can recurse among these two situations to find the embeddings. In this way, we can find the low-cost embeddings without enumerating all possible combinations among the nodes in the final lists.

5. INDEXING

The most expensive parts of Ness are the computation of $R_G(u)$ for all u in G (Line 3 of Algorithm 1) and the determination of $list_1(v)$ for all v in V_Q (Line 5 of Algorithm 1). However, the computation of $R_G(u)$ can be done off-line by performing a breadth first search up to h-hops from each node in G. Its time complexity is $O(|V_G| \cdot d^h)$, where d is the average degree of each node.

To speed up the computation of $list_1(v)$ for all $v \in V_Q$, we use two types of simple index structures. In the first type of indexing, we build a hash table corresponding to each label. The nodes in G are hashed based on their labels. Given a query node v, we use this hash structure to quickly identify the set of possible matches u, such that $L(v) \subseteq L(u)$. If the labels of v are very selective, there will be a limited number of possible matches u and we can quickly determine the nodes u among these matches, for which $cost(u, v) \leq \epsilon$.

Algorithm 3 Neighborhood Based Indexing

```
Off-line Procedure
1: pre compute $R_G(u) = \{\langle l, A_G(u, l)\rangle\}$ for all $u \in V_G$
2: for all label $l$ do
3:    create sorted list $S(l)$ of nodes in descending order of
      $A_G(u, l)$, such that $u_i(l)$ is $i$-th node in $S(l)$
4: end for
On-line Procedure
1: $i \leftarrow 1$
2: $sum(i) \leftarrow \sum_{l \in R(v)} M(A_Q(v, l), A_G(u_i(l), l))$
3: if $sum(i) \leq \epsilon$ then
4:    $i \leftarrow i + 1$
5:    go to step 2
6: else
7:    verify $\forall u_j(l)$ if $cost(u_j(l), v) \leq \epsilon, \quad j < i, l \in R_Q(v)$
8: end if
```

However, if the labels of v are not very selective and there are many possible matches using the hashing technique discussed above, we use the second index structure, which is built on the neighborhood vector $R_G(u)$ following the principle of Threshold Algorithm [12]. The neighborhood vector $R_G(u) = \{\langle l, A_G(u, l)\rangle\}$ for each node $u \in V_G$ is pre computed. Next, for each label l, we generate a sorted list $S(l)$ of nodes u in descending order of their $A_G(u, l)$ values. Let us denote the node at position i from the top of $S(l)$ as $u_i(l)$. In the online phase, we start from the top of the each sorted list $S(l)$ in parallel and go to the next position in the subsequent iteration. For some position i from the top, we compute, $sum(i) = \sum_{l \in R_Q(v)} M[A_Q(v, l), A_G(u_i(l), l)]$. Assume at iteration $i = i_1$, $sum(i_1)$ becomes greater than the cost threshold ϵ. Then, we terminate this iterative procedure and verify for all nodes $u_j(l)$, where $j < i_1, l \in R_Q(v)$, if $cost(u_j(l), v) \leq \epsilon$. For each $v \in V_Q$, we need to verify only $O((i_1 - 1)|l|)$ nodes for their cost; where $|l|$ denotes the number of labels in $R_Q(v)$. This can reduce the complexity of the online algorithm significantly. The complete procedure for neighborhood based indexing is given in Algorithm 3.

Proof of Correctness. Let us denote $S_i(l)$ as all the nodes up to position i from top of the sorted list $S(l)$, i.e. $S_i(l) = \{u_j(l), 1 \leq j \leq i\}$. The following lemma will be useful to prove the correctness of our indexing algorithm.

LEMMA 4. *If $sum(i) > \epsilon$, then for all $u \notin \{S_{i-1}(l) : l \in R_Q(v)\}$, $cost(u, v) > \epsilon$.*

PROOF. It follows directly from the fact that, each $S(l)$ is a sorted list of nodes u in descending order of $A_G(u, l)$ values. □

Therefore, in Algorithm 3, we start from $i = 1$ and find the smallest i, for which $sum(i) > \epsilon$. Following the previous lemma, for any node $u \notin \{S_{i-1}(l) : l \in R_Q(v)\}$, we can eliminate them without actually computing $cost(u, v)$.

We note that, our indexing can be easily implemented in a disk-based manner for very large graphs. Also we can apply external memory breadth first search algorithms, e.g., Ulrich Meyer [1] and Lars Arge [2], to compute the neighborhood vectors $R_G(u)$ for all the nodes.

Dynamic Update. Our indexing structure can efficiently accommodate dynamic updates in G, i.e., insertion/ deletion of nodes, edges and labels. If a node u is added or deleted in G, it will only change the vectors of u's h-hop neighbors. We only need to propagate the labels of these nodes and modify their neighborhood vectors. They also need to be updated in the sorted lists of label l for all $l \in L(u)$. The addition/ deletion of a label can be handled similarly. If an edge (u_1, u_2) is added/ deleted in G, we need to update vectors for the $h - 1$ hop neighbors of both u_1 and u_2.

6. QUERY OPTIMIZATION

In this section, we eliminate the non-discriminative labels both from the target and query graphs at the initial stage of our matching algorithm to make the technique more efficient. The efficiency of the algorithm "Iterative Unlabel" is related to the number of individual node matches for each node in the query graph. If there exists some node which is not very selective in terms of its own labels or the labels present in its neighborhood, there will be many matches corresponding to that node at the initial stage of our algorithm. In order to eliminate the problem posed by these nodes, we first eliminate all the non-discriminative labels both from the target graph and the query graph, and then we also ignore the nodes in the query graph, which do not contain sufficient number of discriminative labels in themselves and in their neighborhoods. These non-discriminative labels are considered at the last stage of our matching algorithm, i.e., when we search for the final matches. In the following discussion, we shall clarify the notion of discriminative and non-discriminative labels in the perspective of node and graph matches.

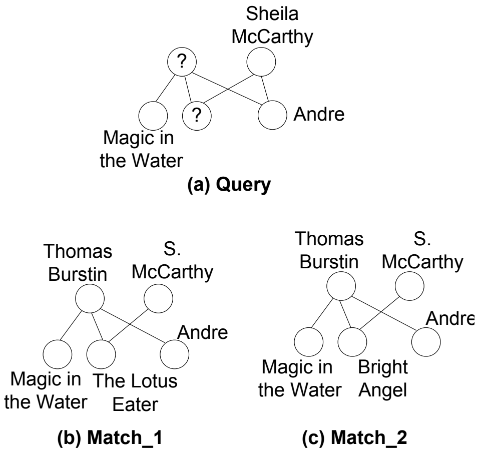


Figure 10: Top-2 Matches (Query 1)

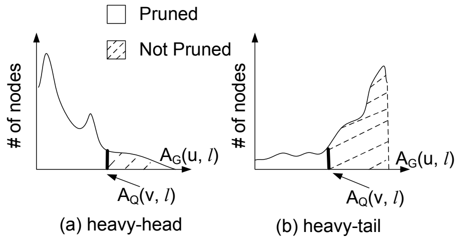


Figure 9: Discriminative (Heavy-Head) vs. Non-Discriminative (Heavy-Tail) Distribution

Let us consider the distribution of $A_G(u,l)$ values of some label l, $<l, A_G(u,l)> \in R_G(u)$, for different nodes $u \in V_G$. Figure 9 shows one example. For a label l, we plot the different $A_G(u,l)$ values along the X-axis. The Y-axis shows the number of nodes u having that particular $A_G(u,l)$ value in their neighborhood vector $R_G(u)$. The distribution in Figure 9(a) is skewed towards the smaller values of $A_G(u,l)$, whereas Figure 9(b) is skewed towards the higher values of $A_G(u,l)$. We call them as *heavy-head* and *heavy-tail* distributions respectively. Given a query node v, since we prune all the nodes u in G for which $\sum_{l \in R_Q(v)} M[A_Q(v,l), A_G(u,l)] > \epsilon$, the labels with heavy-head distribution have more pruning power than those with heavy-tail distribution. Therefore, we should retain labels with heavy-head distribution for node match, as those labels are more discriminative.

7. EXPERIMENTAL RESULTS

In this section, we present the experimental results to demonstrate the effectiveness and the efficiency of the neighborhood based similarity search technique on a number of real-life and synthetic graph datasets including DBLP, Intrusion, Freebase and WebGraph. In order to evaluate the effectiveness, we show two possible applications - RDF query answering and network alignment. We test the robustness of our approach by providing the accuracy of the best matches for queries of different sizes and under the presence of random noise. The efficiency and scalability of our approach are also investigated. All experiments are performed using a single core in a 40GB, 2.50GHz Xeon server.

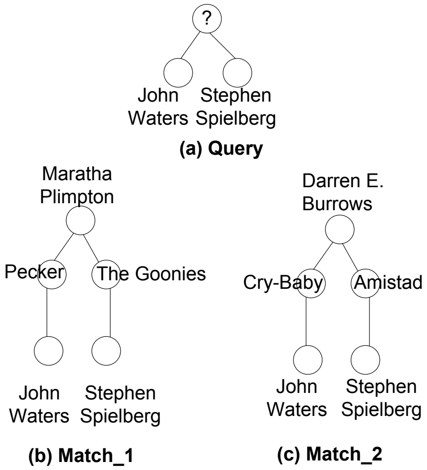


Figure 11: Top-2 Matches (Query 2)

7.1 Graph Data Sets

DBLP Collaboration Graph. The DBLP collaboration graph is downloaded from www.informatik.uni-trier.de/∼ley /db. There are 684K distinct authors and 7M co-author edges among them. We consider the name of each author as the label of that node. There are $683,927$ distinct labels in DBLP. We use the DBLP dataset for efficiency test.

Freebase Entity Relationship Graph. Freebase is a large collaborative knowledge base of structured data harvested from many sources including Wikipedia. We downloaded the 'film' entity relationship graph data from http://download.freebase.com/datadumps /2010-10-07. This graph has 172K nodes, each representing an entity, i.e., actor, movie, director, producer and so on. An edge represents the relationship between two entities. Names of entities are treated as labels. There are total 579K edges and $159,514$ distinct labels in this graph. Freebase graph is used for effectiveness, robustness and efficiency analysis.

Intrusion Alert Network. This network contains the anonymous log data of intrusion alerts in a computer network. It has 200K nodes and 703K edges where each node is a computer and an edge means a possible attack such as Denial-of-Service and TCP Service Sweep. Each node has 25 labels (computer generated alerts in this case) on average. There are around $1,000$ types of alerts. We use this graph for robustness and efficiency experiments.

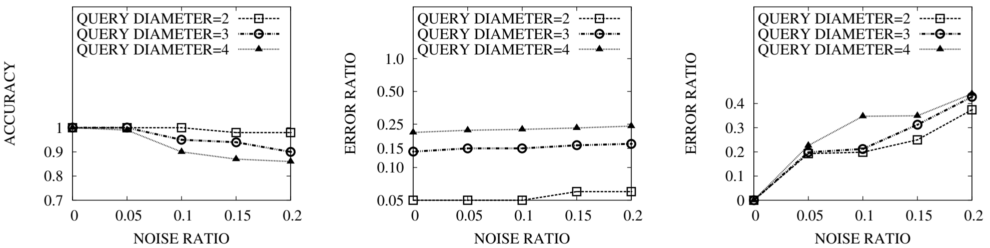
WebGraph with Synthetic Labels. We downloaded the uk-2007-05 web graph data from http://webgraph.dsi.unimi.it [4]. This web graph is a collection of UK web pages. For our experiments, we use a subset that contains 10M pages (i.e. nodes) and 213M hyperlinks (i.e. edges). We uniformly assign $10,000$ synthetically generated labels across various nodes, such that each node gets one label. We test the scalability of our approach on this graph.

7.2 RDF Query Answering

In addition to the query shown in Figure 1, we show two more examples using the Freebase graph dataset.

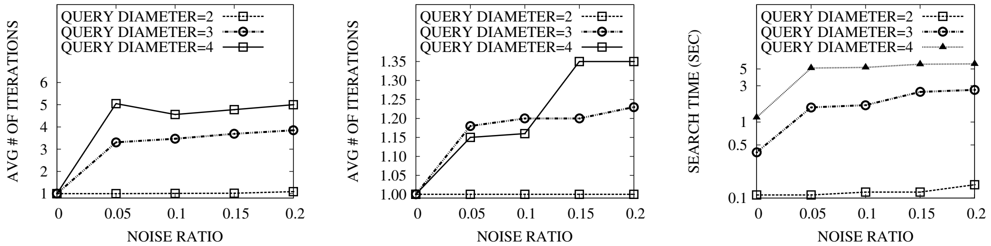
Query 1: *Who did cinematography for at least two 'Sheila McCarthy' movies, one of them being 'Andre'? The person was also cinematographer of the movie 'Magic in the Water'.*

Here, we would like to emphasize that, 'Sheila McCarthy' did not act in the movie 'Andre'. However, as discussed earlier, this type of inaccuracy is common, since the user may not have the ac-



(a) Accuracy (**Intrusion**) (b) Error Ratio (**Freebase**) (c) Error Ratio (**Intrusion**)

Figure 12: Robustness of Network Alignment



(a) Top-k Search (Algorithm 1) (b) Iterative Unlabel (Algorithm 2) (c) Online Search Time

Figure 13: Convergence of Online Search Algorithm (DBLP)

curate information, or there can be some noises in the target graph. Using our approach, we get the following top-2 answers for this query, as shown in Figure 10.

Query 2: *Which actors have appeared in both a "John Waters" movie and a "Steven Spielberg" movie?*

The query and the corresponding top-2 matches are shown in Figure 11. Here, we would like to emphasize that, actors in the Freebase dataset are not directly connected with the directors and cinematographers; rather via some movies. To write a SPARQL query, we need to maintain this structural property. However, given the query graph as shown in Figure 11, which does not maintain this structural property; we still obtain the results, where the embeddings are very close to the query graph.

7.3 Network Alignment

We perform network alignment for query graphs of different sizes and in the presence of various amount of noise. For these experiments, three different sets of query graphs are used with diameters 2, 3, 4 and the number of nodes 100, 150, 200 respectively. These query sets will simulate the situation when we align a small social network to a large one. In each query set, we randomly select 100 subgraphs with the specified diameters and nodes from the original graph datasets. Then we introduce noise by adding edges to the query graphs, which are not present in the original graph. The *noise ratio* is defined as the number of edges added divided by the original number of edges present in the query graph. We use propagation depth 2 and α is selected as described earlier in Section 3.3.

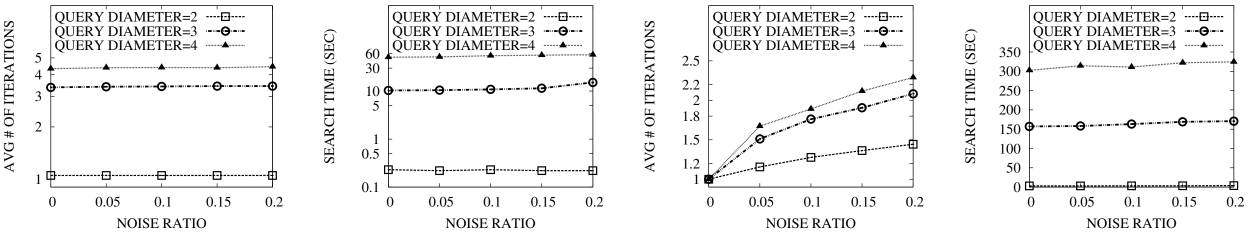
The robustness of our approach in the presence of random noise is measured using two metrics. The *accuracy* is defined as the number of correctly identified nodes of the target graph in all the top-1 matches divided by the total number of nodes in all query graphs in the corresponding query set. The accuracy is 1 for both DBLP and Freebase datasets with different amounts of noise, since these graphs have more number of distinct labels. The accuracy vs. noise ratio plots for Intrusion dataset is shown in Figure 12(a). The accuracy remains at a relatively high level when the noise ratio increases up to 0.2.

We also measure the *error ratio*, which is defined as the number of incorrectly identified nodes of the target graph in all the top-1 matches divided by the total number of nodes in all query graphs in the corresponding query set. The lower is the error ratio, the more distinguishable the nodes are in terms of their neighborhood structure and contents. The error ratio remains close to 0 for DBLP graph at different amount noise. The error ratio vs. noise ratio plots for Freebase and Intrusion are shown in Figure 12(b) and 12(c) respectively. It can be observed that the error ratio remains at a relatively low level for Freebase graph, when the noise ratio increases up to 0.2. Hence, these experiments indicate that DBLP and Freebase is less automorphic compared to the Intrusion network.

7.4 Efficiency Results

We provide the running time of our algorithm for different datasets in Table 1. For these experiments, we randomly select query graphs with 50 nodes and diameter 2 from the original graph datasets. The vectorization and indexing is performed with propagation depth 2 and the search algorithm is used to identify the top-1 matches. It can be observed that our algorithm is very efficient for large graph datasets. The on-line phase for Intrusion graph requires more time because the average number of labels per node is much higher than that in other graphs. This leads to more time used for cost computation (Eq. (7)).

We also verify the convergence rate of our 'Top-k Search' and 'Iterative Unlabel' algorithms for various network alignment experiments discussed earlier. The convergence rate of these algorithms is measured as the average number of iterations required before they terminate. When the noise ratio is increased, our algorithm requires more iterations to satisfy the cost threshold. Thus, the corresponding running time also increases as shown in Figure 13 for the DBLP dataset. Moreover, it requires more time to identify the



(a) Convergence (**Freebase**) (b) Search Time (**Freebase**) (c) Convergence(**Intrusion**) (d) Search Time (**Intrusion**)

Figure 14: Convergence of Online Search Algorithm (Freebase & Intrusion)

matches of a larger query graph. The convergence plots for Freebase and Intrusion networks are given in Figure 14.

Dataset	2-hop Indexing (Off-line)	Top-1 Search (Online)
DBLP (0.7M, 7M, 0.7M)	$1,733$ sec	0.06 sec
Freebase (0.2M, 0.6M, 0.2M)	280 sec	0.22 sec
Intrusion (0.2M, 0.5M, 1K)	227 sec	1.6 sec
WebGraph (10M, 213M, 10K)	$5,125$ sec	0.26 sec

Table 1: Efficiency: Off-line Indexing and Online Search

7.5 Neighborhood-based Cost Function Properties

Recall that we proved in Theorem 1 that our neighborhood-based cost function ensures there is no false negatives when the cost threshold is set to 0. In this subsection, we investigate the false positive rate by using our neighborhood-based cost function with threshold set to 0. This experiment is performed on DBLP, Freebase and Intrusion datasets. In particular, for each dataset, we select 100 small query subgraphs with 10 nodes each from the original graph. For each of the query graphs, by using 2-hop propagation, we identify all matches with cost = 0. Among these matches, we manually verify if there is any false positives, i.e. a match which is not graph isomorphic with the query graph. The percentage of false positives is calculated as the number of false positives divided by the total number of matches obtained. We show the results in Table 2. It can be seen that using our cost function with cost threshold set to 0, the percentage of false positives on real-life social/ information networks is very small.

Dataset	False Positive
DBLP	0%
Freebase	0%
Intrusion	0.3%

Table 2: False Positive Ratio

Dataset	Search with Index&Optimization	Search w/o Index&Optimization
DBLP	0.06 sec	9.63 sec
Freebase	0.22 sec	1.75 sec

Table 3: Benefits of Index and Optimization

As we have discussed earlier, the higher the value of h is, the lower the number of false positives will be. Therefore, for a target graph, we can employ error ratio as a cost function and learn the satisfactory value of h from training queries generated from the target graph. DBLP graph is used in this experiment. We use a training set of 100 small query graphs (with 10 nodes each) generated from the DBLP graph. The queries are generated in such a way that the labels in the query nodes are mostly not unique. Some noise is also added in these query graphs as explained earlier. Next, we start with $h = 0$ and gradually increase h until the error ratio becomes less than a small value. We show the results for DBLP graph in Figure 15. It can be observed that, by setting $h = 2$, we can reduce the error ratio to an acceptable level when the noise ratio is below 0.1. This indicates that for the real-life social/ information networks with few auto-morphism and many distinct labels, we only need a small propagation depth to make the error ratio close to zero.

7.6 Pruning Capacity of Search Algorithm

We verify the pruning capacity of our Top-k search algorithm with respect to the number of distinct labels present in the target graph. For this experiment, we use a subgraph extracted from the WebGraph dataset, which contains $1,000$ nodes and $14,067$ edges. We vary the number of distinct labels from 1 to 800. Given a randomly extracted query graph with the number of nodes $|V_Q| = 8, 10$ and 12 respectively, we check how many subgraphs need to be verified during the "final match" phase of our approach. The smaller this number is, the more powerful the pruning of our algorithm is. We plot the number of subgraphs need to be verified in the "final match" phase vs. the number of distinct labels in Figure 16. Note that the Y axis is in log scale. It can be observed that, when there is only 1 distinct label in the entire graph, we need to verify about 10^{25} subgraphs for a query graph with 8 nodes during the "final match" phase. However, as the number of distinct labels increases, the number of subgraphs that we need to verify decreases rapidly. For 800 distinct labels, we only need to verify a very small number of subgraphs (e.g. 12 subgraphs when $|V_Q| = 8$) in the "final match" phase of our approach. Thus, our algorithm can be very efficient on graphs with few automorphisms and many distinct labels.

7.7 Indexing and Query Optimization

In Table 3, we compare the running time of our online search algorithm with that of a linear scan with no indexing and query optimization. Each of the query graphs has 50 nodes and diameter 2 for this experiment. It can be observed that, our indexing and query optimization techniques can significantly speed up online search.

We also compare the index construction time of dynamic update with the cost of rebuilding the whole index when the target graph is modified. The propagation depth is 2 for these experiments. The results for DBLP dataset are shown in Figure 17. As we can see, for a wide range of updates in the target graph, it is more efficient to update the index structure rather than re-indexing the graph. The

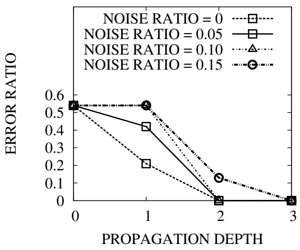


Figure 15: Satisfactory h Value (DBLP)

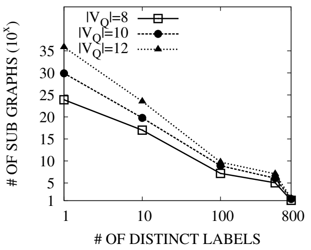


Figure 16: Pruning Capacity (WebGraph)

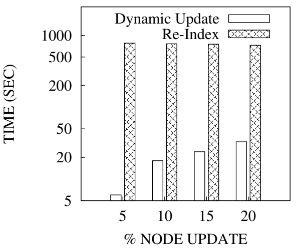
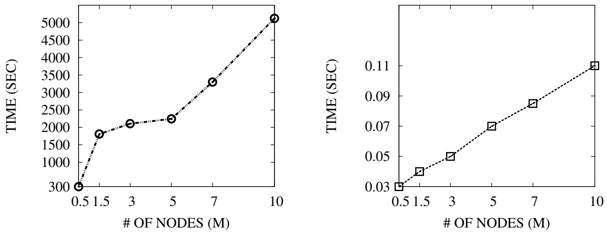


Figure 17: Dynamic Update Index (DBLP)

results also indicate that our index structure is very efficient against dynamic updates in the target graph.

7.8 Scalability

We show the scalability of our approach on the WebGraph dataset. The vectorization time as a function of the number of nodes in the graph is shown in Figure 18(a). Figure 18(b) shows the change trends of the online search time with respect to the number of nodes. The propagation depth is 2 for indexing and we identify the top-1 matches using our search algorithm. Each of the query graphs has 10 nodes and diameter 3 for this experiment. As it can be observed, for a graph with 10 million nodes, our approach can return the top-1 match in 0.11 second. The corresponding index building time is also tolerable. Both the index building time and the online search time is roughly linear in the number of nodes. These results show that our technique is highly scalable for large scale information/ social networks.



(a) Vectorization Time

(b) Search Time

Figure 18: Scalability Results (WebGraph)

8. RELATED WORK

Graph search has been studied in different contexts such as graph isomorphism, graph indexing, structure matching, etc. In XML, where the structures encountered are often trees and lattices, queries built on path expression become popular [28] and their corresponding indices have been developed [9].

In bioinformatics, exact and approximate graph alignment has been extensively studied, e.g., PathBlast [21], Saga [33], NetAlign [23], IsoRank [32]. They are targeting relatively small biological networks with less than 10k nodes. It is difficult to apply them in social and information networks with thousands or even millions of nodes. Kernel based graph matching techniques are also proposed, e.g., common walks [16, 18], shortest path [5], limited-size subgraphs [19] and subtree patterns [20]. Recently, Shervashidze et. al [25] proposed fast subtree pattern kernel based on the Weisfeiler-Lehman method. Kernel methods do not support subgraph search well.

For subgraph search, Shasha et al. [31] extend the path-based technique for full-scale graph retrieval; Yan et al. propose gIndex [37] using frequent subgraphs. These studies inspired new graph index structures such as δ-Tolerance Closed Frequent Subgraphs [8], Tree [40], and GCoding[41]. He et al. [17] develop a closure tree index to perform approximate graph search. Tian et al. [33] design a fragment based index to assemble an approximate match. Shang et al. introduce an efficient algorithm for testing subgraph isomorphism [29]. Ferro et al. propose a novel indexing scheme, SING [26], based on locality information. All these methods are built strictly on graph structures, not good for approximate search shown in Figure 1.

There have been significant studies on inexact graph matching on attributed graphs [30, 7]. Tong et al. [35] propose the best-effort pattern matching in large attributed graphs. It finds the best match not based on the proximity among the labels, rather based on the shape of the query graph. Tian et al. [34] proposed an approximate subgraph matching tool, called TALE, with efficient indexing and high pruning capabilities. Mongiov$\grave{i}$ et. al. introduce a set-cover-based inexact graph matching technique, called SIGMA [24]. Both techniques only use edge misses to measure the quality of graph matching. Therefore, they are not appropriate for the proximity based search scenario studied in this work. There have been some recent work on inexact graph matching, i.e., simulation based cubic time graph pattern matching [13], homomorphism based subgraph matching [14], Belief propagation based net alignment [3], edge-edit-distance based subgraph indexing technique [39] and graph partition based subgraph identification scheme [6].

9. CONCLUSIONS

In this paper, we defined a new graph similarity measure, neighborhood based graph similarity, and proposed an information propagation model to convert a large network into a set of multidimensional vectors, where sophisticated indexing and similarity search algorithms are available. We proved, under this measure, that subgraph similarity search is NP hard, while graph similarity match is polynomial. We introduced a criterion to select the best propagation rate with respect to different node labels in a graph. We further investigated the techniques to index the neighborhood vectors and to compress them by deleting non-discriminative labels, thus optimizing the query processing time. The proposed method, called Ness, is not only efficient, but also robust against structure changes and information loss. Empirical results show that it could quickly and accurately find high-quality matches in large networks, with negligible time cost. In future work, it will be interesting to consider the graph alignment problem, when the node labels in two graphs are not exactly identical, i.e the same user can have slightly different usernames in Facebook and Twitter.

10. ACKNOWLEDGMENTS

This research was sponsored in part by the U.S. National Science Foundation under grant IIS-0954125 and by the Army Research Laboratory under cooperative agreement W911NF-09-2-0053 (NS-CTA). X. Yan was supported in part by the Open Project Program of the State Key Lab of CAD&CG (Grant No. A1001), Zhejiang University. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

11. REFERENCES

[1] D. Ajwani, U. Meyer, and V. Osipov. Improved external memory bfs implementation. In *ALENEX*, 2007.

[2] L. Arge, G. S. Brodal, and L. Toma. On external-memory mst, sssp and multi-way planar graph separation. In *Workshop on Algorithmic Theory, Vol. 1851 of LNCS*, pages 433–447. Springer, 2000.

[3] M. Bayati, M. Gerritsen, D. F. Gleich, A. Saberi, and Y. Wang. Algorithms for large, sparse network alignment problems. *ICDM*, 0:705–710, 2009.

[4] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *WWW*, pages 595–601, 2004.

[5] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *ICDM*, pages 74–81, 2005.

[6] M. Brocheler, A. Pugliese, and V. S. Subrahmanian. Cosi: Cloud oriented subgraph identification in massive social networks. *ASONAM*, 0:248–255, 2010.

[7] S. Chaudhury, K. Ganjam, V.Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, 2003.

[8] J. Cheng, Y. Ke, W. Ng, and A. Lu. FG-Index: Towards verification-free query processing on graph databases. In *SIGMOD*, pages 857 – 872, 2007.

[9] C. Chung, J. Min, and K. Shim. APEX: An adaptive path index for xml data. In *SIGMOD*, pages 121–132, 2002.

[10] S. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[11] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.

[13] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. *PVLDB*, 3(1):264–275, 2010.

[14] W. Fan, J. Li, S. Ma, H. Wang, and Y. Wu. Graph homomorphism revisited for graph matching. *PVLDB*, 3(1):1161–1172, 2010.

[15] Freebase. http://www.freebase.com.

[16] T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *COLT and the 7th Kernel Workshop*, 2003.

[17] H. He and A. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, page 38, 2006.

[18] H.Kashima and A.Inokuchi. Kernels for graph classification. ICDM Workshop on Active Mining, 2002.

[19] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD*, pages 158–167, 2004.

[20] J. J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In *First Int. Workshop on Mining Graphs, Trees and Sequences*, pages 65–74, 2003.

[21] B. P. Kelley, B. Yuan, F. Lewitter, R. Sharan, B. R. Stockwell, and T. Ideker. Pathblast: a tool for alignment of protein interaction networks. *Nucleic Acids Res*, 32:83–88, 2004.

[22] A. Khan, X. Yan, and K.-L. Wu. Towards proximity pattern mining in large graphs. In *SIGMOD*, 2010.

[23] Z. Liang, M. Xu, M. Teng, and L. Niu. Netalign: a web-based tool for comparison of protein interaction networks. *Bioinformatics*, 22(17):2175–2177, 2006.

[24] M. Mongiovì, R. D. Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan. Sigma: a set-cover-based inexact graph matching algorithm. *J. Bioinformatics and Computational Biology*, 8(2):199–218, 2010.

[25] N. N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. pages 1660–1668. Curran, 2010.

[26] R. D. Natale, A. Ferro, R. Giugno, M. Mongiovì, A. Pulvirenti, and D. Shasha. Sing: Subgraph search in non-homogeneous graphs. *BMC Bioinformatics*, 11:96, 2010.

[27] E. Prudhommeaux and A. Seaborne. Sparql query language for rdf. Technical report, W3C, 2007.

[28] C. Qun, A. Lim, and K. Ong. D(k)-index: An adaptive structural summary for graph-structured data. In *SIGMOD*, pages 134–144, 2003.

[29] H. Shang, Y. Zhang, X. Lin, and J. Yu. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. In *VLDB*, pages 364–375, 2008.

[30] L. Shapiro and R. Haralick. Structural descriptions and inexact matching. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.

[31] D. Shasha, J. T.-L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *PODS*, pages 39–52, 2002.

[32] R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *PNAS*, 105(35):12763–12768, 2008.

[33] Y. Tian, R. McEachin, C. Santos, D. States, and J. Patel. SAGA: a subgraph matching tool for biological graphs. *Bioinformatics*, 23(2):232–239, 2006.

[34] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. In *ICDE*, pages 963–972, 2008.

[35] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, pages 737–746, 2007.

[36] D. J. Watts, P. S. Dodds, and M. E. J. Newman. Identity and search in social networks. *Sience*, 296:1302–1305, 2002.

[37] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, pages 335–346, 2004.

[38] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *SIGMOD*, pages 766–777, 2005.

[39] S. Zhang, J. Yang, and W. Jin. Sapper: Subgraph indexing and approximate matching in large graphs. *PVLDB*, 3(1):1185–1194, 2010.

[40] P. Zhao, J. Yu, and P. Yu. Graph indexing: tree + delta >= graph. In *VLDB*, pages 938–949, 2007.

[41] L. Zou, L. Chen, J. Yu, and Y. Lu. A novel spectral coding in a large graph database. In *EDBT*, pages 181–192, 2008.