NeMa: Fast Graph Search with Label Similarity

Arijit Khan Yinghui Wu Charu C. Aggarwal Xifeng Yan

Computer Science
University of California, Santa Barbara
{arijitkhan, yinghui, xyan}@cs.ucsb.edu

IBM T. J. Watson Research
Hawthorne, NY
charu@us.ibm.com

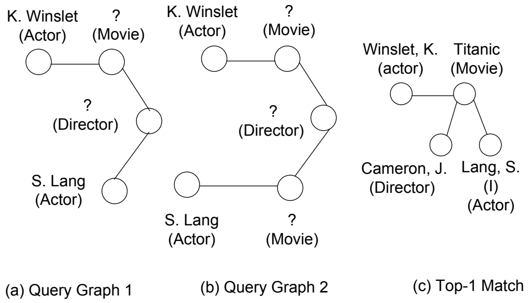
ABSTRACT

It is increasingly common to find real-life data represented as networks of labeled, heterogeneous entities. To query these networks, one often needs to identify the matches of a given *query graph* in a (typically large) network modeled as a *target graph*. Due to noise and the lack of fixed schema in the target graph, the query graph can substantially differ from its matches in the target graph in both structure and node labels, thus bringing challenges to the graph querying tasks. In this paper, we propose NeMa (**Ne**twork **Ma**tch), a neighborhood-based subgraph matching technique for querying real-life networks. (1) To measure the quality of the match, we propose a novel subgraph matching cost metric that aggregates the costs of matching individual nodes, and unifies both structure and node label similarities. (2) Based on the metric, we formulate the minimum cost subgraph matching problem. Given a query graph and a target graph, the problem is to identify the (top-k) matches of the query graph with minimum costs in the target graph. We show that the problem is NP-hard, and also hard to approximate. (3) We propose a heuristic algorithm for solving the problem based on an inference model. In addition, we propose optimization techniques to improve the efficiency of our method. (4) We empirically verify that NeMa is both effective and efficient compared to the keyword search and various state-of-the-art graph querying techniques.

1. INTRODUCTION

With the advent of the Internet, sources of data have increased dramatically, including the World-Wide Web, social networks, genome databases, knowledge graphs, medical and government records. Such data are often represented as *graphs*, where nodes are labeled entities and edges represent relations among these entities [14, 43]. Querying and mining of graph data are essential for a wide range of emerging applications [1, 15, 30].

To query these graphs, one often needs to identify the matches of a given *query graph* in a (typically large) *target graph*. Traditional graph querying models are usually defined in terms of *subgraph isomorphism* and its extensions (*e.g.,* edit distance), which identify subgraphs that are exactly or approximately isomorphic to query graphs [35, 33, 43]. In addition, a wide range of query models and languages are proposed — such as SPARQL and XDD for the XML and RDF data — which require a standard schema of queries and target graphs. Nevertheless, the real-life graphs are *complex* and *noisy*, and often lack standardized schemas [1]. Indeed, (a) the nodes may be heterogeneous, referring to different entities (*e.g.,* persons, companies, documents) [14]. (b) The node labels in a graph often carry rich semantics, *e.g.,* id, urls, personal information, logs, opinions [15]. (c) Worse still, the semantics of entities and their interconnections in various datasets may be different and unknown to the users [1]. In this context, a match may not necessarily be (even approximately) isomorphic to the query graph in terms of label and topological equality. Thus, traditional graph querying techniques are not able to capture good quality matches. Consider the following example over the IMDB movie dataset.



(a) Query Graph 1 (b) Query Graph 2 (c) Top-1 Match

Figure 1: A Query and Its Match (Example 1.1)

Example 1.1. *A user wants to find a movie of actress 'Kate Winslet' that is directed by the same director who also worked with actor 'Stephen Lang'. Even if the schema and exact entity labels of the target network are not available, the user can still come up with some reasonable graph representation of the query [15, 17], as illustrated in Figure 1(a) and 1(b). Observe that such graphical representation may not be unique, and there might not be an exact match of the query graph in the dataset. Indeed, the result in Figure 1(c) (a star-shaped graph) is by no means similar to the query graphs in Figure 1 (a) and (b) (both chain-shaped graphs) under traditional graph similarity definitions. Graph edit distance of the result graph with query graphs* 1 *and* 2 *are* 4 *and* 6*, respectively. The size of the maximum common subgraph is* 3 *in both cases. Nevertheless, 'Titanic' is the correct answer of the query; and hence, the result graph* should *be considered a good match for both the query graphs using some novel graph similarity metric.*

This motivates us to investigate fast subgraph matching techniques suitable for query answering, which can *relax* the rigid structural and label matching constraints of subgraph isomorphism and other traditional graph similarity measures. Our proposed graph similarity metric is based on the following observations: (a)

	NeMa	BLINKS	IsoRank	SAGA	NESS [1]	gStore
Precision (Node)	**0.91**	0.52	0.63	0.75	Filter: 0.17 Filter+Verify: 0.80	0.59
Recall (Node)	**0.91**	0.52	0.63	0.75	Filter: 0.83 Filter+Verify: 0.80	0.59
Precision (Graph)	**0.88**	0.50	0.40	0.69	Filter: 0.39 Filter+Verify: 0.74	0.55
Recall (Graph)	**0.88**	0.50	0.40	0.69	Filter: 0.75 Filter+Verify: 0.74	0.55
Top-1 Match Finding Time (sec)	0.97	1.92	4882.0	15.95	Filter: 0.59 Filter+Verify: 56.16	**0.92**

Table 1: NeMa vs. Keyword Search and Graph Querying Methods: The query graphs are extracted from the *IMDB* graph, and later modified by adding 30% structural noise and 50% label noise. We determine the top-1 match for each query graph using various methods, and measure effectiveness at the level of (a) query nodes, and (b) query graphs. At the node level, precision is defined as the ratio of correctly discovered node matches over all discovered node matches, while recall is measured as the ratio of correctly discovered node matches over all correct node matches. Similarly, at the graph level, precision is defined as the ratio of correctly discovered graph matches over all discovered graph matches, and recall is measured as the ratio of correctly discovered graph matches over all correct graph matches. A graph match is considered correct if at least 70% of its nodes are matched correctly. Since we consider only the top-1 match, precision and recall have the same value. In addition, we also report precision and recall of NESS filtering phase. For details about the query graphs, noise, and evaluation metrics, see Section 7.

if two nodes are close in a query graph, the corresponding nodes in the result graph must also be close. However, (b) there may be some differences in labels of the matched nodes.

While the need for such a graph similarity metric is evident (*e.g.,*SAGA [35], IsoRank [33]), there is little work on subgraph matching in large networks considering both the criteria. Recently, NESS [20] is proposed for subgraph matching that considers the proximity among nodes, but resorts to strict node label matching. The NESS algorithm is based on a *filtering-and-verification* approach. In the filtering phase, the less promising candidate nodes are pruned iteratively, until no more candidates can be pruned. The output of the filtering phase is a limited number of final candidates for each query node. Then, it verifies all possible graph matches formed by these final candidates, in order to find the top-k graph matches. One can modify NESS to leverage for node label differences. However, this modification reduces the effectiveness of its filtering phase, and results in a large number of final candidates for each query node (See Appendix for an example). Indeed, in our experiments, we find a very low precision score for NESS, at the end of its filtering phase (Table 1). Hence, it becomes quite expensive to determine the top-k graph matches from these large number of final candidates. In contrast, our proposed NeMa framework employs an inference algorithm that iteratively boosts the score of more promising candidate nodes, considering both label and structural similarity; and thereby directly finds the top-k graph matches.

Contributions. In this paper, we propose NeMa, a novel subgraph matching framework for querying heterogeneous networks.
(1) We define the query result as the match of a given query graph in a target graph, in terms of a notion of homomorphism-based subgraph matching. To measure the quality of the matches, we further define a novel subgraph matching cost metric between the query graph and its matches (Section 3). In contrast to strict subgraph isomorphism, our proposed metric aggregates the costs of matching individual query nodes, which in turn depends on the cost of matching node labels and their neighborhoods within a certain hops.
(2) Based on the cost metric, we propose the minimum cost subgraph matching problem (Section 4), which is to identify the matches of the query graph with minimum costs in the target graph. We show that the problem is NP-hard and also hard to approximate.
(3) We propose a heuristic method for the minimum cost subgraph matching problem (Section 5). In a nutshell, NeMa converts the underlying graph homomorphism problem into an equivalent inference problem in graphical models [29], and thereby allows us to apply an inference algorithm to heuristically identify the optimal matches. Our method avoids costly subgraph isomorphism and edit distance computations. We further propose indexing and optimization techniques for the problem in Section 6.
(4) We empirically verify the effectiveness and efficiency of NeMa. Our experimental results on real-world networks in Section 7 show that NeMa finds better quality results quickly as compared to keyword search (e.g. BLINKS [16]) and various graph querying techniques (*e.g.,*IsoRank [33], SAGA [35], NESS [20], gStore [43]).

2. RELATED WORK

Subgraph Matching. Ullmann's backtracking method [38], VF2 [9], SwiftIndex [32] are used for subgraph isomorphism checking.

The subgraph matching problem identifies all the occurrences of a query graph in the target network. In bioinformatics, exact and approximate *subgraph matching* have been extensively studied, e.g., PathBlast [19], SAGA [35], NetAlign [22], IsoRank [33]. Among them, SAGA is close to ours in terms of problem formulation. However, these algorithms target smaller biological networks. It is difficult to apply them in large heterogeneous networks.

There have been significant studies on inexact subgraph matching in large graphs. Tong et al. [37] proposed the best-effort pattern matching, which aims to maintain the shape of the query. In contrast, we identify the optimal matches in terms of proximity among entities rather than the shape of the query graph. Tian et al. [36] proposed an approximate subgraph matching tool, called TALE, with efficient indexing. Mongiovi et. al. introduced a set-cover-based inexact subgraph matching technique, called SIGMA [26]. Both the techniques use edge misses to measure the quality of a match; and therefore, cannot incorporate the notion of proximity among entities. There are other works on inexact subgraph matching. An incomplete list (see [13] for surveys) includes homomorphism based subgraph matching [12], belief propagation based net alignment [4], edge-edit-distance based subgraph indexing technique [41], subgraph matching in billion node graphs [34], regular expression based graph pattern matching [3], schema [25] and unbalanced ontology matching [42]. Among them, homomorphism based subgraph matching [12] is close to ours. However, instead of identifying the top-k matches, the paper reports all the subgraphs

[1] In this paper, all experimental results with NESS correspond to its modified version, where we allow two nodes to be matched if their label difference is within a predefined threshold.

where the query edges can be mapped to paths of a given maximum length and the label differences are within a certain threshold.

There are several works on simulation and bisimulation-based graph pattern matching, *e.g.,* [11, 23], which define subgraph matching as a *relation* among the query nodes and target nodes. Compared to them, NeMa, is more strict, since we define subgraph matching as a *function* from the query nodes to target nodes.

Label and Concept Propagation. Label propagation has been widely used in semi-supervised learning, *e.g.,* labeling of unlabeled graph nodes [31]. Concept Propagation /Concept Vector, on the other hand, was originally formulated to measure the semantic similarities between terms/concepts in a taxonomy [21]. We note that the spreading activation theory of memory [2] used a similar idea of activation propagation. CP/CV and spreading activation have been effectively used in [7, 20] for approximate structural matching in trees, graphs and also for information retrieval from associated networks [5]. These works consider only strict node label matching. However, subgraph matching without node labels is a harder problem than subgraph matching with node labels [40]. Therefore, instead of strict node label equality, when one allows approximate node label matching (*e.g.,* in our current work), it significantly increases the complexity of the search problem.

Querying Semi-structured Data. Lorel and UnQl are among the preliminary query languages designed for semi-structured data. Both of them model the input data as labeled graphs, while permitting the users to write queries without detailed knowledge about the schema. Later, the underlying query processing systems convert those queries into standard SQL or structural recursion queries, respectively, for retrieving the answers. This idea of query rewriting has been explored in the context of both relational and semi-structured data, *e.g.,* [10, 28, 39, 15]. Observe that such query rewriting techniques alleviate the users from the complexity of understanding the schema; nevertheless, the underlying query processing system still requires a fixed schema.

In the realm of RDF, the SPARQL is widely used as the query processing language. However, writing of a SPARQL query is often too challenging, because it requires the exact knowledge of structure, node labels and types. gStore [43], which is the first study that considers a subgraph matching-based query answering technique in RDF data, allows approximate node label matching, but adheres to strict structural matches. In contrast, our NeMa framework permits both structural and node label mismatches.

Our work is different from the keyword search in graphs [16, 18], as our queries have both *structure* and keywords (node labels).

3. PRELIMINARIES

We start with a few definitions.

3.1 Target Graphs, Queries and Matching

Target graph. A target graph that represents a heterogeneous network dataset can be defined as a labeled, undirected graph $G = (V, E, L)$, with the node set V, edge set E, and a label function L, where (1) each *target node* $u \in V$ represents an entity in the network, (2) each edge $e \in E$ denotes the relationship between two entities, and (3) L is a function that assigns to each node u a label $L(u)$ from a finite alphabet. In practice, the node labels may represent the attributes of the entities, *e.g.,* name, value, etc.

Query graph. A query graph $Q = (V_Q, E_Q, L_Q)$ is an undirected, labeled graph, with a set of *query nodes* V_Q, a set of query edges E_Q, and a label function L_Q, which assigns to each query node $v \in V_Q$ a label $L_Q(v)$ from a finite alphabet.

$Q(V_Q, E_Q, L_Q)$	query graph
$G(V, E, L)$	target graph
$\phi : V_Q \rightarrow V$	subgraph matching function
Δ_L	label difference function
$\mathbb{M}(v)$	candidate set of node v
$\mathbb{R}_G(u)$	neighborhood vector of node u
$N_\phi(v, u)$	neighborhood matching cost between v and u
$F_\phi(v, u)$	individual node matching cost between v and u
$C(\phi)$	subgraph matching cost function

Table 2: Notations: Target Graphs, Queries and Subgraph Matching

We next define the *subgraph matching* of a (connected) query graph in a large target network.

Given a target graph $G = (V, E, L)$ and a query graph $Q = (V_Q, E_Q, L_Q)$, (1) a node $u \in V$ is a *candidate* for a query node $v \in V_Q$ if the difference in their labels (*i.e.,* $L(u)$ and $L_Q(v)$, respectively), determined by a given (polynomial-time computable) *label difference function* Δ_L, is less than or equal to a predefined threshold ϵ. We denote as $\mathbb{M}(v)$ the candidate set of the query node v. (2) a *subgraph matching* is a many-to-one function $\phi : V_Q \rightarrow V$, such that, for each query node $v \in V_Q$, $\phi(v) \in \mathbb{M}(v)$.

Remarks. **(1)** The label difference function Δ_L between two node labels can be defined by a variety of criteria, such as the Jaccard similarity, string edit distance, or more sophisticated semantic metrics, *e.g.,* ontology similarity [10]. In this work, we use Jaccard similarity measure to determine Δ_L (Section 7). **(2)** In contrast to strict one-to-one mapping as in traditional subgraph isomorphism tests, we consider a more general many-to-one subgraph matching function. Indeed, two query nodes may have the same match [12, 30]. **(3)** In practice, the nodes in the target and query graphs may be annotated with types (*e.g.,* Figure 1 and [15]), where a query node can only be matched with target nodes having the same type. In such cases, our subgraph matching model can be easily adapted to capture the type constraints by refining candidate sets.

3.2 Subgraph Matching Cost Function

There can be many valid matching functions for a given query graph and a target graph [13]. As stated earlier, our novel graph similarity metric must preserve the proximity among node pairs in the query graph, while the labels of the matched nodes should also be similar. Taking this as our guideline, we introduce the *subgraph matching cost function* in NeMa as a metric to measure the goodness of a matching. The function adds up the costs of matching a query node with its candidate, thereby capturing the difference between labels and neighborhood structures of the two nodes. We first introduce the notion of a neighborhood vector.

Neighborhood vectorization. Given a node u in the target graph G, we represent the neighborhood of u with a *neighborhood vector* $\mathbb{R}_G(u) = \{\langle u', P_G(u, u')\rangle\}$, where u' is a node within h-hops of u, and $P_G(u, u')$ denotes the *proximity* of u' from u in G.

$$P_G(u, u') = \begin{cases} \alpha^{d(u,u')} & \text{if } d(u, u') \leq h; \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Here, $d(u, u')$ is the distance between u and u'. The *propagation factor* α is a parameter between 0 and 1; and $h > 0$ is the hop number (effectively, the radius) of the neighborhood for vectorization. The neighborhood vector of node u encodes the proximity information from u to its h-hop neighbors. It often suffices to consider small values of h (*e.g.,* $h = 2$), since the relationship between two entities becomes irrelevant as their social distance increases [6].

Based on neighborhood vectors, we now proceed to model the matching cost of the neighborhoods of a query node and a target node. Let us denote the set of neighboring nodes within h-hops of v

as $\mathbb{N}(v)$. Given a matching function ϕ, the neighborhood matching cost between v and $u = \phi(v)$, denoted by $N_\phi(v, u)$, is defined as:

$$N_\phi(v,u) = \frac{\sum_{v' \in \mathbb{N}(v)} \Delta_+ \left(P_Q(v,v'), P_G\left(u, \phi(v')\right)\right)}{\sum_{v' \in \mathbb{N}(v)} P_Q(v,v')} \qquad (2)$$

where $\Delta_+(x, y)$ is a function defined as

$$\Delta_+(x,y) = \begin{cases} x - y, & \text{if } x > y; \\ 0 & \text{otherwise.} \end{cases} \qquad (3)$$

Intuitively, $N_\phi(v, u)$ measures the matching cost of the neighborhood vectors of v and u. Note that (i) the user issues a query based on her *vague* notion of how the entities are connected in the target graph. Hence, Δ_+ avoids penalizing the cases when two nodes are closer in the target graph, as compared to their corresponding nodes in the query graph. (ii) We normalize $N_\phi(v, u)$ over the neighborhood of v that incurs more cost when same number of node misses occurs in a smaller neighborhood.

Recall that we assume the existence of the label difference function $0 \leq \Delta_L \leq 1$. Now, the individual node matching cost for matching function ϕ is defined as a linear combination of the label difference function and the neighborhood matching cost function.

$$F_\phi(v,u) = \lambda \cdot \Delta_L\left(L_Q(v), L(u)\right) + (1 - \lambda) \cdot N_\phi(v,u), \qquad (4)$$

where $u = \phi(v)$.

This node matching cost combines *both* label matching cost and neighborhood matching cost via a parameter $0 < \lambda < 1$, whose optimal value lies between $0.3 \sim 0.5$ empirically (Section 7).

We are now ready to define our subgraph matching cost function. Given a matching ϕ from the query nodes $v \in V_Q$ to target nodes $\phi(v) \in V$, the subgraph matching cost function is defined as:

$$C(\phi) = \sum_{v \in V_Q} F_\phi(v, \phi(v)) \qquad (5)$$

Intuitively, $C(\phi)$ is the matching cost of ϕ between the query graph Q and the target graph G, and the problem is to find a matching function ϕ that minimizes $C(\phi)$. Note that, assuming Δ_L is non-negative, $F_\phi(v, \phi(v))$ and therefore, $C(\phi)$ are both non-negative, so the minimum value that $C(\phi)$ can take is 0.

3.3 Cost Function Properties

The following properties of our subgraph matching cost function illustrates its connection with subgraph isomorphism.

Property 1. *If the query graph Q is subgraph isomorphic (in terms of structure and node labels equality) to the target graph G, then there exists a minimum cost matching function ϕ with $C(\phi) = 0$.*

Property 1 ensures that all the matching functions ϕ, which identifies exact (isomorphic) matches for Q, must have cost 0. However, a match ϕ of Q, where $C(\phi) = 0$, may not necessarily be isomorphic to Q. We refer to such matches as *false exact matches*.

Example 2.1. Consider a query graph Q, a target graph G (Figure 2), and a subgraph matching function ϕ, where $\phi(v_1)$=u_1, $\phi(v_2)$= $\phi(v_4)$=u_2, and $\phi(v_3)$=u_3. Assuming $h = 1$ and $\alpha = 0.5$, the neighborhood vectors in Q are: $\mathbb{R}_Q(v_1)$={$\langle v_2,\ 0.5\rangle$, $\langle v_3,\ 0.5\rangle$}, $\mathbb{R}_Q(v_2)$={$\langle v_1,\ 0.5\rangle$}, $\mathbb{R}_Q(v_3)$={$\langle v_1,\ 0.5\rangle$, $\langle v_4,\ 0.5\rangle$}, and $\mathbb{R}_Q(v_4)$= {$\langle v_3,\ 0.5\rangle$}. Similarly, we have the following neighborhood vectors in G: $\mathbb{R}_G(u_1)$ = {$\langle u_2,\ 0.5\rangle$, $\langle u_3,\ 0.5\rangle$}, $\mathbb{R}_G(u_2)$ ={$\langle u_1,\ 0.5\rangle$, $\langle u_3,\ 0.5\rangle$}, and $\mathbb{R}_G(u_3)$ = {$\langle u_1, 0.5\rangle$, $\langle u_2,\ 0.5\rangle$}. Therefore, the individual node matching costs F_ϕ is 0 for all $v \in V_Q$, and the subgraph matching cost $C(\phi)$ is 0. Observe that the match identified by ϕ is not isomorphic to Q.

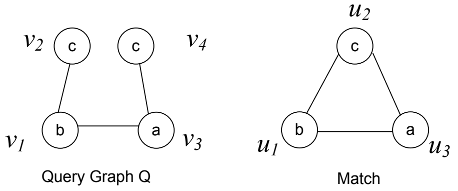


Figure 2: Example of False Exact Match in NeMa

However, if the matching function ϕ is *one-to-one*, the following property shows that the false exact matches can be avoided,.

Property 2. *If the match identified by ϕ is not isomorphic to the query graph Q, and ϕ is a one-to-one function, then $C(\phi) > 0$.*

PROOF. Since Q is connected and ϕ is a one-to-one function, if the match identified by ϕ is not isomorphic to Q, one of the following must hold. (1) There exists some node $v \in V_Q$, s.t., $\Delta_L\big(L_Q(v), L(\phi(v))\big) > 0$. Then, $C(\phi) > 0$, assuming $\lambda \neq 0$ in Eq. 4. (2) There exists an edge (v, v') in E_Q; but the corresponding edge (u, u') is not in graph G. $\phi(v) = u$ and $\phi(v') = u'$. This implies $P_Q(v, v') = \alpha$, but $P_G(u, u') < \alpha$, which in turn implies $N_\phi(v, u) > 0$. Assuming $\lambda \neq 1$ in Eq. 4, we get $C(\phi) > 0$. □

4. PROBLEM FORMULATION

The subgraph matching cost function favors matches with low matching costs. Based on the matching cost function, we introduce the minimum cost subgraph matching problem as follows.

Problem Statement 1. Minimum Cost Subgraph Matching. *Given a target graph G, a query graph Q, and the label noise threshold ϵ, find the minimum cost matching ϕ,*

$$\underset{\phi}{\operatorname{argmin}}\, C(\phi), \qquad (6)$$

$$\textit{s.t.} \qquad \Delta_L\left(L_Q(v), L(u)\right) \leq \epsilon, \forall v \in V_Q, u = \phi(v) \qquad (7)$$

Intuitively, instead of checking subgraph isomorphism, our problem formulation identifies the optimal match by minimizing node label differences as well as node pair distances. The identified matches serve as the answer to the query graph.

The problem is, however, nontrivial. The following theorem shows that the decision version of the problem is intractable, even when the subgraph matching function ϕ is not injective.

Theorem 1. *Given a target network G, a query graph Q, it is* NP-*complete to determine whether there exists a match ϕ with* NeMa *subgraph matching cost $C(\phi) = 0$.*

PROOF. The problem is NP, since there is a nondeterministic algorithm which guesses a matching function ϕ, and verifies whether its cost $C(\phi) = 0$, in polynomial time. We prove the NP-hardness by reduction from the graph homomorphism problem, which is NP-complete [8]. A homomorphism from a graph Q' to a graph G' (both unlabeled) is a function that preserves node adjacency (i.e., each edge in Q' is mapped to an edge in G'). Given an instance of the graph homomorphism problem, we construct an instance of the minimum cost subgraph matching problem, where all nodes in the target graph G and query graph Q have identical labels. We also assume, *w.l.o.g.*, that the depth of vectorization $h = 1$. One may verify that if there exists a homomorphism ϕ' from Q' to G', then there exists a corresponding matching ϕ from Q to G, s.t. $C(\phi) = 0$. Conversely, if ϕ' is not a homomorphic matching, then there exists an edge (v, v') in E_Q, but the corresponding edge $(\phi(v), \phi(v'))$ is not in G. Hence, $C(\phi) > 0$ ($\lambda \neq 1$ in Eq. 4). Therefore, there exists a matching function ϕ from Q to G, where $C(\phi) = 0$, if and only if there is a homomorphic matching ϕ' from Q' to G'. This completes the proof. □

One may want to find a polynomial time approximation algorithm. However, the problem is also hard to approximate.

Theorem 2. *The minimum cost subgraph matching is APX-hard.*

PROOF. We show that this optimization problem is APX-hard by performing a reduction (f, g) from the Maximum Graph Homomorphism (MGH) problem without self loops, which is APX-hard [27]. An MGH problem identifies a matching which maximizes the number of edges of the query graph Q that can be mapped to edges of the target graph G (both unlabeled). Given an instance I of MGH, we construct an instance I' of the minimum cost subgraph matching problem, where all nodes in the target network G and query graph Q have identical labels. Let n_q and e_q be the total number of nodes and edges, respectively, in Q. *w.l.o.g.*, assume the depth of vectorization $h = 1$, and the proportionality constant $\lambda = 1 - \frac{1}{n_q}$. We denote by OPT$(I)$ the value of the optimal solution of problem instance I, and VAL(I, x) the value of a feasible solution x of the problem instance I. Assume OPT$(I) = e_o$ and VAL$(I, x) = e$ for some feasible solution x of instance I. Clearly, $e_o \geq 1$. Hence, (1) OPT$(I') < 1 \leq e_o =$ OPT(I). Also, given some feasible solution y of instance I', one may verify that $|\text{OPT}(I) - \text{VAL}\,(I, g(y))\,| = e_o - e$, and $|\text{OPT}(I') - \text{VAL}(I', y)| \geq \frac{e_o - e}{2n_q e_q}$. Therefore, (2) $|\text{OPT}(I) - \text{VAL}\,(I, g(y))\,| \leq 2n_q e_q |\text{OPT}(I') - \text{VAL}(I', y)|$. Thus, there exists a reduction (f, g) from MGH to the minimum cost subgraph matching problem, and the theorem follows. □

5. QUERY PROCESSING ALGORITHM

In this section, we propose a heuristic solution to identify the minimum cost matchings. We start by introducing the max-sum inference problem in graphical models [29], and show how our graph homomorphism problem underlying the NeMa framework is equivalent to an inference problem in graphical models.

Max-Sum Inference. In graphical models, the joint probability distribution function $p(X)$ of a set of variables $X = \{x_1, x_2, \ldots, x_M\}$ can be expressed as a product of the form $p(X) = \prod_i f_i(X_i)$, where each $X_i \subseteq X$. Alternatively, $\log p(X) = \sum_i \log f_i(X_i)$. The *Max-Sum* inference problem is to find the values of the variables $x_1, x_2, \ldots, x_M$ that result in maximum $p(X)$. In other words, we would like to maximize $\log p(X)$ that can be decomposed as the sum of several functions of the form $\log f_i(X_i)$, each of which depends only on a subset of the original variables.

The objective of the max-sum inference problem is similar to that of the minimum cost subgraph matching problem, which is to *minimize* the overall subgraph matching cost $C(\phi)$. Recall that (1) $C(\phi)$ is an aggregation of the individual node matching costs $F_\phi(v, \phi(v))$ of all query nodes v, and (2) the individual node matching cost of a query node v depends only on the matches of v and its neighbors in $\mathbb{N}(v)$. In light of this, we propose an *iterative inference* algorithm similar to the loopy belief propagation algorithm [29], used for inferencing in graphical models.

5.1 Iterative Inference Algorithm

In this section, we introduce our inference algorithm, denoted as NemaInfer and illustrated in Figure 3.

Overview. Given a query graph Q and a target graph G, NemaInfer first computes the candidate set for each query node using the node label similarity function Δ_L (line 1). Next, it initializes an *inference cost* $U_0(v, u)$ by assigning it to the minimum possible value of the individual node matching costs $F_\phi(v, u)$, over all possible matching functions ϕ, s.t., $\phi(v) = u$ (line 2-3). It then *iteratively* computes an *inference cost* for each query node v and its candidates, and selects the *optimal match* of v as its candidate u with the minimum inference cost. NemaInfer keeps track of the optimal matches for each query node. The procedure repeats until it reaches a fixpoint, where the optimal matches for more than a threshold number of query nodes remain identical in two successive iterations (lines 4-12). Finally, NemaInfer refines the matches of each query node and its neighborhood that it "memorizes" via a memoization technique, and obtains the best match (line 13). The constructed subgraph match $\mathbf{\Phi}$ is then returned (line 14).

```
Algorithm NemaInfer
Input: Target graph $G(V, E, L)$, Query Graph $Q(V_Q, E_Q, L_Q)$.
Output: Minimum cost matching of $Q$ in $G$.
1.  for each node $v \in V_Q$ do compute $\mathbb{M}(v)$;
2.    $i := 0$; flag := true;
3.    Initiate iterative inferencing with Eq. 8;
4.    while flag do
5.      $i := i + 1$;
6.      for each $v \in V_Q$ do
7.        for each $u \in \mathbb{M}(v)$ do
8.          compute $U_i(v, u)$ with Theorem 3;
9.          keep track of the current matches of neighbors $v' \in \mathbb{N}(v)$;
10.       compute optimal match $O_i(v)$ using Eq. 10;
11.     if more than a threshold number of
          query nodes $v$ satisfy $O_i(v) = O_{i-1}(v)$ then
12.       flag := false;
13.   construct $\mathbf{\Phi}$ for all $v \in V_Q$ (with Eq. 11, 12).
14.   return $\mathbf{\Phi}$;
```

Figure 3: Iterative Inference Algorithm NemaInfer

We next introduce several procedures of NemaInfer in detail.

Inference cost and optimal match (lines 3-12). The algorithm NemaInfer improves the quality of the matching in each iteration, based on the notion of an *inference cost* and the *optimal match*.

Inference cost. At each iteration i of NemaInfer, the inference cost $U_i(v, u)$ for each $v \in V_Q$ and $u \in \mathbb{M}(v)$ is defined as follows.

$$U_0(v, u) = \min_{\{\phi : \phi(v) = u\}} F_\phi(v, u) \qquad (8)$$

$$U_i(v, u) = \min_{\{\phi : \phi(v) = u\}} \big[F_\phi(v, u) + \sum_{v' \in \mathbb{N}(v)} U_{i-1}\left(v', u'\right)\big] \qquad (9)$$

We assume $i > 0$, and $u' = \phi(v')$ in Equation 9. Intuitively, the inference cost is the minimum sum of the individual node matching cost $F_\phi(v, u)$ and the previous iteration's inference costs $U_{i-1}\big(v', \phi(v')\big)$ for all neighbors v' of v, over all possible matching functions ϕ, with the constraint $\phi(v) = u$.

Note that although we consider the minimization over all possible matching functions ϕ, s.t., $\phi(v) = u$, in Equation 9, it only depends on the matches of the neighboring nodes in $\mathbb{N}(v)$. As discussed later, inference costs can be computed in polynomial time.

Optimal match. In every iteration, we also define the *optimal match* of each query node. The optimal match of a query node v at iteration i, denoted by $O_i(v)$, is defined as follows.

$$O_i(v) = \operatorname*{argmin}_{u \in \mathbb{M}(v)} U_i(v, u); \quad i \geq 0 \qquad (10)$$

Example 4.1. We illustrate the idea of one iteration of NemaInfer using Figure 4. Assume we have already determined the candidate matches $\mathbb{M}(v)$ for every query node v using the label similarity function Δ_L. For example, $\mathbb{M}(v_2) = \{u_2, u_5, u_9\}$ and

$\mathbb{M}(v_4) = \{u_7, u_{10}\}$ in Figure 4. Also, consider $h = 1$. At $i = 0$, $U_0(v_2, u_5) = U_0(v_2, u_9) = 0$. Therefore, we can not distinguish between u_5 and u_9 in the initialization round, as which one is a better match of v_2. However, observe that $U_0(v_4, u_{10}) < U_0(v_4, u_7)$. u_{10} is a neighbor of u_9, while u_7 a neighbor of u_5. Thus, it not only influences the optimal match $O_0(v_4)$ of v_4 at iteration $i = 0$, but it also makes $U_1(v_2, u_9) < U_1(v_2, u_5)$ at iteration $i = 1$, via Eq. 9. Hence, we improve the matches in each iteration and proceed towards the minimum cost (heuristic) subgraph match.

Invariant. The algorithm NemaInfer posses the following invariant in each of its iteration, which illustrates the connection between the inference cost and the subgraph matching cost (Section 3).

Invariant 1. *If there exists a matching function ϕ from the nodes of Q to the nodes of G, such that, $C(\phi) = 0$, then $U_i(v, \phi(v)) = 0$ for all $v \in V_Q$ and $i \geq 0$.*

However, the converse is not always true. In fact, based on the properties of the loopy belief propagation algorithm, there is no guarantee that our algorithm will converge for *all* query nodes after a certain number of iterations. Therefore, we terminate the procedure when more than a threshold number of query nodes v satisfy the condition $O_i(v) = O_{i-1}(v)$. We empirically verified in Section 7 that our method usually requires about 2 to 3 iterations to terminate — around 95% of query nodes converge using *IMDB* dataset, and also performs well in real-life networks.

Matching refinement (line 13). The optimal match of each query node at the final iteration might not correspond to the subgraph matching function with the minimum (heuristic) aggregate cost [29]. This can happen if there are multiple graph matching functions that result in the minimum cost graph matches. Therefore, we need to refine the optimal node matches from the final round of NemaInfer to identify one such minimum cost subgraph matching function, say $\mathbf{\Phi}$. We refer to the matches of the query nodes corresponding to $\mathbf{\Phi}$ as the *most probable matches*. To find these most probable matches, the standard *memoization* technique can be used after the termination of our iterative inference algorithm. First, a query node, say v, is selected randomly, and its most probable match, denoted by $\mathbf{\Phi}(v) \in \mathbb{M}(v)$, is determined as follows:

$$\mathbf{\Phi}(v) = \underset{u \in \mathbb{M}(v)}{\operatorname{argmin}} U_{i'}(v, u) \tag{11}$$

In Eq. 11, $i = i'$ denotes the final iteration. For the remaining nodes, the most probable matches are determined by memoizing recursively, i.e., we keep track of the matches of the neighboring nodes that give rise to the most probable match of the current node. For example, the most probable matches $\mathbf{\Phi}(v')$ of all $v' \in \mathbb{N}(v)$ are obtained using the most probable match of v as follows.

$$\phi_p = \underset{\{\phi : \phi(v) = \mathbf{\Phi}(v)\}}{\operatorname{argmin}} \Big[F_\phi(v, \mathbf{\Phi}(v)) + \sum_{v' \in \mathbb{N}(v)} U_{i'-1}\left(v', \phi(v')\right) \Big]$$

$$\mathbf{\Phi}(v') = \phi_p(v') \tag{12}$$

The aforementioned memoization technique is performed until the most probable matches of all query nodes are computed.

Computation of Inference Costs. A straightforward approach to determine the inference cost $U_i(v, u)$ for a query node v and its candidate u (Eq. 9) considers all possible combination of matches for all nodes $v' \in \mathbb{N}(v)$, which has exponential time complexity and might be very expensive. In this section, we propose a technique to compute the inference cost in *polynomial time*.

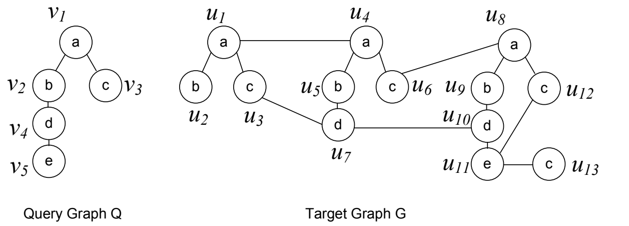


Figure 4: Optimal Subgraph Match Finding Algorithm

Partial inference cost. To evaluate the inference cost $U_i(v, u)$ for a query node v and its candidate u at iteration i of the algorithm NemaInfer, we compute a *partial inference cost* for each node $v' \in \mathbb{N}(v)$, which is denoted by $W_i(v, u, v')$, and defined below.

$$W_i(v, u, v') = \min_{\{\phi : \phi(v) = u\}} \Big[\beta(v) \cdot \Delta_+\left(P_Q(v, v'), P_G\left(u, \phi(v')\right)\right) + U_{i-1}\left(v', \phi(v')\right) \Big] \tag{13}$$

Here, $\beta(v) = [\sum_{v' \in \mathbb{N}(v)} P_Q(v, v')]^{-1}$. To compute $W_i(v, u, v')$, we only need to find the minimum value in Eq. 13 over the candidates in $\mathbb{M}(v')$. Hence, the partial inference cost $W_i(v, u, v')$ can be computed in polynomial time, for each triplet v, u, v', where $u \in \mathbb{M}(v)$ and $v' \in \mathbb{N}(v)$. Next, we show the relation between the partial inference cost and the inference cost.

Theorem 3. *The inference cost $U_i(v, u)$ is computable in polynomial time via the following formula:*

$$U_i(v, u) = \Delta_L\big(L_Q(v), L(u)\big) + \sum_{v' \in \mathbb{N}(v)} W_i(v, u, v') \tag{14}$$

PROOF.

$$\begin{aligned}
&U_i(v, u) \\
&= \min_{\{\phi:\phi(v)=u\}} \Big[\underbrace{\Delta_L\big(L_Q(v), L(u)\big) + \beta(v) \cdot \sum_{v' \in \mathbb{N}(v)} \Delta_+(P_Q(v, v'), P_G(u, \phi(v')))}_{F_\phi(v,u)} \\
&\qquad + \sum_{v' \in \mathbb{N}(v)} U_{i-1}(v', \phi(v')) \Big] \\
&= \min_{\{\phi:\phi(v)=u\}} \sum_{v' \in \mathbb{N}(v)} \Big[\beta(v) \cdot \Delta_+(P_Q(v, v'), P_G(u, \phi(v'))) + U_{i-1}(v', \phi(v')) \Big] \\
&\qquad + \Delta_L\big(L_Q(v), L(u)\big) \\
&= \sum_{v' \in \mathbb{N}(v)} \underbrace{\min_{\{\phi:\phi(v)=u\}} \Big[\beta(v) \cdot \Delta_+(P_Q(v, v'), P_G(u, \phi(v'))) + U_{i-1}(v', \phi(v')) \Big]}_{W_i(v,u,v')} \\
&\qquad + \Delta_L\big(L_Q(v), L(u)\big) \\
&= \Delta_L\left(L_Q(v), L(u)\right) + \sum_{v' \in \mathbb{N}(v)} W_i(v, u, v')
\end{aligned}$$

Hence, the theorem. □

It follows from Theorem 3 that the inference cost $U_i(v, u)$ of nodes v and u can be efficiently computed in polynomial time, by (1) determining the partial inference cost $W_i(v, u, v')$ for each $v' \in \mathbb{N}(v)$, and (2) aggregating these partial inference costs with Δ_L $\big(L_Q(v), L(u)\big)$. The aforementioned technique also keeps track of which matches of the neighboring query nodes give rise to the most probable match of the current query node. This information is required during matching refinement.

Time complexity. We analyze the time complexity of the algorithm NemaInfer. Let us denote the number of nodes in the target graph G and the query graph Q as $|V|$ and $|V_Q|$, respectively. (1) It requires $O(|V_Q| \cdot |V|)$ time to identify $\mathbb{M}(v)$ for each query node $v \in V_Q$ (line 1). (2) We denote the maximum number of candidates per query node as m_Q, and the maximum number of h-hop neighbors of each query node as d_Q. The computation of the optimal match $O_t(v)$ per query node v has time complexity $O(m_Q \cdot d_Q)$ following Theorem 3 (line 10). Therefore, the time required for each iteration of NemaInfer is $O(|V_Q| \cdot m_Q \cdot d_Q)$. If there are total I iterations, the overall time complexity is given by $O(|V_Q| \cdot |V| + I \cdot |V_Q| \cdot m_Q \cdot d_Q)$. Observe that $|V_Q|$, I, $|d_Q|$ and m_Q are typically small. Indeed, as verified in our experiments (Section 7), I is typically less than 4 and m_Q is 35, for query graphs with 5 nodes and real life graphs containing 12M nodes.

5.2 Generalized Queries

In this section, we extend NemaInfer for three generalized cases, namely, *Top-K matches*, *unlabeled queries*, and *labeled edges*.

Top-k Matches. In many applications, the query graph is not subgraph isomorphic to the target network; and hence, we are interested in identifying the top-k matches rather than only the best match. Given the target network G and the query graph Q, the *top-k subgraph matching* problem is to identify the top-k matches for a *selected query node* $v \in V_Q$.

The algorithm NemaInfer can be readily adapted for this problem. (1) The algorithm computes the inference costs and terminates at line 12. (2) We identify the top-k most probable matches of v (Eq. 11). (3) For each of these top-k most probable matches of v, we apply the recursive memoizing technique (Eq. 12) to determine the corresponding most probable matches for other query nodes.

Matching Query with Unlabeled Nodes. A query graph may have nodes with unknown labels, *e.g.,* query graphs constructed from RDF queries. NeMa can be adapted to evaluate such queries. First, we identify all the nodes from the target network that can be matched with some labeled query node based on label similarity. Next, we find the subgraph induced by all those matched nodes from the target network along with their neighbors within h-hops. All nodes in this subgraph are considered as the candidates for the unlabeled query nodes. The algorithm NemaInfer is then invoked to identify the matches. In addition, if the unlabeled query nodes contain type information, the candidate sets can further be refined.

NeMa **with Edge Labels.** The NeMa cost function can be adapted to consider the edge labels. Specifically, we concatenate the edge labels along the shortest path between a pair of nodes, and then update the neighborhood matching cost (Equation 2) as follows.

$$N_\phi(v,u) = \frac{\sum\limits_{v' \in \mathbb{N}(v)} [\Delta_+ \left(P_Q(v,v'), P_G\left(u,u'\right)\right)) + \Delta_L(s(v,v'), s(u,u')))]}{\sum\limits_{v' \in \mathbb{N}(v)} [P_Q(v,v') + 1]}$$

Here, $u = \phi(v)$, $u' = \phi(v')$, and $s(v,v')$ concatenates the edge labels along the shortest path between v and v'. Since, we consider the edge labels along the shortest path between a pair of nodes, the asymptotical time complexity of NeMaInfer remains the same.

6. INDEXING AND OPTIMIZATION

In this section, we discuss indexing and optimization techniques to improve the efficiency of our network matching algorithm.

6.1 Candidate Selection

The candidate set of a query node is defined in terms of the label similarity function (see Section 3), which may include candidate nodes that do not match the query node due to neighborhood mismatch. We introduce optimization techniques to efficiently filter such candidate nodes as much as possible, and thereby improving the performance of our inference algorithm NemaInfer. We first introduce the notion of *isolated candidates*.

Isolated Candidates. Given a query node v and its candidate set $\mathbb{M}(v)$, a node $u \in \mathbb{M}(v)$ is an *isolated candidate* of v, if

$$\{u' : u' \in \mathbb{M}(v'), v' \in \mathbb{N}(v)\} \cap \{u'' : u'' \in \mathbb{N}(u)\} = \emptyset \quad (15)$$

Intuitively, the node u is an isolated candidate of a query node v if none of the candidates within h-hop neighbors of v are in the h-hop neighborhood of u; otherwise, it is a non-isolated candidate of v. Thus, an isolated candidate u of v can not be matched with v.

To efficiently find the non-isolated candidates, we propose an optimization problem, based on *verification cost* and *candidate cover*.

Verification Cost. The verification cost associated with a query node v is defined as the time complexity to verify all nodes in its candidate set $\mathbb{M}(v)$, whether they are non-isolated candidates. Note that the complexity of verifying whether some node $u \in \mathbb{M}(v)$ is a non-isolated candidate is $O\big(|\mathbb{N}(u)| + \sum_{v' \in \mathbb{N}(v)} |\mathbb{M}\left(v'\right)|\big)$.

Candidate Cover. There exists dependencies between two non-isolated candidates: if u is a non-isolated candidate of v, then there must exist a node $u' \in \mathbb{N}(u)$, such that, $u' \in \mathbb{M}(v')$ for some $v' \in \mathbb{N}(v)$. Clearly, u' is a non-isolated candidate of v'. If we verified all candidates $\{u' : u' \in \mathbb{M}(v'), v' \in \mathbb{N}(v)\}$, there is no need to verify the candidates in $\mathbb{M}(v)$ again. Thus, one may reduce redundant verifications using a notion of candidate cover.

We define *candidate cover* $\mathbb{C}(Q)$ as a set of query nodes v, such that, for all $v' \in V_Q$, either $v' \in \mathbb{C}(Q)$, or $v' \in \mathbb{N}(v)$.

$$\mathbb{C}(Q) = \{v : \forall v'(v' \in \mathbb{C}(Q) \vee v' \in \mathbb{N}(v))\} \quad (16)$$

All non-isolated candidate nodes can be identified by verifying *only* the query nodes in $\mathbb{C}(Q)$. We define the verification cost of a candidate cover as the sum of the verification costs of its constituent query nodes. Next, we introduce the candidate cover problem.

Problem Statement 2. Candidate Cover. *Given a query graph Q, find the candidate cover with the minimum verification cost.*

The following result shows that the candidate cover problem is intractable, but approximable within a factor 2 in polynomial time.

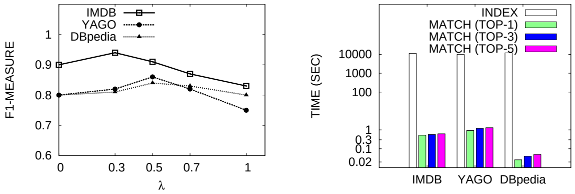
Theorem 4. *The candidate cover problem is (1)* NP*-hard, and (2) 2-approximable.*

PROOF. We show that this problem is NP-hard by reduction from NP-complete weighted minimum vertex cover problem [8]. Given a decision version of the weighted minimum vertex cover problem, we construct an instance of the candidate cover problem, where the vertex weights are considered as the corresponding verification costs. Assuming $h = 1$, the minimum weighted vertex cover will be our candidate cover. One can apply linear programming to solve this problem with 2-approximation [8]. □

6.2 Indexing

We introduce indexing technique to improve the efficiency of the inference algorithm. (1) During the off-line indexing phase, it computes the neighborhood vectors $\mathbb{R}(u)$ for all nodes u in the target network G, and stores the vectors in the index. (2) During the on-line network match technique, if u is selected as a candidate of v, it applies Eq. 15 to verify whether u is an isolated candidate of v. If so, u is eliminated from the candidate set of v.

Our index structure has space and time complexity $O(nd_G^h)$, where $|V| = n$, $d_G =$ average node degree in G, and $h =$depth of



(a) Effectiveness (b) Efficiency

Figure 5: Query Performance

vectorization. For NeMa with edge labels (Section 5.2), the asymptotical time and space complexity of indexing remains the same, since we consider edge labels along the shortest paths.

Dynamic maintenance of the index. Our indexing methods can efficiently accommodate dynamic updates in the target network. If a node u (and the edges attached to it) is added or deleted, only the indexes of u's h-hop neighbors need to be updated. If a single edge (u, u') is added or deleted, only the $h-1$ hop neighbors of both u and u' are updated, thus reducing the redundant computation.

6.3 Optimization for top-k matching

The inference algorithm NemaInfer can be adapted to identify the top-k matches. For small values of k, it is possible to prune candidate nodes by setting a cost threshold. The cost threshold ϵ_c is initially set to a small value ϵ_0. If $U_i(v, u) > \epsilon_c$ for some $u \in \mathbb{M}(v)$ at iteration i of the inference algorithm, then u is eliminated from the candidate set $\mathbb{M}(v)$ for the subsequent iterations. After termination, if the top-k matches cannot be identified, we increase ϵ_c by a small value, and repeat the steps above. The correctness of this method is ensured by Theorem 5.

Theorem 5. *If $U_i(v, u) > \epsilon_c$ at the i-th round of inference algorithm, then for all $j > i$, $U_j(v, u) > \epsilon_c$ at the j-th round of inference algorithm.*

PROOF. It follows directly from Eq. 9 and the fact that $U_i(v, u) \geq 0$ for all $i \geq 0$, over all pairs v, u, where $u \in \mathbb{M}(v)$. □

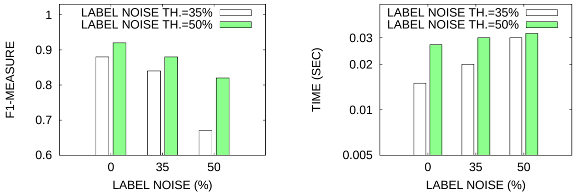
Hence, we can eliminate u from $\mathbb{M}(v)$, whenever $U_i(v, u) > \epsilon_c$ occurs for the first time at some iteration i of NemaInfer.

7. EXPERIMENTAL RESULTS

We present three sets of empirical results over three real-life datasets to evaluate (1) the effectiveness and efficiency (Section 7.2), (2) scalability (Section 7.3), and (3) optimization techniques (Section 7.4) underlying the NeMa framework.

7.1 Experimental Setup

Graph Data Sets. We used the following three real life datasets, each represents a target graph. (1) ***IMDB* Network**[3]**.** The *Internet Movie Database (IMDB)* consists of the entities of movies, TV series, actors, directors, producers, among others, as well as their relationships. (2) ***YAGO* Entity Relationship Graph**[4]**.** *YAGO* is a knowledge base with information harvested from the Wikipedia, WordNet and GeoNames. It contains about 20 million RDF triples. (3) ***DBpedia* Knowledge Base**[5]**.** *DBpedia* extracts information from the Wikipedia. We considered 22 million RDF triples from *DBpedia* article categories, infobox properties, and person data.



(a) Effectiveness (***DBpedia***) (b) Match Time (***DBpedia***)

Figure 6: Performance against Label Noise

Dataset	# Nodes	# Edges
IMDB	2,932,657	11,040,263
YAGO	12,811,149	18,282,215
DBpedia	5,177,018	20,835,327

Table 3: Dataset Sizes

The nodes in *YAGO* and *DBpedia* are annotated with labels, while the nodes in *IMDB* are annotated with both types and labels. Hence, we used the type information associated with the nodes, in addition to their labels, while querying the *IMDB* network.

Query graphs. We generated the query graphs by extracting subgraphs from the target graphs, and then introduced *noise* to each query graph. Specifically, the query generation was controlled by:

- node number and diameter, denoted by $|V_Q|$ and D_Q, respectively, where the *query diameter* is the maximum distance between any two nodes in the query graph Q.
- *structural noise*, the ratio of the number of edge updates (random insertion and deletion of edges) in Q to the number of edges in the extracted subgraph; and
- *label noise*, measured by the Jaccard similarity between the labels of nodes in the extracted subgraph and their updated counterparts in Q, where the updated labels were obtained by inserting randomly generated words to the query node labels.

We used Jaccard similarity as the label similarity measure. Specifically, given a query node v and a target node u, the label difference $\Delta_L(L_Q(v), L(u))$ is defined as $1 - \frac{|w_v \cap w_u|}{|w_v \cup w_u|}$, where w_v and w_u are the set of words in their label $L_Q(v)$ and $L(u)$, respectively. Recall that we allowed some noise in the node labels by varying the label matching cost threshold ϵ in our matching algorithm (Problem Statement 1). A node u in the target network is considered a candidate to match with a query node v if their label difference $\Delta_L(L_Q(v), L(u))$ is less than the predefined cost threshold ϵ, referred to as the *label noise threshold*.

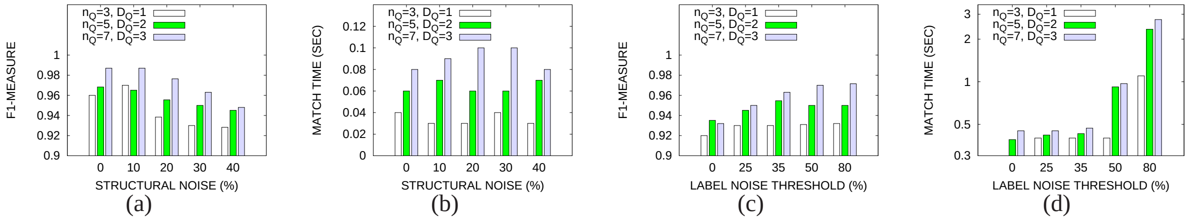
Evaluation metrics. Since the query graphs were extracted from target graphs, one already has the correct node matches. Now, the effectiveness of NeMa is measured as follows. *Precision* (P) is the ratio of the correctly discovered node matches over all discovered node matches. *Recall* (R) is the ratio of the correctly discovered node matches over all correct node matches. *F1-Measure* combines the results of precision and recall, *i.e.,*

$$F1 = \frac{2}{(1/R + 1/P)} \tag{17}$$

We considered the top-1 match to evaluate precision, recall, and F1-measure. Thus, we obtained the same values for them. However, precision and recall will be useful while analyzing NESS.

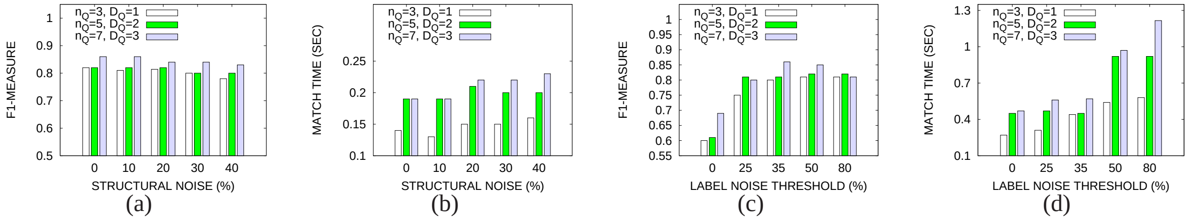
Comparing Methods. We compared NeMa with keyword search (BLINKS [16]) and various graph querying methods: SAGA [35], IsoRank [33], NESS [20], and NeMa$_{\text{gs}}$ - a variation of NeMa following gStore [43]. All these methods were implemented in C++.

[3] `http://www.imdb.com/interfaces#plain`
[4] `http://www.mpi-inf.mpg.de/yago-naga/yago/`
[5] `http://dbpedia.org/About`



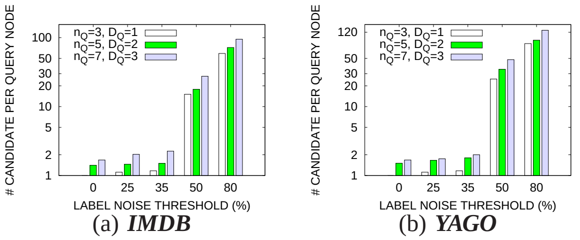
(a) (b) (c) (d)

Figure 7: Query Performance vs. Noise (***IMDB***); n_Q: Query Nodes, D_Q: Query Diameter



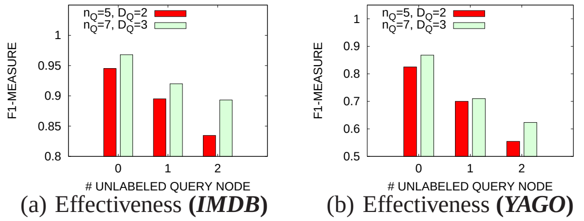
(a) (b) (c) (d)

Figure 8: Query Performance vs. Noise (***YAGO***); n_Q: Query Nodes, D_Q: Query Diameter



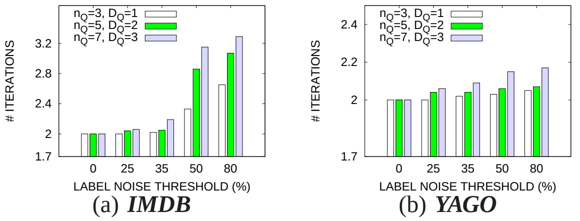
(a) ***IMDB*** (b) ***YAGO***

Figure 9: # Candidates vs. Label Noise



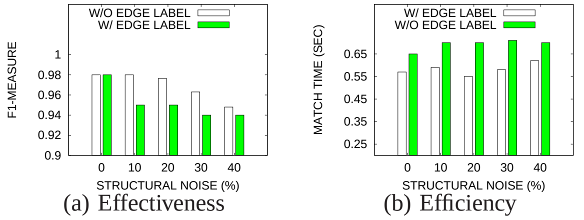
(a) Effectiveness (***IMDB***) (b) Effectiveness (***YAGO***)

Figure 11: Effectiveness with Unlabeled Query Nodes



(a) ***IMDB*** (b) ***YAGO***

Figure 10: # Iterations vs. Label Noise



(a) Effectiveness (b) Efficiency

Figure 12: Performance with Edge Labels (***IMDB***)

In our experiments, (1) propagation factor α and depth of vectorization h (Section 3) were set as 0.5 and 2, respectively [20], (2) the optimal values of the proportionality constant λ (Eq. 4) for different datasets were obtained empirically (Figure 5(a)), (3) the indexes were stored in the hard disk. All the experiments were run using a single core in a 100GB, 2.5GHz Xeon server.

7.2 Effectiveness and Efficiency

7.2.1 Performance over Real-life Data Sets

In these experiments, we evaluated the performance of of NeMa over three real-life graphs, averaged over 100 queries (Figure 5). For each target graph, we randomly generated 100 query graphs with $|V_Q|$ = 7 and D_Q = 3. We fixed the structural noise as 30%, label noise as 50%, and label noise threshold as 50%.

Figure 5(a) shows the efficiency of NeMa over various datasets, and with different values of the proportionality constant λ. For all the three datasets, our algorithm *always* correctly identifies more than 76% of the query nodes. Specifically, the F1-measure is 0.94 for *IMDB*, with $\lambda = 0.3$, even when we introduced 30% structural noise and 50% label noise. The effectiveness is higher over *IMDB* due to the type constraint posed with the query nodes. Besides, the optimal value of λ lies between $0.3 \sim 0.5$ in these datasets.

Figure 5(b) reports the efficiency of NeMa using the same setting as in Figure 5(a), including the running time of off-line index construction (`INDEX`) and online query evaluation (`MATCH`). We observed the following. (a) NeMa identifies the best match in less than 0.2 seconds, over all three datasets [6]. (b) The top-k match finding time does not vary significantly, over different k, since our inferencing method is always executed once. (c) The time required for indexing is modest (*e.g.,* 9862 sec for the *YAGO* dataset with 13M nodes and 18M edges). (d) The indexing and querying times are longer over *IMDB*, due to its higher density.

7.2.2 Performance against Noise

In this set of experiments, we investigated the impact of varying noises on the performance of NeMa. Three sets of query graphs were generated by setting (i) $|V_Q| = 3$, $D_Q = 1$, (ii) $|V_Q| = 5$, $D_Q = 2$, and (iii) $|V_Q| = 7$, $D_Q = 3$. Under each query set, 100 query graphs were generated.

Varying label noise. Fixing the structural noise as 30%, we varied the label noise from 0% to 50%, and investigated the effectiveness of NeMa, when the label noise threshold was set as 35% and 50%,

[6] our indexing and matching algorithm can be parallelized for every node. Hence, one may implement NeMa in a PREGEL [24] platform, for larger graphs.

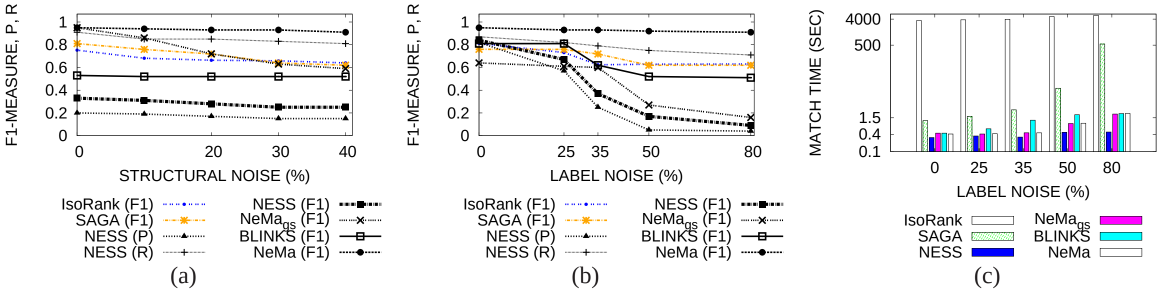


Figure 13: Comparison Results (**IMDB**): NESS, BLINKS Modified for Approximate Label Match. NESS Results Correspond to its *Filtering* Phase.

respectively. As shown in Figure 6(a) over *DBpedia*, (1) the F1-measure decreases as the label noise increases, since the candidate set of each query node may contain more candidates that are not true matches, which in turn reduces the effectiveness, (2) the F1-measure is higher when the label noise threshold is higher, since the candidate sets are more likely to include the correct matches. Observe that the F1-measure is always above 0.60.

Figure 6(b) shows that NeMa efficiency is insensitive to label noise, but more sensitive to label noise threshold. This is because it takes NeMa more time to process the larger candidate sets for the query graph as the label noise threshold increases.

Varying structural noise. Fixing the label noise threshold as 35% and label noise as 35%, we varied the structural noise from 0% to 40% in Figure 7(a) and 7(b). It can be observed that (a) both effectiveness and efficiency decrease as we increase the structural noise, and (b) with the increase of the query size, both effectiveness and efficiency increase. The reason is that (1) larger queries have more constraints in the neighborhood of a query node, which benefit the identification of correct matches, and (2) it takes longer time for NeMa to compare the matching cost for larger queries. Moreover, the F1-measure is always above 0.93, and the running time is always less than 0.1 seconds, even with 40% structural noise.

Varying label noise threshold. Fixing the structural noise as 30% and the label noise as 35%, we investigated the effect of varying the label noise threshold on the query performance.

The effectiveness and efficiency of NeMa over *IMDB* are illustrated in Figures 7(c) and 7(d), respectively. We observed the following. (1) The F1-measure initially increases while we increase the label noise threshold. This is because the query node labels are updated by adding random words. Hence, the higher is the label noise threshold, there is more chance that the correct match of a query node will be selected in its candidate set. (2) When the label noise threshold is more than 35%, the F1-measure does not improve significantly. Therefore, the optimal value of the label noise threshold can be determined empirically based on the query and target graphs. On the other hand, the running time of NeMa increases with the increase of the label noise threshold. This is because (a) the candidate matches per query node increases (see Figure 9), and (b) the number of iterations required for the convergence of our network matching algorithm also increases (see Figure 10). Hence, it takes more time for NeMa to find the matches.

7.2.3 Effectiveness with Unlabeled Query Node

We next verified the effectiveness of NeMa in the presence of *unlabeled nodes* in query graphs. These experiments simulate RDF query answering (see Section 5.2). For these experiments, we randomly selected two sets of 100 query graphs each, where (i) $|V_Q| = 7$, $D_Q = 3$, and (ii) $|V_Q| = 5$, $D_Q = 2$, respectively. Fixing the structural noise as 30%, label noise as 35%, and label noise threshold as 35%, we varied the number of unlabeled query nodes from 0 to 2. As shown in Figure 11, (1) the F1-measure decreases over both datasets while the number of the unlabeled nodes increases, because the unlabeled nodes introduce more candidates, which in turn reduces the effectiveness, (2) the effectiveness is higher over *IMDB* due to the type constraints, and (3) over all the cases, the F1-measure is always above 0.50.

7.2.4 Performance with Propagation Depth

In these experiments, we analyzed the effect of propagation depth h in our query performance. We randomly selected 100 query graphs from *YAGO*, with $|V_Q| = 7$, $D_Q = 3$, structural noise 30%, label noise 50% and label noise threshold 50%. Table 4 shows that the efficiency of NeMa decreases with increasing h, especially the index time increases exponentially with h. However, for $h = 2$, we obtained an acceptable F1-measure of 0.86.

	$h = 1$	$h = 2$	$h = 3$
Index Time (sec)	265	9862	236553
Match Time (sec)	0.58	0.92	2.76
F1-Measure	0.61	0.86	0.87

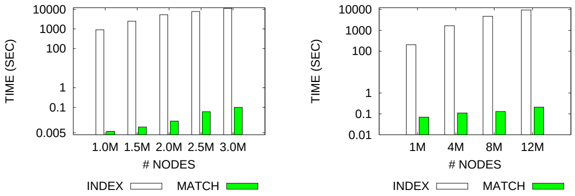
Table 4: Query Performance with Varying h ***(YAGO)***

7.2.5 Performance with Edge Labels

We verified the query performance in the presence of edge labels (Section 5.2). We randomly selected 100 query graphs from *IMDB*, with with $|V_Q| = 7$, $D_Q = 3$, label noise 50% and label noise threshold 50%. We varied the structural noise from 0% to 40%. The labels of the newly inserted edges in the query graphs were assigned by generating random strings. Figure 12 shows that, (1) when no structural noise is added, the F1-measure remains same for both the cases of labeled and unlabeled edges. (2) However, with the addition of structural noise, the F1-measure decreases slightly for the case of labeled edges, since there are more noises in the query graphs due to the labels of newly inserted edges. (3) The running time is higher for the case of labeled edges because additional time is required to measure edge label similarities.

7.2.6 Comparison with Existing Algorithms

We compared the performance of NeMa with IsoRank [33], SAGA [35], NESS [20], gStore [43], and BLINKS [16]. (1) IsoRank and SAGA find optimal graph matches in smaller biological networks considering structure and node label similarities. (2) NESS finds the top-k graph matches from a large network, but with strict node label equality. Hence, we modified NESS by allowing two nodes to be matched if their label difference is within the label noise threshold. (3) We considered a variation of NeMa, namely, NeMa$_{gs}$, which allows label difference but resorts to strict isomorphic matching. Thus, NeMa$_{gs}$ essentially follows the same principle as gStore, which is a subgraph isomorphism-based SPARQL query evaluation method with node label differences. (4) BLINKS [16], a keyword search method, supports only



(a) ***IMDB*** (b) ***YAGO***

Figure 14: Scalability

structural mismatches. Hence, we also modified BLINKS by allowing node label differences within the label noise threshold.

All these methods, except NESS, find the top-1 graph match directly. Hence, we considered the top-1 match corresponding to each query node to evaluate precision, recall, and F1-measure; and thereby obtained the same score for them. In contrast, NESS employs a *filtering-and-verification* approach, where its filtering phase reports a set of high-quality final candidate nodes for each query node. Then, it verifies all possible graph matches formed by these final candidate nodes, in order to find the top-1 graph match. Therefore, we report precision, recall and F1-measure of its filtering phase, which is the most important step in NESS. For fairness, we reported only the running time of its filtering phase in Figure 13(c).

For these experiments, we randomly selected 100 query graphs, where $|V_Q|$ = 7 and D_Q = 3, using the *IMDB* dataset. In each query graph, one node was *unlabeled* and the labels of the remaining nodes were updated by randomly inserting new words. We varied structural noise in Figure 13(a), and fixed both label noise and label noise threshold as 50%. Observe that, with no structural noise, NeMa and NeMa$_{\text{gs}}$ have F1-measure about 0.94; but with the increase in structural noise, NeMa (F1-measure 0.9) outperforms the other methods (F1-measure $0.1 \sim 0.7$)

We varied label noise and label noise threshold in Figures 13(b) and 13(c), and fixed structural noise as 30%. The label noise threshold had the same value as label noise in these figures. With no label noise, NeMa has F1-measure 0.93, whereas NESS, IsoRank, SAGA, and BLINKS have F1-measures about 0.8. However, as we increase label noise, NeMa (F1-measure 0.9) outperforms the other methods (F1-measure $0.1 \sim 0.7$) by a large margin.

NeMa finds the best match in less than 1 sec, while IsoRank takes 5000 sec. SAGA requires 15 sec and 546 sec, with label noise 50% and 80%, respectively.

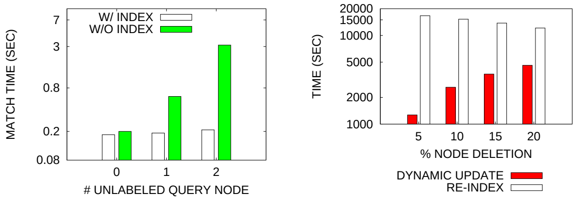
7.3 Scalability

In this section, we analyzed the scalability of NeMa by varying the number of nodes in the *YAGO* and *IMDB* networks. We used 100 randomly selected query graphs, where $|V_Q|$ = 7, d_Q = 3, and fixed the structural noise as 30%, label noise as 35%, and label noise threshold as 35%. Figure 14 shows that NeMa scales well with the size of the target graphs. Specifically, the off-line indexing time increases polynomially, and the online query evaluation time linearly with the increase of the size of the target networks.

7.4 Optimization techniques

In these experiments, we investigated the performance of the optimization techniques of NeMa. We randomly selected 100 query graphs, where $|V_Q|$ = 7 and D_Q = 3, and fixed the structural noise as 30% and both label noise and label noise threshold as 35%. In each query graph, the number of *unlabeled* query nodes is varied from 0 to 2. Figure 15(a) shows that the indexing and optimization techniques significantly improve the efficiency of NeMa, specifically by a factor of 15 in the presence of 2 unlabeled query nodes.

We also compared the index construction time with *dynamic update* against the cost of rebuilding the whole index. In these experiments, we considered only deletion of nodes (and thereby, deletion of the incident edges) from the original network as a method of dynamic updates. Figure 15(b) shows that, 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.



(a) Search Time (***IMDB***) (b) Dynamic Updates (***YAGO***)

Figure 15: Index Performance

8. CONCLUSIONS

In this paper, we have introduced NeMa, a novel graph querying framework via subgraph matching that allows for ambiguity in both the structure and node labels. We convert the neighborhood of each node into a multi-dimensional vector, and then apply an inference algorithm to identify the optimal matches. We further investigate how NeMa can be extended to various graph query-processing applications, such as RDF query answering, graph with edge labels, and top-k approximate matches. Our experimental results over real-life datasets show that NeMa can efficiently find high-quality matches, as compared to state-of-the-art graph querying methods. In future work, one may consider approximate search over graph streams, and also more sophisticated label similarity metrics, *e.g.*, ontology and semantic similarity.

9. ACKNOWLEDGEMENTS

The first author was supported by an IBM Ph.D. Fellowship. This research was also supported 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). The authors would like to thank Doug Bradley from UC Santa Barbara and Shibamouli Lahiri from University of N. Texas for their valuable comments. 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.

10. REFERENCES

[1] S. Abiteboul. Querying Semi-Structured Data. *ICDT*, 1997.

[2] J. R. Anderson. A Spreading Activation Theory of Memory. *J. Verbal Learning and Verbal Behavior*, 1983.

[3] P. Barceló, L. Libkin, and J. L. Reutter. Querying Graph Patterns. *PODS*, 2011.

[4] M. Bayati, M. Gerritsen, D. F. Gleich, A. Saberi, and Y. Wang. Algorithms for Large, Sparse Network Alignment Problems. *ICDM*, 2009.

[5] H. Berger, M. Dittenbach, and D. Merkl. An Adaptive Information Retrieval System based on Associative Networks. *APCCM*, 2004.

[6] N. Buchan and R. Croson. The Boundaries of Trust: Own and Others' Actions in US and China. *J. Econ. Behav. and Org.*, 2004.

[7] V. S. Cherukuri and K. S. Candan. Propagation-Vectors for Trees (PVT): Concise yet Effective Summaries for Hierarchical Data and Trees. *LSDS-IR*, 2008.

[8] S. Cook. The Complexity of Theorem Proving Procedures. *STOC*, 1971.

[9] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Tran. Pattern Anal. and Machine Int.*, 2004.

[10] S. Das, E. I. Chong, G. Eadon, and J. Srinivasan. Supporting Ontology-Based Semantic Matching in RDBMS. *VLDB*, 2004.

[11] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph Pattern Matching: From Intractable to Polynomial Time. *PVLDB*, 2010.

[12] W. Fan, J. Li, S. Ma, H. Wang, and Y. Wu. Graph Homomorphism Revisited for Graph Matching. *PVLDB*, 2010.

[13] B. Gallagher. Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching. *AAAI FS.*, 2006.

[14] J. Han. Mining Heterogeneous Information Networks by Exploring the Power of Links. *ALT*, 2009.

[15] L. Han, T. Finin, and A. Joshi. GoRelations: An Intuitive Query System for DBpedia. *LNCS*, 2011.

[16] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked Keyword Searches on Graphs. *SIGMOD*, 2007.

[17] J. Liu and X. Dong and A. Halevy. Answering Structured Queries on Unstructured Data. *WebDB*, 2006.

[18] M. Kargar and A. An. Keyword Search in Graphs: Finding r-cliques. *PVLDB*, 2011.

[19] 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*, 2004.

[20] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood Based Fast Graph Search in Large Networks. *SIGMOD*, 2011.

[21] J. W. Kim and K. S. Candan. CP/CV: Concept Similarity Mining without Frequency Information from Domain Describing Taxonomies. *CIKM*, 2006.

[22] Z. Liang, M. Xu, M. Teng, and L. Niu. NetAlign: A Web-based Tool for Comparison of Protein Interaction Networks. *Bioinfo.*, 2006.

[23] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo. Capturing Topology in Graph Pattern Matching. *PVLDB*, 2012.

[24] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. PREGEL: A System for Large-Scale Graph Processing. *SIGMOD*, 2010.

[25] S. Melnik, H. G.-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. *ICDE*, 2002.

[26] M. Mongiovì, R. D. Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan. SIGMA: A Set-Cover-Based Inexact Graph Matching Algorithm. *J. Bioinfo. and Comp. Bio.*, 2010.

[27] C. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes. *J. Comp. and Sys. Sc.*, 1991.

[28] Y. Papakonstantinou and V. Vassalos. Query Rewriting for Semistructured Data. *SIGMOD*, 1999.

[29] J. Pearl. Reverend Bayes on inference engines: A distributed Hierarchical Approach. *American Association of AI Conf.*, 1982.

[30] K. Sambhoos, R. Nagi, M. Sudit, and A. Stotz. Enhancements to High Level Data Fusion using Graph Matching and State Space Search. *Inf. Fusion*, 2010.

[31] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 2008.

[32] H. Shang, Y. Zhang, X. Lin, and J. Yu. Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism. *PVLDB*, 2008.

[33] R. Singh, J. Xu, and B. Berger. Global Alignment of Multiple Protein Interaction Networks with Application to Functional Orthology Detection. *PNAS*, 2008.

[34] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient Subgraph Matching on Billion Node Graphs. *PVLDB*, 2012.

[35] Y. Tian, R. McEachin, C. Santos, D. States, and J. Patel. SAGA: A Subgraph Matching Tool for Biological Graphs. *Bioinfo.*, 2006.

[36] Y. Tian and J. M. Patel. TALE: A Tool for Approximate Large Graph Matching. *ICDE*, 2008.

[37] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast Best-Effort Pattern Matching in Large Attributed Graphs. *KDD*, 2007.

[38] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *J. ACM*, 1976.

[39] J. Yao, B. Cui, L. Hua, and Y. Huang. Keyword Query Reformulation on Structured Data. *ICDE*, 2012.

[40] S. Zampelli, Y. Deville, C. Solnon, S. Sorlin, and P. Dupont. Filtering for Subgraph Isomorphism. *CP*, 2007.

[41] S. Zhang, J. Yang, and W. Jin. SAPPER: Subgraph Indexing and Approximate Matching in Large Graphs. *PVLDB*, 2010.

[42] Q. Zhong, H. Li, J. Li, G. Xie, J. Tang, L. Zhou, and Y. Pan. A Gauss Function Based Approach for Unbalanced Ontology Matching. *SIGMOD*, 2009.

[43] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao. gStore: Answering SPARQL Queries via Subgraph Matching. *VLDB*, 2011.

11. APPENDIX

11.1 NeMa vs NESS

To leverage for node label mismatches, one can modify NESS by allowing two nodes to be matched when their label difference is within a predefined cost threshold. However, such modification reduces the pruning capability of NESS, and generates a large number of final candidate nodes at the end of its filtering step. Hence, finding the top-k graph matches by verifying all possible graph matches formed by this large set of final candidate nodes becomes expensive.

Example 10.1. In Figure 16, label a can be matched with a_1, a_2; and label b with b_1, b_2, b_3. Now, consider a matching function ϕ between the query graph Q and match M_1, where $\phi(v_1) = u_1$, $\phi(v_2) = u_2$, $\phi(v_3) = u_3$, $\phi(v_4) = u_5$, $\phi(v_5) = u_6$, and $\phi(v_6) = u_7$. Assuming $h = 1$ and $\alpha = 0.5$, the label-based neighborhood vectors, as defined in NESS, are as follows. $\mathbb{R}_Q(v_4) = \{\langle a, 1\rangle\}$ due to nodes v_3 and v_5 in the 1-hop neighborhood of v_4. Similarly, $\mathbb{R}_\phi(u_5) = \{\langle a_1, 0.5\rangle \langle a_2, 0.5\rangle\}$, due to nodes u_6 and u_7 in the 1-hop neighborhood of u_5. Hence, the label-based neighborhood matching cost $N_\phi(v_4, u_5) = 1-(0.5+0.5) = 0$. Thus, individual node matching cost $F_\phi(v_4, u_5) = \Delta_L(b, b_2)$.

Next, consider another matching ϕ' between Q and match M_2, where $\phi'(v_1) = u_8$, $\phi'(v_2) = u_9$, $\phi'(v_3) = u_{10}$, $\phi'(v_4) = u_{11}$, $\phi'(v_5) = u_{12}$, and $\phi'(v_6) = u_{13}$. As before, $F_{\phi'}(v_4, u_{11}) = \Delta_L(b, b_2)$. In fact, one may verify that $C(\phi) = C(\phi')$, using NESS; although ϕ' is a better match than ϕ. This happens because NESS cannot ensure that the matching of the underlying nodes are preserved when we match the neighborhood vectors based on their label distributions. Hence, the label based neighborhood matching in NESS is weak, and its pruning capacity further reduces when we allow node matching with slightly different labels.

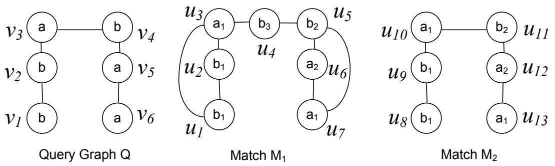


Figure 16: Problem in NESS (Label Based Neighborhood Match)

In contrast, NeMa has improved performance due to two reasons. (1) NeMa computes the neighborhood similarity by directly comparing the neighboring nodes (Eq. 2). This makes the subgraph matching in NeMa more strict as compared to NESS. Indeed, $C(\phi) > C(\phi')$, using NeMa (Figure 16). (2) Unlike the filtering-and-verification approach in NESS, our proposed NeMa framework directly finds the top-k graph matches, considering both label and structural similarity. As verified in Section 7, NeMa outperforms NESS in identifying reasonable matches when the same entity can have different labels in query and target graph.