Neighbor-Aware Search for Approximate Labeled Graph Matching using the Chi-Square Statistics

Sourav Dutta*
Nokia Bell Labs,
Dublin, Ireland
sourav.dutta@nokia.com

Pratik Nayek†
Microsoft Development Center
Bengaluru, India
prnayek@microsoft.com

Arnab Bhattacharya
Computer Sc. & Engg.,
Indian Institute of Technology,
Kanpur, India
arnabb@cse.iitk.ac.in

ABSTRACT

Labeled graphs provide a natural way of representing entities, relationships and structures within real datasets such as knowledge graphs and protein interactions. Applications such as question answering, semantic search, and motif discovery entail efficient approaches for subgraph matching involving both label and structural similarities. Given the NP-completeness of subgraph isomorphism and the presence of noise, approximate graph matching techniques are required to handle queries in a robust and real-time manner. This paper presents a novel technique to characterize the subgraph similarity based on statistical significance captured by chi-square statistic. The statistical significance model takes into account the background structure and label distribution in the neighborhood of vertices to obtain the best matching subgraph and, therefore, robustly handles partial label and structural mismatches. Based on the model, we propose two algorithms, VELSET and NAGA, that, given a query graph, return the top-k most similar subgraphs from a (large) database graph. While VELSET is more accurate and robust to noise, NAGA is faster and more applicable for scenarios with low label noise. Experiments on large real-life graph datasets depict significant improvements in terms of accuracy and running time in comparison to the state-of-the-art methods.

Keywords

Approximate subgraph matching; Statistical significance; Chi-square statistic; Subgraph similarity;

1. INTRODUCTION

With the proliferation of social networks, bioinformatics, chemoinformatics, road networks, etc., the problem of efficient graph querying and mining poses a challenging area of research [1]. The underlying data is generally represented as a *labeled graph* where nodes represent entities and edges capture the relationships and interactions among the entities while the labels on the nodes define the characteristics of the entities. Given a set of graphs, the aim of query graph matching is to retrieve graphs similar to the query [24]. For example, in chemoinformatics or bioinformatics, an important task is to search for compounds or protein structures similar to a query [40] for motif discovery [12]. Traditional approaches for such queries involve *graph isomorphism* where an exact match is required in the context of both graph structure and labels. The problem of graph isomorphism has recently been shown to be quasi-polynomial [3], while *subgraph isomorphism* is NP-complete [10].

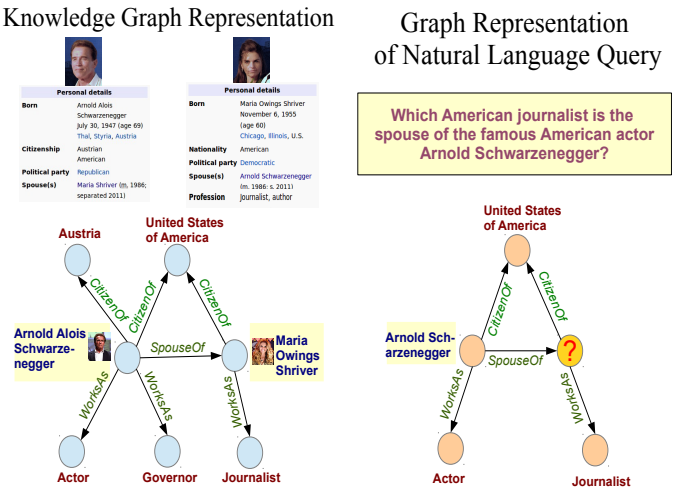


Figure 1: Approx. subgraph matching for question answering.

Consequently, *approximate subgraph matching* has been explored [24,27]. With the availability of extremely large graphs in the form of social networks, road networks, etc., the field of graph querying has extended to the scenario where the aim is to find similar subgraphs from a large database graph [24, 27]. The match can be approximate, allowing for insertions and deletions of structural elements such as nodes and edges as well as label mismatches.

1.1 Applications of Approximate Matching

The problem of approximate subgraph matching caters to myriad applications across various domains including knowledge graph based search for question answering, bioinformatics for genome and protein interaction detection, and social network analysis.

For example, consider Fig. 1 to depict the snapshot of a knowledge graph (KG) containing attributes and relationships about the famous actor Arnold Schwarzenegger. Facts about named-entities (e.g., person, organization, etc.) extracted from Wikipedia infoboxes or other Web sources are typically represented as labeled graphical structures, with labeled vertices denoting the entities and the edges denoting relationships between the entities. Such KGs form the backbone repository for question answering systems such as IBM Watson or Google semantic search enabling efficient and precise

*Work done while the author was at Max Planck Institute for Informatics, Germany.
†Work done while the author was at Indian Institute of Technology, Kanpur, India.

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052561

computation of natural language queries or entity-based search. An example is "*Which American journalist is the spouse of the famous American actor Arnold Schwarzenegger?*"

Downstream processing of such queries entails the extraction of subgraphs (representing the input query) having similar structures and labels from the underlying knowledge graph [6, 20]. As such, "similar" subgraphs demonstrating exact matches or with slight variations in labels (mis-spellings or different surface forms involving acronyms, aliases, etc.) and/or edges (absence of information) with that of the query provide probable candidate answers for further investigation to generate the desired result (as seen in Fig. 1).

In bioinformatics, approximate graph matching plays a vital role in detecting regions in genome that undergo abnormal mutations and interaction of proteins and chemicals therein [37, 47].

However, use of exhaustive techniques utilizing elaborate subgraph isomorphism tests does not scale to large graphs. Thus, modern applications resort to heuristics to retrieve subgraphs that are "similar" to the query graph but may not match exactly. Since it is computationally infeasible to either generate or store all such subgraphs, subgraph querying techniques depend on paradigms that generate subgraphs on-the-fly and quickly filter them based on various pruning conditions. Although "similar" subgraph generation based on various measures have been proposed [24,27,38,41], limited work exists for efficient extraction of matching subgraphs in terms of both running time and accuracy considering both structural and label similarity. We propose robust subgraph matching based on *statistical significance*, previously shown to be successful for mining connected subgraphs [2], and sequence mining [14, 35].

1.2 Motivation for Statistical Significance

The degree of structural similarity between a query graph and regions in the graph database provides a paramount criteria for approximate subgraph matching. As such, various works have captured these characteristics using a variety of features, including number of vertices (with a certain label), number of edges, degree distribution, centrality, diameter, and neighborhood [24, 27].

Nevertheless, the one critical information that has not been captured is the relationship of the labels in the query graph with the *underlying background distribution* of the labels of the larger graph that is being searched. Assuming a uniform distribution of graph labels (from a countable set), the probability of near exact label matches in contiguous regions (i.e., neighborhood) of the database for the entire query is quite low. Hence, such regions tend to exhibit high statistical significance, and can be detected using such.

Using an analogy from the domain of string matching, suppose the query string $q = abac$ is to be searched within a large string $s = cabacbac$ composed from the alphabet set $\Sigma = \{a, b, c\}$. Note that, 3-gram level matching of triplets from q over s matches at positions 2-4, 3-5 and 6-8. Assuming a probability distribution on Σ where a is less frequent than b and c, the statistical significance of matching at position 2 is greater than that at 3 and 6, as the chance of two a's occurring is less likely. Another interesting feature of this approach is its robustness to *noise* or near-exact matches. For example, a query $q' = cbab$ in s would also return $cbac$ as the best matching region with high statistical significance. This strategy was successfully used in [13] for approximate substring matching.

The above intuition of using statistical significance can be captured formally by the notion of *p-value* [34]. The *p-value* of an observed event is the probability of occurrence of events *at least as extreme* as this one. For example, in a coin toss experiment, the p-value of observing 8 heads in a series of 10 tosses is the probability of obtaining 8, 9 or 10 heads. Since extreme events are rare, the *lesser* the p-value, the *more* statistically significant is the event.

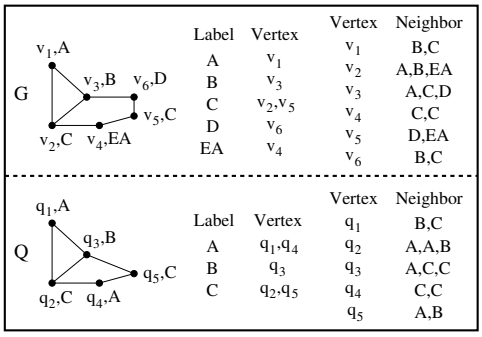


Figure 2: Running example: Input graph G and query Q.

Adopting the notion for graph structures, each vertex in the graph database is represented by a label set composed of the labels of its neighboring vertices. Similarly, the vertices of the query graph are also represented as label sets. The vertices with higher degrees of match depict higher statistical significance, thereby serving as better candidates for subgraph matching. Such a strategy is not only computationally fast but also inherently robust in capturing the approximation criteria for both label and structural similarities.

As an example, in Fig. 2, the vertex $v_2 \in G$ is represented by the label set $\{A, B, EA\}$ (pertaining to its neighbors), while $q_2 \in Q$ and $q_5 \in Q$ are represented as $\{A, A, B\}$ and $\{A, B\}$ respectively. Considering only structural similarity, v_2 can be matched with either q_2, or even q_3, since it has three adjacent nodes. Using labels, v_2 can be matched with q_2 or q_5. Only using both the criteria, and taking into consideration the background distribution of labels, can it be found out the best match of v_2 is with q_2 (depicting higher statistical significance). In other words, q_2 provides a higher degree of match (based on label and structure) and would, thus, depict a higher statistical significance. This forms the *basis* of our proposed algorithms for approximate labeled graph querying.

Since computing the p-value is practically inefficient due to the exponential number of possibilities, the *Pearson's chi-square statistic* [34] has been shown to provide a good estimate [33]. The chi-square statistic, χ^2, computes the (normalized squared) difference between the expected and observed occurrences of outcome values:

$$\chi^2 = \sum_{\forall i} \left[(O_i - E_i)^2 / E_i \right] \quad (1)$$

where O_i and E_i are the *observed* and *expected* number of occurrences, respectively, of the possible outcomes i of an experiment. If p_i represents the probability of occurrence of event i, the expected counts are $E_i = len \times p_i$ for len number of observations.

The χ^2 statistic follows the *chi-square distribution* [34] with degrees of freedom one less than the number of possible outcomes. Higher the χ^2 statistics, lower is the p-value and, thus, higher is the statistical significance. Hence, retrieving a more "similar" subgraph, i.e., one with more statistical significance, translates to finding a subgraph with a larger χ^2 value.

1.3 Problem Statement

We propose to solve the approximate subgraph matching problem utilizing the statistical significance of matches (based on chi-square measure) as the similarity measure based on both structural and label similarity. Formally,

PROBLEM 1. *Given a labeled undirected graph G and a query graph Q where the vertices of G and Q are labeled, retrieve from G the top-k statistically significant subgraphs similar to Q.*

In this paper, we only consider the cases of vertices being labeled (with a single label) and that the labels are drawn from a count-

able set. The scenarios of edge labels (handled via construction of auxiliary vertices) and missing labels, are discussed in Sec. 4.6.

Contributions: In a nutshell, our major contributions are:

- We propose the formulation of *approximate* "similar" subgraph querying problem in terms of *statistical significance*, and use the *chi-square statistics* to jointly incorporate graph label and structural similarities. To the best of our knowledge, there exists no prior literature that works along these lines.
- We present two algorithms to efficiently and effectively solve the above subgraph matching problem. The first algorithm, VELSET, is more generic and robust to noisy edges and/or noisy labels.
- We also present an efficient algorithm, NAGA, that uses an aggressive pruning strategy based on labels.
- We perform extensive experiments on real-life graph datasets to depict improved running time performance and accuracy compared to the state-of-the-art approaches.

2. RELATED WORK

Graph Matching: Graph querying is a well researched problem catering to various application domains. While the graph isomorphism problem is quasi-polynomial [3], exact subgraph isomorphism is NP-complete [10]. Some earlier works targeting this problem involve tree-pruning [46], Swift-Index [36] and VF2 [11]. Biological networks are handled in PathBlast [23], SAGA [41], NetAlign [31] and IsoRank [38]. They are mostly suited for smaller networks, though. An ordered-search algorithm for determining error-correcting isomorphism was proposed in [45]. An algorithm based on random walk to find the best matching in a large graph was proposed in [43, 44]. GraphGrep [18] uses hash-based fingerprinting for subsequent filtering. A variant of the graph isomorphism problem, with vertices characterized by their labels as the only attribute and edges denoting the connectivity in a data graph for a predefined number of hops, was shown to be polynomial-time bounded in [16], with a cost-effective incremental version presented in [15].

Another similar domain of work involves frequent subgraph finding and pattern mining over a set of graphs or a single large graph. The frequent sub-graph discovery algorithm [28] uses sparse graph representation. gSpan [49] uses a canonical labeling system, supporting depth-first search, to find frequently occurring subgraphs. Similar works include GREW [29], mining proximity pattern [25] and frequent patterns in large sparse graphs [30].

In [2], the chi-square measure was used to mine subgraphs that are statistically significantly different from the background distribution of labels in a vertex-labeled graph.

For large data with noise, the approximate subgraph matching problem was formulated to tolerate a small amount of mismatch. An efficient index based approximate subgraph matching tool, TALE [42], maintains the topology of the query using a best-effort pattern matching, and uses a maximum weighted bipartite graph matching algorithm. Zhang et al. [53] proposed another indexing approach based on graph distance. A set-cover-based inexact subgraph matching technique, SIGMA [32], and a regular expression based approach for graph pattern matching [4]. were also studied. SAPPER [54], based on edge edit distance, was shown to be efficient and scalable for large noisy graph databases. APGM [21] proposes a method to mine useful patterns from noisy graph databases. C-Tree [56] proposed a technique for a generalized representation of graphs usable for both subgraph querying and subgraph similarity computation. A semantic-based search algorithm, GString [22], introduced a sequencing method to capture the semantics of the underlying graph data. Some other relevant subgraph matching works include gIndex [50], FG-Index [8], iGraph [19], Grafil [51], Gcoding [55], GPTree [52], cIndex [7], and iGQ [48].

NeMa [24] and SIM-T [27] are two recently proposed works tackling both structural and label noises for the approximate graph matching problem. NeMa involves a fast neighborhood-based subgraph matching algorithm with efficient indexing. It uses a cost-metric based node selection mechanism which helps in covering both structural and label noise. SIM-T performs *tabu search* on an initialized known subset of optimal solution, and shows high accuracy when initialized with a potentially good solution.

Two extensive surveys on graph matching are in [9, 17].

Statistical Significance: Statistical models and measures involve establishing a relation between the empirical or observed results of an experiment with factors affecting the system or to pure chance. An observation is deemed to be *statistically significant* if its presence cannot be attributed to randomness alone. The experimental setup for computing the *p-value* of an observed event is described using a *null hypothesis*. However, the computation can be practically infeasible as it can entail the generation of all possible outcomes. To alleviate this problem, various branch-and-bound methods have been studied [5]. The literature hosts a number of statistical models to approximate the p-value, such as *chi-square*, *z-score*, *log-likelihood ratio* (G^2), and *Hotelling's* T^2 *measure* [34].

The chi-square distribution (χ^2) is widely used to compute the goodness-of-fit of a set of observations to the theoretical model describing the null hypothesis. In most situations, the chi-square distribution provides a good approximation of the p-value [33]. Hence, in this paper, we measure the statistical significance using the *Pearson's* χ^2 *measure* based on the chi-square distribution.

3. THE VELSET ALGORITHM

Both our proposed approaches, VELSET (Sec. 3) and NAGA (Sec. 4), follow a common framework as briefly described next (Fig. 3). The details of the individual algorithms are described later.

3.1 Algorithmic Framework

In the *offline* phase when only the complete graph G (knowledge repository) is available, inverted lists for vertex labels, vertex neighbor-label lists, and other similar information synopses (as shown in Fig. 2) are created. When the query graph Q is made available at *runtime*, similar lists are also created for Q. Note that since Q is small, creation of these lists is computationally lightweight. The lists for the database graph computed offline remain the same irrespective of Q and, hence, do not affect the query time.

Using the sets of lists for G and Q, the candidate vertex pairs are then located in G. These candidates serve as "seeds" and the regions around them are subsequently grown or expanded to generate the matches. For each candidate pair obtained, the chi-square value is computed. Higher the chi-square, higher is the chance that these candidate pairs are good matches. Thus, after insertion of the candidate pairs in a max-heap (called the *primary heap*), the best pair is extracted for exploration.

The neighborhood of the extracted pair is then explored and, vertex pairs generated from the neighborhood are then stored in another max-heap (called the *secondary heap*) along with their corresponding chi-square values. The best ones are again extracted from this heap and their neighborhoods are re-explored to grow the possible matching regions. This continues till the entire query graph is processed. Iterating this k times generates the top-k matches.

It is evident from the above discussion that the mode of searching to find the best matching region follows the *greedy* paradigm by using the chi-square values. Thus, it is not guaranteed to return the optimal match. However, as the experiments show (Sec. 5), the quality of the proposed heuristics are high. We next describe in details the individual algorithmic phases of VELSET.

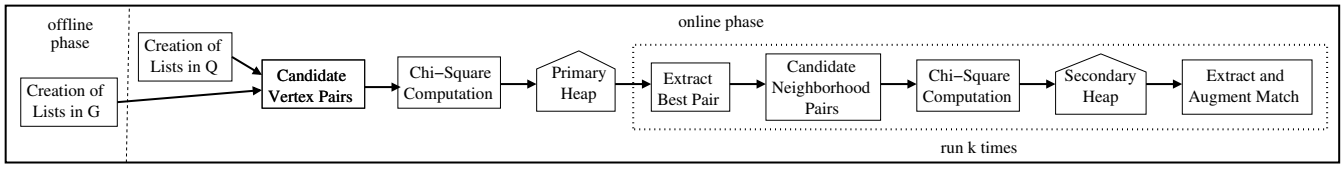


Figure 3: Algorithmic framework overview for VELSET and NAGA.

Algorithm 1 VELSET

Input: Database graph G, Query graph Q, Number of matches k
Output: Top-k subgraphs of G similar to Q

Compute IL^G and LNL^G of input graph G ▷ Offline
Compute IL^Q and LNL^Q of query graph Q
$VP \leftarrow \{\langle u, v\rangle \mid u \in G \wedge v \in Q \wedge l_u^G \simeq l_v^Q\}$
Primary max-heap $PH \leftarrow \Phi$
for all $\langle u, v\rangle \in VP$ **do**
  $T_u, T_v \leftarrow$ triplets centered at u, v respectively
  Initialize O_u to empty string of length $len_v = \binom{deg(v)}{2}$
  for all triplet $t_v \in T_v$ **do**
    $t_u \leftarrow$ largest overlapping match of t_v
    Symbol $s \leftarrow$ match of t_u with t_v ▷ s is s_2, s_1 or s_0
    Append O_u with s
  end for
  $E_u \leftarrow len_v \times [Pr(s_0), Pr(s_1), Pr(s_2)]$ ▷ Eqs. (6), (7), (8)
  Compute $\chi^2_{\langle u,v\rangle}$ using expected E_u and observed O_u ▷ Eq. (1)
  Insert $\langle u, v, \chi^2_{\langle u,v\rangle}.JS(l_u, l_v)\rangle$ into PH
end for
for $i = 1$ to k **do**
  repeat
    $\langle u, v\rangle \leftarrow$ Extract(PH)
  until neither u nor v are marked "done"
  `Match(i)` $\leftarrow \langle u, v\rangle$
  Mark u and v as "done"
  Secondary max-heap $SH \leftarrow \Phi$
  $U', V' \leftarrow$ adjacent nodes of u, v respectively
  Insert $\{\langle u' \in U', v' \in V', \chi^2_{\langle u',v'\rangle}.JS(l_{u'}, l_{v'})\rangle \mid l_{u'}^G \simeq l_{v'}^Q\}$ into SH
  while $|$`Match(i)`$| \neq |Q|$ and $SH \neq \Phi$ **do**
    Expand `Match(i)` by "best" unmarked vertex pair $\langle u', v'\rangle$ from SH
    Insert unmarked vertex pairs from neighborhood of u', v' into secondary heap SH
  end while
end for
return Top-k matches `Match(1), ..., Match(k)`

3.2 Creation of Different Lists

We initially propose the *VErtex Label Similarity on Edge Triplets* or *VELSET* algorithm (pseudocode shown in Algorithm 1). The first step involves the construction of different *indexing lists* summarizing the labels of the vertices and their neighbors. We create *inverted lists* mapping the vertex labels to the corresponding vertices, and are denoted as IL^G and IL^Q for graphs G and Q respectively. The second index, the *label neighbor list* (denoted by *LNL*) stores the label information of the neighbors for each vertex. Thus, the set of labels in the neighbors of $u \in G$ is denoted by LNL_u^G, and those in the neighbors of $v \in Q$ is denoted by LNL_v^Q, etc. As a running example throughout the paper, consider the input graph G and query Q shown in Fig. 2 with the corresponding inverted lists and label neighbor lists.

3.3 Vertex Pair Construction

For each label in Q, VELSET next identifies *similar* labels in G. The notion of similarity between two labels, either identical or fuzzy matching above a threshold, can be appropriately defined by the application. If a label l_i^Q in Q is similar to l_j^G in G, it is denoted as $l_i^Q \simeq l_j^G$; otherwise, they are not similar: $l_i^Q \nsim l_j^G$.

Adopting the approach of [24], we use the Jaccard similarity above a threshold to identify similar labels. The *Jaccard similarity* between two strings s and t composed of characters from the same alphabet set is defined as the number of common characters in s and t, over the total number of characters in the two strings: $JS(s,t) = |s \cap t|/|s \cup t|$. Hence, in Fig. 2, considering a similarity threshold of $\tau = 0.5$, the label EA is similar to the label A.

For each vertex v in Q with label l_v^Q, all the vertices u in G having labels l_u^G where $l_v^Q \simeq l_u^G$, are identified. The vertex pairs $\langle u, v\rangle$ then form the candidate vertex pairs, denoted by *VP*. Formally,

$$VP = \{\langle u, v\rangle \mid u \in G \wedge v \in Q \wedge l_u^G \simeq l_v^Q\} \quad (2)$$

VP can be constructed by performing a join operation on the inverted lists. The vertex pairs obtained for the example in Fig. 2 are $VP = \{\langle v_1, q_1\rangle, \langle v_1, q_4\rangle, \ldots, \langle v_5, q_5\rangle, \ldots, \langle v_4, q_4\rangle\}$.

3.4 Statistical Significance Computation

VELSET next chooses the *most statistically significant* vertex pair from *VP* using the following procedure. The chi-square statistic, $\chi^2_{\langle u,v\rangle}$ for each candidate vertex pair $(u, v) \in VP$, is computed on the basis of "label triplets" using the following steps.

(1) Triplet Generation: A *triplet* of vertices centered at $u \in G$ is the 3-tuple $\langle x, u, y\rangle$ where vertices x and y are adjacent to u. The corresponding *label triplet* $\langle l_x^G, l_u^G, l_y^G\rangle$ of u is then constructed from the respective vertex labels of x, u, y. The ordering of neighboring labels within a triplet does not matter in subsequent computations, i.e., $\langle l_x^G, l_u^G, l_y^G\rangle \equiv \langle l_y^G, l_u^G, l_x^G\rangle$. For the example in Fig. 2, vertex v_3 with label B have neighbors v_1, v_2 and v_6 with labels A, C and D respectively. Hence, the possible label triplets centered at v_3 are $\langle A, B, C\rangle$, $\langle C, B, D\rangle$, and $\langle D, B, A\rangle$. Similarly, the only label triplet generated with q_5 at the center is $\langle B, C, A\rangle$.

While algorithms can be developed to capture two-hop or more detailed neighborhoods, there are two main reasons for not doing so. First, the neighborhood information becomes progressively less precise with more hops. Second, the resulting statistical significance model becomes unnecessarily complex.

(2) Degree of Label Overlap: For a vertex pair $\langle u, v\rangle$, the triplets of $v \in Q$ are then matched with those of $u \in G$. A *one-to-one* matching is considered and the number of *triplet matches* is the number of triplets of $u \in G$ corresponding to $v \in Q$.

The similarity between two label triplets is characterized by the *degree of overlap* of the labels. Consider the labels $\langle l_x^G, l_u^G, l_y^G\rangle$ and $\langle l_a^Q, l_v^Q, l_b^Q\rangle$ for $(u, v) \in VP$ (i.e., $l_u^G \simeq l_v^Q$) centered at $u \in G$ and $v \in Q$ respectively. Based on label similarity, the degree of overlap is classified into 3 classes (with symbols s_2, s_1, and s_0) as:

- s_2: Both the neighboring vertex labels of u and v are similar.

$$s_2 : (l_x^G \simeq l_a^Q \wedge l_y^G \simeq l_b^Q) \quad (3)$$

- s_1: Only one adjacent vertex label is similar.

$$s_1 : (l_x^G \simeq l_a^Q \wedge l_y^G \nsim l_b^Q) \vee (l_x^G \nsim l_a^Q \wedge l_y^G \simeq l_b^Q) \quad (4)$$

- s_0: None of the neighboring vertex labels are similar.

$$s_0 : (l_x^G \nsim l_a^Q \wedge l_y^G \nsim l_b^Q) \quad (5)$$

A triplet match gets the *highest* symbol it can achieve as maximum amount of matching is preferred for best candidate extraction; hence, $s_2 \succ s_1 \succ s_0$. The matched vertex in G is then represented by a sequence of symbols, O, corresponding to the triplet match with the query vertex. The fact that two triplets are matched establishes the structural similarity between the regions centered at u and v while the degree of overlap establishes the label similarity.

Consider the vertex pair $\langle v_3, q_3 \rangle$ in the example. The label triplet $\langle A, B, C \rangle$ of q_3 matches exactly with $\langle A, B, C \rangle$ generated from $\langle v_1, v_3, v_2 \rangle$ and, therefore, corresponds to symbol s_2. The next query label triplet $\langle C, B, C \rangle$ is best matched with $\langle C, B, D \rangle$, thereby providing symbol s_1. Similarly, the best match of $\langle C, B, A \rangle$ of q_3 is with $\langle D, B, A \rangle$, yielding another s_1 symbol. Thus, v_3 is represented by the sequence $\{s_2, s_1, s_1\}$ when matched with q_3.

Note that for a query with less than 3 vertices, pseudo-vertices with dummy labels matching any other label, are assumed.

(3) Expected Symbol Occurrence: The computed symbol sequence O_u for a candidate vertex $u \in G$ signifies its degree of subgraph matching with Q. The statistical significance of the match is measured by computing the chi-square statistic of O_u having length l. The expected probabilities of the symbols characterize their chance of occurring at random. For simplicity, we assume that there are L groups of equally likely labels, and labels in the same group are necessarily similar and are, therefore, matched.

Consider the event when the label of an adjacent vertex of u is *not* similar to the particular label l_a^Q of the query triplet. The chance of this happening is $(1 - 1/L)$. Since there are $d = deg(u)$ such adjacent vertices, the probability that *none* of the adjacent vertices is similar to l_a^Q is $(1 - 1/L)^d$. Considering the two query labels l_a^Q and l_b^Q to be *independent*, the probability that the triplet match gets the symbol s_0 (i.e., there is no label match), therefore, is,

$$Pr(s_0) = \left((1 - 1/L)^d\right)^2 \tag{6}$$

The event of a *matching label* is complementary to all d adjacent vertices having dissimilar labels and is, therefore, $1 - (1 - 1/L)^d$. As for s_1 one of the triplet labels do not match, the probability is,

$$Pr(s_1) = 2.\left(1 - (1 - 1/L)^d\right).\left((1 - 1/L)^d\right) \tag{7}$$

The factor 2 depicts that any one label of the triplet can be similar.

The symbol s_2 encodes the situation when both the labels of a triplet are similar, and its probability of occurrence is given by,

$$Pr(s_2) = \left(1 - (1 - 1/L)^d\right)^2 \tag{8}$$

Note that the symbols are exhaustive and they complete the event space: $Pr(s_0) + Pr(s_1) + Pr(s_2) = 1$.

Since the maximum number of possible triplets for $v \in Q$ with degree $deg(v)$ is $len_v = \binom{deg(v)}{2}$, the vertex $u \in G$ is represented by a sequence of len_v symbols. No triplet in G is matched to more than one query triplet. If the number of triplets centered at u is less than that centered at v, the rest of the $deg(v) - deg(u)$ triplets are assumed to match with null triplets with their symbols as s_0.

(4) Chi-Square Computation: VELSET computes the *expected* number of symbol occurrences for vertex u by multiplying the above probabilities (Eqs. (6) to (8)) by len_v. The *observed* number of each symbol is calculated from the actual match. The *chi-square statistic*, $\chi^2_{\langle u,v \rangle}$, for vertex pair $\langle u, v \rangle$ is computed by Eq. (1).

For modeling both structural and label similarity for subgraph matching, the computed chi-square values of the candidate vertices are multiplied with the Jaccard similarities of their labels to the query vertex labels, and the 3-tuples, $\langle u, v, (\chi^2_{\langle u,v \rangle} \cdot JS(l_u, l_v)) \rangle$, are inserted in a *primary* max-heap structure, *PH*. While higher Jaccard similarity prefers better matching labels, higher chi-square prefers closer structural matches.

Returning to the example vertex pair $\langle v_3, q_3 \rangle$, the symbol sequence for v_3 is $\{s_2, s_1, s_1\}$, as explained earlier. Hence, the observed count is $O[l_{v_3}] = [0, 2, 1]$. Using $|L| = 4$ and $d = deg(q_3) = 3$, the expected count vector for v_3, using Eqs. (6)–(8), is $E[l_{v_3}] = 3 \times [0.178, 0.488, 0.334]$. Hence, $\chi^2_{\langle v_3, q_3 \rangle} = 0.731$ (using Eq. (1)).

3.5 Generating Approximate Match

After the primary heap *PH* is populated, the pair $\langle u, v \rangle$ with the largest χ^2 is extracted from *PH*. It seeds the further subgraph search and is marked "done" to avoid duplicate and conflicting pairing. Any triplet containing either u or v as one of the vertices cannot be matched any further. This ensures that the same region is not explored more than once.

Next, a *secondary* max-heap *SH* is created to expand the match region neighborhood. Triplets centered at the adjacent vertices of u and v are created and inserted into *SH* along with their χ^2 values. The best vertex pair according to its χ^2 value from these adjacent vertex pairs is then extracted from *SH*; the corresponding vertices are marked as "done", and related triplets are deleted.

Assume that for the example, $\langle v_2, q_2 \rangle$ exhibits the maximum χ^2 value. It is extracted from *PH* and vertices v_2 and q_2 are marked as "done". The top-1 matching subgraph `Match(1)` is initialized to $\{(v_2, q_2)\}$. Its neighbor sets $\{v_1, v_3, v_4\}$, and $\{q_1, q_3, q_4\}$ are examined. The similar vertex pairs $\{\langle v_1, q_1 \rangle, \langle v_1, q_4 \rangle, \langle v_4, q_1 \rangle, \langle v_4, q_4 \rangle, \langle v_3, q_3 \rangle\}$ along with their χ^2 values are inserted into *SH* as 3-tuples. From *SH*, the highest scoring pair $\langle v_1, q_1, \chi^2_{\langle v_1, q_1 \rangle} \rangle$ is extracted and `Match(1)` is expanded to $\{(v_2, q_2), (v_1, q_1)\}$.

The process is repeated till Q is exhausted, i.e., the size of the subgraph match becomes equal to the size of Q, or, *SH* becomes empty. The subgraph of G thus matched, is reported as the *top-1 approximate matching subgraph* to Q.

To retrieve more approximate matches in case of a top-k query, the secondary heap is reset and the process starts again by picking up the currently best unmarked vertex pair from the primary heap *PH*. The procedure is then repeated till k matches are obtained.

3.6 Analysis

The running time of VELSET mainly depends on the vertex pair construction and the heap operations. For input graph G containing n vertices and query graph Q comprising p vertices, the inverted index construction for Q requires $O(p)$ time (and $O(n)$ *offline* time for G). Assuming that the label matching operation, i.e., computing Jaccard similarity, can be performed in $O(1)$ time, the time to find similar labels is $|IL^Q| \times |IL^G|$. In the worst case, every label in IL^Q is similar to every label in IL^G and, therefore, the total number of vertex pairs is $\nu = O(n.p)$. Since each vertex pair is inserted at most once into the primary and secondary heaps, the heap operations require $O(\nu \log \nu)$ time. Since the process is repeated k times, the total time for VELSET is $O(k.\nu \log \nu)$.

The index construction time for VELSET for building the inverted list for G is $O(n)$, and the space required is $O(n)$. During running time, the extra space consumed by VELSET is $O(p)$ for the inverted index of Q and $O(\nu)$ for the max-heaps.

4. THE NAGA ALGORITHM

The construction of vertex pairs in VELSET using label similarities is a double-edged sword: while it provides robustness in detecting approximate subgraphs (noisy scenarios) with high accuracy, it also makes the algorithm computationally expensive. To cater to real-time applications favoring faster running times with a less stringent criteria for accuracy, we propose *Neighbor-Aware Greedy Algorithm* or *NAGA* (pseudocode in Algorithm 2).

Algorithm 2 NAGA

Input: Database graph G, Query graph Q, Number of matches k
Output: Top-k subgraphs of G similar to Q

Compute IL^G and LNL^G of input graph G ▷ Offline
Compute IL^Q and LNL^Q of query graph Q
$VP \leftarrow \{\langle u, v\rangle \mid u \in G \wedge v \in Q \wedge l_u^G = l_v^Q\}$
Primary max-heap $PH \leftarrow \Phi$
for all $\langle u, v\rangle \in VP$ **do**
  $S_u, S_v \leftarrow$ label strings of LNL_u, LNL_v respectively
  Initialize O_u to empty string of length $len_v = |LNL_v| - 1$
  for all window $w_v \in S_v$ **do**
    $w_u \leftarrow$ largest overlapping match of w_v
    Symbol $s \leftarrow$ match of w_u with w_v ▷ s is s_2, s_1 or s_0
    Append O_u with s
  end for
  $E_u \leftarrow len_v \times [Pr(s_0), Pr(s_1), Pr(s_2)]$ ▷ Eqs. (13) – (15)
  Compute $\chi^2_{\langle u,v\rangle}$ using expected E_u and observed O_u ▷ Eq. (1)
  Insert $\langle u, v, \chi^2_{\langle u,v\rangle}\rangle$ into PH
end for
for $i = 1$ to k **do**
  repeat
    $\langle u, v\rangle \leftarrow$ Extract(PH)
  until neither u nor v are marked "done"
  `Match(i)` $\leftarrow \langle u, v\rangle$
  Mark u and v as "done"
  Secondary max-heap $SH \leftarrow \Phi$
  $U', V' \leftarrow$ adjacent nodes of u, v respectively
  Insert $\{\langle u' \in U', v' \in V', \chi^2_{\langle u',v'\rangle}\rangle \mid l^G_{u'} = l^Q_{v'}\}$ into SH
  while $|$`Match(i)`$| \neq |Q|$ and $SH \neq \Phi$ **do**
    Expand `Match(i)` by "best" unmarked vertex pair $\langle u', v'\rangle$ from SH
    Insert unmarked vertex pairs from neighborhood of u', v' into secondary heap SH
  end while
end for
return Top-k matches `Match(1), ..., Match(k)`

The steps in NAGA broadly follow the same outline as VELSET. It primarily differs in label similarity computation and the manner in which the neighborhood labels of a vertex are modeled.

4.1 Creation of Different Lists

The creation of inverted lists for input graph G and query graph Q is exactly the same as VELSET (Sec. 3.2).

4.2 Vertex Pair Construction

The first difference of this approach is that, unlike VELSET, NAGA considers only those vertex pairs that have the *exact* same label for the vertex pair construction:

$$VP = \{\langle u, v\rangle \mid u \in G \wedge v \in Q \wedge l_u^G = l_v^Q\} \tag{9}$$

This reduces the cost of computing and storing *VP* considerably. The vertex pairs thus obtained for the example in Fig. 2 are $VP = \{\langle v_1, q_1\rangle, \langle v_1, q_4\rangle, \ldots, \}$; it does *not* include $\langle v_4, q_1\rangle$ and $\langle v_4, q_4\rangle$.

4.3 Statistical Significance Computation

The next step is to choose the *most statistically significant* vertex pair from *VP*. NAGA takes a different approach from VELSET by considering all the labels in the neighborhood of the candidate vertex *together*. The different steps involved are:

(1) Choosing Vertex Pairs: The vertex pair $\langle u, v\rangle$ that provides the *maximal* label overlap between the vertex neighbor lists LNL_u^G and LNL_v^Q is inserted into the primary heap *PH* with its χ^2 value. However, to consider the intersection, the lists are considered as *sets*. Thus, $LNL_{q_2} = \{A, A, B\}$ is modified into set $LNL'_{q_2} = \{A, B\}$. Since vertex labels rarely repeat in real graphs such as knowledge graphs, the reduction is rarely significant.

The query vertex q_2 in Fig. 2 is involved in two vertex pairs with v_2 and v_5. Since the intersection of $LNL'_{q_2} = \{A, B\}$ with $LNL'_{v_2} = \{A, B, EA\}$ is larger than that with $LNL'_{v_5} = \{D, EA\}$, q_2 is matched with v_2, and $\langle v_2, q_2\rangle$ is inserted into *PH*. All other vertex pairs of the form $\langle u', v\rangle (u' \neq u)$ are removed from *VP* so that each $v \in Q$ is matched only once. Hence, $\langle v_5, q_2\rangle$ is pruned.

(2) Matching: The χ^2 value of each vertex pair in *PH* is computed by considering the entire label neighborhood as a *string* with the labels arranged alphabetically. The neighborhood label string LNL_v of $v \in Q$ is queried over the neighborhood label string LNL_u of $u \in G$ using the procedure suggested in [13], as described next.

Both the strings LNL_u and LNL_v are first divided into sliding windows of size 2. Consider the window $l_x^G l_y^G$ from LNL_u. Since it pertains to the vertex $u \in G$, it is augmented with l_u^G. Thus, the 3-window label set considered is l_x^G, l_u^G, l_y^G. Similarly, considering labels $l_a^Q l_b^Q$ from LNL_v, the label window l_a^Q, l_v^Q, l_b^Q is obtained.

For each window in LNL_u, the best matching window in LNL_v is found. Depending on label overlap, the match is characterized into 3 symbols s_2, s_1, and s_0 (note that $l_u^G = l_v^Q$ in NAGA):

- s_2: All the labels in the window match.

$$s_2 : (l_x^G = l_a^Q \wedge l_y^G = l_b^Q) \tag{10}$$

- s_1: Only one of the labels in the window matches.

$$s_1 : (l_x^G = l_a^Q \wedge l_y^G \neq l_b^Q) \vee (l_x^G \neq l_a^Q \wedge l_y^G = l_b^Q) \tag{11}$$

- s_0: None of the labels match.

$$s_0 : (l_x^G \neq l_a^Q \wedge l_y^G \neq l_b^Q) \tag{12}$$

As in VELSET, a window triplet match assumes the symbol with the largest matching, i.e., the preference order is $s_2 \succ s_1 \succ s_0$.

Consider the example vertex pair $\langle v_2, q_2\rangle$. The label neighbor lists obtained are respectively $LNL_{v_2} = \{A, B, EA\}$ and $LNL_{q_2} = \{A, A, B\}$. The first window of v_2 is $\{A, B\}$, and is best matched with the query window $\{A, B\}$ of q_2, yielding the symbol s_2. Similarly, the next window $\{B, EA\}$ of v_2 is best matched with $\{A, B\}$ of q_2 to produce s_1 (only one label matched). Similar to VELSET, the obtained symbols are cast as a match sequence for the vertex u. Thus, the observed symbol vector for $\langle v_2, q_2\rangle$ is $\{s_2, s_1\}$.

Note that the same query window is used to match with both the input windows. This apparent anomaly is resolved later when the neighborhood is expanded and matched.

If query Q has less than 3 vertices, as in VELSET, we assign pseudo-vertices with dummy labels that match any other label.

(3) Expected Symbol Occurrence: The expected probabilities of the symbols are computed in the same manner as in VELSET. Since no label is similar to any other label except itself in NAGA, $L = |\Sigma|$, i.e., the label alphabet cardinality. Thus, for W possible windows, the probabilities are,

$$Pr(s_0) = \left((1 - 1/|\Sigma|)^W\right)^2 \tag{13}$$

$$Pr(s_1) = 2.\left(1 - (1 - 1/|\Sigma|)^W\right).\left((1 - 1/|\Sigma|)^W\right) \tag{14}$$

$$Pr(s_2) = \left(1 - (1 - 1/|\Sigma|)^W\right)^2 \tag{15}$$

The number of windows in $v \in Q$ denotes the cardinality of the symbol sequence for the candidate vertex u. Hence, for vertex pair $\langle u, v\rangle$, u is represented by a sequence of $len_v = |LNL_v| - 1$ symbols, where $|LNL_v|$ is the length of the neighboring list of v.

(4) Chi-Square Statistic Computation: The above computed symbol probabilities are multiplied by len_v to obtain the *expected oc-*

currence counts of the symbols. The overlap between the label lists LNL_u^G and LNL_v^Q provides the actual *observed* occurrences for the symbols. Using these, the χ^2 value is computed (from Eq. (1)). The 3-tuple $\langle u, v, \chi^2_{uv} \rangle$ is pushed into the primary heap, *PH*, as a candidate. This procedure is performed for all the vertex pairs.

Returning to our example, the observed symbol count for $\langle v_2, q_2 \rangle$ is $O[l_{v_2}] = [0, 1, 1]$ corresponding to s_0, s_1, s_2 respectively. Using Eq. (13) to (15), the expected symbol counts ($W = 2$, $|\Sigma| = 5$) are $E[l_{v_2}] = 2 \times [0.4096, 0.4608, 0.1296]$. The chi-square statistical significance, therefore, is $\chi^2_{\langle v_2, q_2 \rangle} = 2.9424$.

4.4 Generating Approximate Match

The approximate subgraph generation process for NAGA is similar to VELSET, albeit with one significant difference; it greedily considers only the adjacent vertices present in the neighborhood of both G and Q (with exact label match constraint).

The top match `Match(1)` is initialized with the pair $\langle u, v \rangle$ having the maximum χ^2 value. An empty secondary max-heap *SH* is initialized, and the vertices u and v are marked as "done". The neighborhoods of u and v are then explored by visiting their adjacent vertices. Vertex pairs of the form $\langle x, y \rangle$ with x and y having the same label, and adjacent to u and v respectively, are constructed. The χ^2 values for the new vertex pairs are computed and the corresponding 3-tuples are inserted into *SH*. The tuple with the maximum χ^2 value is next extracted from *SH*, the corresponding pair is added to `Match(1)`, and neighbors of the newly extracted vertices are visited. The procedure is iterated till Q is exhausted or *SH* is empty. `Match(1)` is reported as the best approximate matching subgraph.

To obtain top-k matches, *SH* is reset, the next best candidate from *PH* is extracted, and the process is repeated.

4.5 Analysis

Although the worst-case theoretical time complexity of NAGA remains the same as that of VELSET, the use of exact label match during vertex pair generation along with overlapped neighborhood search for subgraph matching significantly reduces the number of candidate vertex pairs constructed by NAGA, thereby leading to lesser heap operations during the subsequent operational stages. Hence, overall, NAGA is much faster than VELSET. Although NAGA can withstand structural noise robustly, it is susceptible to noisy vertex labels due to the constraint of exact label matching.

In effect, NAGA provides an efficient "light-weight" approach for approximate subgraph matching for real-time applications as compared to VELSET. Hence, the combination of them caters to varied applications with diverse characteristics.

4.6 Missing Labels and Edge Labels

We now briefly discuss possible modifications to VELSET and NAGA for applicability to different input scenarios.

The graphs may suffer from *missing labels* due to unknown values or garbled information obtained during the graph construction. Both VELSET and NAGA handle that by matching only the vertices with *available* labels. The subsequent adjacent vertices with missing labels are considered to match any other label and will, thus, match any vertex for insertion into the secondary heap.

If the graphs contain *labeled edges*, the algorithms can be adapted by augmenting the label alphabet set with the edge labels. Thereafter, the graphs are modified by replacing the labeled edge by inserting an extra vertex, with its label as the original edge label, in between the pair of vertices connected by the edge. In this manner, the entire graph is converted to have only vertex labels.

5. EXPERIMENTAL RESULTS

5.1 Experimental Setup

Datasets: We use the following two large publicly available knowledge graph corpora for our evaluation:

- *YAGO Entity Relationship Graph* (http://www.yago-knowledge.org/): This dataset provides a knowledge graph [39] containing named-entities (i.e., persons, organizations, etc.) harvested from Wikipedia and the inter-relations linking the entities. The underlying graph, stored as RDF triples (subject-predicate-object) [26], contains 12,811,149 vertices and 18,282,215 edges with 9,505,202 unique named entities forming the node labels.
- *IMDB Network* (http://www.imdb.com/interfaces): The *Internet Movie Database* contains named-entities as movies, actors, etc., along with their relationships represented as RDF triples comprising of 2,932,657 vertices and 11,040,166 edges. The entities are considered as vertex labels (and are, thus, unique), while the type information for each vertex was ignored in our setup.

Query Generation: The queries were generated by extracting subgraphs from the target graph and subsequently introducing noise:
(1) *structural noise*: a small fraction (up to 33%) of edges in the subgraph were randomly and uniformly inserted or deleted,
(2) *label noise*: a small fraction (up to 33%) of vertex labels were randomly and uniformly modified by insertion of random strings.

We thus benchmark the performance of the algorithms across four different query scenarios:
(a) exact matching subgraphs with no noise (*Exact Match*),
(b) subgraphs with structural noise only (*Noisy Edges*),
(c) subgraphs with label noise only (*Noisy Labels*), and
(d) subgraphs with both structural and label noise (*Combined*).

For each dataset and for each query scenario, we generated 20 random queries with the following query sizes: $3, 5, 7, 9, 11, 13$.

Competing Methods: We compare our proposed algorithms against two state-of-the-art approaches that claim to the best in running time and/or accuracy: (1) NeMa [24], and (2) SIM-T [27].

Quality Evaluation: For each query graph Q, the original subgraph of G perturbing which Q was generated, was considered as the "ground truth" of the best match. Although other subgraphs of G might match a query subgraph, the probability of such chance matches under random noise insertion is quite low. Further, since there is no way to ascertain the ground truth for second or later best subgraph matches, the quality results were assessed only for $k = 1$. We report *precision* (P), *recall* (R), and *F1 score* (F1) for the algorithms. The Jaccard similarity threshold was fixed to $\tau = 0.5$.

Setup: The algorithms were implemented in C++ and experiments conducted on 64-bit 4-core 2.4GHz Intel Xeon with 512GB RAM.

5.2 Comparison Results

We use the aforementioned 480 $(4 \times 6 \times 20)$ queries to compare against the two benchmark algorithms for both the datasets.

Fig. 4 and Fig. 5 depict the obtained results. Table 1 and Table 2 summarize the overall averaged results over the different query sizes, and also show the individual precision recall, and F1 scores.

Exact Match: When an *exact subgraph* of G is posed as a query, both VELSET and NAGA exhibit very high accuracy. This happens as exact vertex matches have the largest χ^2 values. NeMa also shows good accuracy although SIM-T suffers a bit. The running time increases with the query size since the number of neighborhood candidate regions increases in a super-linear manner with the number of vertices. For the IMDB dataset, NAGA always finds the best match and has, therefore, a 100% F1 score. NAGA is also the fastest algorithm, although for larger query sizes in YAGO, NeMa is seen to be slightly better. Overall, NAGA performs the best in terms of both running time and quality as evident from the tables.

Table 1: Average F1 score (%) and running time results over varying graph matching queries for YAGO dataset.

Query Scenario	Quality Performance												Running Time Performance (in sec)			
	NeMa			SIM-T			VELSET			NAGA			NeMa	SIM-T	VELSET	NAGA
	P	R	F1	P	R	F1	P	R	F1	P	R	F1				
Exact Match	99.11	86.08	92.14	82.19	82.19	82.19	95.24	95.24	95.24	98.91	98.91	**98.91**	224.94	564.83	275.99	**177.58**
Noisy Edges	99.57	86.46	92.55	86.40	86.40	86.40	96.63	96.63	96.63	97.32	97.32	**97.32**	226.14	644.25	272.22	**185.70**
Noisy Labels	98.61	84.62	91.08	68.72	68.72	68.72	95.26	95.26	**95.26**	97.32	52.78	68.44	226.67	544.70	262.69	**163.96**
Combined	99.55	86.59	92.62	71.63	71.63	71.63	94.67	94.67	**94.67**	95.64	53.18	68.36	225.48	631.97	248.19	**196.70**
Average	**99.21**	85.94	92.10	77.23	77.23	77.23	95.45	**95.45**	**95.45**	97.30	75.55	85.06	225.81	596.44	264.77	**180.98**

Table 2: Average F1 score (%) and running time results over varying graph matching queries for IMDB dataset.

Query Scenario	Quality Performance												Running Time Performance (in sec)			
	NeMa			SIM-T			VELSET			NAGA			NeMa	SIM-T	VELSET	NAGA
	P	R	F1	P	R	F1	P	R	F1	P	R	F1				
Exact Match	92.70	92.31	92.50	80.29	80.29	80.29	95.95	95.95	95.95	100	100	**100**	29.43	152.75	43.32	**7.03**
Noisy Edges	91.22	90.98	91.10	79.68	79.68	79.68	98.94	98.94	98.94	100	100	**100**	29.50	170.18	42.35	**6.98**
Noisy Labels	92.65	62.16	74.40	60.31	60.31	60.31	98.90	98.90	**98.90**	100	53.73	69.90	29.33	153.02	42.79	**6.34**
Combined	91.44	58.61	71.43	63.48	63.48	63.48	97.84	97.84	**97.84**	98.33	45.92	62.61	28.98	171.76	45.31	**6.51**
Average	92.00	76.01	83.25	70.94	70.94	70.94	97.91	**97.91**	**97.91**	**99.58**	74.91	85.50	29.31	161.93	43.44	**6.71**

Table 3: Overall performance synopsis of the algorithms.

Methods	YAGO			IMDB		
	F1 (%)	Running Time (s)	Indexing Time (s)	F1 (%)	Running Time (s)	Indexing Time (s)
VELSET	**95.45**	264.77	1,205	**97.91**	43.44	209
NAGA	85.06	**180.98**	**250**	85.50	**6.71**	132
NeMa	92.10	225.81	∼10,000	83.25	29.31	∼10,000
SIM-T	77.23	596.44	374	70.94	161.93	**107**

Noisy Edges: When only structural mismatch is introduced in the form of *noisy edges*, NAGA again performs the best. Interestingly, it still maintains an F1 of 100% for IMDB 97% for YAGO. This establishes the robustness of the formulated chi-square measure in extracting the best regions. Since in IMDB, the labels are unique and, therefore, there is only a slight increase in the number of candidate vertices with increase in query size, NAGA exhibits almost constant running time. The results also show that the overtly conservative strategy of generating all vertex pairs that have similar labels (and not strictly the same) in VELSET is an overkill when there are no noisy labels. In fact, it may lead to extraction of wrong subgraph regions. NeMa is reasonably accurate and fast as well.

Noisy Labels: However, the situation is quite different when *noisy labels* are introduced but the structure is unaltered. VELSET outperforms NAGA by a large margin for both the datasets. NAGA is much faster than NeMa. VELSET is much better overall with only slightly more running time than NeMa. SIM-T does not seem to perform well for any dataset. For YAGO, NeMa shows reasonable accuracy. However, for IMDB, NeMa performs much worse as all the labels in IMDB are unique and it fails to leverage the closeness of labels in the query to that in the input graph. The recall of NAGA suffers since it has a strict label equality requirement and, hence, refuses to examine nodes with even a slight label mismatch. Consequently, it misses out on good regions. As a result, the F1 score is low as well. However, the candidate regions returned by NAGA are always good, thereby letting it achieve high precision.

Combined: The fourth and final query scenario where both structural and label noise are *combined* show similar trends: NAGA is extremely fast but has poor recall while VELSET has very high accuracy and has comparable running times with NeMa. SIM-T performs poorly in terms of both running time and quality. Thus, VELSET is able to handle both types of errors effectively.

Overall Comparison: The overall synopsis of the comparisons is shown in Table 3. It is clear that VELSET is the most *accurate* heuristic. The comparative advantage is larger for IMDB than YAGO due to larger fraction of unique labels. VELSET achieves close to 95% F1 score or more in all the datasets and query scenarios. NAGA achieves more than 85% F1 score for both the datasets and is comparable with NeMa for IMDB. Strangely enough, the tabu search based SIM-T (designed for such applications) fails to achieve a high quality. This may be due to its dependency on the initial solution guess which, if bad, does not allow it to recover.

When running times are compared, NAGA is undoubtedly the fastest. For the million scale IMDB dataset, it produces results within 7 seconds. NeMa is 4 times slower while VELSET requires 7 times more running time. SIM-T is orders of magnitude slower. On YAGO, NAGA is again the most efficient, and runs 1.25 times faster than NeMa. VELSET is slightly slower than NeMa. SIM-T is slower due to the inherent complexity of the tabu search method.

Table 3 also shows the indexing time required for the different algorithms. The only index information required for VELSET and NAGA are the lists created from the input graph. Consequently, the pre-processing time spent offline is quite low. This also imparts an *update-friendly* nature to these proposed algorithms. Any change in the input graph in either the structure or the labels including addition and deletion of nodes and edges can be quickly absorbed by both VELSET and NAGA. VELSET requires more pre-processing time than NAGA due to computation of Jaccard similarities to ascertain which labels are considered similar. SIM-T does not require much indexing either. NeMa, on the other hand, builds an elaborate index based on the two-hop neighborhood of each vertex, thereby requiring a substantial amount of indexing time.

Finally, the index sizes required by the different algorithms are ∼5GB for NeMa, ∼8GB for SIM-T, and ∼10GB for VELSET and NAGA. Hence, all of them are good enough to run in standard commodity machines with 16GB of RAM or more.

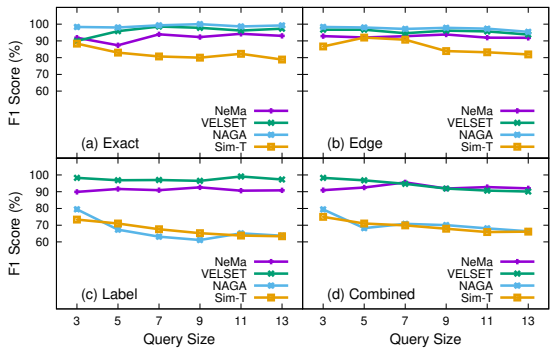
Summary: The above empirical results help establish the following insights. If accuracy is of paramount importance, the VELSET algorithm should be chosen. However, if there is no fuzziness in terms of labels and only structural mismatches can happen, or runtime is the decisive criterion, NAGA should be employed. Thus, our suite of algorithms, VELSET and NAGA, cater to different application scenarios and can be selected based on need.

5.3 Effect of Parameters

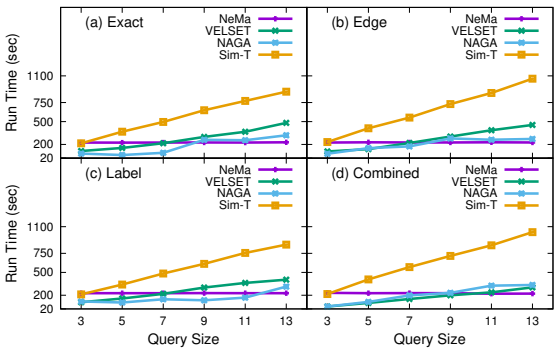
We next explored the performance of VELSET and NAGA with variations in parameter settings. Results on the IMDB dataset were averaged over 120 randomly generated queries of varying lengths ranging from 3 to 13 uniformly across the different query scenarios.

Top-k: We varied the number of *top-k* approximate subgraph matches. Fig. 6(a) shows the results. The running time of both VELSET and NAGA shows a linear increase with k, due to the iterative nature of extracting the top-k matches. The time spent on initial candidate generation is constant irrespective of k. NAGA shows better scalability than VELSET since the sizes of its secondary heaps are smaller due to more stringent label matching criterion.

Label Similarity Threshold: The label similarity threshold, τ, controls the number of candidate vertex pairs generated for VELSET. (NAGA does not use τ.) A low value of τ provides an aggressive approach, whereby even weakly matching vertex labels are considered as candidates, thereby leading to the generation of large

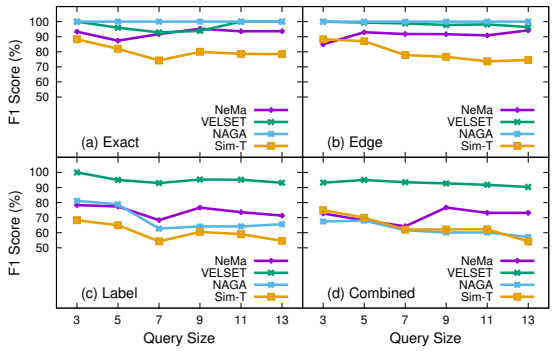


(i) Quality

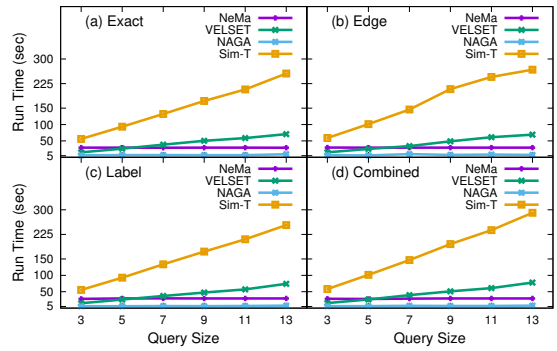


(ii) Running time

Figure 4: YAGO dataset: (i) F1 score and (ii) Running time performance for varying query sizes for different query scenarios.

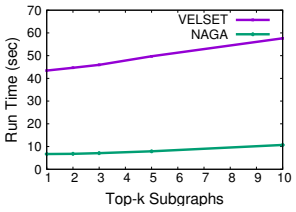


(i) Quality

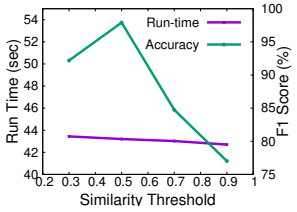


(ii) Running time

Figure 5: IMDB dataset: (i) F1 score and (ii) Running time performance for varying query sizes for different query scenarios.

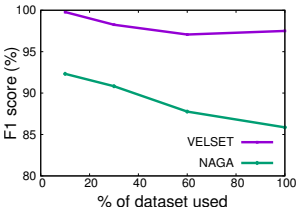


(a) Effect of k

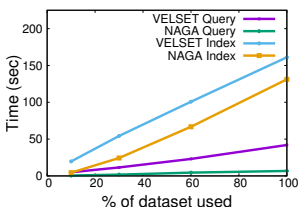


(b) Effect of τ

Figure 6: Performance on IMDB dataset: Effect of (a) size of answer set, k, on running time, (b) similarity threshold τ on VELSET.



(a) Quality

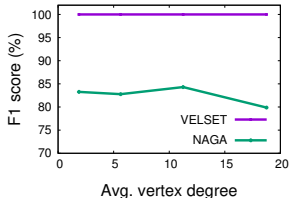


(b) Running time

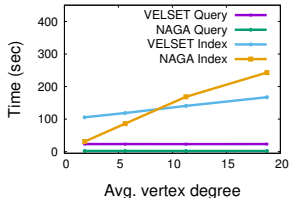
Figure 7: Scalability results on IMDB dataset.

number of candidate vertex pairs. A large τ has the opposite effect. Hence, the running time of VELSET decreases with τ, as shown in Fig. 6(b). Smaller values of τ, however, affect the quality since too many vertex candidate pairs are generated. On the other hand, a large value of τ does not allow enough candidate pairs to be explored. An extreme $\tau = 1$ ensures that the labels must be exactly the same, making VELSET behave similar to NAGA. Empirically, $\tau = 0.5$ provided the best balance, and was hence set as default.

Scalability: We next evaluated the impact of varying input graph sizes. We extracted different sized datasets from IMDB by randomly sampling 10% to 100% of its vertices and the associated edges. With increase in size, the number of matching vertices increases, thereby generating more vertex pairs and increasing the



(a) Quality



(b) Running time

Figure 8: Robustness results on dense Pokec dataset.

running time, as shown in Fig. 7. The increments in running time are sub-linear, thereby showcasing the scalability of the algorithms.

Density: To evaluate the robustness of the proposed algorithms in tackling denser graphs, we used the *Pokec social network* from snap.stanford.edu/data/soc-pokec.html. The network consists of 1,632,803 vertices (with unique labels) and 30,622,564 friendship links representing edges. We created multiple datasets by varying the average vertex degree by randomly sampling 10% to 100% of the edges while retaining all the vertices. The results are shown in Fig. 8. We observed the density of the input graph to have little impact on the quality of the algorithms. VELSET depicts an accuracy of 100%, while NAGA attains a slightly lower accuracy. Interestingly, although the number of neighbors per vertex increased, the density of the graph had only a slight impact on query runtime.

6. CONCLUSIONS

In this paper, we introduced the use of statistical significance to efficiently discover the best approximate subgraphs matching a query. We proposed two algorithms, VELSET and NAGA, tuned towards accuracy and efficiency respectively, to cater to different application settings. We experimentally showed them to outperform existing state-of-the-art algorithms on real-life large datasets.

In future, we plan to study the applicability of statistical significance model to other graph problems.

7. REFERENCES

[1] C. C. Aggarwal and H. Wang. *Managing and Mining Graph Data*, chapter Graph Data Management and Mining: A Survey of Algorithms and Applications, pages 13–68. Springer, 2010.

[2] A. Arora, M. Sachan, and A. Bhattacharya. Mining statistically significant connected subgraphs in vertex labeled graphs. In *SIGMOD*, pages 1003–1014, 2014.

[3] L. Babai. Graph isomorphism in quasipolynomial time. In *STOC*, pages 684–697, 2016.

[4] P. Barcelo, L. Libkin, and J. L. Reutter. Querying graph patterns. In *PODS*, pages 199–210, 2011.

[5] G. Bejerano, N. Friedman, and N. Tishby. Efficient exact p-value computation for small sample, sparse and surprisingly categorical data. *JCB*, 11(5):867–886, 2004.

[6] A. Bordes, S. Chopra, and J. Weston. Question answering with subgraph embeddings. In *EMNLP*, pages 615–620, 2014.

[7] C. Chen, X. Yan, P. S. Yu, J. Han, D. qing Zhang, and X. Gu. Towards graph containment search and indexing. In *VLDB*, pages 926–937, 2007.

[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] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298, 2004.

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

[11] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for large graphs. *PAMI*, 26(10):1367–1372, 2004.

[12] M. K. Das and H. K. Dai. A survey of DNA motif finding algorithms. *BMC Bioinf.*, 8(7):1–13, 2007.

[13] S. Dutta. MIST: Top-k approximate sub-string mining using triplet statistical significance. In *ECIR*, pages 284–290, 2015.

[14] S. Dutta and A. Bhattacharya. Most significant substring mining based on chi-square measure. In *PAKDD*, pages 319–327, 2010.

[15] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu. Incremental graph pattern matching. In *SIGMOD*, pages 925–936, 2011.

[16] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. *PVLDB*, 3(1-2):264–275, 2010.

[17] B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. In *AAAI*, pages 45–53, 2006.

[18] R. Giugno and D. Shasha. GraphGrep: A fast and universal method for querying graphs. *ICPR*, 2:201–212, 2002.

[19] W.-S. Han, J. Lee, M.-D. Pham, and J. X. Yu. iGraph: A framework for comparisons of disk-based graph indexing techniques. *PVLDB*, 3(1-2):449–459, 2010.

[20] B. Hixon, P. Clark, and H. Hajishirzi. Learning knowledge graphs for question answering through conversational dialog. In *NAACL*, pages 851–861, 2015.

[21] Y. Jia, J. Zhang, and J. Huan. An efficient graph-mining method for complicated and noisy data with real-world applications. *Knowledge and Information Systems*, 28(2):423–447, 2011.

[22] H. Jiang, H. Wang, P. S. Yu, and S. Zhou. GString: A novel approach for efficient search in graph dbs. In *ICDE*, pages 566–575, 2007.

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

[24] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. NeMa: Fast graph search with label similarity. *VLDB*, 6(3):181–192, 2013.

[25] A. Khan, X. Yan, and K.-L. Wu. Towards proximity pattern mining in large graphs. *SIGMOD*, pages 867–878, 2010.

[26] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. http://www.citeulike.org/group/2170/article/532408, 2006.

[27] S. Kpodjedo, P. Galinier, and G. Antoniol. Using local similarity measures to efficiently address approximate graph matching. *Disc. Appl. Math.*, 164:161–177, 2014.

[28] M. Kuramochi and G. Karypis. Frequent subgraph discovery. *ICDM*, pages 313–320, 2001.

[29] M. Kuramochi and G. Karypis. GREW – A scalable frequent subgraph discovery algorithm. *ICDM*, pages 439–442, 2004.

[30] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Mining & Know. Disc.*, 11(3):243–271, 2005.

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

[32] M. Mongiovi, R. Di Natale, R. Guigno, A. Pulvirenti, A. Ferro, and R. Sharan. SIGMA: A set-cover-based inexact graph matching algorithm. *JBCB*, 8(2):199–218, 2010.

[33] T. Read and N. Cressie. Pearson's χ^2 and the likelihood ratio statistic G^2: a comparative review. *Int. Statistical Review*, 57(1):19–43, 1989.

[34] T. R. C. Read and N. A. C. Cressie. *Goodness-of-fit statistics for discrete multivariate data*. Springer Series in Statistics, 1988.

[35] M. Sachan and A. Bhattacharya. Mining statistically significant substrings using the χ^2 statistic. *PVLDB*, 5(10):1052–1063, 2012.

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

[37] R. Shen and C. Guda. Applied graph-mining algorithms to study biomolecular interaction networks. *BioMed Research Int.*, 2014(article 439476):11, 2014.

[38] 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.

[39] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[40] B. Sun. *Mining, Indexing, and Search Approach to Entity and Graph Information Retrieval for Chemoinformatics*. PhD thesis, Pennsylvania State Univ., 2008.

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

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

[43] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

[44] 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.

[45] W. H. Tsai and K. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern recognition. *IEEE Systems, Man, and Cybernetics*, 9(12):757–768, 1979.

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

[47] F. Vandin, E. Upfal, and B. J. Raphael. Algorithms for detecting significantly mutated pathways in cancer. *JCB*, 18(3):507–522, 2011.

[48] J. Wang, N. Ntarmos, and P. Triantafillou. Indexing query graphs to speedup graph query processing. In *EDBT*, pages 41–52, 2016.

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

[50] X. Yan, P. S. Yu, and J. Han. Graph indexing based on discriminative frequent structure analysis. *TODS*, 30(4):960–993, 2005.

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

[52] S. Zhang, J. Li, H. Gao, and Z. Zou. A novel approach for efficient supergraph query processing on graph databases. In *EDBT*, pages 204–215, 2009.

[53] S. Zhang, S. Li, and J. Yang. GADDI: Distance index based subgraph matching in biological networks. In *EDBT*, pages 192–203, 2009.

[54] S. Zhang, J. Yang, and W. Jin. SAPPER: Subgraph indexing and approximate matching in large graphs. *PVLDB*, 3:1185–1194, 2010.

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

[56] Q. Zou, S. Liu, and W. W. Chu. Ctree: A compact tree for indexing XML data. In *WIDM*, pages 39–46, 2004.