ChiSeL: Graph Similarity Search using Chi-Squared Statistics in Large Probabilistic Graphs

Shubhangi Agarwal
Computer Sc. & Engg.
Indian Institute of Technology
Kanpur, India
sagarwal@cse.iitk.ac.in

Sourav Dutta
Huawei Research Centre
Dublin
Ireland
sourav.dutta2@huawei.com

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

ABSTRACT

Subgraph querying is one of the most important primitives in many applications. Although the field is well studied for deterministic graphs, in many situations, the graphs are probabilistic in nature. In this paper, we address the problem of subgraph querying in large probabilistic labeled graphs. We employ a novel algorithmic framework, called ChiSeL, that uses the idea of statistical significance for approximate subgraph matching on uncertain graphs that have uncertainty in edges. For each candidate matching vertex in the target graph that matches a query vertex, we compute its statistical significance using the chi-squared statistic. The search algorithm then proceeds in a greedy manner by exploring the vertex neighbors having the largest chi-square score. In addition to edge uncertainty, we also show how ChiSeL can handle uncertainty in labels and/or vertices. Experiments on large real-life graphs show the efficiency and effectiveness of our algorithm.

PVLDB Reference Format:
Shubhangi Agarwal, Sourav Dutta, Arnab Bhattacharya. ChiSeL: Graph Similarity Search using Chi-Squared Statistics in Large Probabilistic Graphs. *PVLDB*, 13(10): 1654-1668, 2020.
DOI: https://doi.org/10.14778/3401960.3401964

Keywords

Probabilistic Graph; Subgraph Query; Chi-Square Statistic; Statistical Significance; Greedy Neighborhood Search

1. INTRODUCTION

Motivation. The current impetus on Internet-of-Things (IoT) and hyper-connectivity for future cities, factories and organizations has fueled a dramatic proliferation of linked open data, social communities and inter-connected network structures across the World Wide Web. Such large data sources are best represented as *labeled graphs*, where entities are modeled as vertices, relationships and inter-dependencies are captured by edges, and labels define the characteristics of entities and relations. These large graphs form the backbone for several real-life domains such as social networks [1], protein and chemical interaction data [8], bioinformatics [30], route planning [17], etc. Efficient large-scale graph matching techniques, thus, provide a fundamental area of research [63, 80].

Probabilistic Graphs. In real-world scenarios, however, the obtained data is often inherently *uncertain* due to noisy measurements (e.g., missing edges or labels) [2], hardware limitations [6], inference models [28], multiple data-source merging [60], privacy-preserving perturbation [11], etc. For example, pairwise protein interactions are usually derived by statistical models while potential interactions in social networks are based on trust and influence factors [42]. Similarly, confidence values may be associated with extracted facts by information extraction systems [12]. Further, in many scenarios, information is often vague or ambiguous, expressing subjective opinions and judgment concerning market analysis, medical diagnosis or even personal evaluation. Such uncertainties can be classified into three categories [69]:
(1) *identity uncertainty* where an entity is represented by multiple objects or references in the data,
(2) *attribute uncertainty* about the attribute values of entities (e.g., vertex existence or label uncertainty), and
(3) *relationship uncertainty* about whether a particular relationship exists (e.g., edge existence or label uncertainty).

A natural way to capture graph uncertainty is to represent them as *probabilistic graphs* [43, 87], where each entity, relation, or characteristics is associated with a probability quantifying the existence likelihood [33], for cases like representative graphs [79] and in named entity resolution [69]. For example, in automated creation of knowledge bases such as YAGO and DBpedia, extraction of entities, facts and relationships have an associated confidence depending on the veracity of the data source, information extraction technique, and errors in the pipeline. Without loss of generality, in this work, we only consider the existence of *edge uncertainties*, i.e., each relationship between entities has an associated probability or confidence of existence.

Existing literature defines two main representations of edge uncertainty in probabilistic graphs [38, 64]:
(1) *edge-existential model*, where each edge is quantified with a probability indicating the chance of edge existence, and
(2) *weight-distribution model*, where each edge is associated with a probability distribution of weight values.

We adopt the former representation model in this work. Later, in Section 4, we discuss how various other scenarios involving vertex, edge, and label uncertainties can be handled by our algorithm.

Graph Querying. Various modern-day applications like question answering, motif discovery, etc., necessitate efficient approaches for label and structural pattern matching on large graphs [3]. This involves efficient indexing and querying approaches for large graphs. A common well-studied deterministic graph query primitive is to search, within a large graph, for query structures that

Proceedings of the VLDB Endowment, Vol. 13, No. 10
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3401960.3401964

are similar both in terms of labels and structure. This allows one to express related groups of entities with given attributes and link structures as subgraph pattern matching queries.

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 been shown to be quasi-polynomial [7], while subgraph isomorphism is known to be NP-complete [19]. Consequently, efficient and accurate subgraph matching heuristics have been explored [27, 48, 102]. However, due to noisy and non-deterministic nature of probabilistic graphs, traditional approaches applicable to deterministic graph problems cannot be directly applied owing to scalability and accuracy issues. Hence, *approximate graph matching* techniques are required to handle queries in a robust and real-time manner. It has been shown that sometimes methods for probabilistic matches outperform exact matching algorithms [98] in many aspects. An overview of challenges and techniques for various problems related to uncertain graphs can be found in [47]. This work proposes a robust and efficient algorithm for *approximate subgraph matching in large probabilistic graphs*.

Possible World Semantics. One important concept in relation to probabilistic data querying is the *possible world semantics* (PWS). It specifies that the (approximate) matching subgraph (to the query) computed should ideally be the one with the maximum overall existential probability across all the possible combinatorial instances of the uncertain graph considering the probabilities of the entities, relationships, and attributes. Here, each "possible world" is a deterministic instance of the uncertain graph [21, 97] considering the existential probabilities of the graph attributes.

It is important to observe that the exponential nature of the possible world semantics usually renders exact query evaluation computationally prohibitive. In fact, even simple queries become #P-complete on graphs [71, 87]. A common solution is to apply intelligent sampling based techniques (e.g., Monte-Carlo) to assess the query on a subset of possible worlds [71]. However, in general, a large number of samples are required to obtain a good approximation and, for complex queries, such sampling methods become ineffective [71]. We empirically demonstrate this behavior in Section 5.5.4. This necessitates efficient algorithms for subgraph matching in uncertain graphs. In this work, we consider the above PWS concept, and also consider the edge probabilities to be independent, following existing literature [42, 43, 72].

Challenges in PWS. It can be observed that PWS introduces a major challenge for approximate graph matching queries. There exists a trade-off between match similarity (in terms of graph structure and labels) and probability of existence, which is not currently addressed in the literature (to the best of our knowledge).

For example, assume a chain query graph A—B—C—D. Consider the resultant matching subgraph (from an input target graph) to return two competing results: $\langle A$—B—E—$D\rangle$ demonstrating large structural and label similarity and $\langle A$—B—$D\rangle$ having the highest existential probability w.r.t. PWS. In such cases it might be difficult to assess as to which is a better match to q. Current methods aim to tackle such problem by introducing *thresholds* [33, 93, 95], i.e., only results having structural similarity and/or existential probability greater than some thresholds are returned.

We propose an *integrated measure* based on a formulation using statistical significance to capture both the PWS concept and the above trade-off for approximate matching subgraphs – the returned results demonstrate both high structural similarity and a high probability of existence. We also showcase that this approach inherently bypasses the efficiency and scalability issues introduced by the enumeration requirement of PWS.

1.1 Real-World Use-Cases

With the proliferation of *electronic medical records*, one of the current research areas in the medical domain is *symptom-disease-diagnostics knowledge graphs* [75, 88] that capture the relationships between symptoms and diseases. The nodes represent symptoms (e.g., cough, fever, etc.) and diseases (e.g., flu) while the edges provide the probabilities of symptom-disease association. For example, if cough is a common symptom for flu but not malaria, the edge connecting cough to flu will have a high probability, whereas the edge connecting cough to malaria will be either absent (if it is not a symptom at all) or have a very low probability.

An important application scenario using the above knowledge graph is the following. Given a patient's symptoms (modeled as a query graph), an approximate probabilistic subgraph matching algorithm would extract the most probable diseases linked to the current symptoms. The results would be then presented to experienced medical professionals for planning the course of treatment for the patient. The possible worlds semantics (PWS) automatically models the highest overall probability of association between the symptoms and the diseases, while approximate matching helps to take into account symptoms that might have not surfaced yet, or are linked to some other condition. A natural extension of this use case can incorporate symptom-disease-prognosis-drugs-side-effects interactions as captured, for example, in the *patient-disease-drug (PDD) graph* [88]. Thus, effective algorithms for approximate subgraph matching on probabilistic graphs enable better "precision medicinal treatment" and early detection of rare diseases.

Consider also the domain of information extraction and natural language processing for automated question answering platforms such as IBM Watson [13] (`researcher.watson.ibm.com/researcher/view_group.php?id=2099`). Large knowledge graphs providing relationships between entities and concepts form the backbone of such systems. However, automated extraction of entities, facts and relations via web crawling faces challenges in terms of accuracy. For example, the *Never-Ending Language Learning (NELL)* project [66, 67], while learning continuously and incrementally, extracts facts from the Web to populate its created ontology. The dataset is created in a semi-supervised setting and contains millions of beliefs. Each belief triple thus learned automatically has a score associated with it that indicates the system's confidence that it is correct (`rtw.ml.cmu.edu/rtw`). Each belief is modeled as an edge in the graph with the confidence score captured by the probability of the existence of the edge between the corresponding pair of entities. Answering natural language understanding based questions (like in Jeopardy [13]) then involves finding the closest matching subgraph with the highest possible probability of truthfulness and existence of facts. Such a large graph (NELL contains 50 million beliefs) poses the need to be efficiently queried for the user to be able to effectively extract relevant information. Since a query in general retrieves multiple answers, it is further important to rank them. However, the probabilistic nature of the graph makes it challenging for deterministic approaches to determine which result is better and, hence, methods that work directly with probabilistic graphs are preferable.

Probabilistic graph querying is also used regularly in biology. For example, protein-protein interaction networks have been modeled as stochastic graphs, where statistical significance measures have been used for motif discovery [41]. We show case this scenario using actual real life queries in Section 5.7.

1.2 Problem Definition

We consider uncertain graphs, where vertices are associated with labels denoting attributes and edges are characterized by their ex-

istential probabilities. Formally, without loss of generality, we represent an input graph as $G = (V, E, \mathcal{L}, \mathcal{P})$ where V is the set of vertices and E is the set of edges. The vertex labels $\mathcal{L} : V \rightarrow$ strings is a mapping of vertices to labels drawn from a finite set $\mathcal{L} = \{l_1, l_2, \cdots l_L\}$ of cardinality L. The mapping $\mathcal{P} : E \rightarrow [0, 1]$ is defined on the set of edges to obtain the associated probability of existence for each edge. We assume that G is undirected and does not contain any hyper-edge. The query graph, represented by $Q = (V_Q, E_Q, \mathcal{L}_Q)$, specifies the structure and the labels on the vertices. Since users generally specify a query completely, there is no uncertainty in the edges. Without loss of generality, $\mathcal{L}$ is assumed to include the query labels, $\mathcal{L} \supseteq \mathcal{L}_Q$. (Applications with other kinds of probabilistic graphs are discussed in Section 4.)

Given a deterministic query Q and a large probabilistic graph G as above, the problem of *approximate similar subgraph matching (ASSM)* aims to find the best matching subgraphs in G that are similar to Q. Observe that multiple instances of such approximate query graph matching might occur in different parts of the input graph. The top-k results extracted should then be ordered based on their degree of similarity and probability of existence in accordance with the possible world semantics (PWS) concept.

Formally, given a graph G with labeled vertices and probabilistic edges and a deterministic query graph Q, our aim is to find the *top-k statistically significant subgraphs* of G that are the best approximate matches of Q.

1.3 Contributions

In this paper, we propose the *Chi-Square based Search in Large ProbabiListic Graphs* (CHISEL) algorithm to efficiently solve the above approximate subgraph matching (ASSM) problem for large probabilistic graphs. The working of CHISEL hinges on the computation of *statistical significance* scores encoding the degree of similarity of subgraphs in the input graph to the query graph. The underlying principle is that regions in G demonstrating a higher match with Q would be characterized by higher statistical significance scores as the matching cannot be attributed to random chances alone. This provides an efficient searching mechanism for CHISEL.

In a nutshell, our contributions in this paper are:

1. A novel subgraph matching algorithmic framework, based on statistical significance measures, for effectively handling large probabilistic graphs adhering to the possible world semantics;
2. The CHISEL algorithmic framework for efficient approximate subgraph matching on labeled input and query graphs;
3. Extensive experimental evaluation showcasing the enhanced performance of our algorithm, both in terms of run-time and accuracy, over state-of-the-art graph querying approaches on real-life graphs; and
4. Techniques on how various scenarios of uncertainty in graph structures and attributes can be adapted within our algorithm.

2. RELATED WORK

2.1 Deterministic Graph Matching

Graph and subgraph matching provide fundamental primitives for applications pertaining to graph analytics and pattern mining in network structures [18]. Classical studies in this domain include tree-pruning [86], Swift-Index [77] and VF2 [20]. Since they depend majorly on backtracking and tree-search algorithms, they are computationally costly for large modern-day graphs. Using a set of feasibility rules defined in [20] for structural match, the VF2++ algorithm [44] improves over VF2. In [55], tree search method for isomorphism is sped up by another heuristic derived from constraint satisfaction. The NP-completeness of subgraph isomorphism [19] led to efforts towards graph edit distance (GED) measures for exact matches; however, optimal solution for GED was shown to be NP-hard [99]. These approaches used state space representation and feasibility rules for graph isomorphism.

Shang et al. [77] proposed QuickSI for testing, and Swift-Index for filtering using prefix tree indexing. Most methods that target biological networks, such as PathBlast [46], SAGA [81], NetAlign [61] and IsoRank [78], work mostly for small networks. Tsai and Fu [85] proposed an ordered-search algorithm for determining error-correcting isomorphism and pattern classification combining both structural and statistical techniques. GraphGrep [32], a graph querying algorithm, uses hash-based fingerprinting for subsequent filtering. Such *filtering-and-verification* based approaches worked with threshold-based distance computation, identifying common substructures, and candidate fragment computation. Identifying graphs in a graph database containing a query subgraph have also been studied [80]. However, our target applications require the identification of the precise locations in the graph(s) where the best match of the query is found.

In general, such methods involved only exact subgraph matching, whereas this work involves approximate matching to extract the best matching subgraphs.

2.2 Approximate Graph Matching

Subsequently, research in subgraph matching has mostly focused on approximate subgraph similarity matches [3], wherein a small amount of mis-match is tolerable. An efficient index-based approximate subgraph matching tool, TALE [82], uses maximum weighted bipartite graph matching. Different heuristics based on predefined graph distance and radius thresholds [101], set cover (SIGMA) [68], edge edit-distance (SAPPER) [102], regular expressions [9], etc. have been proposed. APGM [39] proposes a method to mine useful patterns from noisy graph databases. C-Tree [106] proposes a generalized representation of graphs that can be used for both subgraph querying and subgraph similarity computation. However, these methods are computationally quite expensive and are, hence, infeasible for modern web-scale graphs. Our proposed method based on statistical significance is considerably faster.

A semantic-based search algorithm using a sequencing method to capture the semantics of the underlying graph data was proposed in GString [40]. S^4 system [92] finds the subgraphs with identical structure and semantically similar entities of query subgraph. Other relevant works in the subgraph matching domain are gIndex [90], FG-Index [16], iGraph [36], Grafil [91], Gcoding [105], GPTree [100], and cIndex [14]. An extensive survey about graph matching algorithms was presented in [18]. Random walks to find the best matching in large graphs were used in [83, 84]. Subgraph matching considering the similarity between objects associated with two matching vertices [104], and a maximum likelihood estimation approach [4] were also proposed.

Recent approaches [5, 27] employ statistical analysis methods to find *statistically significant subgraph* matches that deviate from the expected subgraph pattern significantly. Dutta et al. [27] improve upon NeMa [48] and SIM-T [51] approaches based on neighborhood search. A concise description and comparison of various techniques available for graph matching in large graphs is given by [63]. An interesting survey of existing graph-based structural and pattern matching approaches across diverse applications such as computer vision, biology, networks, etc., has been presented in [29].

These approaches consider deterministic graphs and are difficult to generalize for probabilistic setting wherein varying types of uncertainties might be present. The closest approach to our work in terms of using statistical significance is [27].

Another similar domain of work involves frequent subgraph finding and pattern mining over a set of graphs or a single large graph. FSD [52] uses sparse graph representation for candidate generation and counting. gSpan [89] uses a canonical labeling system, supporting depth-first search, to find frequently occurring subgraphs. Similar works include GREW [53], mining proximity pattern [49], and frequent patterns in large sparse graphs [54]. A frequent pattern or decomposition approach is also often employed for structural match, as described in [65]. The aim of this contribution, approximate subgraph matching, is, however, orthogonal to these.

2.3 Probabilistic Graph Matching

Recently, with the advent of uncertain and probabilistic graphs such as knowledge graphs, RDF stores, etc., algorithms for probabilistic graph querying are being studied. Initially, convex optimization methods were proposed [37]. However, they found it difficult to handle large and noisy real-life graphs.

Inexact graph matching for uncertain graphs majorly comprises a three-phase framework: *structural pruning*, *probabilistic pruning* and *verification*. Often the uncertainty information in the graph is ignored and candidate answers are searched using conventional structural pruning algorithms, followed by probabilistic pruning of candidates and verification of the answer candidates. Utilizing this framework, [95, 96] compute tight probabilistic bounds for pruning of approximate subgraph matching based on threshold for uncertain graphs with local correlations and adhering to the possible world semantics. Efficient upper and lower bounds were computed for relaxed query graphs by transforming the problem to set cover and integer quadratic programming problems in [96]. Additionally, [94] constructs an optimal probabilistic index based on edge-cuts of target graph to compute an upper bound on *pattern matching probability* for pruning. The above *filter-and-search* framework performs well in subgraph matching. On arrival of a query, it retrieves and sorts the promising positions in the underlying deterministic graph with the help of index structures incorporating PWS, and verifies the results by checking for subgraph isomorphism. However, given a database of probabilistic graphs, it only returns graphs that contain the entire query subgraph, and does not report the exact location of the query subgraph. Hence, we do not consider these approaches as baselines in our experiments. Although CHISEL creates a similar index on semantics (label) of a node, the use of statistical significance measure provides a more accurate rendition of both the structural similarity and the possible world semantics.

The direct approach described in [33] extends TreeSpan by efficient incremental similarity computation mechanism intertwined with structural pruning, with no attention to uncertainty information. Sampling has been shown to be effective in dealing with the hardness of managing and mining uncertain graphs [56]. The performance of sampling-based approaches depend heavily on the samples considered, though. Weighted subgraph matching algorithms were considered using the probability values as weights [38, 47]. However, such techniques were unable to handle the various uncertain scenarios that CHISEL can take into account.

Hua et al. [38] proposed three novel types of probabilistic path queries using basic principles. For probabilistic graph settings, a host of diverse problems such as frequent subgraph mining [15, 58, 70, 107], clustering [50], reliable subgraphs [62], shortest-path [38], and maximum flow [35] have been actively worked upon. Potamias et al. [72] studied k-nearest neighbor queries over uncertain graphs, and propose sampling algorithms for tackling #P-completeness of reachability problems. Lian et al. [59] proposed customization over existing tree-indexing strategies based on uncertain graph decomposition for k-NN queries. Reverse k-NN queries for uncertain graphs were studied in [31]. Approximate subgraph matching queries on fuzzy RDF graphs using *path decomposition* was recently shown to be efficient [22, 57]. A systematic introduction to the topic of managing and mining uncertain data can be found in [97], while [45] provides a detailed overview of state-of-the-art methods on uncertain graph mining. Such path decomposition-based similarity techniques have been shown to be effective and, hence, we compare the empirical performance of CHISEL with them in Section 5. Interestingly, the neighborhood search step in CHISEL can be considered to be similar to such path-based approaches taking into account the structure around a vertex.

2.4 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. In such scenarios, an observation is deemed to be statistically significant if its presence cannot be attributed to randomness alone. The classical *p-value* computes the chance of rejecting the null hypothesis, i.e., the observation is drawn from a known probability model characterizing the experimental setup. In other words, the less the p-value, the less likely it is that the null hypothesis is true. However, the computation of p-value is generally computationally infeasible since it entails the generation of all possible outcomes. To alleviate this problem various branch-and-bound methods have been studied [10]. In systems where such accuracy in measurement is not a necessity and a small factor of error can be tolerated, an approximation of the p-value can be calculated using other statistical measures. The literature hosts a number of statistical models to capture the uniqueness of observations, including *z-score*, *log-likelihood ratio* (G^2), and *Hotelling's* T^2 *measure* [74].

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 to the p-value [73]. In this paper, we consider statistical significance using the *Pearson's* χ^2 *measure* based on the chi-square distribution, which uses the frequency of occurrences of outcomes to test the fit of a model with the set of theoretical frequencies of the events, where the events are assumed to be mutually exclusive and independent.

The use of p-value for motif discovery in biological stochastic networks was studied in [41]. The use of χ^2 statistical measure has been shown to perform well in diverse applications such as significant substring [24, 25, 76] and connected subgraph extraction [5] for anomaly detection, substring similarity search [23], and maximum clique finding [26]. Hence, in this work, we also use the chi-square measure since it is efficient and reasonably accurate.

3. THE CHISEL ALGORITHM

In this section, we describe the working of our proposed CHISEL algorithm. As defined previously in Section 1, we denote the input target probabilistic graph by $G = (V, E, \mathcal{L}, \mathcal{P})$, and the query graph by $Q = (V_Q, E_Q, \mathcal{L}_Q)$. Without loss of generality, we assume that G is simple and undirected (containing no hyper-edges) with edge probabilities, and since users generally specify a query accurately, there is no uncertainty in the edges of Q. We handle all other special cases that depart from the above model in Section 4. Figure 1 provides sample graphs G and Q as a running example.

3.1 Overview

The CHISEL algorithm has two distinct phases, namely, the *indexing* and the *querying* phases. In the indexing phase, several *inverted index lists* for mapping between vertices, labels, and neigh-

Table 1: Computing observed values of symbols s_0, s_1 and s_2 for vertex pair $\langle v_1, q_1 \rangle$ for the example in Figure 1.

Possible worlds	Neighbors of v_1(A) v_2(B)	v_3(C)	v_4(A)	$P(W_x)$	Best matches for query triplets of q_1(A) and symbols $\langle B, A, C \rangle$	Sym.	$\langle C, A, D \rangle$	Sym.	$\langle B, A, D \rangle$	Sym.
W_0	0	0	0	$\overline{0.7} \times \overline{0.6} \times \overline{0.8} = 0.024$	$\langle \phi, A, \phi \rangle$	s_0	$\langle \phi, A, \phi \rangle$	s_0	$\langle \phi, A, \phi \rangle$	s_0
W_1	0	0	1	$\overline{0.7} \times \overline{0.6} \times 0.8 = 0.096$	$\langle \phi, A, \phi \rangle$	s_0	$\langle \phi, A, \phi \rangle$	s_0	$\langle \phi, A, \phi \rangle$	s_0
W_2	0	1	0	$\overline{0.7} \times 0.6 \times \overline{0.8} = 0.036$	$\langle C, A, \phi \rangle$	s_1	$\langle C, A, \phi \rangle$	s_1	$\langle \phi, A, \phi \rangle$	s_0
W_3	0	1	1	$\overline{0.7} \times 0.6 \times 0.8 = 0.144$	$\langle C, A, \phi \rangle$	s_1	$\langle C, A, \phi \rangle$	s_1	$\langle \phi, A, \phi \rangle$	s_0
W_4	1	0	0	$0.7 \times \overline{0.6} \times \overline{0.8} = 0.056$	$\langle B, A, \phi \rangle$	s_1	$\langle \phi, A, \phi \rangle$	s_0	$\langle B, A, \phi \rangle$	s_1
W_5	1	0	1	$0.7 \times \overline{0.6} \times 0.8 = 0.224$	$\langle B, A, \phi \rangle$	s_1	$\langle \phi, A, \phi \rangle$	s_0	$\langle B, A, \phi \rangle$	s_1
W_6	1	1	0	$0.7 \times 0.6 \times \overline{0.8} = 0.084$	$\langle B, A, C \rangle$	s_2	$\langle C, A, \phi \rangle$	s_1	$\langle B, A, \phi \rangle$	s_1
W_7	1	1	1	$0.7 \times 0.6 \times 0.8 = 0.336$	$\langle B, A, C \rangle$	s_2	$\langle C, A, \phi \rangle$	s_1	$\langle B, A, \phi \rangle$	s_1
Distribution of $[s_0, s_1, s_2]$ for each query triplet					$[0.12, 0.46, 0.42]$		$[0.40, 0.60, 0.00]$		$[0.30, 0.70, 0.00]$	

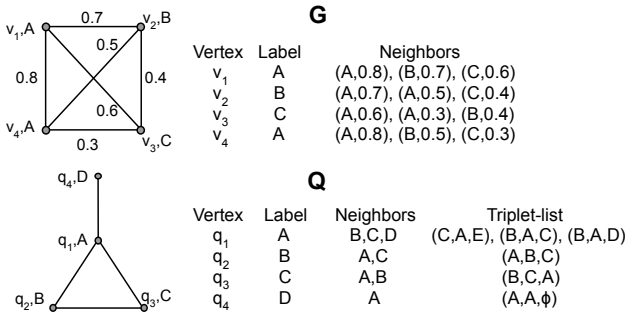


Figure 1: Running example of a subgraph matching query Q for the input probabilistic graph G.

bors are constructed. Observe that this phase is a one-time pre-processing step for an input graph (to boost the performance of querying later) and is independent of the query (i.e., performed even before the queries arrive). The querying phase is initiated when a query arrives, and the index structures are then used to efficiently compute statistical significance scores to guide the search process for finding the best matching subgraph. We next describe in detail the working of the two phases in CHISEL.

3.2 Indexing Phase

In the offline indexing phase, CHISEL constructs several indexes to store vertex and neighborhood information from the input target graph G. Specifically, we construct the following.

3.2.1 Vertex-Label Inverted Index

The vertex-label inverted index stores the mapping between the labels and the vertices of the input graph, G. Here, the labels present in $\mathcal{L}$ are considered to be keys, while the vertices associated with those labels form the values.

3.2.2 Neighbor Labels List

For each vertex in G, CHISEL stores the neighboring vertices along with their labels and the corresponding edge existential probabilities by using an adjacency list structure. Figure 1 depicts the construction of the above indexes for our example graphs.

3.2.3 Expected Degree Computation

Since the graph is probabilistic, the existence of neighbors of a vertex is uncertain. Assuming that the existence of each edge is an *independent* event, the *expected degree* of a vertex is computed as the sum of probabilities of edges incident on it. Thus, if a vertex v has n_v neighbors connected with edges with probabilities $p_1, p_2, \cdots, p_{n_v}$, the expected degree, δ_v, of v is given by $\delta_v = \mathcal{E}[deg(v)] = \sum_{i=1}^{n_v} p_i$.

We prove this by induction. Assume the base case where there is only one neighbor with probability p_1. The expected number of neighbors, therefore, is $\delta = p_1 \times 1 + (1 - p_1) \times 0 = p_1$. Assume that for $n-1$ neighbors, the expected degree is $\delta = \sum_{i=1}^{n-1} p_i$. If another vertex with edge probability p_n is added, then with probability p_n, the number of neighbors is $\delta + 1$, while with probability $1 - p_n$, it remains δ. Hence, the new expected degree is $\delta' = p_n \times (\delta + 1) + (1 - p_n) \times \delta = p_n + \delta = \sum_{i=1}^{n} p_i$.

Hence, the expected degree of vertex v_1 in the example of Figure 1 is $\delta_{v_1} = 0.8 + 0.7 + 0.6 = 2.1$.

3.2.4 Neighbor Label Probabilities

For each vertex in G, we also compute the probability that a particular label would occur in its neighborhood. This is used during the querying phase and, hence, CHISEL performs the computation during the offline pre-processing phase for faster querying.

Consider vertex v to have n_v neighbors with corresponding labels $l_1, l_2, \cdots, l_{n_v}$, each connected by edges with existential probabilities $p_1, p_2, \cdots, p_{n_v}$ respectively. Note that vertex labels may not be unique and may repeat.

Assume label l_j to be associated with ψ neighbors of v, which are connected via edges with probabilities $p_1^{l_j}, p_2^{l_j}, \cdots, p_\psi^{l_j}$. The event that *no instance* of label l_j is present in the neighborhood of v occurs when none of these vertices are neighbors of v, i.e., these edges do not exist. Denoting the number of instances of label l_j in the neighborhood by $\#l_j$, the probability of this event is

$$P(\#l_j = 0) = (1 - p_1^{l_j}) \times \cdots \times (1 - p_\psi^{l_j}) = \prod_{i=1}^{\psi} \overline{p}_i^{l_j} \quad (1)$$

where $\overline{p}_i^{l_j}$ denotes the probability of the edge between the i^{th} neighbor and v not existing. Using the above equation, the probability that label l_j occurs *at least once* in the neighborhood of v is,

$$P(\#l_j \geq 1) = 1 - P(\#l_j = 0) \quad (2)$$

Similarly, the probability that label l_j occurs *exactly once* in the neighborhood of v is given by

$$P(\#l_j = 1) = \sum_{i=1}^{\psi} \left[p_i^{l_j} \cdot \prod_{\iota \neq i} \overline{p}_\iota^{l_j} \right] = P(\#l_j = 0) \cdot \sum_{i=1}^{\psi} \frac{p_i^{l_j}}{\overline{p}_i^{l_j}} \quad (3)$$

If there is no neighbor of v with the label l_j, then it cannot occur in the neighborhood of v and, consequently, $P(\#l_j = 0) = 1$ and $P(\#l_j \geq 1) = P(\#l_j = 1) = 0$.

The occurrence probabilities of labels in the neighborhood of a vertex is computed using equations (1)-(3) during pre-processing.

3.3 Querying Phase

After the indexing phase is completed, the *querying phase* commences on arrival of a query graph $Q = (V_Q, E_Q, \mathcal{L}_Q)$. We next describe the details of the steps involved in this phase.

3.3.1 Inverted Lists and Neighborhood Information

Similar to the input target graph G, as described in Section 3.2, the vertex-label inverted index and neighbor label list structures are computed for Q as well. Since query graphs are generally relatively small in size (in the order of tens of vertices), this step is fast.

3.3.2 Vertex Pair Generation

The querying phase proceeds by constructing similar matching vertex pairs between graphs G and Q. Specifically, a *vertex pair*, $\langle v, q\rangle$, is constructed with vertex $v \in V$ and $q \in V_Q$ having the same label, i.e., $l_v = l_q$, where l_v and l_q denote the labels of vertices v and q respectively. Given a query vertex q, such vertex pairs can be easily formed by using the vertex-label inverted index structure of G. Formally, the entire vertex pair set for query Q is

$$\mathcal{VP} = \{\langle v, q\rangle \mid v \in V, q \in V_Q, l_v = l_q\} \tag{4}$$

The vertex pair set for the example in Figure 1 is $\{\langle v_1, q_1\rangle, \langle v_2, q_2\rangle, \langle v_3, q_3\rangle, \langle v_4, q_1\rangle\}$.

Vertex pairs play an important role in pruning the search space, by providing initial seeds for neighborhood exploration for finding matching subgraph. The CHISEL framework works on the computation of statistical significance scores of these vertex pairs to obtain the top-k potential candidate regions in G for extracting the best (approximate) matching subgraph to Q. The next section explains how the χ^2-value of a vertex pair is computed.

3.4 Vertex Pair Chi-Square Computation

3.4.1 Label Triplet Generation

For each query vertex q, CHISEL initially constructs *triplets* of vertices $\langle x, q, y\rangle$ where x and y are the neighboring vertices of q in Q. The corresponding label triplet, $\langle l_x, l_q, l_y\rangle$, is computed, where l_x, l_q, and l_y denote the labels of the vertices x, q and y respectively in Q. The neighbors are considered to be symmetric, i.e., $\langle l_x, l_q, l_y\rangle$ is equivalent to $\langle l_y, l_q, l_x\rangle$. Therefore, without loss of generality, we order the labels in a label triplet alphabetically.

Considering vertex $v \in G$ that forms the vertex pair $\langle v, q\rangle$ with q, similar neighbor triplets $\langle u, v, w\rangle$ and their corresponding label triplets $\langle l_u, l_v, l_w\rangle$ of v are extracted.

3.4.2 Triplet Pair Matching

For the vertex pair $\langle v, q\rangle$, CHISEL next characterizes the similarity between the label triplets obtained from the vertices v and q. By definition of vertex pair construction, we have $l_q = l_v$. However, the other two neighboring vertex labels (in the triplets) may or may not match. Based on the degree of label matching between the obtained label triplets (of a vertex pair), i.e., label triplet pairs $\langle l_x, l_q, l_y\rangle$ and $\langle l_u, l_v, l_w\rangle$, the *triplet pair similarity* is characterized into three different classes, as:

- s_2: When both the neighboring labels in the triplets match.

$$s_2 : (l_x = l_u \wedge l_y = l_w) \tag{5}$$

- s_1: When only one of the neighboring triplet labels match.

$$s_1 : ((l_x = l_u \wedge l_y \neq l_w) \vee (l_x \neq l_u \wedge l_y = l_w)) \tag{6}$$

- s_0: When none of the neighboring labels match.

$$s_0 : (l_x \neq l_u \wedge l_y \neq l_w) \tag{7}$$

Since a vertex in the target graph have neighbors connected by probabilistic edges, it will have varying number of triplets in the different "possible worlds". We next explain how the above triplet pair matching assigns symbols in this probabilistic setting.

3.4.3 Possible Worlds

We adopt the *possible world semantics* (PWS) graph uncertainty model, where each vertex v with d neighbors is connected by edges with probabilities of existence. The existence of edges are considered to be independent of each other. Thus, in each possible world, either an edge exists or it does not exist. There are 2^d possible worlds, and each such possible world occurs with a particular probability (as shown in Table 1).

3.4.4 Observed Match Symbol Vector

Assume vertices $t_1, t_2, \cdots, t_d$ to be the neighbors of vertex v, with corresponding labels $l_1, l_2, \cdots, l_d$ respectively. Consider a particular possible world W_i where edges to neighbors $t_1, t_2, \ldots, t_w$ exist but those to $t_{w+1}, t_{w+2}, \cdots, t_d$ do not exist, for some $1 \leq w \leq d$. The probability of world W_i is, thus,

$$P(W_i) = p_1 \times \cdots \times p_w \times \bar{p}_{w+1} \times \cdots \times \bar{p}_d \tag{8}$$

The vertex triplets of v that exist in W_i are $\mathcal{T}_i = \{\langle l_1, l_v, l_2\rangle, \langle l_1, l_v, l_3\rangle, \cdots, \langle l_{w-1}, l_v, l_w\rangle\}$.

Assuming the vertex pair $\langle v, q\rangle$, the query triplet $\langle l_x, l_q, l_y\rangle$ is now matched with all the possible triplets in $\mathcal{T}_i$. The matching of a query triplet is considered to be the one that produces the *best* scenario, i.e., where there are more triplet label matches. The order of preference of match classes (or symbols) is, thus, $s_2 \succ s_1 \succ s_0$.

For each possible world W_i, a match symbol s_j (either s_2, s_1, or s_0) is, hence, associated with the query triplet, denoted as $s_j \odot W_i$. The overall probability of a particular match symbol is the *sum* of probabilities of the possible worlds to which it gets associated:

$$P(s_j) = \sum_{\forall W_i,\ s_j \odot W_i} P(W_i) \tag{9}$$

Thus, corresponding to each query triplet, and a vertex pair, there is a probability distribution of the symbols s_2, s_1 and s_0. This forms the *observed counts* for the match symbols that implicitly incorporate the PWS concept based on the different possible worlds.

Based on the above formulation, Table 1 shows the observed symbol vectors for query triplets corresponding to the vertex pair $\langle v_1, q_1\rangle$ with label A. Since v_1 has 3 neighbors, the number of possible worlds is $2^3 = 8$. They are denoted by $W_0, \ldots, W_7$.

Consider the possible world W_1 where only neighbor v_4 with label A occurs but neighbors v_2 and v_3 do not occur. The probability of this world is $P(W_1) = (1 - 0.7) \times (1 - 0.6) \times 0.8 = 0.096$. The label triplets around v_1 are $\langle A, A, \phi\rangle$ and $\langle \phi, A, \phi\rangle$ where ϕ is a dummy label that is used when at most one neighbor label exists. Comparing any label with ϕ, including ϕ itself results in a mis-match. For the query triplet $\langle B, A, C\rangle$, the best match, therefore, is $\langle \phi, A, \phi\rangle$. Since none of the neighbors match, this results in the symbol s_0. Similarly, for the world W_2, the query triplet $\langle B, A, C\rangle$ is best matched with the vertex triplet $\langle C, A, \phi\rangle$ yielding the symbol s_1. In the world W_7, the possible vertex triplets are $\langle A, A, B\rangle$, $\langle A, A, C\rangle$ and $\langle B, A, C\rangle$. The highest match corresponds to $\langle B, A, C\rangle$ and it yields the symbol s_2. Thus, the world W_7 contributes to the symbol s_2.

Adding up the probabilities of the worlds in which the symbols occur, the observed symbol vector for the query triplet $\langle B, A, C\rangle$ is $[0.12, 0.46, 0.42]$. Since the probabilities of the possible worlds add up to 1, so do the probabilities of the symbols.

3.4.5 Multiple Query Triplets

When there are multiple query triplets corresponding to a vertex pair, the observed counts of the match symbols are added to form the cumulative *observed count vector*, O, for the vertex pair. In Table 1, considering all the 3 query triplets pertaining to q_1, the cumulative observed symbol vector obtained is $O = [0.82, 1.76, 0.42]$.

CHISEL considers the matching symbols of each of the query triplets to find the best matching subgraph.

3.4.6 Efficiently Computing Observed Vectors

However, in the above process of computing the cumulative observed count vectors, enumerating all the possible worlds is computationally inefficient. For some vertices with large degrees, it is practically infeasible. Fortunately, since we are only concerned with the presence or absence of relevant neighbor vertex labels in the worlds, CHISEL efficiently computes the observed symbol vectors based on the probabilities obtained in the indexing phase.

Consider a query triplet $\langle l_x, l_q, l_y \rangle$ and a corresponding vertex v in G with label $l_v = l_q$. The vertex v can have multiple neighbors with label l_x (or l_y). Equation (1) to Equation (3) in Section 3.2.4 give the probabilities of l_x (or l_y) occurring various number of times in the neighborhood.

We first consider the case when $l_x \neq l_y$. The symbol s_2 occurs in those worlds where both the labels l_x and l_y occur at least once. The occurrence of such an event has the probability

$$O[s_2] = P(\#l_x \geq 1) \times P(\#l_y \geq 1) \quad \text{[using Eq. (2)]} \tag{10}$$

Similarly, symbol s_0 occurs when none of the instances of the labels l_x and l_y occur:

$$O[s_0] = P(\#l_x = 0) \times P(\#l_y = 0) \quad \text{[using Eq. (1)]} \tag{11}$$

Since the probability of the match symbols add up to 1,

$$O[s_1] = 1 - O[s_2] - O[s_1] \quad \text{[using Eq. (10) and (11)]} \tag{12}$$

We next consider the case when $l_x = l_y$. The symbol s_0 occurs in those worlds where no instance of label l_x occurs:

$$O[s_0] = P(\#l_x = 0) \quad \text{[using Eq. (1)]} \tag{13}$$

Similarly, symbol s_1 occurs when exactly one instance of l_x occurs:

$$O[s_1] = P(\#l_x = 1) \quad \text{[using Eq. (3)]} \tag{14}$$

Consequently,

$$O[s_2] = 1 - O[s_0] - O[s_1] \quad \text{[using Eq. (13) and (14)]} \tag{15}$$

The above computation avoids the exponential enumeration of the possible worlds and is only *linear* in terms of number of neighbors of a vertex. More importantly, equations (1)-(3) are computed in the *offline* phase before any query arrives. The querying phase only uses the information and is, therefore, very fast in practice.

3.4.7 Expected Symbol Vector for Triplet Match

The chi-square statistic computes the deviation of the observations from the expectations. The expected count of triplet match symbols for a vertex triplet is computed as follows.

Suppose the input target graph contains L labels that are assumed to be equally likely in terms of occurrence. Consider a vertex v with *expected degree* δ (as computed in Section 3.2.3). The chance that a neighbor of v has a particular label l_x is $1/L$. Therefore, the chance that none of the neighbors of v has label l_x is given by $(1 - 1/L)^\delta$.

Now, considering a label triplet of v, if the highest match symbol (to a query label triplet) is s_0, then none of the triplets have label l_x. Since there are two neighboring vertices in a triplet, assuming independence of labels, the probability of the event s_0 is

$$P(s_0) = \left((1 - 1/L)^\delta\right)^2 \tag{16}$$

The chance that there is at least one neighbor with label l_x is $1 - (1 - 1/L)^\delta$. Thus, the probability that v has a triplet where both the neighboring vertices match is given by

$$P(s_2) = \left(1 - (1 - 1/L)^\delta\right)^2 \tag{17}$$

The event s_1 occurs when there is one vertex in a triplet that matches a label while the other does not. Since there are two ways of enumerating vertices in a triplet, the probability of the event s_1 is

$$P(s_1) = 2 \cdot (1 - 1/L)^\delta \cdot \left(1 - (1 - 1/L)^\delta\right) \tag{18}$$

Observe that the match symbols are mutually exhaustive, i.e., the probabilities add up to 1. If a query has len triplets, the expected counts of the match symbols ($i = 0, 1, 2$) are

$$E_i = len \cdot P(s_i) \tag{19}$$

The expected counts of the match symbols are represented as a vector, referred to as the *expected symbol vector*. Importantly, the calculation of the above probabilities is *independent* of the query. Hence, they are actually done in the *offline* pre-processing phase of CHISEL, contributing to the efficiency of the querying phase.

The vertex v_1 in Figure 1, as shown earlier in Section 3.2.3, has an expected degree of 2.1. Assume the total number of labels in $\mathcal{L}$ to be $L = 4$. Hence, $P(s_0) = ((1 - 1/4)^{2.1})^2 = 0.30$, $P(s_1) = 0.49$, and $P(s_2) = 0.21$. Thus, the expected symbol vector for the vertex pair $\langle v_1, q_1 \rangle$ having three query triplets is computed as $E = 3 \cdot [0.30, 0.49, 0.21] = [0.90, 1.47, 0.63]$.

3.4.8 Chi-Square of a Vertex Pair

Using the *observed symbol vector* and the *expected symbol vector* as computed above, CHISEL finally computes the *chi-square* value, $\chi^2_{\langle v,q \rangle}$, of each candidate vertex pair $\langle v, q \rangle$ as:

$$\chi^2_{\langle v,q \rangle} = \sum_{i=0}^{2} \frac{(O[s_i] - E[s_i])^2}{E[s_i]} \tag{20}$$

In the example in Figure 1, the chi-square of the vertex pair $\langle v_1, q_1 \rangle$ is $\chi^2_{\langle v_1,q_1 \rangle} = \frac{(0.82-0.90)^2}{0.90} + \frac{(1.76-1.47)^2}{1.47} + \frac{(0.42-0.63)^2}{0.63} = 0.13$.

3.5 Top-k Subgraph Search

CHISEL next computes the χ^2 scores for each of the vertex pairs obtained from the query Q. The best candidate regions for subgraph matching are then explored to obtain the final answer by use of two heap structures as discussed next.

3.5.1 Primary Heap

All the vertex pairs along with their χ^2 values are inserted into a *max-priority heap* called the *primary heap*. To compute the similarity for subgraph matching, vertex pairs are picked up from the primary heap in the priority order, i.e., the one with the largest chi-square score is picked first, and so on, to form the *seed* of the neighborhood search process. A vertex that has been matched earlier is not picked up any further. This ensures that the subgraphs returned are disjoint. This is done to avoid duplicate answers.

3.5.2 Secondary Heap

For each vertex pair picked from the primary heap, its neighborhood is searched to see if the query subgraph can be completed.

Suppose, $\langle v, q \rangle$ is chosen. The neighbors of vertices v and q that share the same label (i.e., form a vertex pair themselves) are extracted, constructed into a vertex pair, and inserted into another priority-max heap, referred to as the *secondary heap*.

Assume a neighbor vertex pair $\langle n_v, n_q \rangle$. In addition to its chi-square value, $\chi^2_{\langle n_v, n_q \rangle}$, the probability, $p(v, n_v)$, of the edge connecting v to n_v is also inserted in the secondary heap. The neighbor vertex pair having the maximum value of $p(v, n_v) \cdot \chi^2_{\langle n_v, n_q \rangle}$ is then picked and the neighborhood search continues using this as the new seed. The edge probability is used along with the chi-square so that more probable edges are preferred.

3.5.3 Top-k Search

The growth of the neighborhood continues till the query is completely matched or the subgraph in the target graph cannot grow any further due to lack of matching vertex pairs.

The above process is initiated k times to find the top-k matching subgraphs. The subgraphs are constrained to be disjoint from each other, by marking vertex pairs as "visited" in both the heaps.

3.6 Summation of Chi-Square Values

For every matching vertex pair of a query, the chi-square value follows the χ^2 *distribution* with 2 *degrees of freedom* (since there are 3 possible match symbols). The χ^2 values for the vertices of a matching subgraph are added to produce the total chi-square statistical significance score of the match. Since addition of chi-square distributions results in another chi-square distribution [34], for a query Q of size $|V_Q|$, the total chi-square value of the matching subgraph follows the chi-square distribution with degrees of freedom $2 \cdot |V_Q|$. This enables approximating the *p-value* of the match, if required. The top-k matching subgraphs are sorted based on their total χ^2 values.

On one hand, the observed match symbol vector models the PWS concept by considering the match symbol (of triplets) across all the "possible" worlds. On the other hand, the χ^2 computation and the greedy neighborhood search takes into account the structural and label match. Thus, our proposed framework provides an integrated measure to obtain the best approximate subgraph matches.

3.7 Complexity Analysis

Assume that G has n vertices and m edges while Q has q vertices and p edges. Building the query graph indexes require $O(p)$, while creating the vertex pairs $\mathcal{VP}$ require at most $n \cdot q = O(n)$ time, since p and q are small and considered to be constants in practice. For each vertex pair in $\mathcal{VP}$, computing the expected vector requires $O(1)$ time, while that for the observed vector requires $O(d_v)$ time, where d_v is the degree of vertex v. The complexity of the above operations for all vertex pairs is approximately $O(m)$.

Considering $m \leq n^2$, the average degree of vertices in G is $a = 2m/n \sim O(n)$. The size of the primary heap is bounded by $O(n)$. For a vertex in the primary heap, the secondary heap is initially populated by $O(a)$ neighboring vertex pairs. Subsequently, the best candidate from the secondary heap is extracted and its $O(a)$ neighboring vertex pairs are added to the heap. Observe, at most q such iterations are performed on the secondary heap to extract the matching subgraph, providing a total time complexity of $O(q \cdot a) \sim O(n)$ for the heap operations. Hence, for top-k subgraph matches, the time complexity of ChiSeL is $O(m + k \cdot n)$.

The size of the primary heap is $O(n)$, while each secondary heap can grow to a maximum of another $O(a) \sim O(n)$ neighbors. The extra space overhead, hence, is at most $O(n)$.

Empirical evaluation, however, shows that the sizes of primary and secondary heaps are mostly small constants in general.

4. SPECIAL CASES OF UNCERTAINTY

We have so far described ChiSeL for vertex labeled probabilistic graphs where the uncertainty is only in the existence of edges. In this section, we briefly outline how other cases of uncertainty and noise can be handled within our framework.

4.1 Edge Labels

If, in addition to vertices, edges are also labeled, then for a vertex to match, the corresponding edge labels also need to match. To handle this easily, for a neighboring vertex, we prepend the corresponding edge label to the vertex label of the neighbor. This ensures that the edge labels are also taken into account during triplet matching and symbol generation.

4.2 Uncertain Vertices

Consider the vertices in the target input graph to be also uncertain, i.e., each vertex exists with a probability. This scenario can be handled in our current model by absorbing the vertex probabilities into the probabilities of the edges incident on it. Thus, if vertices u, v, w exist with probabilities p_u, p_v, p_w respectively, and the edges $e_1 = (u, v), e_2 = (w, v)$ exist with probabilities p_1, p_2 respectively, then the graph can be appropriately modified such that the edge probabilities are updated to $p'_1 = p_1 \cdot p_u \cdot p_v$ and $p'_2 = p_2 \cdot p_w \cdot p_v$. The vertices are then no longer treated as uncertain, and the rest of the framework remains same.

4.3 Noisy Labels

For scenarios where the labels in the vertices might be noisy (e.g., mis-spelled), similarity measures such as *Jaccard similarity* can be used for matching purposes. For example, vertex labels having a Jaccard similarity score greater than a pre-defined threshold will be considered to be a match. Also, *semantic similarity* of the vertex labels can be considered in such cases.

4.4 Label Uncertainty

The label of a vertex or an edge might also be uncertain, that is, there exists a probability distribution over a set of labels for each vertex or edge. For example, suppose vertex v has label l_{1_v} with probability 0.6 and label l_{2_v} with probability 0.4. Given a query vertex q with label l_q, a suitable distance function (such as Jensen-Shannon divergence measure) can be used to ascertain the similarity between the two label distributions. Similar to noisy labels, only if this distance is below a threshold, the vertices are said to match.

4.5 Uncertain Query Graphs

Finally, if the query graph is also uncertain in nature (this uncertainty may be in edges and/or vertices and/or labels), we can adopt deviation thresholds of probability for each uncertainty, i.e., only candidate matches having a probability deviation from the query vertex/edge matches within the threshold will be considered.

5. EXPERIMENTAL EVALUATION

In this section, we empirically evaluate the efficacy of our proposed ChiSeL algorithm, and benchmark its performance against state-of-the-art competing algorithms on large real-life datasets.

5.1 Empirical Setup

All the algorithms were implemented in C++. The experiments were performed on an Intel(R) Xeon(R) 2.6GHz CPU E5-2697v3 processor with 504GB RAM running CentOS Linux 7.2.1511.[1]

[1]The source code of ChiSeL is available from https://github.com/Shubhangi-Agarwal/ChiSeL.

Table 2: Characteristics of datasets used.

Dataset	# Vertices	# Edges	# Labels	Avg. Degree
PPI-complete	7.6M	1.2B	0.2M	316
PPI-small	12.0K	10.7M	2.4K	1789
YAGO	4.3M	11.5M	4.0M	5
IMDb	3.0M	11.0M	3.0M	7

5.1.1 Datasets

We use the following datasets as our input target graphs:

- *STRING DB* or *PPI* v10.5 (`version-10-5.string-db.org`): a database of known and predicted protein-protein interactions created automatically by collecting information from various sources. We extract the COG mappings of proteins and their links. The proteins are considered as the vertex set with the orthologous groups as their labels, and the links form the edges. Importantly, each protein link is annotated with a confidence score (from 0 to 1) that represents how likely it is that the interaction exists. The total graph consists of around 7.6M vertices and 1.2B edges with around 200K unique vertex labels. This dataset is referred to as *PPI-complete* henceforth.
  - ⋆ *PPI-small*: We also randomly extract a smaller graph from PPI-complete consisting of 12K vertices and 10.7M edges with around 2.4K unique vertex labels.
- *YAGO* (`www.yago-knowledge.org`): an open source knowledge graph consisting of extracted entities and relations from Wikipedia, WordNet and GeoNames. Each relationship is associated with a confidence value. It comprises of nearly 4.3M vertices and 11.5M edges, with 4M unique labels.
- *IMDb* (`www.imdb.com/interfaces`): dataset with information on movies, actors, directors, etc., with around 3M uniquely labeled vertices and 11M edges. We randomly assign edge probabilities to model scenarios where the edge existential probability distribution is not known apriori.

Table 2 summarizes the characteristics of the various datasets.

5.1.2 Query Generation

Exact. The query graphs were constructed from each of the above dataset by initially selecting a random vertex, and then exploring its neighborhood using a random walk till q vertices are visited. Finally, the subgraph induced by the visited vertices is considered as the query Q. Without loss of generality, we consider the query graphs to be connected; else, the algorithms can be independently executed on the disjoint components.

We generated query graphs of different sizes, with the number of vertices varying from 3 to 13, with a step size 2. For PPI-complete, the query graph size varied from 3 to 25. The number of query edges varied from 2 to 80. Further, for each query graph size, 20 different graphs were randomly extracted from each of the datasets. We refer to this set of query graphs as *exact* queries.

Noisy. To study the performance of the algorithms in presence of noise, we further create a *noisy* query graph set by introducing structural noise as follows. For each of the above exact queries obtained, we randomly insert, delete or modify the probabilities of some edges such that the number of edit operations is 33% of the edges present in the exact query.

Thus, for each dataset, in general, we considered $(6 \times 20 \times 2) = 240$ queries. For PPI-complete, the number of queries were $(12 \times 20 \times 2) = 480$. Unless otherwise mentioned, results presented henceforth are averages over the corresponding query sets.

5.1.3 Competing Methods

Two major strategies are used in the literature: distance/error bounded pruning approaches and tree/graph traversal based methodologies. Thus, in the same spirit, we compare the ChiSeL algorithm against two recent state-of-the-art methods:

(i) *PBound* [33], that performs maximal subgraph matching by incrementally computing similarity probabilities, and using probability upper bounds for pruning.

(ii) *Fuzzy* [57], that uses path-based graph decomposition and k-partite based joining techniques to obtain best matching approximate subgraph on LUBM benchmark, and has been shown to outperform SPath [103] and SAPPER [102].

Table 3: Overall performance comparison of the algorithms on all the datasets averaged over both exact and noisy query graph sets. (The results for PPI-complete are not shown since PBound and Fuzzy could not be run on them. ChiSeL achieves an accuracy of 0.84 on PPI-complete with a running time of 0.14s.)

Method	YAGO		IMDb		PPI-small	
	Acc.	Time (s)	Acc.	Time (s)	Acc.	Time (s)
ChiSeL	0.89	1.62	0.87	0.05	0.96	16.69
PBound	0.26	560.92	0.57	101.95	0.29	3134.09
Fuzzy	0.61	1.82	0.62	2.19	0.01	13970.94

5.1.4 Parameter Setting

For the above competing methods, we performed the best effort implementation as the original code was not available publicly or from the authors.

We experimentally studied the performance of the baselines with different parameter settings. Fuzzy was tested on varying edit-distance parameter combinations from the set $\{0, 0.25, 0.5, 0.75, 1.0\}$, and different distance and probability threshold parameters for PBound. For best performance, the parameters of PBound were set to their default values (as reported in [33]), while in Fuzzy the probability threshold was set to 0.1 with the insertion, deletion and replacement costs for string edit-distance set to 1.0 each.

We set $k = 10$ for the number of top-k subgraphs returned in our framework.

5.1.5 Evaluation Measures

We evaluate the quality of the matching subgraphs reported and the performance of the algorithms using the following measures:

(i) *Mean maximum accuracy* – reports the average over the maximum accuracy of a subgraph match found, for each query, within the top-k results reported. We define the *accuracy* as the number of matching edges present in the answer subgraph against the number of edges required for a complete match (i.e., edges in the query).

(ii) *Runtime* – compares the computation efficiency of the algorithms by reporting the wall clock running times.

5.2 Overall Results

The overall performances of the algorithms on the different datasets are tabulated in Table 3.

PBound enumerates all minimum spanning trees and Fuzzy lists all source-destination paths. Since these numbers are extremely large for PPI-complete that contains more than a billion edges, none of the queries for PBound and Fuzzy could finish within a reasonable amount of time (1 hour) for PPI-complete. ChiSeL could easily scale to such massive graphs along with high accuracy. For a fair evaluation of all the three algorithms, a smaller dataset, PPI-small, was therefore, randomly extracted from PPI-complete.

In terms of runtime, on all the datasets, ChiSeL is the fastest. PBound particularly suffers in compute time as it iterates over all possible minimum spanning trees. While running time for Fuzzy is low owing to enumeration of all the source-destination paths, it requires a very high indexing time. ChiSeL requires considerably

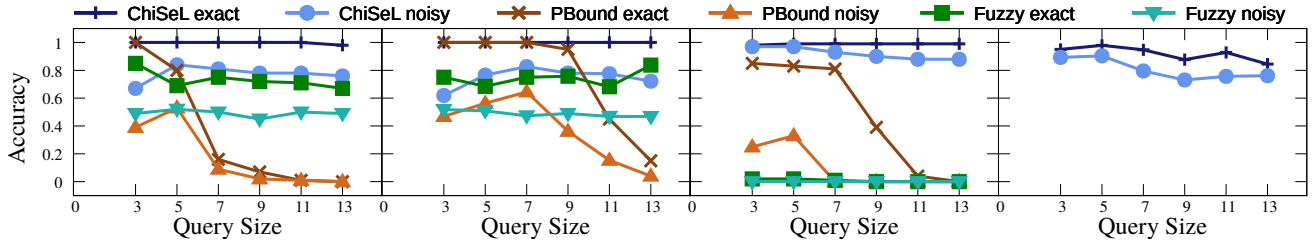


Figure 2: Accuracy comparison of different algorithms over all datasets.

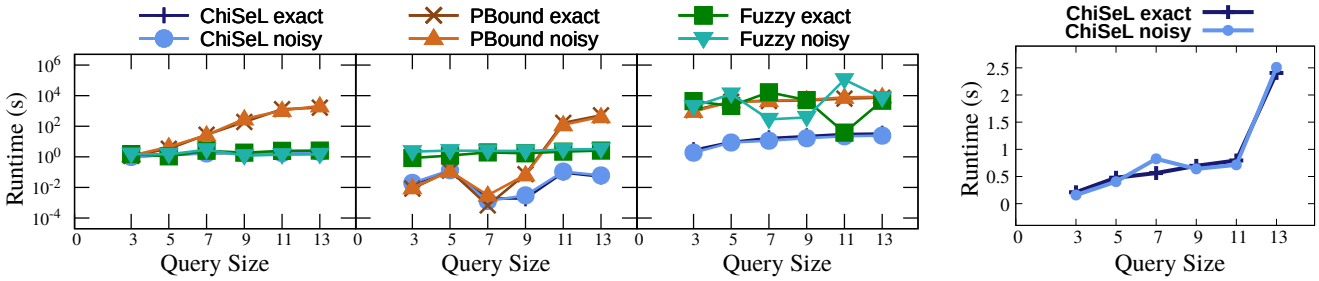


Figure 3: Runtime comparison of different algorithms over all datasets.

more running time for YAGO due to the fact that the vertex labels in YAGO are very long and can have special characters and, therefore, label matching requires a lot of time. Interestingly, CHISEL took the longest time for PPI-small, almost 15 times more than even the total PPI-complete dataset. The reason was the very large average degree of vertices in PPI-small. To understand this effect further, we did more experiments on the average degree (see Section 5.5.1).

The quality of CHISEL was also the best for all the datasets. While Fuzzy produced medium quality results for YAGO and IMDb, it completely failed for PPI-small. Fuzzy had too many paths to search since the number of labels were low and, hence, it ended up not finding the desired subgraphs for almost all the queries.

CHISEL, thus, efficiently extracts subgraphs with better structural similarity (edge and label matches) to the query.

5.3 Detailed Analysis

To further analyze the performance of the algorithms, we conducted experiments to capture the accuracy and runtime over different query types and query sizes. Figure 2 and Figure 3 report the accuracy and the runtime performance of all the algorithms.

CHISEL was seen to perform the best in terms of quality, with nearly 100% accuracy for exact queries for almost all the query sizes for all the datasets except PPI-complete. The accuracy numbers were also observed to be stable across different query sizes, thereby demonstrating the robustness of our proposed framework. Even for the noisy query scenario, the accuracy was around 0.8 for most of the datasets and more than 0.9 for PPI-small.

PBound depicted a sharp drop in accuracy with increase in query sizes. This is due to its inability to iterate over minimum spanning trees with distinct vertex sets within a reasonable runtime. Fuzzy showed a steady performance and was not affected much by the query size. It, however, performed very poorly for PPI-small.

Figure 3 shows that CHISEL was the fastest for almost all the scenarios. Fuzzy quickly becomes impractical for larger query sizes and requires a very large time for query sizes 9 and above. Both PBound and Fuzzy performed very poorly on PPI-small due to the extremely high average degree. The number of paths and subgraphs are too many when the density of the graph is high,and, hence, these algorithms do not scale for such dense graphs.

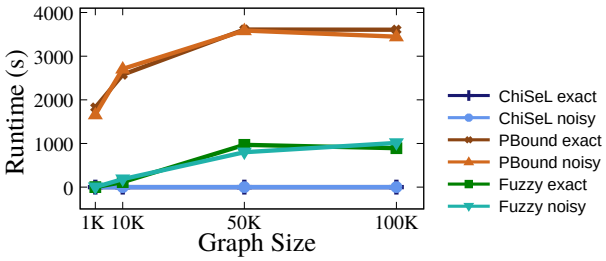


Figure 4: Scalability of CHISEL, PBound and Fuzzy on graphs sampled from PPI-complete dataset.

An interesting anomalous behavior of runtime with query sizes was observed for IMDb and PPI-complete datasets for CHISEL. This is explored and explained in detail later in Section 5.5.5.

5.4 Scalability Study

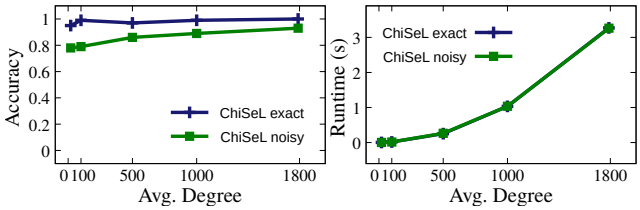
We next study the scalability of the different algorithms with respect to input graph size. For that, different sized subsets from the PPI-complete dataset were created maintaining the average degree in each subset to be around 250. We then pose queries of size 5, for both exact and noisy scenarios, on these subsets. Figure 4 reports the running times observed on these sampled subgraphs. PBound performs poorly due to its enumeration of all minimum spanning trees. While for very small graphs, Fuzzy is comparable in runtime with CHISEL, the overall scalability of CHISEL is better. For graphs with size greater than 50K, the runtime of CHISEL is orders of magnitude lesser than the competing approaches. Overall, the scalability over graph size for all algorithms is at most linear.

5.5 Analysis of Effect of Parameters

In this section, we empirically explore the robustness of performance for our proposed CHISEL algorithm. We vary different characteristics of the input and query graphs to study their impact on the accuracy and runtime of CHISEL.

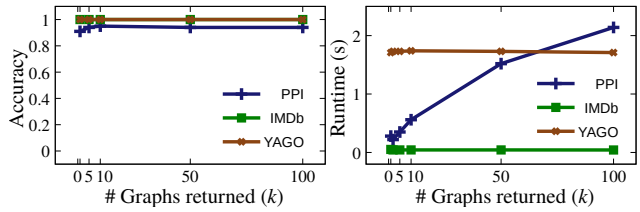
5.5.1 Average Graph Degree

We first analyze the effect of average degree of vertices in the input graph over the performance of CHISEL. To that end, we took the entire PPI-small dataset and created various edge-subsets of it.



(a) Accuracy (b) Running time

Figure 5: Effect of average degree.



(a) Accuracy (b) Running time

Figure 6: Effect of number of subgraphs returned.

We created different samples of decreasing density from it by randomly deleting edges. The number of unique labels were more or less conserved. Queries of size 5 were then posed on these graphs.

While we observed no appreciable effect on accuracy, the runtime increased considerably with increase in average degree (Figure 5). This is due to the increase in the number of neighborhoods, affecting the time taken by CHISEL to explore and populate the secondary heap. To understand it further, we measured the size of the secondary heap, averaged across different queries, for the different scenarios. We see that while the secondary heap size is only 9 when the average degree is 20, it increases steadily across increasing average degree and reaches 543 when average degree is 1789. Consequently, the running time increases considerably.

5.5.2 *Number of Subgraphs Returned, Top-k*

The next set of experiments measures the effect of number of subgraphs returned, i.e., the parameter k for the top-k search. Figure 6 shows that there is negligible effect of k on accuracy for all the datasets and on runtime for IMDb and YAGO datasets. The runtime increases only slightly for PPI-complete mainly because of the increase in the number of secondary heap initializations.

5.5.3 *Perturbation of Edge Probabilities*

Since edge probabilities model the uncertainty in structure for our problem setting, we next study how changes in edge existence probabilities affect the performance of CHISEL. For this study, we use PPI-complete, albeit with a slight modification. The input graph edges that are present in the query are modified to be *certain*, i.e., the probabilities of those edges in the original graph are set to 1.0. We refer to this modified dataset as *perturbed*, while *unperturbed* denotes the original un-altered PPI-complete graph. Further, as before, we consider both the *exact* and *noisy* query scenarios. Figure 7 depicts the accuracy of CHISEL with changes in the probability model of an input graph with varying query sizes.

The accuracy increases considerably for the *perturbed* version of PPI-complete. This intuitively follows from the *possible world scenario* concept. Since the exact matches to the query edges are modified to have probability 1.0, the exact query subgraph is extracted and returned by CHISEL with higher chi-square (at the top of the ranking) in most cases, thereby leading to an increase in accuracy. This provides a valuable insight that our proposed framework for subgraph matching based on statistical significance inherently takes into account the possible world modeling.

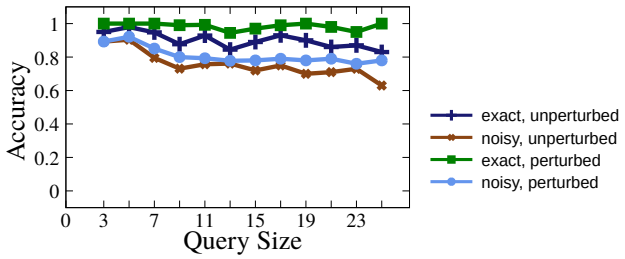
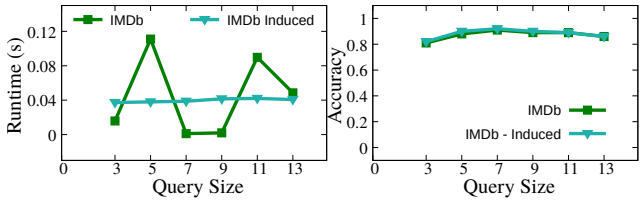


Figure 7: Effect of perturbation of edge probabilities.



(a) Running time (b) Accuracy

Figure 8: Effect of query degree distribution on runtime and accuracy for original and controlled degree distribution.

5.5.4 *Possible World Semantics Modeling*

To understand the effects of sampling of possible worlds, we did the following experiment. From the PPI-small dataset, we created various "certain" graphs by sampling the edges according to their existence probabilities. In other words, in each such sampled possible world, an edge is present with its corresponding existence probability. Each such sampled world, therefore, contains a subset of the original set of edges, but is *certain* in nature and no longer a probabilistic graph. For each sampled certain graph, we ran an approximate graph querying algorithm that works for certain graphs. We chose the NAGA algorithm [27] since it reports fairly accurate results and works on the same principles of statistical significance.

For each scenario, we created a number of sampled possible world graphs, varying from 100 to 10000. We then ran NAGA for queries of size 5 on every possible world graph, and report the *best* accuracy obtained over any possible world graph. Despite creating as large as 10000 possible worlds, the best accuracy obtained over a query graph was only 0.33. On an average, the best accuracy over all the queries was only 0.15.

Since the number of possible worlds for PPI-small is extremely large, it may be that 10000 samples were not enough. We, thus, chose a very small graph – the input graph G shown in the example in Figure 1. Since there are only 6 edges in it, the total number of possible worlds is only $2^6 = 64$. We sampled 20 possible world graphs ($\sim 31\%$) from it, and ran the query Q in Figure 1 against the possible worlds using NAGA. For these samples, the best accuracy achieved by NAGA was only 0.50, while CHISEL demonstrated an accuracy of 0.75. The best accuracy level of 0.75 was attained by NAGA only after sampling 41 worlds ($\sim 64\%$).

This shows that sampling the possible worlds and running an approximate graph matching algorithm that works only for certain graphs is not enough to obtain good results, and is neither effective nor scalable. CHISEL successfully avoids this expensive sampling procedure by utilizing the possible world modeling directly in its framework to find good matches.

5.5.5 *Query Degree Distribution*

An interesting erratic effect was observed over different query sizes for runtime for both IMDb and PPI-complete datasets (as earlier pointed out in Section 5.3). As shown in Figure 8(a), the query processing time was highest for 5-vertex sized queries while particularly low for the larger 7- and 9-vertex sized queries. On fur-

Table 4: Indexing time and memory consumption of CHISEL.

Measures	PPI-complete	YAGO	IMDb
Time (s)	5616.28	66.71	60.16
Memory (GB)	440.00	8.40	7.00

ther analysis, it was observed that queries, consisting of vertices whose label-matching vertices in the input graph exhibit large degrees, adversely affected the matching subgraph computation time. Since the query sets for each query size were chosen *randomly*, the set for 5-vertex size had a particularly high number of such high-degree graphs than 7- and 9-vertex sets and, hence, the "anomaly".

To confirm the above behavior and to alleviate the above occurrence, we sampled a query set from IMDb with similar vertex degree distribution. We refer to this query set as *IMDb-Induced*. The query set was chosen in a progressive manner as follows. First, random queries of size 13 were chosen. Then, queries of size 11 were chosen by considering subgraphs from this set, and so on. This ensures that the degree distribution of the different query sets do not vary widely. On this modified query set, the query processing time increases only slightly and smoothly across the query sizes (Figure 8(b)). This confirms the effect of query degree distribution on the runtime complexity of CHISEL. Thus, it was the presence of such anomalous high-degree query vertices (in the randomly generated queries) that attributed to the erratic behavior at some points of Figure 3. Removing such anomalous vertices resulted in a smoother curve, as shown in Figure 8(a).

Figure 8(b) shows that there is no appreciable effect of query degree distribution on the accuracy of the algorithms. Similar results were observed for the PPI-complete dataset as well.

5.6 Indexing Requirements

Table 4 tabulates the indexing time and memory requirements of CHISEL for the different datasets. Even for the very large PPI-complete dataset with more than a billion edges, the memory footprint is not very high and the indexing time is less than 2 hours. The smaller-sized YAGO and IMDb datasets require only a minute and less than 10GB of memory. This shows that CHISEL is applicable for diverse applications using commodity hardware.

5.7 Real World Use Case: StringDB

In this section, we show the performance of CHISEL in a real life use case for example queries from String DB (`version10.string-db.org/cgi/input.pl`) containing synthetases and regulators connected to long chain fatty acyl-CoA synthetase. We obtain the "most confident" gold annotated result for 2 different queries (having 11 and 16 vertices) obtained by varying the number of interactors. The result consists of the exact locations (in terms of vertex IDs) of the PPI-complete graph where this query structure is important (in terms of score). Detection of such sites is useful in identifying mutation regions for early detection of cancer and other diseases.

We compare the performance of CHISEL and NAGA [27] in extracting subgraphs similar to the queries. Figure 9 depicts the obtained matching subgraphs from the algorithms. (Node identifiers shown in the figure are protein names and are different from labels of orthologous groups that are queried.) CHISEL demonstrates a high structural similarity for both the queries and is seen to outperform NAGA. The accuracy gap of CHISEL is more for the larger query since the discriminative power from the probabilistic modeling and structural similarity via statistical significance increases when more number of vertices and edges are involved. The inherent modeling of PWS by CHISEL (by taking the edge existential

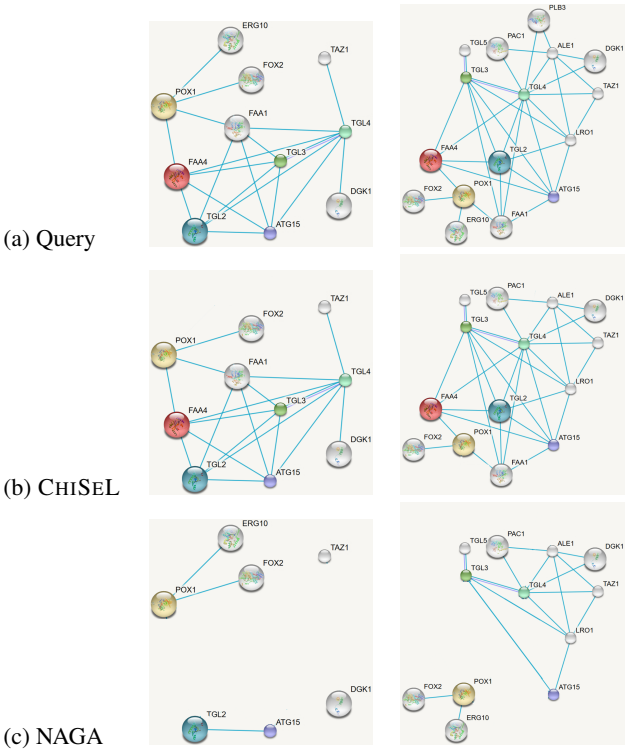


Figure 9: Evaluation on real dataset: (a) String DB queries, and corresponding results from (b) CHISEL and (c) NAGA.

probabilities into account) enables it to accurately identify the location of the query subgraphs that are most important in terms of the protein interactions. It needs to be noted that there are several occurrences of the same query subgraph based on labels and edges in the PPI-complete graph. However, the vertex IDs provided as part of the ground truth provide the exact list of vertices that are the most important. As shown in Figure 9, CHISEL correctly finds most of them and outperforms NAGA. The average query processing time for the two queries for NAGA was 27.6s, while CHISEL took only 0.31s.

Hence, it can be concluded that CHISEL provides an effective and efficient algorithm for approximate subgraph matching in uncertain graphs for real-life use cases.

6. CONCLUSIONS

This paper proposed CHISEL to handle approximate subgraph querying in large labeled probabilistic graphs. Our working principle hinges on the principle of *statistical significance* measured using the *chi-squared statistic* and *greedy neighborhood search* integrating both *structural similarity* and *possible world semantics*. Experiments showed that CHISEL is effective and efficient on large-scale graphs. Using it for a real-life use case showed that the results found were highly relevant and accurate.

In future, we would like to extend the solution to other types of graphs such as weighted graphs, hyper-graphs, etc.

Acknowledgments

We thank the anonymous reviewers for their constructive comments, Garima Gaur for her helpful suggestions, and the authors of [27] for the implementation details of NAGA.

7. REFERENCES

[1] A. Abou-Rjeili and G. Karypis. Multilevel Algorithms for Partitioning Power-law Graphs. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 124–124, 2006.

[2] E. Adar and C. Re. Managing Uncertainty in Social Networks. *Data Engineering Bulletin*, 30(2):15–22, 2007.

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

[4] A. Armiti and M. Gertz. Geometric Graph Matching and Similarity: A Probabilistic Approach. In *International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 1–12, 2014.

[5] A. Arora, M. Sachan, and A. Bhattacharya. Mining Statistically Significant Connected Subgraphs in Vertex Labeled Graphs. In *International Conference on Management of Data (SIGMOD)*, pages 1003–1014, 2014.

[6] S. Asthana, O. D. King, F. D. Gibbons, and F. P. Roth. Predicting Protein Complex Membership using Probabilistic Network Reliability. *Genome Research*, 14(6):1170–1175, 2004.

[7] L. Babai. Graph Isomorphism in Quasipolynomial Time. In *Symposium on Theory of Computing (STOC)*, pages 684–697, 2016.

[8] F. R. Bach, M. Zaslavskiy, and J. P. Vert. Global Alignment of Protein LC Protein Interaction Networks by Graph Matching Methods. *Bioinformatics*, 15(12):259–267, 2009.

[9] P. Barcelo, L. Libkin, and J. L. Reutter. Querying Graph Patterns. In *Symposium on Principles of Database Systems (PODS)*, pages 199–210, 2011.

[10] G. Bejerano, N. Friedman, and N. Tishby. Efficient Exact p-value Computation for Small Sample, Sparse and Surprisingly Categorical Data. *Journal of Computational Biology*, 11(5):867–886, 2004.

[11] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa. Injecting Uncertainty in Graphs for Identity Obfuscation. *PVLDB*, 5(11):1376–1387, 2012.

[12] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr., and T. M. Mitchell. Toward an Architecture for Never-ending Language Learning. In *Conference on Artificial Intelligence (AAAI)*, pages 1306–1313, 2010.

[13] R. Chandrasekar. Elementary? Question Answering, IBM's Watson, and the Jeopardy! Challenge. *Resonance*, 19:222–241, 2014.

[14] C. Chen, X. Yan, S. Y. Philip, J. Han, D. Q. Zhang, and X. Gu. Towards Graph Containment Search and Indexing. In *PVLDB*, pages 926–937, 2007.

[15] Y. Chen, X. Zhao, X. Lin, Y. Wang, and D. Guo. Efficient Mining of Frequent Patterns on Uncertain Graphs. *Transactions on Knowledge and Data Engineering*, 31(2):287–300, 2019.

[16] J. Cheng, Y. Ke, W. Ng, and A. Lu. FG-Index: Towards Verification-free Query Processing on Graph Databases. In *International Conference on Management of Data (SIGMOD)*, pages 857–872, 2007.

[17] T. Y. Cheung. State of the Art of Graph-based Data Mining. *Transactions on Software Engineering*, 5(1):59–68, 1983.

[18] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty Years of Graph Matching in Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

[19] S. A. Cook. The Complexity of Theorem-proving Procedures. In *Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

[20] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph Isomorphism Algorithm for Matching Large Graphs. *Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.

[21] N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. *International Journal of Very Large Data Bases*, 16(4):523–544, 2007.

[22] R. de Virgilio, A. Maccioni, and R. Torlone. Approximate Querying of RDF Graphs via Path Alignment. *Distributed Parallel Databases*, 33:555–581, 2015.

[23] S. Dutta. MIST: Top-k Approximate Sub-string Mining Using Triplet Statistical Significance. In *European Conference on Information Retrieval (ECIR)*, pages 284–290, 2015.

[24] S. Dutta and A. Bhattacharya. Most Significant Substring Mining based on Chi-square Measure. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 319–327, 2010.

[25] S. Dutta and A. Bhattacharya. Mining Statistically Significant Substrings Based on the Chi-Square Measure. In *Pattern Discovery Using Sequence Data Mining: Applications and Studies*, pages 73–82. IGI Global, 2012.

[26] S. Dutta and J. Lauri. Finding a Maximum Clique in Dense Graphs via Chi-Square Statistics. In *International Conference on Information and Knowledge Management (CIKM)*, pages 2421–2424, 2019.

[27] S. Dutta, P. Nayek, and A. Bhattacharya. Neighbor-Aware Search for Approximate Labeled Graph Matching using the Chi-Square Statistics. In *International Conference on World Wide Web (WWW)*, pages 1281–1290, 2017.

[28] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective Community Search Over Large Spatial Graphs. *PVLDB*, 10(6):709–720, 2017.

[29] B. Gallagher. Matching Structure and Semantics: A Survey on Graph-based Pattern Matching. In *Conference on Artificial Intelligence (AAAI)*, pages 45–53, 2006.

[30] H. H. Gan, D. Fera, J. Zorn, N. Shiffeldrim, M. Tang, U. Laserson, N. Kim, and T. Schlick. RAG: RNA-As-Graphs Database - Concepts, Analysis, and Features. *Nutrition and Health*, 5(1-2):1285–1291, 1987.

[31] Y. Gao, X. Miao, G. Chen, B. Zheng, D. Cai, and H. Cui. On Efficiently Finding Reverse k-nearest Neighbors over Uncertain Graphs. *The VLDB Journal*, 26:467–492, 2017.

[32] R. Giugno and D. Shasha. GraphGrep: A Fast and Universal Method for Querying Graphs. In *International Conference on Pattern Recognition (ICPR)*, pages 112–115, 2002.

[33] Y. Gu, C. Gao, L. Wang, and G. Yu. Subgraph Similarity Maximal All-matching over a Large Uncertain Graph. In *International Conference on World Wide Web (WWW)*, pages 755–782, 2016.

[34] P. Hall. Chi-Squared Approximations to the Distribution of a Sum of Independent Random Variables. *The Annals of Probability*, 11(4):1028–1036, 1983.

[35] S. Han, Z. Peng, and S. Wang. The Maximum Flow Problem of Uncertain Network. *Information Sciences*, 265:167–175, 2014.

[36] 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.

[37] P. Hintsanen and H. Toivonen. Finding Reliable Subgraphs from Large Probabilistic Graphs. In *International Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 3–23, 2008.

[38] M. Hua and J. Pei. Probabilistic Path Queries in Road Networks: Traffic Uncertainty Aware Path Selection. In *International Conference on Extending Database Technology (EDBT)*, pages 347–358, 2010.

[39] 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.

[40] H. Jiang, H. Wang, P. S. Yu, and S. Zhou. GString: A Novel Approach for Efficient Search in Graph Databases. In *International Conference on Data Engineering (ICDE)*, pages 566–575, 2007.

[41] R. Jiang, Z. Tu, T. Chen, and F. Sun. Network Motif Identification in Stochastic Networks. *Proceedings of the National Academy of Sciences*, 103(25):9404–9409, 2006.

[42] R. Jin, L. Liu, and C. C. Aggarwal. Discovering Highly Reliable Subgraphs in Uncertain Graphs. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 992–1000, 2011.

[43] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-constraint Reachability Computation in Uncertain Graphs. *PVLDB*, 4(9):551–562, 2011.

[44] A. Jüttner and P. Madarasi. VF2++ - An Improved Subgraph Isomorphism Algorithm. *Discrete Applied Mathematics*, 242:69–81, 2018.

[45] V. Kassiano, A. Gounaris, A. N. Papadopoulos, and K. Tsichlas. Mining Uncertain Graphs: An Overview. In *International Symposium on Algorithmic Aspects of Cloud Computing (ALGOCLOUD)*, pages 87–116, 2016.

[46] B. P. Kelley, B. Yuan, F. Lewitter, R. Sharan, B. Stockwell, and T. Ideker. PathBLAST: A Tool for Alignment of Protein Interaction Netw. *Nucleic Acids Research*, 32:83–88, 2004.

[47] A. Khan and L. Chen. On Uncertain Graphs Modeling and Queries. *PVLDB*, 8(12):2042–2043, 2015.

[48] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. NeMa: Fast Graph Search with Label Similarity. *PVLDB*, 6(3):181–192, 2013.

[49] A. Khan, X. Yan, and K. L. Wu. Towards Proximity Pattern Mining in Large Graphs. In *International Conference on Management of Data (SIGMOD)*, pages 867–878, 2010.

[50] G. Kollios, M. Potamias, and E. Terzi. Clustering Large Probabilistic Graphs. *Transactions of Knowledge and Data Engineering*, 25(2):325–336, 2011.

[51] S. Kpodjedo, P. Galinier, and G. Antoniol. Using Local Similarity Measures to Efficiently Address Approx. Graph Matching. *Discrete Appl. Maths.*, 164:161–177, 2014.

[52] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *International Conference on Data Mining (ICDM)*, pages 313–320, 2001.

[53] M. Kuramochi and G. Karypis. GREW – A Scalable Frequent Subgraph Discovery Algorithm. In *International Conference on Data Mining (ICDM)*, pages 439–442, 2004.

[54] M. Kuramochi and G. Karypis. Finding Frequent Patterns in a Large Sparse Graph. *Data Mining & Knowledge Discovery*, 11(3):243–271, 2005.

[55] J. Larrosa and G. Valiente. Constraint Satisfaction Algorithms for Graph Pattern Matching. *Mathematical Structures in Computer Science*, 12(4):403–422, 2002.

[56] J. Leskovec and C. Faloutsos. Sampling from Large Graphs. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 631–636, 2006.

[57] G. Li, L. Yan, and Z. Ma. An Approach for Approximate Subgraph Matching in Fuzzy RDF Graph. *Fuzzy Sets and Systems*, 376:106–126, 2019.

[58] J. Li, Z. Zou, and H. Gao. Mining Frequent Subgraphs over Uncertain Graph Databases under Probabilistic Semantics. *The VLDB Journal*, 21(6):753–777, 2012.

[59] X. Li, R. Cheng, Y. Fang, J. Hu, and S. Maniu. Scalable Evaluation of k-NN Queries on Large Uncertain Graphs. In *International Conference on Extending Database Technology (EDBT)*, pages 181–192, 2018.

[60] X. Lian, L. Chen, and G. Wang. Quality-Aware Subgraph Matching Over Inconsistent Probabilistic Graph Databases. *Transactions on Knowledge and Data Engineering*, 28(6):1560–1574, 2016.

[61] Z. Liang, M. Xu, M. Teng, and L. Niu. NetAlign: A Web-based Tool for Comparison of Protein Interaction Networks. *Bioinfomatics*, 22(17):2175–2177, 2006.

[62] L. Liu, R. Jin, C. Aggrawal, and Y. Shen. Reliable Clustering on Uncertain Graphs. In *International Conference on Data Mining (ICDM)*, pages 459–468, 2012.

[63] A. Mahmood, H. Farooq, and J. Ferzund. Large Scale Graph Matching (LSGM): Techniques, Tools, Applications and Challenges. *International Journal of Advanced Computer Science and Applications*, 8(4):494–499, 2017.

[64] S. Maniu, R. Cheng, and P. Senellart. An Indexing Framework for Queries on Probabilistic Graphs. *Transactions on Database Systems*, 42(2):1–34, 2017.

[65] B. T. Messmer and H. Bunke. Efficient Subgraph Isomorphism Detection: A Decomposition Approach. *Transactions on Knowledge and Data Engineering*, 12(2):307–323, 2000.

[66] T. Mitchell, W. Cohen, E. Hruscha, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohammad, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-Ending Learning. In *Conference on Artificial Intelligence (AAAI)*, pages 2302–2310, 2015.

[67] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-Ending Learning. *Communications of the ACM*, 61(5):103–115, 2018.

[68] M. Mongiovi, R. Di Natale, R. Guigno, A. Pulvirenti, A. Ferro, and R. Sharan. SIGMA: A Set-Cover-Based Inexact Graph Matching Algo. *Journal of Bioinformatics and Computational Biology*, 8(2):199–218, 2010.

[69] W. E. Moustafa, A. Kimmig, A. Deshpande, and L. Getoor. Subgraph Pattern Matching over Uncertain Graphs with Identity Linkage Uncertainty. In *International Conference on Data Engineering (ICDE)*, pages 904–915, 2014.

[70] O. Papapetrou, E. Ioannou, and D. Skoutas. Efficient Discovery of Frequent Subgraph Patterns in Uncertain Graph Databases. In *International Conference on Extending*

Database Technology (EDBT), pages 355–366, 2011.

[71] P. Parchas, F. Gullo, D. Papadias, and F. Bonchi. The Pursuit of a Good Possible World: Extracting Representative Instances of Uncertain Graphs. In *International Conference on Management of Data (SIGMOD)*, pages 967–978, 2014.

[72] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest Neighbors in Uncertain Graphs. *PVLDB*, 3(1-2):997–1008, 2010.

[73] T. Read and N. Cressie. Pearson's χ^2 and the Likelihood Ratio Statistic G^2: A Comparative Review. *International Statistical Review*, 57(1):19–43, 1989.

[74] T. R. C. Read and N. A. C. Cressie. *Goodness-of-fit Statistics for Discrete Multivariate Data*. Springer, 1988.

[75] M. Rotmensch, Y. Halpern, A. Tlimat, S. Horng, and D. Sontag. Learning a Health Knowledge Graph from Electronic Medical Records. *Nature Scientific Rep.*, 7, 2017.

[76] M. Sachan and A. Bhattacharya. Mining Statistically Significant Substrings using the Chi-Square Statistic. *PVLDB*, 5(10):1052–1063, 2012.

[77] 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.

[78] R. Singh, J. Xu, and B. Berger. Global Alignment of Multiple Protein Interaction Networks with Application to Func. Orthology Detection. *Proceedings of the National Academy of Sciences*, 105(35):12763–12768, 2008.

[79] S. Song, Z. Zou, and K. Liu. Triangle-based Representative Possible Worlds of Uncertain Graphs. In *International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 283–298, 2016.

[80] S. Sun and Q. Luo. Scaling Up Subgraph Query Proc. with Efficient Subgraph Matching. In *International Conference on Data Engineering (ICDE)*, pages 220–231, 2019.

[81] Y. Tian, R. McEachin, C. Santos, D. States, and J. Patel. SAGA: A Subgraph Matching Tool for Biological Graphs. *Bioinformatics*, 23(2):232–239, 2006.

[82] Y. Tian and J. Patel. TALE: A Tool for Approximate Large Graph Matching. In *International Conference on Data Engineering (ICDE)*, pages 963–972, 2008.

[83] H. Tong and C. Faloutsos. Center-piece Subgraphs: Prob. Defn. and Fast Solutions. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 404–413, 2006.

[84] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast Best-effort Pattern Matching in Large Attributed Graphs. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 737–746, 2007.

[85] W. H. Tsai and K. Fu. Error-correcting Isomorphisms of Attributed Relational Graphs for Pattern Recognition. *Transactions on Systems, Man, and Cybernetics*, 9(12):757–768, 1979.

[86] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *Journal of ACM*, 23(1):31–42, 1976.

[87] L. G. Valiant. The Complexity of Enumeration and Reliability Prob. *J. of Computing*, 8(3):410–421, 1979.

[88] M. Wang, J. Zheng, J. Liu, W. Hu, S. Wang, X. Li, and W. Liu. PDD Graph: Bridging Electronic Medical Records and Biomedical Knowledge Graphs via Entity Linking. In *International Semantic Web Conference (ISWC)*, pages 219–227, 2017.

[89] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. In *International Conference on Data Mining (ICDM)*, pages 721–724, 2002.

[90] X. Yan, P. S. Yu, and J. Han. Graph Indexing Based on Discriminative Frequent Structure Analysis. *Transactions on Database Systems*, 30(4):960–993, 2005.

[91] X. Yan, P. S. Yu, and J. Han. Substructure Similarity Search in Graph Databases. In *International Conference on Management of Data (SIGMOD)*, pages 766–777, 2005.

[92] X. Yu, Y. Sun, P. Zhao, and J. Han. Query-driven Discovery of Semantically Similar Substructures in Heterogeneous Networks. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1500–1503, 2012.

[93] Y. Yuan, G. Wang, and L. Chen. Pattern Match Query in a Large Uncertain Graph. In *International Conference on Information and Knowledge Management (CIKM)*, pages 519–528, 2014.

[94] Y. Yuan, G. Wang, L. Chen, and B. Ning. Efficient Pattern Matching on Big Uncertain Graphs. *Information Sciences*, 339:369–394, 2016.

[95] Y. Yuan, G. Wang, L. Chen, and H. Wang. Efficient Subgraph Similarity Search on Large Probabilistic Graph Databases. *PVLDB*, 5(9):800–811, 2012.

[96] Y. Yuan, G. Wang, L. Chen, and H. Wang. Graph Similarity Search on Large Uncertain Graph Databases. *The VLDB Journal*, 24(2):271–296, 2015.

[97] Y. Yuan, G. Wang, H. Wang, and L. Chen. Efficient Subgraph Search over Large Uncertain Graphs. *PVLDB*, 4(11):876–886, 2011.

[98] R. Zass and A. Shashua. Probabilistic Graph and Hypergraph Matching. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 34–41, 2008.

[99] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing Stars: On Approximating Graph Edit Distance. *PVLDB*, 2(1):25–36, 2009.

[100] S. Zhang, J. Li, H. Gao, and Z. Zou. A Novel Approach for Efficient Supergraph Query Processing on Graph Databases. In *International Conference on Extending Database Technology (EDBT)*, pages 204–215, 2009.

[101] S. Zhang, S. Li, and J. Yang. GADDI: Distance Index Based Subgraph Matching in Biological Networks. In *International Conference on Extending Database Technology (EDBT)*, pages 192–203, 2009.

[102] S. Zhang, J. Yang, and W. Jin. SAPPER: Subgraph Indexing and Approximate Matching in Large Graphs. *PVLDB*, 3(1-2):1185–1194, 2010.

[103] P. Zhao and J. Han. On Graph Query Optimization in Large Networks. *PVLDB*, 3(1-2):340–351, 2010.

[104] L. Zou, L. Chen, and Y. Lu. Top-k Subgraph Matching Query in a Large Graph. In *PhD Workshop: International Conference on Information and Knowledge Management (CIKM)*, pages 139–146, 2007.

[105] L. Zou, L. Chen, J. X. Yu, and Y. Lu. A Novel Spectral Coding in a Large Graph Database. In *International Conference on Extending Database Technology (EDBT)*, pages 181–192, 2008.

[106] Q. Zou, S. Liu, and W. W. Chu. CTree: A Compact Tree for Indexing XML Data. In *International Workshop on Web Info. and Data Management (WIDM)*, pages 39–46, 2004.

[107] Z. Zou, H. Gao, and J. Li. Discovering Frequent Subgraphs over Uncertain Graph Databases under Probabilistic Semantics. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 633–642, 2010.