Towards Frequent Subgraph Mining on Single Large Uncertain Graphs

Yifan Chen† Xiang Zhao†§ Xuemin Lin‡ Yang Wang‡

† *National University of Defense Technology, China*
{yfchen, xiangzhao}@nudt.edu.cn

‡ *The University of New South Wales, Australia*
{lxue, wangy}@cse.unsw.edu.au

§ *Collaborative Innovation Center of Geospatial Technology, China*

***Abstract*—Uncertainty is intrinsic to a wide spectrum of real-life applications, which inevitably applies to graph data. Representative uncertain graphs are seen in bio-informatics, social networks, etc. This paper motivates the problem of frequent subgraph mining on single uncertain graphs. We present an *enumeration-evaluation* algorithm to solve the problem. By showing support computation on an uncertain graph is #P-hard, we develop an approximation algorithm with accuracy guarantee for this purpose. To enhance the solution, we devise optimization techniques to achieve better mining performance. Experiment results on real-life data confirm the usability of the algorithm.**

I. Introduction

Uncertainty is intrinsic to a wide spectrum of real-life applications, either endogenous or extraneous. For example, in a professional collaboration networks, given `Bill` and `Matthew`, it may not be possible to definitely assert a relation of the form "`Bill` cooperates well with `Matthew`", using available information at hand. Our confidence in such relation is commonly quantified by probability. We say that the relation exists with a probability of p, and the value of p can be determined manually by domain experts using available information, or automatically by information extraction and generation rules. Thus, this paper focuses on *uncertain graphs*, where our knowledge is presented as a graph with uncertainty associated to edges. Besides social networks [1, 13], uncertain graph model has been incorporated in communication [8] and wireless sensor networks [6], protein interaction [2, 22] and regulatory networks in biology [10], etc. Research efforts have been dedicated to a number of interesting problems on uncertain graphs, [3, 11, 17, 19] to list a few.

Frequent pattern mining has been a focused theme in data mining for more than a decade. Graph patterns, or frequent subgraphs, are of particular interest lately, which are subgraphs found from a *collection of small graphs* [23] or *single large graph* [15] with support no less than a threshold. Frequent subgraphs encode important properties of graphs, and hence, are useful at characterizing graph datasets, classifying and clustering graphs, and building structural indices [5].

While the notion of frequent subgraph and mining methods on deterministic graphs are well understood, the case becomes more intriguing and less studied on uncertain graphs. An uncertain graph is a special edge-weighted graph, where the weight on each edge (u, v) is the existence probability of the connection between vertices u and v. Lately, research

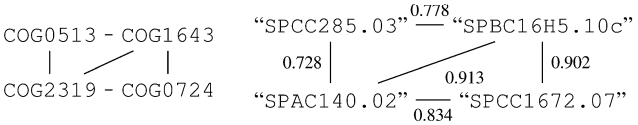


Fig. 1. Protein Interactions of *Fission Yeast*

effort has been dedicated to frequent subgraph mining (FSM) on a collection of small uncertain graphs. Being equally important, however, the problem on single large uncertain graphs remains open, given that real-life large networks are increasingly involved with uncertainty in nature. Thus, this research comes in response trying to fill the gap.

We investigate FSM on single uncertain graphs, which finds applications in bio-medicine, social behavior analytics, e.g., currently in **biology**, biologists are interested in identifying *functional modules* and *evolutionarily conserved subnetworks* from biological networks such as protein interaction networks. However, owing to measurement limit, protein interaction is subject to uncertainties. Uncovering the expected functional association patterns enables us to predict protein functions [2].This motivates us to discover subgraphs that frequently appear in an uncertain graph with high probability.

Currently, initial inquiry into real-life protein interaction network has unveiled the usability of our research. On the left of Figure 1 is a frequent subgraph mined from uncertain protein interactions of *fission yeast* [1], where vertices are annotated with clusters of orthologous group (COG) functions as labels. The frequent subgraph suggests a functional association pattern among proteins, with possible interpretation by gene expression procedure. COG0513 proteins unwind RNA for COG0724 proteins to bind, while producing ATP for COG1643 proteins. Both COG1643 and COG0724 proteins help the translation of COG2319 proteins, which is to modify the assembly dynamics of microtubules. We also provide an instance (the node IDs names the proteins while the node labels (omitted) corresponds to the pattern) of such interaction found in fission yeast on the right of Figure 1 for better appreciation.

While a subgraph on a deterministic graph is measured by *support*, this notion does not make sense on uncertain graphs, since the containment relationship becomes vague, or non-deterministic, due to the probabilistic presence of structure.

[1] http://www.string-db.org

1550-4786/15 $31.00 © 2015 IEEE
DOI 10.1109/ICDM.2015.88

Existing work defined *expected support* on a collection of small uncertain graphs [29], which counts the contribution from an implied graph as long as it contains a subgraph at least *once*. Extending the notion, we define expected support on single uncertain graphs as the aggregated support weighted to its existence probability over all possible graphsSubgraphs surpassing a given threshold are considered *frequent*. Due to the *shift of definition*, existing algorithm is no longer applicable to single uncertain graphs. Therefore, we propose an efficient solution with accuracy guarantee, where the *computational challenge* of proper handling edge-based probability and vertex-based support is addressed.

Contributions. To the best of our knowledge, this work makes a first effort to FSM on single uncertain graphs. In summary, we make the following contributions:

(1) To capture subgraphs that not only appear frequently but also have high confidence, in terms of uncertainty, we define frequent subgraph on single uncertain graphs based on *expected support*. Patterns under expected semantics are useful in exploring motifs in an uncertain network. We show that computing expected support over an uncertain graph is #P-hardand propose an approximation algorithm to obtain an interval containing the real value with accuracy guarantee.

(2) By casting the relations between the interval and threshold σ into three cases, we guarantee that, with probability at least $1-\delta$, any subgraph with expected support no less than σ will be output, but any subgraph with expected support less than $(1-\varepsilon)\sigma$ will not be output, where ε is an error tolerance, and $1-\delta$ is a degree of confidence, $\varepsilon, \delta \in [0,1]$. We propose to evaluate support via Mont-Carlo simulations following fully polynomial randomized approximation scheme (FPRAS), to achieve both high accuracy and efficiency. To further expedite, we devise two optimization techniques to share the computation among samples, and to early prune candidate subgraphs, respectively.

(3) Using real-life data, we conduct an extensive experimental study of our algorithm fanta. We find that the framework works on single large uncertain graphs, which is able to catch frequent subgraphs in terms of expected existence probability. The approximation algorithm provides fairly close estimation for support evaluation by tuning quality-control parameters δ and ε. Moreover, the optimization techniques are effective in reducing the running down to 71.3% at most.

Organization. We state the problem in Section II, and Section III presents the framework. We investigate support evaluation and optimizations in Sections IV and V, respectively. Section VI describes experiments, followed by conclusion.

Related Work. All existing work on FSM on uncertain graphs is developed on transaction settings, i.e., multiple small/medium uncertain graphs. FSM on uncertain graph transactions under *expected* semantics considers a subgraph frequent if its expected support is greater than the threshold. Algorithm MUSE was proposed to address the NP-hard problem [29], which is a combination of exact and approximation algorithms. The algorithm was later improved by leveraging edge and connectivity indices [20]. Besides, FSM under *probabilistic* semantics was also investigated [26], where a subgraph is frequent if its ϕ-frequent probability is greater the threshold. According to existing discussion [16], frequent subgraphs under the expected semantics is suitable for exploring motifs in an uncertain graph while frequent subgraphs under the probabilistic semantics is suitable for exploring features in an uncertain graph.

Our research is different in that we focus on single large uncertain graphs, which can be regarded as a general case of transaction setting allowing disconnectivity. Hence, our problem is more challenging due to not only the #P-hardness of support computation, but also the difficulty of elegant handling vertex-based support and edge-based uncertainty.

Mining maximal cliques is of interests on uncertain graphs [19, 28]. Reliable subgraphs was for finding connected subgraphs with a high probability [9, 11]. Other mining tasks over uncertain graphs include clustering [14, 17], core decomposition [3], etc. Querying uncertain graphs also receives much attention, and a natural problem is about reachability [12]. As to complex structures, subgraph search [18, 24, 25] was investigated. Additionally, the notion of structural-context similarity was proposed to analyze uncertain graphs [27].

For various types of FSM problems and algorithms on deterministic graphs, we refer readers to [5] for a recent survey.

II. Preliminary

A. Deterministic Graph

For ease of exposition, we assume graph is undirected with neither self-loops nor multi-edges. A *deterministic graph* G is a tuple $(V_G, E_G, l_G, \Sigma_G)$, where V_G is a set of vertices, $E_G \subseteq V_G \times V_G$ is a set of edges, and $l_G : V_G \cup E_G \to \Sigma_G$ is a labeling function that assigns labels to vertices and edges.

A graph g is *subgraph isomorphic* to another graph G, denoted by $g \sqsubseteq G$, if there exists an *injection* $f : V_g \to V_G$ such that (1) $\forall v \in V_g$, $f(v) \in V_G \wedge l_g(v) = l_G(f(v))$; and (2) $\forall (u,v) \in E_g$, $(f(u), f(v)) \in E_G \wedge l_g(u,v) = l_G(f(u), f(v))$. Hence, g is a *subgraph* of G, G is a *supergraph* of g, and $f(g)$ is an *embedding* of g in G. g is a *direct supergraph* of g' if $g' \sqsubseteq g$ and $|E_g| = |E_{g'}| + 1$.

Consider two graphs $g \sqsubseteq G$, and a *support threshold* τ, assume there is a function to measure the *support* of g in G, denoted by $sup(g, G)$. If $sup(g, G) \geq \tau$, we say g is a *frequent subgraph* of G. There are several ways to define the support of a subgraph g in a single graph G, and the most intuitive way is to count the isomorphisms of g in G. Unluckily, however, one may easily verify that this measure is not anti-monotonic [15].

Anti-monotonicity is crucial to the development of algorithms that can effectively prune the search space, without which they have to carry out exhaustive search of the whole pattern space. As a consequence, existing literature presents several anti-monotone support measures based on (1) minimum image (MI) [4], (2) harmful overlap (HO) [7], and (3) maximum independent sets (MIS) [15]. These measures were all established on subgraph isomorphisms, but differ in the extent of *compatible* overlap among them. Particularly, MI is the only measure that can be computed efficiently, while HO and MIS involve solving NP-complete problems; the result set of MI is always a superset of that of HO/MIS, and thus, the

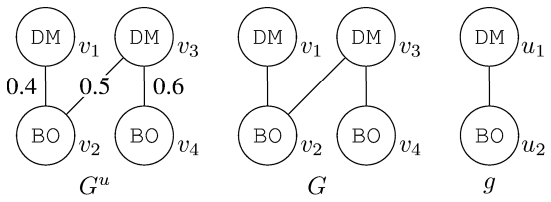


Fig. 2. Example Graphs

desired answers can be derived from the results of MI with additional computation. Therefore, we adopt MI as the support measure in the following discussion, whereas the algorithms are readily to be extended to others with minor effort.

Consider a set of distinct subgraph isomorphisms F from g to G. $F(v)$ denotes the set of (distinct) vertices v' such that there exists an isomorphism mapping $v \in V_g$ to $v' \in V_G$. The *minimum image based support* (shortened to "support" hereafter) of g in G is $sup(g, G) = \min_{v \in V_g}\{|F(v)|\}$.

EXAMPLE *1:* Consider in Figure 2 deterministic graphs G and g, which model professional social networks with members represented by vertices and collaborations by edges. Each vertex takes profession as its label, e.g., BO indicates an biologist; edge labels are omitted for clarity. There are 3 subgraph isomorphisms between g and G, i.e., (u_1, u_2) to (v_1, v_2), (v_3, v_2) and (v_3, v_4). Thus, $sup(g, G) = \min\{2, 2\} = 2$.

B. Uncertain Graph

An *uncertain graph* is a tuple $G^u = (G, P)$, where G is a deterministic graph, and $P : E_G \to (0, 1]$ is a probability function that assigns each edge e with an existence probability, denoted by $P(e)$, $e \in E_G$. G is the *backbone graph*.

Upon determination of each edge, a deterministic graph G^i is *implied*. An uncertain graph G^u implies $2^{|E_G|}$ possible graphs in total, each of which is a structure G^u may exist as.

We consider the model where independence among edges holds, which finds various applications [2, 8, 10]. The probability of G^u implying G^i, or *existence probability* of G^i, can be computed by including or excluding the edges:

$$P(G^u \Rightarrow G^i) = \prod_{e \in E_{G^i}} P(e) \prod_{e \in E_G \setminus E_{G^i}} (1 - P(e)).$$

While the classic notion of support becomes intriguing on uncertain graphs, we resort to expected support, which is a probability distribution over the support in implied graphs.

DEFINITION *1:* The *expected support* of a subgraph g in an uncertain graph G^u is defined as:

$$esup(g, G^u) = \sum_{i=1}^{2^{|E_G|}} P(G^u \Rightarrow G^i) \cdot sup(g, G^i), \qquad (1)$$

where G^i is an implied graph of G^u.

Given an expected support threshold σ, a subgraph g is said to be *frequent* if the expected support of g in G^u is no less than the threshold, i.e., $esup(g, G^u) \geq \sigma$.

PROPOSITION *1 (Anti-monotonicity):* Consider 2 graphs $g' \sqsubseteq g$ and uncertain graph G^u. $esup(g, G^u) \leq esup(g', G^u)$.

EXAMPLE *2:* Consider Figure 2, and assume support threshold $\sigma = 1$. Associating every edge of G with an existence probability, we construct an uncertain graph G^u, where probability models collaboration closeness between people. G is the backbone graph, $P(G^u \Rightarrow G) = 0.12$. Recall Example 1 that $sup(g, G) = 2$. Thus, $P(G^u \Rightarrow G) \cdot sup(g, G) = 0.24$. Accumulating the values from all 8 implied graphs, we have $esup(g, G^u) = 1.12 \geq \sigma$, and thus, g is a frequent subgraph.

PROBLEM *1:* Given an uncertain graph $G^u = (G, P)$ and an expected support threshold σ, *FSM on an uncertain graph* finds all subgraphs g whose expected support is no less than the threshold, i.e., $\mathcal{G} = \{g \mid esup(g, G^u) \geq \sigma \wedge g \sqsubseteq G\}$.

We annotate the semantics of Definition 1. Suppose $esup(g, G^u) = 10$, and G_r denotes a randomly and independently chosen implied graph. It is *expected* that there are at least 10 distinct occurrences of g in G_r. Frequent subgraphs under the expected semantics is suitable for exploring motifs in an uncertain graph. Domain G^u is omitted onward when there is no ambiguity, e.g., "$esup(g)$".

III. ALGORITHM FRAMEWORK

We present an *enumeration-evaluation* algorithm named fanta (frequent subgraph mining on uncertain graphs):

- **Enumeration**: enumerate all possible candidate subgraphs;
- **Evaluation**: for each subgraph, compute its expected support, and decide whether to output as a result.

The enumeration phase is the same as that for FSM on an deterministic graph. Thus, any enumeration strategy leveraging the Apriori property can be used. The Apriori property states that supergraphs of an infrequent subgraph cannot be frequent. Specifically, all subgraphs of an uncertain graph can be organized in a rooted directed acyclic graph (DAG), where the nodes represent candidate subgraphs (with the root being $null$). An arc in the DAG from a pattern g' to g denotes that g' is a direct supergraph of g. We enumerate all possible subgraphs by starting from frequent single edges and attaching every time a new edge to those frequent subgraphs, so that subgraphs consisting of n edges can be found at level-n of the DAG. Only the children of a frequent subgraph will be enumerated. To avoid duplicate enumeration of a subgraph while retaining completeness, existing approach gSpan [23] imposes a lexicographic order among the subgraphs. We also employ this strategy for elegant enumeration.

The evaluation phase determines whether an enumerated subgraph is frequent by comparing its expected support with the threshold. A naïve procedure is to generate all implied graphs, compute and aggregate the support of the subgraph in every implied graph, and then derive the expected support and compare with the threshold. This can be rather time-consuming due to the large number of implied graphs and the high complexity of support computation, and hence, becomes unbearable to end-users. In order to achieve better runtime performance, we seek every opportunity to return answers within reasonable time. In the sequel, we will look into this problem and address it with an efficient algorithm.

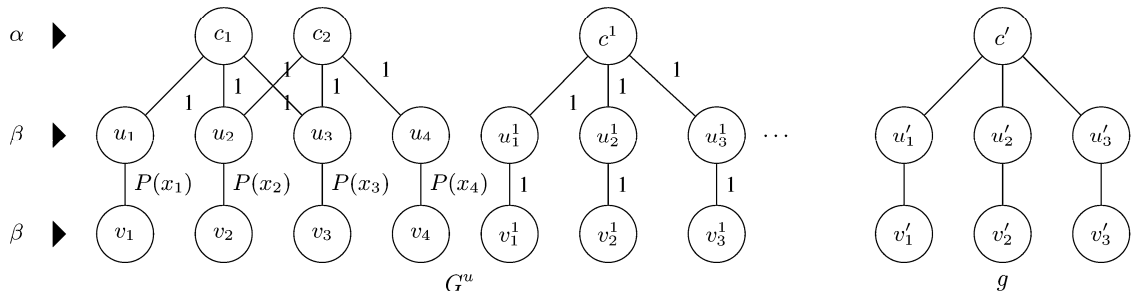


Fig. 3. Example of Graph Construction

IV. Support Evaluation

Computing expected support directly by Equation (1) is excessively complex, as there exist $2^{|E_G|}$ implied graphs of G^u. That is, for each candidate subgraph, we need to calculate its support in an exponential number of deterministic graphs, where expensive subgraph isomorphism tests are frequently involved. This section investigates whether the computational complexity can be reduced, and develops countermeasures.

A. Reformulation

We first reformulate expected support in Definition 1. Let $P(sup(g) = j)$ denote the aggregate possibility that support of subgraph g in an implied graph equals j, i.e., $P(sup(g) = j) = \sum_{G^i \in \Lambda_j(g)} P(G^u \Rightarrow G^i)$, where $\Lambda_j(g) = \{G^i \mid sup(g, G^i) = j\}$. It is then not difficult to rewrite Equation (1) as

$$esup(g) = \sum_{j=1}^{M_s} P(sup(g) = j) \cdot j, \tag{2}$$

where $M_s = sup(g, G)$ is the maximum support of g among all implied graphs of G^u. We further denote $P_j(g)$ the aggregate probability that the support of g in an implied graph is no less than j, i.e., $P_j(g) = \sum_{G^i \in \Delta_j(g)} P(G^u \Rightarrow G^i)$, where $\Delta_j(g) = \{G^i \mid sup(g, G^i) \geq j\}$.

Proposition *2:* $esup(g) = \sum_{j=1}^{M_s} P_j(g)$.

Proof: By Equation (2), we have

$$\begin{aligned} esup(g) &= \sum_{j=1}^{M_s-1} (P_j(g) - P_{j+1}(g)) \cdot j + P_{M_s}(g) \cdot M_s \\ &= \sum_{j=1}^{M_s-1} P_j(g) \cdot j - \sum_{j=2}^{M_s} P_j(g) \cdot (j-1) + P_{M_s}(g) \cdot M_s \\ &= P_1(g) + \sum_{j=2}^{M_s-1} P_j(g) + P_{M_s}(g). \end{aligned}$$

Therefore, $esup(g) = \sum_{j=1}^{M_s} P_j(g)$. ■

B. Computational Complexity

We first prove the #P-hardness of computing $P_\varsigma(g)$, where ς is an integer constant, and then, carry it on to that of $esup(g)$.

Theorem *1:* It is #P-hard to compute $P_\varsigma(g)$.

Proof: We reduce from the DNF counting problem, shown to be #P-hard. While the detailed proof is omitted in the interest of space, we use Example 3 to illustrate the idea. ■

Example *3:* Consider a boolean formula in disjunctive normal form (DNF) $D = (x_1 \wedge x_2 \wedge x_3) \vee (x_2 \wedge x_3 \wedge x_4)$ with probability $P(x_1), P(x_2), P(x_3), P(x_4)$ that x_1, x_2, x_3, x_4 are assigned **true**, respectively. We construct subgraph g first, the vertices of which are divided into three groups, $\{c'\}$ (labeled α), $\{u'_1, u'_2, u'_3\}$ and $\{v'_1, v'_2, v'_3\}$ (labeled β), showing in the rightmost of Figure 3. As to uncertain graph G^u, $\varsigma - 1$ (disconnected) isomorphisms are first established, and the edge probabilities are all equal to 1, to guarantee $P(sup(g, G^u) \geq \varsigma - 1) = 1$. An additional part is built for G^u, involving $\{c_1, c_2\}$ (labeled α), $\{u_1, u_2, u_3, u_4\}$ and $\{v_1, v_2, v_3, v_4\}$ (labeled β). $\{c_1, c_2\}$ corresponds to the clauses of D, and u_i, v_i to x_i; an edge is added between c_1 and $\{u_1, u_2, u_3\}$ with probability 1, as x_1, x_2, x_3 are included in clause C_1, which is the same with c_2 and $\{u_2, u_3, u_4\}$; an edge is added between u_i and v_i with probability $P(x_i)$. Hence, the constructed uncertain graph is shown in the left of Figure 3.

Corollary *1:* It is #P-hard to compute $esup(g)$.

C. Evaluation Algorithm

Due to the #P-hardness, we propose an approximate evaluation algorithm with error tolerance ε. As an approximation algorithm, it is desirable that a subgraph will be returned if it is frequent (true positive); to achieve this, we have to compromise with infrequent subgraphs in the result set (false positive). For this purpose, we interpret the output of an evaluation as a closed interval $[\underline{esup}, \overline{esup}]$ that approximately contains the true value of $esup$, and carefully handle the following cases when evaluating subgraph g against support threshold σ:

- **Case 1**: If $\overline{esup}(g) < \sigma$, do not output g, since it is certain that $esup(g) < \sigma$;
- **Case 2**: If $\underline{esup}(g) \geq (1 - \varepsilon)\sigma$ and $\overline{esup}(g) \geq \sigma$, output g, as it is certain that $esup(g) \geq (1 - \varepsilon)\sigma$, and it is probable that $esup(g) \geq \sigma$; and
- **Case 3**: If $\overline{esup}(g) \geq \sigma$ and $\underline{esup}(g) < (1 - \varepsilon)\sigma$, it cannot be determined whether or not to output g, because we cannot decide whether $esup(g) \geq \sigma$ or $esup(g) < (1 - \varepsilon)\sigma$.

By intuition Case 3 is not desirable, in which case we cannot make a decision on g. Nevertheless, we observe that if the width of interval $[\underline{esup}, \overline{esup}]$ is within length $\varepsilon\sigma$, Case 3 will not happen. Thus, by enforcing the interval width at

most $\varepsilon\sigma$, it is sufficient to approximate $esup(g)$ by the interval $[\underline{esup}(g), \overline{esup}(g)]$, where only Cases 1 and 2 happen. This is crucial to the algorithm design, and we rely on this to determine whether to include g as a result.

Our approximation algorithm is based on Monte-Carlo simulations. By Proposition 2, we first approximate each $P_j(g)$, $j \in [1, M_s]$, and then aggregate the values to derive the interval $[\underline{esup}(g), \overline{esup}(g)]$. To ensure the internal is no larger than $\varepsilon\sigma$, we require the absolute error for each approximated $P_j(g)$ to not exceed $\frac{\varepsilon\sigma}{M_s}$. This requirement recalls a class of randomized algorithms - *randomized approximation scheme* - that can provide accuracy guarantee.

Given a confidence coefficient $\delta \in [0, 1]$, and an absolute error tolerance ε', we can use the value $\hat{p}$ produced by a randomized approximation scheme to estimate p, if $P(|\hat{p}-p| < \varepsilon') \geq 1-\delta$, where $1-\delta$ is a confidence degree. Therefore, to approximate $P_j(g)$ under the conditions of δ and ε', we resort to an algorithm based on Hoeffding's inequality.

LEMMA *1:* Let $X_1, X_2, \ldots, X_n$ be independent identically distributed Bernoulli random variables, and $X_i = 1$ with the probability p. The following inequality holds:

$$P(|\frac{1}{n}\sum_{i=1}^{n} X_i - p| \geq \varepsilon') \leq 2\exp(-2\varepsilon'^2 n).$$

Lemma 1 tells that the average mean of n sample observations provides an approximation of p with accuracy guarantee, and the sample size required for satisfying confidence degree $1-\delta$ and absolute error tolerance $\varepsilon' = \frac{\varepsilon\sigma}{2M_s}$ is

$$N \geq \frac{\ln(\frac{2}{\delta})}{2\varepsilon'^2} = \frac{2{M_s}^2 \ln(\frac{2}{\delta})}{\varepsilon^2\sigma^2}.$$

We present the baseline algorithm for evaluating a subgraph in Algorithm 1. It takes as input a subgraph g, an uncertain graph G^u, an error tolerance ε and a real number δ; and it outputs a boolean that indicates whether g is frequent. In particular, Line 1 allocates an empty array for storing the observations of $P_j(g)$'s, and computes in M_s the maximum support of g on the backbone graph G of G^u, where the embeddings are recorded and identified by ID's. Line 2 initializes the variables, as well as the sample size N. Then, we apply the Monte-Carlo method. Line 3 collects N randomly drawn implied graphs, or *sample graphs* G_i of G^u. Note that "sample graph" differs from "implied graph" in that two sample graphs may correspond to the same implied graph. As a minor optimization, instead of sampling on all edges, we only consider the *embedding edges* of g, i.e., $\mathcal{E}_m = \{e_i \mid e_i \in F(g)\}$, where F is a set of isomorphisms from g to G, $i \in [1, |\mathcal{E}_m|]$. This shrinks the probability space but does not affect the correctness of the result that samples over the whole uncertain graph. We still refer it as a sample graph of G^u, although it contains only (partial) embedding edges. Lines 4 – 7 simulate the support of g on each G_i, and aggregate the probabilities to corresponding variables. That is, to increase the *observation probability*, or approximated probability of $P_j(g)$ by $P(G^u \Rightarrow G_i)$, if j is no larger than $sup(g, G_i)$. After screening all sample graphs, Line 9 evaluates the approximate support via EvaluateSup.

Algorithm 1: CollectApproxEval(g, G^u, ε, δ)

Input : g is a subgraph; G^u is an uncertain graph; ε is the error tolerance; δ is a real number in $[0, 1]$.
Output: boolean - **true** if g is a frequent subgraph; **false**, otherwise.

1 $\mathcal{P} \leftarrow \emptyset$, $M_s \leftarrow$ ComputeSup(g, G);
2 $Y \leftarrow X_0 \leftarrow \cdots \leftarrow X_{M_s} \leftarrow 0$, $N \leftarrow \frac{2{M_s}^2 \ln(\frac{2}{\delta})}{\varepsilon^2\sigma^2}$;
3 $\Omega \leftarrow$ randomly draw N implied graphs of G^u;
4 **foreach** sample graph $G_i \in \Omega$ **do**
5 $\quad p \leftarrow$ ComputeProb(G_i), $sup \leftarrow$ ComputeSup(g, G_i);
6 $\quad$ **for** $j = 1$ to sup **do** $X_j \leftarrow X_j + p$;
7 $\quad Y \leftarrow Y + p$;
8 **for** $j = 1$ to M_s **do** $\mathcal{P} \leftarrow \mathcal{P} \cup \{\frac{X_j}{Y}\}$;
9 **switch** EvaluateSup$(\mathcal{P}, M_s)$ **do**
10 $\quad$ **case** 1 **return false**;
11 $\quad$ **case** 2 **return true**;
12 $\quad$ **case** 3 do nothing; /* invalid */

In general, function EvaluateSup works as follows. It takes as input the observation probabilities $\mathcal{P}[j]$ for $P_j(g)$'s, and an integer x, and outputs either Cases 1 or 2 (Case 3 should not be reached). Particularly, the current observation probability (support) $\hat{P}(g) = \sum_{j\in[1,M_s]} \mathcal{P}[j]$, and the output is bounded by $esup = \hat{P}(g) - \frac{M_s\varepsilon\sigma}{2x}$, and $\overline{esup} = \hat{P}(g) + \frac{M_s\varepsilon\sigma}{2x}$. Upon the interval, we make decision against the threshold σ.

Correctness and Complexity. The correctness of Algorithm 1 is guaranteed by Lemma 1, which ensures that any output subgraph meets the error tolerance, as long as there are enough sample graphs. As we memorize the embeddings of subgraph g along with the pattern growth, we compute M_s in $O(|F(g)||V_g|)$, where $F(g)$ is the set of embeddings of g. Then, we perform Monte-Carlo simulation. The complexity of support and probability computation is $O(|F(g)||V_g|)$ and $O(|E_G|)$, respectively. As the total number of drawn samples is bounded by $O((\frac{M_s}{\varepsilon\sigma})^2 \ln\frac{1}{\delta})$, the overall complexity is $O((\frac{M_s}{\varepsilon\sigma})^2 \ln\frac{1}{\delta}|F(g)||V_g|)$. Since it is bounded by a polynomial in $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$, Algorithm 1 belongs to FPRAS.

We remark that there can be false negatives, i.e., frequent subgraphs but not returned, when applying Algorithm 1 in the mining algorithm. This may lead to children of those subgraphs not to be tested, and hence, trigger more false negatives. However, some of them may still be visited via other paths in the DAG and included as answer again. Additionally, in practice, by carefully the specifying quality-control parameters δ and ε, we are able to further reduce the chance. Experiment result in Section VI-B confirms our argument.

V. OPTIMIZATION TECHNIQUES

A. Sharing Computation among Samples

A straightforward way to implement the Monte-Carlo based support evaluation is to compute subgraph support in each sample graph, and aggregate the observation probabilities. This can be time-consuming, and we go in quest of speedup.

EXAMPLE *4:* Consider in Figure 4 two sample graphs G' and G'' of uncertain graph G^u in Figure 2, and a subgraph p of

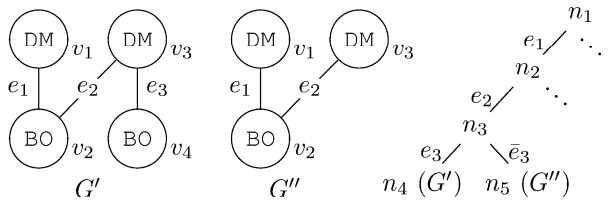


Fig. 4. Example of Computation Sharing

single edge as "(DM,BO)". (1) After computing $P(G^u \Rightarrow G'')$, we can further look at e_3 for calculating $P(G^u \Rightarrow G')$; (2) An embedding of p exists in G'', we can confirm G' containing that embedding without additional examination; and (3) To derive $sup(p, G'')$, we construct a data structure as

$$F((\texttt{DM}, \texttt{BO})) = \begin{pmatrix} (v_1, v_2) \\ (v_3, v_2) \end{pmatrix},$$

which can be leveraged for computing $sup(p, G')$ by adding another row of (v_3, v_4) in it.

Evidenced by the example, given two sample graphs $G'' \sqsubset G'$, three types of computational costs can be shared: (1) calculating existence probability; (2) testing whether an embedding exists; and (3) computing support of subgraphs. Next, we generalize the aforementioned ideas to multiple sample graphs such that containment does not necessarily hold.

1) Sharing Types (1) and (2) Costs: We construct a binary *sharing tree* with all sample graphs as leaf nodes. The tree is of depth $|\mathcal{E}_m|$, sample graphs are at depth $|\mathcal{E}_m|$, and the branching at depth i is based on the inclusion or exclusion of edge e_i.That is, the root is at depth 1, and its left child n_l leads to sample graphs having edge e_1, while the right child n_r comprises the remaining graphs that do not have e_1. This branching process carries on from e_1 till the last in $\mathcal{E}_m$, and conceptually, there are $2^{|\mathcal{E}_m|}$ leaf nodes. Thus, we can find a position for every sample graphs at the level of leaf nodes.

In our implementation, we do not generate the whole tree but only materialize a branch if there is a sample graph as its leaf. Particularly, we adopt the following method to generate sample graphs and the branches. At the root, we randomly decide for every G_i whether it goes to the left child n_l or right child n_r. If n_l is not empty, we materialize n_l in the tree, and then, randomly decide the whereabouts of the sample graphs; if n_l is empty, we stop growing this branch. The same procedure applies to n_r, and the iterations terminate when all sample graphs reach the level of leaf nodes. The existence probabilities of the sample graphs are calculated as a by-product during the growth, and sharing of calculation is maximized.

Afterwards, we compute the support on every sample graph. The first step is to determine the embeddings contained by every sample graph. To this end, we maintain an inverted index for embedding edges, with each embedding edge as an entry, and embeddings that contain the edge as its postings. The index helps find the embeddings missing due to the non-existence of an embedding edge, which can be easily built during support computation on the backbone graph.

For a sample graph, we trace back to the root from the leaf node where the graph is, and collect the *right* edges that encounters on the way. The whole set of embeddings except the sets of embeddings containing any collected edges are the those contained by the sample graph in question. Moreover, to enable computation reuse, we record the intermediate results at the tree nodes when processing every sample graph. Then, we conduct incremental computation on the intermediate results at the *lowest parent*, i.e., the first node having been processed on the aforementioned path, rather than on the root.

EXAMPLE *5:* Further to Example 4, on the right of Figure 4 is a partial sharing tree for G' and G'', where n_1 is the root, and n_4 (resp. n_5) is a leaf node accommodating G' (resp. G''). After processing G', we have at node n_3 the set of embeddings $\{(v_1, v_2), (v_3, v_2), (v_3, v_4)\}$ that can be contained by any sample graph under this branch. Then, due to the missing of edge e_3, we retrieve from the inverted index the embeddings that should not exist - $\{(v_3, v_4)\}$. Then, we eliminate it and obtain the embeddings contained by G'' - $\{(v_1, v_2), (v_3, v_2)\}$.

2) Sharing Type (3) Cost: Subsequently, we calculate the support of a subgraph g on the sample graphs in a serial fashion. Given two sample graphs G' and G'', we first describe how to compute $sup(g, G')$. For each embedding contained by G', we parallelize the *embedding vertices* to the vertices of g. Note that since we have already obtained the isomorphic mappings, this operation is much easier and faster than the general graph alignment. In specific, for a vertex v_g, we employ a map to keep a record - each embedding vertex in G' as a key, and the number of occurrences in all embeddings as its value. When we encounter an embedding having u as the embedding vertex for v_g, we increment the value of u in the map for v_g by one. After counting all vertices in all embeddings, we can derive $sup(g, G')$ by retrieving the size of the smallest map under every vertex of g.

To reuse the aforementioned intermediate result for G'', instead of building the maps from scratch, we first eliminate the embeddings not contained by G'', decrementing the corresponding values in the maps, and then append the embeddings that are not covered previously by G', incrementing the corresponding values. Afterwards, we can derive $sup(g, G'')$.

Model the cost of a unit increment/decrement to be fixed as c, and that of retrieving the size of the smallest map to be c'. Given a collection of sample graphs G_i each containing a set of embeddings $F_i \subseteq F$, where $i \in [1, N]$, the cost of support computation from scratch is $|F_i| \cdot c + c'$. Define $m_i^j \triangleq |(F_i \cup F_j) \setminus (F_i \cap F_j)|$. The cost of support computation from G_i (resp. G_j) to G_j (resp. G_i) is $m_i^j \cdot c + c'$ (resp. $m_j^i \cdot c + c'$), and $m_i^j = m_j^i$. If $|F_j| < m_i^j$, it is less costly to compute for G_j from scratch. To maximize the computational sharing, we formulate the following problem.

PROBLEM *2:* Given a collection of sample graphs $G_i \in \Omega$ each containing a set of embeddings $F_i \subseteq F$, where $i \in [1, N]$, the computational cost from G_i to G_j is $\min(m_i^j, |F_j|) \cdot c + c'$, find a *computation sequence with smallest cost* such that each graph is processed exactly once.

It is conjectured that Problem 2 is NP-hard, and hence, an efficient implementation is sought. In specific, we construct a bit array for each sample graph, each bit corresponding to an embedding in F. A bit is set to 1 if the graph contains the corresponding embedding, and 0, otherwise. In this way, the

cost between two graphs can be easily obtained by calculating the Hamming distance between their bits. Then, we adopt a heuristic computation sequence - from left to right according graph positions in the level of leaf nodes. The intuition is that two consecutive leaf nodes like G' and G'' in Figure 4 are usually of small distance, and hence, little incremental cost.

Compiling things together, we present Algorithm 2. To integrate it into the baseline algorithm, we replace Lines 4 – 8 of Algorithm 1 with "$\mathcal{P} \leftarrow$ ShareCompTree$(g,\ \Omega)$".

Algorithm 2: ShareCompTree$(g,\ \Omega)$

Input : g is a subgraph; Ω is a set of sample graphs.
Output: $\mathcal{P}$ is an array of probabilities.

1 construct a sharing tree, and associate every sample graph G_i of probability p_i to a leaf node;
2 mark the root as processed;
3 **foreach** sample graph $G_i (i \in [1, |\Omega|])$ **do**
4 find a route to the lowest parent node n having been processed via the path from G_i to root;
5 **foreach** tree node n' on the route **do**
6 mark n' as processed, compute the set of valid embeddings under this branch;
7 construct an embedding bit array B_i for G_i;
8 $sup \leftarrow$ compute from scratch for G_1;
9 **for** $j = 1$ to sup **do** $X_j \leftarrow X_j + p_1$;
10 $Y \leftarrow Y + p_1$;
11 **foreach** sample graph $G_i (i \in [2, |\Omega|])$ **do**
12 $d \leftarrow$ HammingDistance$(B_{i-1},\ B_i)$;
13 **if** $d < |F_i|$ **then** $sup \leftarrow$ incrementally compute based on the intermediate result for G_{i-1} ;
14 **else** $sup \leftarrow$ compute from scratch for G_i;
15 **for** $j = 1$ to sup **do** $X_j \leftarrow X_j + p_i$;
16 $Y \leftarrow Y + p_i$;
17 **for** $j = 1$ to M_s **do** $\mathcal{P} \leftarrow \mathcal{P} \cup \{\frac{X_j}{Y}\}$;
18 **return** $\mathcal{P}$

B. Pruning at Evaluation Checkpoints

While $N = \frac{2{M_s}^2 \ln(\frac{2}{\delta})}{\varepsilon^2 \sigma^2}$ samples are required to guarantee Case 3 not happen, we could make decision for a subgraph g earlier before the confirmation on interval width, if Cases 1 or 2 have already been satisfied. In these cases, we can stop the simulation early, and hence, save computation. A subsequent extreme of using this idea is to check the conditions for every sample graph, where the loss may outweigh the gain. To achieve the best of both worlds, we incorporate a periodical checkpoint mechanism such that early-stop can be achieved at certain checkpoints, as shown in Algorithm 3.

Checkpoints are given sample sizes when periodical check is conducted. Suppose we check r times, when the sample size reaches $c_1, \ldots, c_k, \ldots, c_r$, respectively, where $0 = c_1 < \cdots < c_k < \cdots < c_r = N$. For each checkpoint, we conduct incremental computation over $c_{k+1} - c_k$ graphs. In particular, Lines 3 – 4 retrieve the sample graphs required from Ω. Then, Line 5 computes with Ω' the current observation probabilities $P_j(g)$'s. Eventually, Line 6 determines whether to output the subgraph by EvaluateSup. Note that Case 3 here is not always invalid when it has not reached the terminal checkpoint.

Algorithm 3: CheckpointMech$(g,\ \Omega,\ C)$

Input : g is a subgraph; Ω is a set of sample graphs; $C = \{c_k\}$ is a set of checkpoints, $k \in [1, r]$.
Output: boolean - **true** if g is a frequent subgraph; **false**, otherwise

1 $\Omega' \leftarrow \emptyset$;
2 **for** $k = 1$ to $r - 1$ **do**
3 $c \leftarrow c_{k+1} - c_k$, $\Omega \leftarrow \Omega \setminus \Omega'$;
4 $\Omega' \leftarrow$ the first c sample graphs in Ω;
5 $\mathcal{P} \leftarrow$ ShareCompTree$(g,\ \Omega')$;
6 **switch** EvaluateSup$(\mathcal{P}, k)$ **do**
7 **case** 1 **return false**;
8 **case** 2 **return true**;
9 **case** 3 do nothing;

To introduce the mechanism into the framework, we replace Lines 4 – 12 of Algorithm 1 with Algorithm 3. Note that for each subgraph the computation sharing data structures are cached across checkpoints. Shortly, we will discuss the method details and choice of checkpoints.

1) A Structure-based Upper Bound: In the naïve implementation of EvaluateSup, an interval is obtained in terms of absolute error. However, absolute error can be quite large, especially when the sample size is small. Reversely, this advises that the evaluation performance can be improved if there is a tight bound to assist our decision. In nature, we attempt to find tighter bounds by considering structural property, rather than obtain bounds directly in terms of absolute error.

An upper bound $\mathcal{U}(g)$ of $esup(g)$ is used to prune a subgraph; if $\mathcal{U}(g) < \sigma$, g is determined not to be frequent. By Apriori property, $P_j(g)$ is bounded by $P_j(g')$, where g is a direct supergraph of g', i.e., $P_j(g) \leq P_j(g')$, if $g' \sqsubseteq g$ and $|E_g| = |E_{g'}| + 1$. While this inequality is straightforward, can we enhance its pruning power? An immediate idea is to multiply $P_j(g')$ with $P_j(e)$, where e is the additional edge in g. However, this is *incorrect* in general, and we argue that this is due to the special constraints of vertex-based support definition on single graphs. Nonetheless, thanks to Lemma 2, we can still leverage the pruning rule in many cases.

LEMMA *2:* Consider two graphs g and g', where g is a direct supergraph of g' such that $E(g) = E(g') \cup \{e\}$ and $e \not\sqsubseteq g'$. The inequality holds: $P_j(g) \leq P_j(g') \cdot P_j(e)$.

To apply the pruning rule, we define an indicator variable

$$I = \begin{cases} 1, & e \not\sqsubseteq g'; \\ 0, & otherwise. \end{cases}$$

Specifically, when e is distinct from all edges in g', $I = 1$ advises that we can leverage the tighter bound in Lemma 2; otherwise, the Apriori-based bound is employed.

PROPOSITION *3 (Upper bounds):* $esup(g)$ is bounded by

$$\mathcal{U}^x(g) = \sum_{j=1}^{x} P_j(g) + \sum_{j=x+1}^{M_s} P_j(g')(P_j(e))^I, \qquad (3)$$

where g is a direct supergraph of g', and e is a distinct edge.

By "distinct", we mean that there does not exist another edge $e' \in E_{g'}$ with identical labels to e. In this way, we can

derive a series of upper bounds $\mathcal{U}^x(g)$ for subgraph g, $x \in [1, M_s]$, such that $esup(g) = \mathcal{U}^{M_s}(g) \leq \cdots \leq \mathcal{U}^x(g) \leq \cdots \leq \mathcal{U}^1(g)$. Hence, there are many choices of x to apply the pruning rule. While any of them could be used, the pruning power varies due to different tightness of the bounds.

One may notice that this upper bound requires $P_j(g')$ and $P_j(e)$, the true values of which are both unknown. Fortunately, the observation probabilities of them are available. Specifically, the former is computed previously, and the latter can be pre-computed and stored globally. Then, we may take the observation probability plus the absolute error for $P_j(g')$ and $P_j(e)$, i.e., $\bar{P}_j(g') = \hat{P}_j(g') + \varepsilon', \bar{P}_j(e) = \hat{P}_j(e) + \varepsilon'$. It is worth noting that if the error ε' is large due to not large enough sample size, both $\bar{P}_j(g')$ and $\bar{P}_j(e)$ can be large, limiting the pruning power. Therefore, we need to guarantee the accuracy of $\hat{P}_j(g')$ and $\hat{P}_j(e)$ to assure the effectiveness.

2) Incorporating Upper Bound Pruning: Depending on when to execute pruning, we present an evaluation procedure, namely *eager evaluation*, as shown in Algorithm 4. It is "eager" in the sense that it tries to filter our infrequent subgraphs as early as possible. To incorporate the technique, we replace Line 6 of Algorithm 3 with "**switch** PruneEval($\mathcal{P}, k$) **do**".

Algorithm 4: PruneEval($\mathcal{P}, x$)

Input : $\mathcal{P}$ is an array; x is an integer.
Output: Case 1, 2 or 3.
1 $\mathcal{U} \leftarrow 0, I \leftarrow 0, \varepsilon' \leftarrow \frac{\varepsilon\sigma}{2x}$;
2 **if** $e \not\sqsubseteq g'$ **then** $I \leftarrow 1$;
3 **for** $j = 1$ to x **do** $\mathcal{U} \leftarrow \mathcal{U} + \mathcal{P}[j] + \varepsilon'$;
4 **for** $j = x + 1$ to M_s **do**
5 $\quad \mathcal{U} \leftarrow \mathcal{U} + \min\{\bar{P}_j(g') \cdot (\bar{P}_j(e))^I, \mathcal{P}[j] + \varepsilon'\}$;
6 **if** $\mathcal{U} < \sigma$ **then return** Case 1;
7 **if** x equals to M_s **then return** Case 2;
8 **return** Case 3;

Indeed, Algorithm 4 may sample for the maximum times, in order to keep the absolute error no larger than $\varepsilon' = \frac{\varepsilon\sigma}{2M_s}$, which applies to all frequent subgraphs. In other terms, it turns out that (1) if g is frequent, it works identically to Algorithm 1 incorporating the checkpoint mechanism; however, (2) if it is not frequent, the sample size can be largely reduced.

It is worth noting that the series of upper bounds "happen" to provide a good choice of the checkpoints. More precisely, we can ensure $\sum_{i=j}^{x} P_j(g)$ of $U^x(g)$ to be within $\varepsilon\sigma$ when employing the upper bound by Equation (3). Then, the sample size for each upper bound is naturally the checkpoints, i.e., $\frac{1^2 \ln(2/\delta)}{\varepsilon^2\sigma^2}, \frac{2^2 \ln(2/\delta)}{\varepsilon^2\sigma^2}, \ldots, \frac{{M_s}^2 \ln(2/\delta)}{\varepsilon^2\sigma^2}$, respectively.

VI. Experiments

A. Experiment Setup

All the algorithms evaluated in the experiments were implemented by C++ with STL support. In particular, the algorithm framework proposed in Section IV was implemented first, namely the baseline, and on top of it incorporated the optimization techniques. We chose Amazon EC2 as the experiment platform, where `c3.2xlarge` instance was selected (28 compute units, 8 virtual CPU and 15GB memory).

Experiments were conducted on real-life graphs - reference network of articles and collaboration network of professionals:

- **CIT**(`linqs.cs.umd.edu/projects//projects/lbc/`): CIT is a typical citation network of published articles based on CiteSeer. Each vertex has a single label representing an area in computer science (e.g. DM & DB), and there are totally six distinct labels. Each edge comes with a score (from 0 to 100) of *dissimilarity* between the corresponding pair of publications, and a smaller score denotes a stronger similarity. We modeled the dissimilarity as the edge probability and normalized the scores into (0,1],
- **COL**(`www.informatik.uni-trier.de/~ley/db/`): COL is a subset of a professional social network based on DBLP, which was provided by authors of [21]. The vertex label indicates the major direction that a person works on, and the edge probabilities express the strength of collaboration between two authors. Edge probabilities were derived from an exponential CDF of mean μ to the number of collaborations; if two authors collaborated t times, the corresponding probability is $1 - e^{t/\mu}$, where $\mu = 5$ following the convention [12, 21].

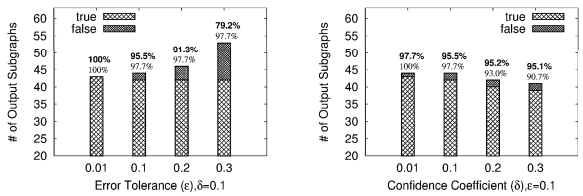
We summarize the statistics of the datasets in Table I. For comparison, the two uncertain graphs are of different structural characteristics. Density of graph is measured by $|E|/|V|$. Thus, CIT has the density of about 1.43, and for COL it is roughly 3.50. While COL is larger and denser, the number of distinct vertex labels of COL is greater. Additionally, the average existence probability on edges are 0.12 and 0.23, respectively.

TABLE I. Dataset Statistics

Dataset	$\|V\|$	$\|E\|$	$\|l_V\|$	min/avg/max $P(e)$
CIT	3,312	4,732	6	0.01/0.12/1
COL	100,000	349,684	130	0.13/0.23/0.99

B. Evaluating Approximation Quality

In this set of experiments, we show the proposed approximation algorithm framework, namely Baseline, can effectively discover desired answers, and evaluate its approximation quality. Recall that the exact algorithm has to enumerate all possible implied graphs. Since it is infeasible to exactly find all the true frequent patterns on even slightly large graphs, we had to conduct the experiments on a small portion of CIT with 30 vertices, which were extracted from the original graph while keeping the density roughly the same. Subgraphs discovered under $\varepsilon = 0.01$ and $\delta = 0.01$ were regarded as the true frequent subgraphs, with support threshold σ set to 2. Note that under this parameter setting, Baseline also becomes quite slow, which requires a large number of sample graphs.



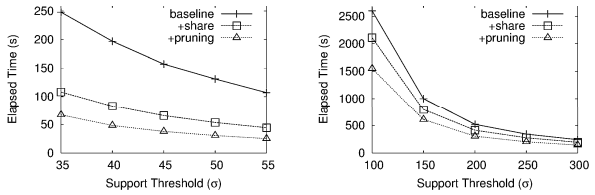
(a) Part. CIT, # of Output Subgraphs (b) Part. CIT, # of Output Subgraphs

Fig. 5. Results - Approximation Quality

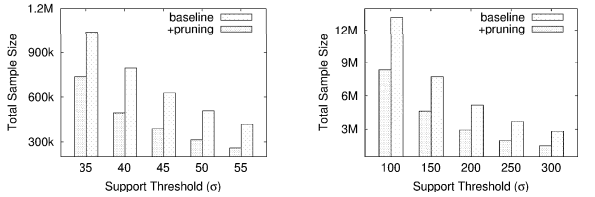
We examine approximation quality with respect to parameters ε and δ, which is measured by *precision* and *recall*; precision is the percentage of true frequent subgraphs in the output, and recall is the percentage of output subgraphs in the true frequent subgraphs. The comparison results on CIT and COL are presented in Figures 5(a) and 5(b), respectively.

In Figure 5(a), we varied ε from 0.01 to 0.3 and $\delta = 0.1$. The percentages above the each bar indicate the precision (in bold) and the recall rates. It reveals that the precision of Baseline decreases while the recall remains stable with the growth of ε. This is because (1) when ε becomes larger, more false frequent subgraphs are returned, reducing the precision; and (2) when δ is fixed, the probability of a frequent subgraph being returned is fixed, and thus, the number of true frequent subgraphs output does not fluctuate significantly.

Figure 5(b) shows the results along with the changes of confidence coefficient δ from 0.01 to 0.3, and $\varepsilon = 0.1$. We observe that while the precision levels off at about 95%, the recall decreases with the growth of δ. The reason boils down to that (1) fixed ε determines the expected number of false frequent subgraphs to be returned, and thus, the precision remains stable; and (2) while δ increases, the probability of a frequent subgraph to be output decreases, and hence, returned true frequent subgraphs reduce, bringing down the recall.



(a) CIT, Elapsed Time (b) COL, Elapsed Time



(c) CIT, Total Sample Size (d) COL, Total Sample Size

Fig. 6. Results - Proposed Techniques

C. Evaluating Proposed Techniques

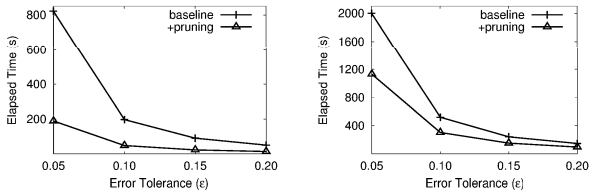
This set of experiments evaluate the effect of the optimization techniques under the framework, involving:

- **Baseline**: Baseline is the basic approximation algorithm for mining single uncertain graphs via the Monte-Carlo simulation, i.e., Algorithm 1 proposed in Section IV;
- **+Share**: Applying the computation sharing technique on top of Baseline results in +Share, as in Section V-A, which shares three types of costs among sample graphs; and
- **+Pruning**: Further incorporating the pruning at checkpoint mechanism, introduced in Section V-B, with +Share, we have +Pruning that integrates all proposed techniques.

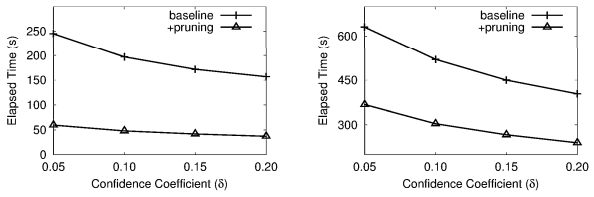
Figure 6(a) plots the elapsed time on CIT. We fixed ε and δ both at 0.1, and varied σ from 35 to 55. The figure reads that the execution time drops gradually. Moreover, +Share improves the efficiency of Baseline substantially, and +Pruning further improves the speed, though the benefit over +Share is relatively moderate. In particular, when $\sigma = 35$, time-saving effect is the most evident, reducing 73.1% of the elapsed time.

We then carried out the experiment on COL, and put the results in Figure 6(b), where similar trend was reflected. Besides, it is worth noting that when $\sigma = 100$, the proposed techniques save the most time by approximately 1,000s.

To further appreciate the pruning technique, we look into the total number of samples. Figures 6(c) and 6(d) depict the value on CIT and COL, respectively. Since +Share does not involve pruning, which requires the same number of samples as Baseline, we omit it here. The key observation is that pruning makes it possible to terminate earlier with less samples. Both figures unveil the effectiveness of the pruning rules, which reduce samples by up to $\frac{1}{3}$ as compared to Baseline.



(a) CIT, Elapsed Time (b) COL, Elapsed Time



(c) CIT, Elapsed Time (d) COL, Elapsed Time

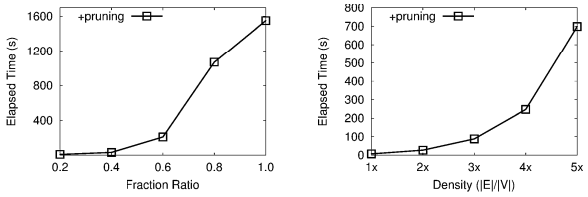
Fig. 7. Results - Impact of Parameters

D. Evaluating Impact of Parameters

Apart from σ, user-defined parameters ε and δ also influence the algorithms. This set of experiments demonstrate the influence by varying ε and δ, respectively.

We fixed $\tau = 40$ and $\delta = 0.1$ on dataset CIT, and increased ε by 0.05 each time. The results are illustrated in Figure 7(a). The general trend is that execution time reduces with the growth of ε, and reduction is less remarkable as ε gets larger. It is noticed that +Pruning is less sensitive to the change of ε, and displays better performance compared with Baseline, which demonstrates the priority and robustness of +Pruning. The same experiment was conducted on COL, with $\sigma = 200$ and $\delta = 0.1$. Similar trend is observed in Figure 7(b); nonetheless, it showcases relatively greater influence of ε.

Then, we evaluate the impact of δ, the value of which was selected from $\{0.05, 0.10, 0.15, 0.20\}$. Figures 7(c) and 7(d) summarize the results on CIT and COL, respectively. Both figures show that the running gradually lowers with δ. In comparison with the impact of ε, the influence of δ on the efficiency is less noteworthy. In summary, +Pruning is less sensitive to the change of ε and δ. Besides, both algorithms



(a) Syn. COL, Elapsed Time (b) Syn. COL, Elapsed Time

Fig. 8. Results - Dataset Scalability

are influenced more by ε than δ. This is justifiable, as the time complexity of the algorithms is proportional to $\frac{1}{\varepsilon^2}$ and $\ln \frac{1}{\delta}$.

E. Evaluating Dataset Scalability

The scalability of +Pruning is evaluated in this set of experiments in two aspects - dataset size and density. The graphs were sampled from COL, and density is measured by $\frac{|E|}{|V|}$. Specifically, dataset size is reflected by fraction of vertices with respect to the whole COL; we randomly sampled vertices, increasing the fraction ratio from 0.2 to 1.0 while keeping the density rough the same as the original COL. Then, for the second experiment, we varied density and fixed $|V|$.

First, +Pruning is evaluated on 0.2,0.4,0.6,0.8 fractions of COL, $\sigma = 100$. Figure 8(a) shows rapid growth especially when fraction ratio rises from 0.6 to 0.8, though it lowers down at 0.8. Note that, with the increase of data size, the number of frequent subgraphs does not grow linearly but mounts in an exponential rate. Actually, we conducted another experiment (omitted due to space constraint) where support threshold rose with fraction ratio, which saw a linear growing trend instead.

The experiment against graph density was run on a sample of COL with 20,000 vertices and various numbers of edges. The x-axis in Figure 8(b) expresses density, i.e., $|E| = |V|$,$2|V|$,$3|V|$,$4|V|$,$5|V|$, respectively, and σ was fixed at 100. The figure reads an increase in the elapsed time, and the growth rate keeps increasing. We contend that this is due to similar reason for the previous results in Figure 8(a).

VII. Conclusion

In this paper, we have investigated FSM on single uncertain graphs. Based on an enumeration-evaluation framework, we propose an effective algorithm fanta to achieve elegant mining performance. The #P-hard support computation problem is resolved by approximating the true value into an interval with accuracy guarantee. Optimization techniques are developed to improve runtime performance. Extensive experiments indicate that the methods yield a promising solution for real-life data.

Acknowledgment

X. Zhao were supported by NSFC 61402494 and 61402498, NSF Hunan 2015JJ4009; X. Lin was supported by NSFC61232006, DP150102728, 140103578 and 120104168.

References

[1] E. Adar and C. Re. Managing uncertainty in social networks. *IEEE Data Eng. Bull.*, 30(2):15–22, 2007.

[2] S. Asthana and et al. Predicting protein complex membership using probabilistic network reliability. *Genome Res.*, 14(6):1170–1175, 2004.

[3] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *KDD*, 2014.

[4] B. Bringmann and S. Nijssen. What is frequent in a single graph? In *PAKDD*, pages 858–863, 2008.

[5] H. Cheng, X. Yan, and J. Han. Mining graph patterns. In *Frequent Pattern Mining*, pages 307–338. 2014.

[6] D. Chu, A. Deshpande, and et al. Approximate data collection in sensor networks using probabilistic models. In *ICDE*, page 48, 2006.

[7] M. Fiedler and C. Borgelt. Support computation for mining frequent subgraphs in a single graph. In *MLG*, 2007.

[8] J. Ghosh, H. Q. Ngo, and S. Yoon. On a routing problem within probabilistic graphs and its application to intermittently connected networks. In *INFOCOM*, pages 1721–1729, 2007.

[9] P. Hintsanen and H. Toivonen. Finding reliable subgraphs from large probabilistic graphs. *DMKD*, 17(1):3–23, 2008.

[10] R. Jiang, Z. Tu, T. Chen, and F. Sun. Network motif identification in stochastic networks. *PNAS*, 103(25):9404–9409, 2006.

[11] R. Jin, L. Liu, and C. C. Aggarwal. Discovering highly reliable subgraphs in uncertain graphs. In *KDD*, pages 992–1000, 2011.

[12] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-constraint reachability computation in uncertain graphs. *PVLDB*, 4(9):551–562, 2011.

[13] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[14] G. Kollios, M. Potamias, and E. Terzi. Clustering large probabilistic graphs. *IEEE TKDE*, 25(2):325–336, 2013.

[15] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *DMKD*, 11(3):243–271, 2005.

[16] J. Li, Z. Zou, and H. Gao. Mining frequent subgraphs over uncertain graph databases under probabilistic semantics. *VLDB J.*, 21(6):753–777, 2012.

[17] L. Liu, R. Jin, C. C. Aggarwal, and Y. Shen. Reliable clustering on uncertain graphs. In *ICDM*, pages 459–468, 2012.

[18] W. E. Moustafa, A. Kimmig, A. Deshpande, and L. Getoor. Subgraph pattern matching over uncertain graphs with identity linkage uncertainty. In *ICDE*, pages 904–915, 2014.

[19] A. P. Mukherjee, P. Xu, and S. Tirthapura. Mining maximal cliques from an uncertain graph. In *ICDE*, 2015.

[20] O. Papapetrou, E. Ioannou, and et al. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *EDBT*, 2011.

[21] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. k-nearest neighbors in uncertain graphs. *PVLDB*, 3(1):997–1008, 2010.

[22] D. Rhodes, S. Tomlins, and et al. Probabilistic model of the human protein-protein interaction network. *Nat Biotechnol.*, 23(8):1–9, 2005.

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

[24] Y. Yuan, G. Wang, and L. Chen. Pattern match query in a large uncertain graph. In *CIKM*, pages 519–528, 2014.

[25] Y. Yuan, G. Wang, H. Wang, and L. Chen. Efficient subgraph search over large uncertain graphs. *PVLDB*, 4(11):876–886, 2011.

[26] Z. Zou, H. Gao, and J. Li. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *KDD*, 2010.

[27] Z. Zou and J. Li. Structural-context similarities for uncertain graphs. In *ICDM*, pages 1325–1330, 2013.

[28] Z. Zou, J. Li, H. Gao, and S. Zhang. Finding top-k maximal cliques in an uncertain graph. In *ICDE*, pages 649–652, 2010.

[29] Z. Zou, J. Li, H. Gao, and S. Zhang. Mining frequent subgraph patterns from uncertain graph data. *IEEE TKDE*, 22(9):1203–1218, 2010.