Towards Frequent Subgraph Mining on Single Large Uncertain Graphs

Yifan Chen  Xiang Zhao  
National University of Defense Technology, China  
Email: {yfchen, xiangzhao}@nudt.edu.cn

Xuemin Lin  Yang Wang  
The University of New South Wales, Australia  
Email: {lxue, wangy}@cse.unsw.edu.au

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], [14], uncertain graph model has been incorporated in communication [3], [9] and wireless sensor networks [7], protein interaction [2], [22] and regulatory networks in biology [11], etc. Research efforts have been dedicated to a number of interesting problems on uncertain graphs, [4], [12], [17], [19] to name 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 [16] 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 [6].

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 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.

In this paper, we investigate FSM on single uncertain graphs, which finds applications in bio-medicine, social behavior analytics, etc. For example, 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 for observed proteins or poorly characterized genomes, which comply with discovered association patterns [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 in STRING database 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 of such interaction found in fission yeast on the right of Figure 1 for better appreciation.

While the significance of 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. 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 once. Extending the notion, we define expected support

\[ \text{expected support} = \max \left( \sum p_{(u,v)} \right) \]

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

Fig. 1. Protein Interactions of Fission Yeast

\[
\begin{array}{c|c|c|c}
\text{COG0513 - COG1643} & \text{SPCC285.03} & 0.778 & \text{“SPBC16H5.10c”} \\
\text{COG2319 - COG0724} & \text{SPAC140.02} & 0.834 & \text{“SPCC1672.07”} \\
\end{array}
\]
on single uncertain graphs as the aggregated support weighted to its existence probability over all possible graphs, which is a probability distribution over the support in all implied graphs of the uncertain graph. Subgraphs surpassing a given threshold are considered frequent. Due to the shift of definition, existing algorithm on a collection of uncertain graphs 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, to exist in reality, 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. To appreciate the hardness of the problem, we show that computing expected support over an uncertain graph is #P-hard, by reducing from the DNF counting problem. Thus, we 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 \( \sigma \) into three cases, we guarantee that, with probability at least \( 1 - \delta \), any subgraph with expected support no less than \( \sigma \) will be output, but any subgraph with expected support less than \( (1 - \varepsilon)\sigma \) will not be output, where \( \varepsilon \) is an error tolerance, and \( 1 - \delta \) is a degree of confidence, \( \varepsilon, \delta \in [0, 1] \). This shapes our enumeration-evaluation framework, where we judiciously enumerate candidate subgraphs and evaluate them one by one.

We propose to evaluate support via 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. In particular, we first seek to share computation among samples via an online-built binary sharing tree; moreover, under a checkpoint mechanism, delicate structure-based pruning rules are devised to filter out non-promising candidates at checkpoints.

3. Using real-life data, we conduct an extensive experimental study of our algorithm. 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 \( \delta \) and \( \varepsilon \). Moreover, the optimization techniques are effective in reducing 71.3% of running time at most, and hence, improve mining performance and user experience.

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 \( \phi \)-frequent probability is greater than the threshold. 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 disconnectedness. 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]. The concept of reliable subgraphs was uniquely established for finding connected subgraphs with high confidence [10], [12]. Other mining tasks over uncertain graphs include clustering [15], [17], core decomposition [4], etc. Querying uncertain graphs also receives much attention, and a natural problem to ask is about reachability [13]. As to complex structures, subgraph search [18], [25], [24] 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 [6] for a recent survey.

II. PRELIMINARY

A. Deterministic Graph

For ease of exposition, we assume graph is undirected 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 \( l_G : V_G \cup E_G \rightarrow \Sigma_G \) is a labeling function that assigns labels to vertices and edges. \(|V_G|\) and \(|E_G|\) are the numbers of vertices and edges, respectively.

A graph \( g \) is subgraph isomorphic to another graph \( G \), denoted by \( g \subseteq G \), if there exists an injection \( f : V_g \rightarrow V_G \) such that (1) \( \forall v \in V_g, f(v) \in V_G \land l_G(v) = l_G(f(v)) \); and (2) \( \forall (u, v) \in E_g, (f(u), f(v)) \in E_G \land l_G(u, v) = l_G(f(u), f(v)) \). 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 \( g' \subseteq g \) and \( |E_g| = |E_{g'}| + 1 \).

Consider two graphs \( g \subseteq G \), and a support threshold \( \tau \), 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 [16].

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) [5], (2) harmful overlap (HO) [8], and (3) maximum independent sets (MIS) [16]. 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 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 $\text{sup}(g, G) = \min_{v \in V_g} |F(v)|$.

**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 a biologist; edge labels are omitted for clarity. There are $3$ subgraph isomorphisms between $g$ and $G$, i.e., $(u_1, v_2)$ to $(v_1, v_2)$, $(v_3, v_2)$ and $(v_3, v_4)$. Thus, $\text{sup}(g, G) = \min\{2, 2\} = 2$.

**B. Uncertain Graph**

An uncertain graph is a tuple $G^u = (G, P)$, where $G$ is a deterministic graph, and $P : E_G \rightarrow (0, 1]$ 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 $|E_G|$ 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 in [2], [9], [11]. 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:

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

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

Given an expected support threshold $\sigma$, 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., $\text{esup}(g, G^u) \geq \sigma$.

**Proposition 1** (Anti-monotonicity): Consider $2$ graphs $g' \subseteq g$ and uncertain graph $G^u$. $\text{esup}(g, G^u) \leq \text{esup}(g', G^u)$.

**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 $\text{sup}(g, G) = 2$. Thus, $P(G^u \Rightarrow G) \cdot \text{sup}(g, G) = 0.24$. Accumulating the values from all $8$ implied graphs, we have $\text{esup}(g, G^u) = 1.12 \geq \sigma$, and thus, $g$ is a frequent subgraph.

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

We annotate the semantics of Definition 1. Suppose $\text{esup}(g, G^u) = 10$, 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., “$\text{esup}(g)$”.

**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 uncertain 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 naive 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.

IV. Support Evaluation

Computing expected support directly by Equation (1) is excessively complex, as there exist $2^{|E_G|}$ implied graphs of $G^n$. That is, for each candidate subgraph, we need to calculate its support in an exponential number of exact 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., $P(sup(g) = j) = \sum_{G' \in \Delta_j(g)} P(G^n \Rightarrow G')$, where $\Delta_j(g) = \{G' \mid sup(g, G') = j\}$. It is then not difficult to rewrite Equation (1) as

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

where $M_s = sup(g, G)$ is the maximum support of $g$ among all implied graphs of $G^n$. 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., $P_j(g) = \sum_{G' \in \Delta_j(g)} P(G^n \Rightarrow G')$, where $\Delta_j(g) = \{G' \mid sup(g, G') \geq j\}$.

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

Proof: By Equation (2), we have

$$esup(g) = \sum_{j=1}^{M_s-1} (P_j(g) - P_{j+1}(g)) \cdot j + P_{M_s}(g) \cdot M_s$$

$$= \sum_{j=1}^{M_s-1} P_j(g) \cdot j - \sum_{j=2}^{M_s} P_j(g) \cdot (j-1) + P_{M_s}(g) \cdot M_s$$

$$= P_1(g) + \sum_{j=2}^{M_s-1} P_j(g) + P_{M_s}(g).$$

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

B. Computational Complexity

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

THEOREM 1: It is #P-hard to compute $P_\zeta(g)$.

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) $D = (x_1 \land x_2 \land x_3) \lor (x_2 \land x_3 \land x_4)$ 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 $\alpha$), $\{u'_1, u'_2, u'_3\}$ and $\{v'_1, v'_2, v'_3\}$ (labeled $\beta$), showing in the rightmost of Figure 3. As to uncertain graph $G^n$, $\zeta - 1$ (disconnected) isomorphisms are first established, and the edge probabilities are all equal to 1, to guarantee $P(sup(g, G^n) \geq \zeta - 1) = 1$. An additional part is built for $G^n$, involving $\{c_1, c_2\}$ (labeled $\alpha$), $\{u_1, u_2, u_3, u_4\}$ and $\{v_1, v_2, v_3, v_4\}$ (labeled $\beta$). $\{c_1, c_2\}$ 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 $\varepsilon$. 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 $[esup, esup]$, that approximately contains the true value of $esup$, and carefully handle the following cases when evaluating subgraph $g$ against support threshold $\sigma$:

- Case 1: If $esup(g) < \sigma$, do not output $g$, since it is certain that $esup(g) < \sigma$.
- Case 2: If $esup(g) \geq (1 - \varepsilon)\sigma$ and $esup(g) \geq \sigma$, output $g$, as it is certain that $esup(g) \geq (1 - \varepsilon)\sigma$, and it is probable that $esup(g) \geq \sigma$.
- Case 3: If $esup(g) \geq \sigma$ and $esup(g) < (1 - \varepsilon)\sigma$, 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 $[esup, esup]$ 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 $[esup(g), esup(g)]$, 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 $[esup(g), esup(g)]$. To ensure the interval 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 $\varepsilon'$, 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 $\delta$ and $\varepsilon'$, 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:

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

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

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 $\varepsilon$ and a real number $\delta$; and it outputs a boolean that indicates whether $g$ is frequent. In particular, Line 1 allocates an empty array for storing the embeddings of $G$. As we memorize the embeddings of $G$, 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 $G$. 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., $E_m = \{e_i \mid e_i \in E(g)\}$, where $E$ is a set of isomorphisms from $g$ to $G$, $i \in [1, |E_m|]$. 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 $G$, 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 $P(G \Rightarrow G_i)$, if $j$ is no larger than $sup(g, G_i)$. After screening all sample graphs, Line 9 evaluates the approximate support via EvaluateSup.

In general, function EvaluateSup works as follows. It takes as input the observation probabilities $P_j[j]$ 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) $P(g) = \sum_{j \in [1, M_s]} P_j[j]$, and the output is bounded by $\varepsilon sup = P(g) - \frac{M_s \varepsilon}{2x}$, and $\tilde{\varepsilon sup} = P(g) + \frac{M_s \varepsilon}{2x}$. Upon the interval, we make decision against the threshold $\sigma$.

**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 $O(\sup(g)||V_g||)$, where $F(g)$ is the set of embeddings of $g$. Then, we perform Monte-Carlo simulation. The complexity of support and probability computation is $O(\sup(g)||V_g||)$ 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 $O((\frac{M_s}{\varepsilon\sigma})^2 \ln \frac{1}{\varepsilon} G || V_g||)$. 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 $\delta$ and $\varepsilon$, 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.
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 subtracting the sets of embeddings containing any collected edges are the ones 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 $e_2$, 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 $\text{sup}(g, G')$. 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 $\text{sup}(g, G')$ 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 covered previously by $G'$, incrementing the corresponding values. Afterwards, we can derive $\text{sup}(g, G'')$.

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 $m_i^2 \triangleq |(F_i \cup F_j) \setminus (F_i \cap F_j)|$. The cost of support computation from $G_i$ (resp. $G_j$) to $G_j$ (resp. $G_i$) is $m_i^2 \cdot c + c'$ (resp. $m_j^2 \cdot c + c'$), and $m_i^2 = m_j^2$. If $|F_i| < m_i^2$, it is less costly to compute for $G_i$ 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 $\min(m_i^2, |F_j|) \cdot c + c'$, 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 “$P \leftarrow \text{ShareCompTree}(g, \Omega)$”.

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

**Input**: $g$ is a subgraph; $\Omega$ is a set of sample graphs.  
**Output**: $P$ 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 $X_j \leftarrow X_j + p_i$;  
10. $Y \leftarrow Y + p_i$;  
11. foreach sample graph $G_i (i \in [2,|\Omega|])$ do  
12. $d \leftarrow \text{HammingDistance}(B_{i-1}, B_i)$;  
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 $P \leftarrow P \cup \{X_j^{\Omega}\}$;  
18. return $P$

**B. Pruning at Evaluation Checkpoints**

While $N = \frac{2M^2 \ln(\frac{1}{\beta})}{\varepsilon^2 \sigma^2}$ 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 $\Omega$. Then, Line 5 computes with $\Omega'$ 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; $\Omega$ 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 $\Omega$;  
   5. $P \leftarrow \text{ShareCompTree}(g, \Omega')$;  
   6. switch EvaluateSup($P, k$) 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 $U(g)$ of $esup(g)$ is used to prune a subgraph; if $U(g) < \sigma$, $g$ 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 $g' \subset g$ 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 $s$ and $s'$, where $g$ is a direct supergraph of $g'$ such that $E(g) = E(g') \cup \{e\}$ and $e \not\subset g'$. The inequality holds: $P_j(g) \leq P_j(g') \cdot P_j(e)$.

To apply the pruning rule, we define an indicator variable

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

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

$$U^\varepsilon(g) = \sum_{j=1}^{x} P_j(g) + \sum_{j=x+1}^{M} P_j(g')(P_j(e))^I,$$  
(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 \( U^r(g) \) for subgraph \( g, x \in [1, M_x] \), such that \( e_{\sup}(g) = U^{M_x}(g) \leq \cdots \leq U^r(g) \leq \cdots \leq U^1(g) \). 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., \( P_j(g') = P_j(g') + \varepsilon', P_j(e) = P_j(e) + \varepsilon' \). It is worth noting that if the error \( \varepsilon' \) is large to not large enough sample size, both \( P_j(g') \) and \( P_j(e) \) can be large, limiting the pruning power. Therefore, we need to guarantee the accuracy of \( P_j(g') \) and \( P_j(e) \) 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 “\textbf{switch} PruneEval(\mathcal{P}, k) \textbf{do}”.

<table>
<thead>
<tr>
<th>Algorithm 4: PruneEval(\mathcal{P}, x)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: ( \mathcal{P} ) is an array; ( x ) is an integer.</td>
</tr>
<tr>
<td><strong>Output</strong>: Case 1, 2 or 3.</td>
</tr>
<tr>
<td>1 ( U \leftarrow 0, I \leftarrow 0, \varepsilon' \leftarrow \frac{\varepsilon g}{2} ).</td>
</tr>
<tr>
<td>2 if ( e \not\in g' ) then ( I \leftarrow 1 );</td>
</tr>
<tr>
<td>3 for ( j = 1 ) to ( x ) do ( U \leftarrow U + \mathcal{P}[j] + \varepsilon' );</td>
</tr>
<tr>
<td>4 for ( j = x + 1 ) to ( M_x ) do</td>
</tr>
<tr>
<td>5 ( U \leftarrow U + \min{P_j(g'), P_j(e)}, \mathcal{P}[j] + \varepsilon' };</td>
</tr>
<tr>
<td>6 if ( U &lt; \sigma ) then return Case 1;</td>
</tr>
<tr>
<td>7 if ( x ) equals to ( M_x ) then return Case 2;</td>
</tr>
<tr>
<td>8 return Case 3;</td>
</tr>
</tbody>
</table>

Indeed, Algorithm 4 may sample for the maximum times, in order to keep the absolute error no larger than \( \varepsilon' = \frac{\varepsilon g}{2} \), 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 \( \sum_{i=1}^{M_x} P_j(g) \) of \( U^x(g) \) to be within \( \varepsilon \delta \) when employing the upper bound by Equation (3). Then, the sample size for each upper bound is naturally the checkpoints, i.e., \( \frac{1^x \ln(2/\delta)}{\varepsilon \delta}, \frac{2^x \ln(2/\delta)}{\varepsilon \delta}, \ldots, \frac{M_x \ln(2/\delta)}{\varepsilon \delta} \), respectively.

VI. EXPERIMENTS
A. Experiment Setup

All the algorithms involved and evaluated in the experiments were implemented using C++ with STL support. In particular, the algorithm framework proposed in Section IV was implemented first, namely the baseline, and then optimization techniques were added on top of it. We chose Amazon EC2 as the standard experiment platform, where c3.2xlarge instance was selected. Each instance has 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** (links.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 and DB), and there are six distinct labels in total. 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 a kind of confidence in asserting two publications are identical, and normalized the scores into \( (0, 1) \), captured by edge probability in an uncertain graph.

- **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 \( \mu \) 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 [13], [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>
<thead>
<tr>
<th>Table I. Dataset Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
</tr>
<tr>
<td>CIT</td>
</tr>
<tr>
<td>COL</td>
</tr>
</tbody>
</table>

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 \( \sigma \) set to 2. Note that under this parameter setting, Baseline also becomes quite slow, which requires a large number of sample graphs.

We examine approximation quality with respect to parameters \( \varepsilon \) and \( \delta \), 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 $\varepsilon$ 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 $\varepsilon$. This is because (1) when $\varepsilon$ becomes larger, more false frequent subgraphs are returned, reducing the precision; and (2) when $\delta$ 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 $\delta$ 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 $\delta$. The reason boils down to that (1) fixed $\varepsilon$ determines the expected number of false frequent subgraphs to be returned, and thus, the precision remains stable; and (2) when $\delta$ increases, the probability of a frequent subgraph to be output decreases, and hence, returned true frequent subgraphs reduce, bringing down the recall.

Figure 6(a) plots the elapsed time on CIT. We fixed $\varepsilon$ and $\delta$ both at 0.1, and varied $\sigma$ 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). Similar trend is reflected in the figure while the difference lies from two aspects. First, the decrease of execution time is more evident on COL than that on CIT, especially when support threshold is small. Second, the advancement of +Pruning over +Share is more apparent, demonstrating that pruning is more effective on COL. Additionally, 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 total number of samples required on CIT and COL, respectively. Since +Share does not involve pruning, which requires the same number of samples as Baseline, we omit it in this comparison. 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 at most $\frac{1}{3}$ of the samples required by Baseline.

Seeing the progressive improvement, the complete version of algorithm fanta, i.e., +Pruning, will be used hereafter.

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 $\varepsilon$ and $\delta$ both at 0.1, and varied $\sigma$ 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). Similar trend is reflected in the figure while the difference lies from two aspects. First, the decrease of execution time is more evident on COL than that on CIT, especially when support threshold is small. Second, the advancement of +Pruning over +Share is more apparent, demonstrating that pruning is more effective on COL. Additionally, 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 total number of samples required on CIT and COL, respectively. Since +Share does not involve pruning, which requires the same number of samples as Baseline, we omit it in this comparison. 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 at most $\frac{1}{3}$ of the samples required by Baseline.

Seeing the progressive improvement, the complete version of algorithm fanta, i.e., +Pruning, will be used hereafter.

D. Evaluating Impact of Parameters

Apart from $\sigma$, user-defined parameters $\varepsilon$ and $\delta$ also influence the algorithms. This set of experiments demonstrate the influence by varying $\varepsilon$ and $\delta$, respectively.
We fixed $\tau = 40$ and $\delta = 0.1$ on dataset CIT, and increased $\varepsilon$ 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 $\varepsilon$, and reduction is less remarkable as $\varepsilon$ gets larger. It is noticed that fanta is less sensitive to the change of $\varepsilon$, and displays better performance compared with Baseline, which demonstrates the priority and robustness of fanta. 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 $\varepsilon$.

Then, we evaluate the impact of $\delta$, 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 execution time gradually lowers with the growth of $\delta$. In comparison with the impact of $\varepsilon$, the influence of $\delta$ on the efficiency is less noteworthy. In summary, fanta is less sensitive to the change of $\varepsilon$ and $\delta$. Besides, both algorithms are influenced more by $\varepsilon$ than $\delta$. This is justifiable, as the time complexity of the algorithms is proportional to $\frac{1}{\varepsilon}$ and $\ln \frac{1}{\delta}$.

![Figure 8](image_url)

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

**E. Evaluating Dataset Scalability**

The scalability of fanta 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, fanta is evaluated on 0.2,0.4,0.6 and 0.8 fractions of COL, and $\sigma = 100$. 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., $|E| = \sum_{i}^{V} \sum_{j}^{V} E_{ij}$, respectively, and $\sigma$ 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).

We also evaluated the impact of edge probabilities, and found that higher probabilities and larger distribution variances can bring about more frequent subgraphs, and thus, increase execution time. We omit the results in the interest of space.

**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.

**REFERENCES**

