Efficient Probabilistic Supergraph Search over Uncertain Graphs

Ke Zhu[1] Gaoping Zhu[1]
Xuemin Lin[1]

[1] University of New South Wales, Australia
{kez,gzhu, lxue}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-201308
February 2013

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Given a query graph q, retrieving the data graphs g from a set D of data graphs such that q contains g, namely supergraph containment search, is fundamental in graph data analysis with a wide range of real applications. It is very challenging due to the NP-Completeness of subgraph isomorphism testing. Consider that in many applications, graph data are often uncertain due to various reasons. In this paper, we study the problem of probabilistic supergraph search; that is, given a set D of uncertain data graphs, a certain query graph q and a probability threshold θ, we retrieve the data graphs g^u from D such that the probability of q containing g^u is not smaller than θ. We show that besides the NP-Completeness of subgraph isomorphism testing, the problem of calculating probabilities is #P-Complete; thus it is even more challenging than the supergraph containment search.

To tackle the computational hardness, we first propose two novel and effective pruning rules to efficiently prune non-promising data graphs. Then, efficient verification algorithms are developed with the aim of sharing computation and terminating non-promising computation as early as possible. Extensive performance studies on both real and synthetic data demonstrate the efficiency and effectiveness of our techniques in practice.

1 Introduction

A wide spectrum of recent applications reveal the popularity of graphs in modeling complex data. These include chem-informatics, bio-informatics, road networks, social networks, WWW, etc. The increasing needs to efficiently manage and analyze graph data lead to an extensive literature on various fundamental graph structure search problems such as *graph containment search* [20, 24, 26, 31, 34], *graph similarity search* [19, 23], *graph all-matching* [29, 30, 33] and *frequent subgraph mining* [12, 16, 22], etc.

Supergraph Containment Search. Given a set D of data graphs and a query graph q, a *supergraph containment search* is to retrieve the data graphs g from D such that q is a *supergraph* of g (i.e., q contains g). This problem is fundamental and driven by many real applications [6, 27], including bio-informatics and chem-informatics. For example, in bio-informatics a species is usually described by a protein-protein interaction (PPI) network [32]. A PPI network may be represented by a labeled graph such that each vertex corresponds to a protein, its label represents a particular type of protein, and the edge between two proteins (vertices) represents their mutual interaction. It is observed [4] that the proteins of a PPI network may contain various key structural complexes (e.g. pathway) - connected subgraphs of the PPI network, such that the proteins in each structural complex function and evolve in a corrected way. Consequently, it is important and fundamental to identify those well known structural complexes (graphs) contained by a newly discovered or evolved species (PPI network) in investigating the evolutionary history of discovered species; that is, supergraph containment search.

In chem-informatics, it is popular to model molecular descriptors as graphs. Each descriptor is a chemical substructure indicating specific chemical functionality. Given a database of descriptors (graphs) and a molecule (graph), conducting molecular analysis demands efficient supergraph containment search techniques for functionality prediction since it needs to identify all descriptors contained by the given molecule. More applications of a supergraph containment search can be found in [6, 27].

Uncertainty in Graph Data. The conventional supergraph containment search assumes that both query and data graphs are exact. In fact, graph data in many applications may be uncertain: (1) In bio-informatics, the pairwise interactions in a PPI network are only suggested based on the experimental statistics, they can be mistakenly/accidently observed or missed during experiments [1, 2]. For instance, the STRING [1] dataset and BioGRID [2] dataset contain uncertain PPI networks where each interaction (i.e., edge) in the networks is associated with an occurrence probability provided by statistical predications. (2) In chem-informatics, chemical compounds are graphs structured with atoms and chemical bonds as vertices and edges. A chemical bond is caused by the electromagnetic attraction between electrons and nuclei, which depends on many probabilistic electron localization functions [14]. Consequently, chemical bonds are naturally uncertain.

Probabilistic Supergraph Search. Motivated by the above, recently a number of studies towards analyzing uncertain graphs have been conducted (e.g. [13, 17, 18, 25]). Consider that the supergraph containment search is fundamental to many applications and data graphs may often be uncertain in some applications. In this paper, we investigate the problem of supergraph containment search over uncertain data graphs g^u where each edge in a g^u has an occurrence probability, namely *probabilistic supergraph search*. The goal is to identify the total probabilities that all vertices in an

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

[2] http://thebiogrid.org/

uncertain data graph g^u are connected together to form a *connected spanning* subgraph g' of g^u and such g' is contained by a certain query graph q, denoted by $P_c(q \supseteq g^u)$. Specifically, the probabilistic supergraph search problem is to retrieve the data graphs g^u from a given set D of uncertain graphs such that $P_c(q \supseteq g^u) \geq \theta$ where q is a given query graph and θ is a given threshold. While the problem will be formally defined in Section 2, below is an example.

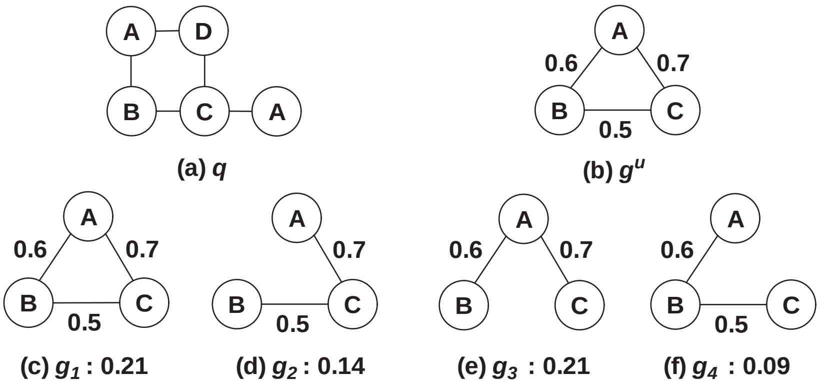


Figure 1.1: Probabilistic Supergraph Search

Example 1 *Figure 1.1(b) depicts an uncertain data graph g^u where the occurrence probability of each edge is also illustrated. Figures 1.1(c)-(f) depict all* 4 connected spanning *subgraphs of g^u. For each spanning subgraph g_i ($1 \leq i \leq 4$), its occurrence probability is recorded. For example, the occurrence probability of g_1 is 0.21 ($= 0.6 \times 0.5 \times 0.7$). For g_2 to occur, the edge (A, B) cannot occur; thus its occurrence probability is 0.14 ($= 0.4 \times 0.5 \times 0.7$). Similarly, the occurrence probabilities of g_3 and g_4 are 0.21 and 0.09, respectively.*

Given a certain query graph q depicted in Figure 1.1(a), $P_c(q \supseteq g^u) = 0.23$ ($= 0.14 + 0.09$) since q contains g_2 and g_4 but does not contain g_1 and g_3. Consequently, regarding a probabilistic supergraph search, g^u is in the result set if $\theta = 0.2$, and is not in the result set if $\theta = 0.3$. □

If we view all vertices of a g^u as a group, then the connectivity enforcement may lead to a *group interaction* - the interaction among all group members (vertices). Such a group interaction is often required as a focus point in the applications of chem-informatics, bio-informatics, etc. A collection of disjoint parts of g^u may hardly provide the information of such a group interaction since their relationships are very diverse: from one hop to many hops. These are the rationales behind the connectivity requirement in a probabilistic supergraph search in this paper.

Challenges & Contributions. To the best of our knowledge, this is the first paper investigating the probabilistic supergraph search problem. The probabilistic supergraph search problem is quite challenging. Firstly, testing subgraph isomorphism is NP-Complete [8]. In this paper, we also show that the calculation of the probabilities $P_c(q \supseteq g^u)$ is #P-Complete even if the subgraph isomorphism testing has been conducted.

In [25], a dual version of probabilistic supergraph search, the problem of *probabilistic subgraph search*, is investigated, where conducting probabilistic subgraph search follows the filtering and verification framework. The filtering techniques in [25] are based on the following observation. Given a query graph q, an uncertain graph g^u, and a subgraph f of g^u, it is shown in [25] that 1) if the probability $P(f \subseteq g^u)$ that f is contained by g^u is smaller than θ ($P(f \subset g^u) < \theta$) and q contains f, then the probability $P(q \subseteq g^u)$ that q is contained by g^u is smaller than θ (i.e. $P(f \subset g^u) < \theta$); and 2) if $P(f \subseteq g^u) \geq \theta$ and f contains q, then $P(f \subseteq g^u) \geq \theta$. In Section 2, we will show that the above conditions in the IF parts of in 1) and 2) do not give any indictions of the probabilities that q contains g^u. Regarding verification, the verification technique in [25] is essentially based on the enumeration of subgraphs of g^u which are *supergraphs* of q. Nevertheless, the nature of our problem probabilistic supergraph search needs to efficiently enumerate the **connected** subgraphs of g^u which are *subgraphs* of q. Thus, the probabilistic subgraph search problem is inherently different than the probabilistic supergraph search problem and the proposed filtering and verification techniques are inherently unapplicable to our problem.

To tame the computational hardness of probabilistic supergraph search, we follow the filtering-verification framework. Particulary, we propose two novel pruning rules to effectively and efficiently filter non-promising data graphs in D. We also develop practically efficient algorithm to conduct verifications. Our contributions are summarized as follows.

- We develop two novel, effective pruning rules. One is based on a probabilistic signature and the other is based on graph structures (features).
- As the pre-computation of probabilities in these two pruning rules involve #P-Completeness, we develop two PTIME based probability upper-bounds, respectively, to speed up the pre-computation while effectively retaining the pruning powers.
- Finally, we develop two novel algorithms to efficiently verify candidates. One efficiently searches connected spanning subgraphs of g^u with the aim of computation sharing, and the other goes one step further to employ a synchronization traversal strategy to terminate a non-promising search as early as possible.

We conduct experiments on both real and synthetic graph data to demonstrate the effectiveness and efficiency of our proposed filtering and verification techniques.

Organization. The rest of paper is organized as follows. Related work is summarized in the remainder of this Section. Section 2 presents the problem definition and the complexity studies. Section 3 gives our filtering techniques. Our verification algorithms are presented in Section 4. The performance evaluation is reported in Section 5. Section 6 concludes the paper.

Related Work. Extensive research has been conducted in recent years in graph structure search against certain graphs. This includes *subgraph containment search* [7, 20, 21, 24, 26, 31] and *supergraph containment search* [6, 27]. The problem of subgraph containment search is to find the graphs from a given set of data graphs which contain a query graph, while its dual version, the problem of supergraph containment search, is described earlier. The problem of finding data graphs from a given set of data graphs which approximately contain a query graph, namely *similarity subgraph search*, has been studied in [19, 23], and the problem of exact or approximate *all-matching* has been studied in [28, 29, 30, 33].

Related work on processing uncertain graphs may be found in [9, 13, 17, 18, 25]. Jin et al. [13] propose to answer reachability queries in an uncertain graph (i.e., to decide if the probability that a vertex v can reach another vertex u within a constrained distance is not smaller than a given probability threshold). Potamias et al. [18] approach the kNN problem over an uncertain graph (i.e., to compute the k closest vertices to a query vertex in an uncertain graph). Both [18] and [13] tackle their problems with the sampling paradigm. Kollios and Potamias [9] investigate the problem of clustering large probabilistic graphs by extending the graph edit distance from certain graphs to uncertain graphs, while mining frequent subgraphs is conducted in [17].

The most related work is conducted by Yuan et al. [25] proposing to retrieve all the uncertain data graphs g^u from a set D of uncertain data graphs such that the probability of g^u containing a certain query graph q is not smaller than a given threshold, namely probabilistic subgraph search. [25] proposes upper and lower bounds of probabilities to prune non-promising uncertain data graphs and validate true answers before the costly probability computation. Due to the inherently different problem natures as discussed in the last Section and detailed in Section 2, the techniques in [25] are not applicable to the probabilistic supergraph search problem.

2 Probabilistic Supergraph Search

In this paper, we focus on *simple* (no self-loops or multiple edges) [10], *connected* [10], *undirected* [10], *vertex-labeled* graphs. Nevertheless, the developed techniques can be immediately extended to directed and/or edge-labeled graphs.

2.1 Problem Definition

To distinguish graphs with uncertainty involved, we refer certain graphs without uncertainty involved as *exact* graphs. An *exact* graph is defined as $g = (V, E, l)$, where V and $E \subseteq V \times V$ are the sets of vertices and undirected edges, respectively. Given a set of labels Σ, the label function $l : V \to \Sigma$ assigns a label $l(v)$ to each vertex $v \in V$. We denote the vertex and edge sets of g by $V(g)$ and $E(g)$.

Definition 1 (Subgraph Isomorphism Mapping)
Given two exact graphs $g_a = (V_a, E_a, l_a)$ and $g_b = (V_b, E_b, l_b)$, a subgraph isomorphism mapping *is an injective function $\mathcal{F} : V_a \to V_b$ such that (1) $\forall v \in V_a$, $\mathcal{F}(v) \in V_b$ and $l_a(v) = l_b(\mathcal{F}(v))$; (2) $\forall(v_1, v_2) \in E_a$, $(\mathcal{F}(v_1), \mathcal{F}(v_2)) \in E_b$.*

Given a subgraph isomorphism mapping $\mathcal{F}$ from g_a to g_b, we say that g_a is a *subgraph* of g_b (or g_a is contained by g_b), denoted by $g_a \subseteq g_b$. We also say that g_b is a *supergraph* of g_a (or g_b contains g_a). $\mathcal{F}(g_a) = \{ (\mathcal{F}(u), \mathcal{F}(v)) | (u, v) \in E(g_a) \}$ denotes the *match* of g_a induced by $\mathcal{F}$, which is the set of edges in g_b mapped from g_a by $\mathcal{F}$.

Following [17], an *uncertain graph* is defined as follows.

Definition 2 (Uncertain Graph) *An* uncertain graph *is defined as $g^u = (V, E, l, P)$ where the corresponding $g^{exc} = (V, E, l)$ is an exact graph defined above, and P is a probability function $P : E \to [0, 1]$ that assigns an* occurrence probability *$P(e)$ to each edge $e \in E$.*

Given an uncertain graph $g^u = (V, E, l, P)$, we refer the corresponding exact graph $g^{exc} = (V, E, l)$ as the *underlying* exact graph of g^u. As discussed in Section 1, in this paper we consider *connected* graphs only. *Thus, the underlying graph g^{exc} of an uncertain graph g^u referred in this paper, thereafter, is always connected if not otherwise specified.*

We use $P(\bar{e})$ to denote the *non-occurrence probability* of an edge $e \in E$; that is, $P(\bar{e}) = 1 - P(e)$.

Definition 3 (Possible World Graph) *A* possible world graph *(PWG)* $g' = (V', E', l')$ *of an uncertain graph* $g^u = (V, E, l, P)$ *is a* spanning *subgraph of the underlying exact graph* g^{exc} *of* g^u*; that is,* $V' = V$*,* $E' \subseteq E$ *and* $l' = l$*.*

Note that the underlying exact graph g^{exc} of g^u is the PWG containing all edges in $E(g^u)$; that is, $V(g^{exc}) = V(g^u)$ and $E(g^{exc}) = E(g^u)$. We use $\mathcal{W}(g^u)$ ($\mathcal{W}_c(g^u)$) to denote the set of all (connected) PWGs of g^u. The occurrence probability of a PWG g', denoted by $P(g')|_{g^u}$, is defined below in (2.1).

$$P(g')|_{g^u} = \prod_{e \in E(g')} P(e) \times \prod_{e \in E(g^u) - E(g')} P(\bar{e}) \tag{2.1}$$

Containment Probabilities. Given an exact graph g and an uncertain graph g^u, the probability that $g \supseteq g^u$ without (with) the connectivity constraint may be expressed as follows, denoted by $P(g \supseteq g^u)$ ($P_c(g \supseteq g^u)$).

$$P(g \supseteq g^u) = \sum_{g' \in \mathcal{W}(g^u) \wedge g \supseteq g'} P(g')|_{g^u} \tag{2.2}$$

$$P_c(g \supseteq g^u) = \sum_{g' \in \mathcal{W}_c(g^u) \wedge g \supseteq g'} P(g')|_{g^u} \tag{2.3}$$

The probability that $g \subseteq g^u$ without (with) connectivity constraint, denoted by $P(g \subseteq g^u)$ ($P_c(g \subseteq g^u)$), is expressed below.

$$P(g \subseteq g^u) = \sum_{g' \in \mathcal{W}(g^u) \wedge g \subseteq g'} P(g')|_{g^u} \tag{2.4}$$

$$P_c(g \subseteq g^u) = \sum_{g' \in \mathcal{W}_c(g^u) \wedge g \subseteq g'} P(g')|_{g^u} \tag{2.5}$$

Problem Statement. As discussed in Section 1, in this paper we investigate *connected* PWGs only. Given a set $D = \{\, g^u_1, ..., g^u_n \,\}$ of uncertain data graphs, an exact query graph q and a probability threshold θ, the probabilistic supergraph search retrieves the set A_q of all the uncertain data graphs $g^u_i \in D$ where $P_c(q \supseteq g^u_i) \geq \theta$.

Note that the underlying exact graph of each $g^u_i \in D$ is connected and q is also connected. For presentation simplicity, an uncertain data graph and a query graph are hereafter abbreviated to a *data graph* and a *query*, respectively.

Table 2.1 gives the notations used throughout this paper.

2.2 #P-Completeness

Theorem 1 *The problem of computing* $P_c(q \supseteq g^u)$ *is #P-Complete.*

Proof 1 *It is shown [3] that the problem of counting the number of connected spanning subgraphs in an exact graph* g^{exc} *is #P-Complete. Below, we show that a special case of the problem of computing* $P_c(q \supseteq g^u)$ *is equivalent to the above counting problem.*

Given an exact graph g, we can convert g to g^u *by assigning the occurrence probability* $\frac{1}{2}$ *to each edge in g. Assume that q is a supergraph of g. It is immediate that* $P(g')|_{g^u} = (\frac{1}{2})^{|E(g^u)|}$ *for any subgraph* g' *of* g^u*. Consequently,* $P_c(q \supseteq g^u) = |\mathcal{W}_c(g^u)| \times (\frac{1}{2})^{|E(g^u)|}$*. Thus, the theorem holds.*

Table 2.1: General Notations

Symbol	Description
g^u	an uncertain data graph
g^{exc}	the underlying graph of g^u (exact graph)
q/f	query / feature graph
D	a set of uncertain data graphs
g'	a PWG of an uncertain data graph
$\mathcal{W}(g^u)$	the set of all PWGs of g^u
$\mathcal{W}_c(g^u)$	the set of all connected PWGs of g^u
$P(e)/P(\bar{e})$	the occurrence / non-occurrence probability of an edge e
$P(g')\|_{g^u}$	the occurrence probability of a PWG g' of g^u
$P_c(q \supseteq g^u)$	the probability that $q \supseteq g^u$ with connectivity constraint
C_q/A_q	the candidate / answer set of a query graph q

The proof of Theorem 1 implies that computing $P_c(q \supseteq g^u)$ is #P-Complete even if q is known to contain g^{exc}. Moreover, testing if q contains a PWG of g^u is NP-Complete [8]; this, together with Theorem 1, makes the probabilistic supergraph search problem computationally very expensive. Motivated by these, this paper follows the filtering-verification framework to effectively and efficiently filter non-promising data graphs; that is, avoid the exact verification which is NP-Complete to conduct subgraph isomorphism test [8] and #P-Complete to conduct probability computation. We first propose our filtering techniques in Section 3 and then develop efficient verification algorithms in Section 4.

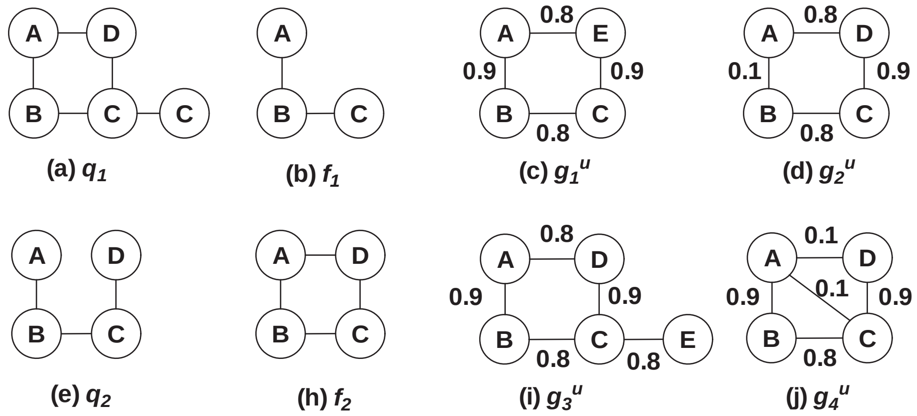
2.3 Compared to Probabilistic Subgraph Search

Pruning and Validation. In [25], the authors propose to effectively filter uncertain data graphs (i.e., prune false answers and validate true answers) without computing containment probability. Given a query q and a threshold θ, (1) the *Pruning Condition* is: If f is a subgraph of q and $P(f \subseteq g^u) < \theta$, then $P(q \subseteq g^u) < \theta$ and q can be pruned as a false answer to q. (2) the *Validation Condition* is: If q is a subgraph of f and $P(f \subseteq g^u) \geq \theta$, then $P(q \subseteq g^u) \geq \theta$ and q can be validated as a true answer to q. For pruning and validation purpose, [25] mines discriminative features f (i.e., frequent subgraphs) from D and index $P(f \subseteq g^u)$ for each g^u such that $f \subseteq g^{exc}$. However, the conditions in the IF parts of (1) and (2) has no indication on $P_c(q \supseteq g^u)$. Below is an example.

Example 2 *Figure 2.1(c) and (d) depict two data graphs g_1^u and g_2^u containing a feature f_1 depicted in Figure 2.1(b). Assume $\theta = 0.5$. Consider the query q_1 in Figure 2.1(a). Clearly, q_1 contains f_1 and it can be computed that $P(f_1 \subseteq g_1^u) = 0.72 \geq \theta$ and $P(f_1 \subseteq g_2^u) = 0.08 < \theta$. However, against the pruning condition in [25], g_1^u is a false result (i.e., $P_c(q_1 \supseteq g_1^u) = 0$), while g_2^u, on the contrary, is a true answer (i.e., $P_c(q_1 \supseteq g_2^u) = 0.61$).*

Figure 2.1(i) and (j) depict two data graphs g_3^u and g_4^u containing a feature f_2 depicted in Figure 2.1(h). Assume $\theta = 0.5$. Consider the query q_2 in Figure 2.1(e). Clearly, q_2 is contained by f_2 and it can be computed that $P(f_2 \subseteq g_3^u) = 0.51 \geq \theta$ and $P(f_2 \subseteq g_4^u) = 0.06 < \theta$. However, against the validation condition in [25], g_3^u is a false result (i.e., $P_c(q_2 \supseteq g_3^u) = 0$), while g_4^u is a true answer (i.e., $P_c(q_2 \supseteq g_4^u) = 0.52$).

Verification. In [25], the verification is based on the enumeration of all subgraphs g' of g^u which are *supergraph* of q. Meanwhile, our verification for probabilistic supergraph


(a) q_1 (b) f_1 (c) $g_1{}^u$ (d) $g_2{}^u$

(e) q_2 (h) f_2 (i) $g_3{}^u$ (j) $g_4{}^u$

Figure 2.1: Pruning and Validation Examples

search demands the enumeration of all spanning *connected* subgraphs g' of g^u which are *subgraphs* of q. Clearly, the verification techniques of the two problems differs on (1) the containment relationships between g' and q; (2) the connectivity constraint on g'. Thus the verification technique in [25] is also not applicable to our problem.

In conclusion, problems investigated in [25] and this paper are inherently different. Novel filtering and verification techniques are strongly demanded for efficient probabilistic supergraph search.

3 Filtering Techniques

In this section, we first propose two pruning rules, one is based on graph features and the other is based on a probabilistic signature of g^u. Since both pruning rules involve #P-Completeness, we propose two effective upper bounds, respectively, for these two pruning rules to reduce the pre-computation time of building filtering index, while these two upper bounds are still very effective.

3.1 Two Pruning Rules & Filtering Framework

Probabilistic Signature. We use the total occurrence probabilities $P_c(g_u)$ of each g' in $\mathcal{W}_c(g^u)$ as a *probabilistic signature* of g^u; specifically,

$$P_c(g_u) = \sum_{g' \in \mathcal{W}_c(g^u)} P(g')|_{g^u} \tag{3.1}$$

Theorem 2 *Given a query q and a data graph g^u, $P_c(q \supseteq g^u) \leq P_c(g^u)$.*

Proof 2 *Let $q' \supseteq g^{exc}$; recall g^{exc} is the underlying graph of g^u. It is immediate that $P_c(q \supseteq g^u) \leq P_c(q' \supseteq g^u)$. From (2.3) and (3.1), it follows that $P_c(q' \supseteq g^u) = P_c(g^u)$.*

Theorem 2 immediately implies the following pruning rule.

Pruning Rule 1 *Given a query q, a data graph g^u, and a threshold θ, if $P_c(g^u) < \theta$ then g^u can be pruned from the result set regarding θ and q.*

Feature Based. The probabilistic signature based pruning rule, Pruning Rule 1, does not utilize any structure information of a query q. Next, we propose a feature based pruning rule to further prune non-promising data graphs. The theorem below is a key observation.

Theorem 3 *Given a data graph g^u, a query q, and an exact graph f, suppose that $f \not\subseteq q$. Then, $P_c(q \supseteq g^u) \leq P_c(g^u) - P_c(f \subseteq g^u)$.*

Proof 3 *Let A denote the set of connected PWGs g' of g^u such that $g' \subseteq q$ and $f \not\subseteq g'$, and B denote the set of connected PWGs g' of g^u such that $g' \supseteq f$. Since $f \not\subseteq q$, $P_c(q \supseteq g^u) = \sum_{g' \in A} P(g')|_{g^u}$. As $A \subseteq \mathcal{W}_c(g^u) - B$, we have:*

$$\begin{aligned}\sum_{g' \in A} P(g')|_{g^u} &\leq \sum_{g' \in \mathcal{W}_c(g^u)} P(g')|_{g^u} - \sum_{g' \in B} P(g')|_{g^u} \\ &= P_c(g^u) - P_c(f \subseteq g^u)\end{aligned}$$

Theorem 3 implies the following pruning rule.

Pruning Rule 2 *Given a data graph g^u, a query q, a probability threshold θ, and an exact graph f such that $q \not\supseteq f$, if $P_c(g^u) - P_c(f \subseteq g^u) < \theta$ then g^u can be pruned regarding q and θ.*

Filtering Framework. Regarding Pruning Rule 2, we correspond each uncertain data graph $g_i^u \in D$ to its underlying graph g_i^{exc} and then we mine a set F of discriminative frequent subgraphs from g_i^{exc}s using the technique in [24]. The feature based index $I = \{ (f, D_f) | f \in F \}$ consists of the features in F and their corresponding GID-list (GraphID-list) D_f where D_f is represented as follows.

$$D_f = \{ (g_i^u, P_c(g_i^u) - P_c(f \subseteq g_i^u)) \mid g_i^u \in D, f \subseteq g_i^{exc} \} \tag{3.2}$$

Note that if $f \not\subseteq g_i^{exc}$, then $P_c(f \subseteq g_i^u) = 0$; thus such g_i^u is excluded from D_f since $P_c(g_i^u) - P_c(f \subseteq g_i^u)$ equals the probabilistic signature $P_c(g_i^u)$ in this case.

Clearly, $P_c(g^u) - P_c(f \subseteq g^u) < P_c(g^u)$ if $P_c(f \subseteq g^u) \neq 0$; this means that $P_c(g^u) - P_c(f \subseteq g^u)$ is tighter than $P_c(g^u)$; nevertheless, the probabilistic signature $P_c(g^u)$ can be still used to prune g^u if g^u is not in any D_f where $f \not\subseteq q$. Thus, we conduct feature based pruning first. Below is the filtering algorithm.

Algorithm 1: Filtering (q, F, D, θ)

```
Input  : $q$: query graph; $F$: set of features; $\theta$: probability threshold; $D$: set of uncertain
         data graphs;
Output : $C_q$: the candidate set;
$C_q := D$;
foreach $f \in F$ with $q \not\supseteq f$ do
    foreach $g^u \in D_f$ with $P_c(g^u) - P_c(f \subseteq g^u) < \theta$ do
        $C_q := C_q - \{ g^u \}$;
foreach data graph $g^u \in C_q$ with $P_c(g^u) < \theta$ do
    $C_q := C_q - \{ g^u \}$;
return $C_q$;
```

Note that to compute $P_c(g^u)$ and $P_c(f \subseteq g^u)$, we need to enumerate all connected PWGs of g^u. In fact, the proof of Theorem 1 implies that computing $P_c(g^u)$ is #P-Complete; thus, computing $P_c(f \subseteq g^u)$ is also #P-Complete. Below we develop effective upper bounds of $P_c(g^u)$ and $P_c(g^u) - P_c(f \subseteq g^u)$, respectively, to reduce the computation costs.

3.2 From #P-Complete To PTIME

Upper Bound of Probabilistic Signature. Below is the basic observation. Any connected PWG g' of g^u must contain a spanning tree T of g^{exc}. Given a spanning tree T of g^{exc} where $T = \{ T_1, ..., T_{n-1} \}$, each T_i is an edge in T, and $n = |V(g^u)|$, we use $T[1,h] = \{ T_1, ..., T_h \}$ to denote the set of the first h spanning edges in T and we assume that the edges in T are labeled such that for any h ($1 \leq h \leq n-1$), $T[1,h]$ is a sub-tree (i.e., connected). For any h ($1 \leq h \leq n-1$), $\mathcal{W}_c(g^u)$ can be divided into the following $(h+1)$ disjoint sets: $\mathcal{W}_{T[1,h]}$ - the set of connected PWGs of g^u containing $T[1,h]$; for $1 \leq i \leq h$, $\mathcal{W}_{T[1,i-1] \wedge \overline{T_i}}$ - the set of connected PWGs containing $T[1,i-1]$ but excluding T_i. We assume that $T[1,0]$ contains a dummy edge T_0 with $P(T_0) = 1$. For example, as illustrated by Figure 3.1, the solid lines in Figure 3.1(b) give a spanning tree of the graph in Figure 3.1(a).

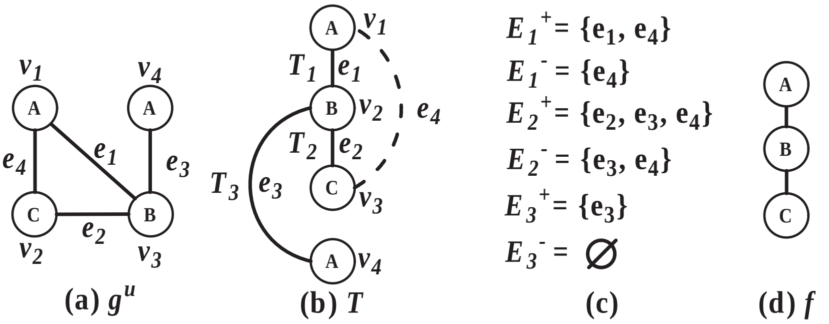


Figure 3.1: Example

Let $P(\mathcal{W}_{T[1,h]}) = \sum_{g' \in \mathcal{W}_{T[1,h]}} P(g')|_{g_u}$, and for $1 \leq i \leq h$, $P(\mathcal{W}_{T[1,i-1] \wedge \overline{T_i}}) = \sum_{g' \in \mathcal{W}_{T[1,i-1] \wedge \overline{T_i}}} P(g')|_{g_u}$

$$P_c(g^u) = P(\mathcal{W}_{T[1,h]}) + \sum_{i=1}^{h} P(\mathcal{W}_{T[1,i-1] \wedge \overline{T_i}}) \tag{3.3}$$

To tighten the upper-bound, we make use of a simple necessary condition of the connectivity as follows. Let E_h^+ denote the set of edges which connect a vertex in $T[1,h-1]$ to a vertex not in $T[1,h-1]$, $E_h^- = E_h^+ - \{T_h\}$. For example, Figure 3.1(c) shows E_i^- and E_i^+ regarding T and g^u in Figure 3.1(b).

Given A is a set of edges, P_A denotes the probability that at least one edge in A occurs; that is,

$$P_A = 1 - \prod_{e \in A} (1 - P(e)) \tag{3.4}$$

Since g^{exc} (the underlying exact graph of g^u) is connected, $E_h^+ \neq \emptyset$ if $h \leq n-1$ and $E_n^+ = \emptyset$. Note that $P_{E_n^+} = 1$ since $\prod_{e \in \emptyset}(1 - P(e)) = 0$. It can be immediately verified that:

$$P(\mathcal{W}_{T[1,h]}) \leq (\prod_{i=1}^{h} P(T_i))P_{E_{h+1}^+}. \tag{3.5}$$

It is clear that $E_h^- = \emptyset$ if T_h is a cut edge of g^u; for example, T_3 in Figure 3.1(c) is a cut edge. Regarding $P(\mathcal{W}_{T[1,i-1] \wedge \overline{T_i}})$, if T_i is a cut edge of g^{exc}, we have $P(\mathcal{W}_{T[1,i-1] \wedge \overline{T_i}}) = 0$; otherwise, there must be a connected PWG g' containing a spanning tree T' of g^{exc} such that (1) $T[1, i-1] = T'[1, i-1]$ and (2) $T_i \in T$ is replaced by $T'_i \in E_i^-$. Therefore, we assign $P_{E_i^-}$ by 0 if T_i is a cut, and $P_{E_i^-}$ is calculated according to (3.4) otherwise. It can be immediately verified that:

$$P(\mathcal{W}_{T[1,i-1] \wedge \overline{T_i}}) \leq (\prod_{j=1}^{i-1} P(T_j))P(\overline{T_i})P_{E_i^-} \tag{3.6}$$

Combining the two inequalities (3.5) and (3.6) gives an upper-bound $UB(g^u, T, h)$ of $P_c(g^u)$; that is:

$$\begin{aligned} UB(g^u, T, h) &= (\prod_{i=1}^{h} P(T_i))P_{E_{h+1}^+} + \\ & \sum_{i=1}^{h}(\prod_{j=1}^{i-1} P(T_j))P(\overline{T_i})P_{E_i^-} \end{aligned} \tag{3.7}$$

Theorem 2 and (3.7) imply the following pruning rule.

Pruning Rule 3 *Given a data graph g^u, a spanning tree T of g^{exc} and a probability threshold θ, if $UB(g^u, T, h) < \theta$ for an h ($1 \leq h \leq |V(g^u)|$) then g^u can be pruned against q and θ.*

Generating Tighter Upper Bound. Below we first show that given a T, $UB(g^u, T, h)$ is decreasing when h is increasing; that is, $UB(g^u, T, h)$ is the minimum when $h = n - 1$.

Theorem 4 *Given an uncertain graph g^u, a spanning tree T of g^{exc} and an integer $1 \leq h \leq |V(g^u)| - 1$, $UB(g^u, T, h)$ monotonically decreases when h increases.*

Proof 4 *We prove that $UB(g^u, T, h-1) \leq UB(g^u, T, h)$ for any $2 \leq h \leq |V(g^{exc})| - 1$. Let $UB_{\triangle}(h) = UB(g^u, T, h) - UB(g^u, T, h-1)$.*

Let V be the vertices included in the sub-tree $T[1, h-1]$. Let v be the new vertex in $T[1, h]$ induced by T_h (i.e., not in $T[1, h-1]$). Let V' be the vertices in $V(g^u) - V - \{v\}$.

We use A to denote the set of edges between v and any vertices in V excluding T_h in the original spanning tree T. We use B to denote the set of edges between any vertex in V and any vertex in V'. We use C to denote the set of edges between v and any vertices in V'. We have $E_h^+ = \{T_h\} \cup A \cup B$, $E_{h+1}^+ = B \cup C$ and $E_h^- = A \cup B$.

Let $P(\overline{A})$, $P(\overline{B})$, $P(\overline{C})$ denote the probabilities that all edges in A, B, and C, respectively, do not occur. Below can be immediately verified.

$$
\begin{aligned}
P(T_h)P_{E^+_{h+1}} &= P(T_h)(1 - P(\overline{B})P(\overline{C})) && (3.8)\\
P_{E^+_h} &= (1 - P(\overline{T_h})P(\overline{A})P(\overline{B}))) && (3.9)\\
P(\overline{T_h})P_{E^-_h} &= P(\overline{T_h})(1 - P(\overline{A})P(\overline{B})) && (3.10)
\end{aligned}
$$

From (3.7), we have that if T_h is a cut,

$$
\begin{aligned}
UB_{\Delta}(h) &= (\prod_{i=1}^{h-1} P(T_i))(P(T_h)P_{E^+_{h+1}} - P_{E^+_h})\\
&\leq (\prod_{i=1}^{h-1} P(T_i))(P(T_h)P_{E^+_{h+1}} - P_{E^+_h}\\
&\quad +P(\overline{T_h})P_{E^-_h})\\
&= (\prod_{i=1}^{h-1} P(T_i))(-P(T_h)P(\overline{B})P(\overline{C})) \leq 0
\end{aligned}
$$

If T_h is not a cut, then

$$
\begin{aligned}
UB_{\Delta}(h) &= (\prod_{i=1}^{h-1} P(T_i))(P(T_h)P_{E^+_{h+1}} - P_{E^+_h}\\
&\quad +P(\overline{T_h})P_{E^-_h})\\
&= (\prod_{i=1}^{h-1} P(T_i)) \times (-P(T_h)P(\overline{B})P(\overline{C}))\\
&\leq 0
\end{aligned}
$$

Theorem 4 implies that given an uncertain graph g^u and any spanning tree T of g^{exc}, $UB(g^u, T, n-1)$ is the smallest upper bound regarding T where $n = |V(g^u)|$; that is, $\prod_{i=1}^{n-1} P(T_i) + \sum_{i=1}^{n-1}(\prod_{j=1}^{i-1} P(T_j))P(\overline{T_i})P_{E^-_i}$ is the minimum. Consequently, the optimality of the upper bound only depends on the choice of T and the ordering of edges in T.

Finding an optimal T to minimize $UB(g^u, T, n-1)$ tends to be computationally hard. We use a greedy algorithm to generate T as follows in Algorithm 2.

Algorithm 2: UBounding (g^u)

```
Input  : $g^u$: a data graph with $n$ vertices;
Output : $T = \{T_1, T_2, ...T_{n-1}\}$: spanning tree of $g^{exc}$;
Iteratively choose an edge $e$ in $g^u$ to be included in $T$ such that
    • $e$ and the current $T[1, h]$ form a tree.
    • the current $UB(g^u, T, h+1)$ is minimized.
```

Clearly, UBounding runs in $O(|V(g^u)||E(g^u)|^2)$ time, which can also produce $UB(g^u, T, n-1)$ in the end.

Upper-Bound of $P_c(g^u) - P_c(f \subseteq g^u)$. Note that the upper bound that we devised for probabilistic signature is based on a spanning tree T of g^{exc}. Now, we present an upper-bound of $P_c(g^u) - P_c(f \subseteq g^u)$ based on a subtree of T to reduce the computation costs.

Recall that $\mathcal{W}_c(g^u)$ can be divided into $(h+1)$ disjoint sets: $\mathcal{W}_{T[1,h]}$ and for $1 \leq i \leq h$, $\mathcal{W}_{T[1,i-1] \wedge \overline{T_i}}$. Let $f = \{T_{j_1}, ..., T_{j_l}\}$ be a subtree of T where $j_1 < j_2, ... < j_l$. Suppose that $q \not\subseteq f$. From (3.5), (3.6), and the proof of Theorem 3, it immediately follows that

$$P_c(g^u) - P_c(f \subseteq g^u) \leq \sum_{i=1}^{j_l} (\prod_{x=1}^{i-1} P(T_x)) P(\overline{T_i}) P_{E_i^-} \tag{3.11}$$

Note that in (3.11), $\sum_{i=1}^{j_l} (\prod_{x=1}^{i-1} P(T_x)) P(\overline{T_i}) P_{E_i^-}$ is increasing when j_l increases and a tree f may have multiple mappings in T, we always choose a mapping with the minimum value of j_l to minimize $\sum_{i=1}^{j_l} (\prod_{x=1}^{i-1} P(T_x)) P(\overline{T_i}) P_{E_i^-}$. For example, the tree f in Figure 3.1(d) has two mappings, (T_1, T_2) and (T_2, T_3), to the spanning tree in Figure 3.1(b); we choose the mapping (T_1, T_2) to minimize the upper-bound in (3.11). The following pruning rule is immediate.

Pruning Rule 4 *Suppose that $T = \{\, T_1, ..., T_{n-1} \,\}$ is a spanning tree of the underlying exact graph g^{exc} of g^u, and f is a tree, and the subtree $\{\, T_{j_1}, ..., T_{j_l} \,\}$ of T is the match induced by an isomorphic mapping from f to T such that j_l is minimized. If $\sum_{i=1}^{j_l} (\prod_{x=1}^{i-1} P(T_x)) P(\overline{T_i}) P_{E_i^-} < \theta$ and $f \not\subseteq q$, then g^u can be pruned regarding q and θ.*

$\sum_{i=1}^{j_l} (\prod_{x=1}^{i-1} P(T_x)) P(\overline{T_i}) P_{E_i^-}$ can be re-written below in (3.12).

$$\begin{aligned} & UB(g^u, T, n-1) - \prod_{i=1}^{n-1} P(T_i) \\ & - \sum_{i=j_l+1}^{n-1} (\prod_{x=1}^{i-1} P(T_x)) P(\overline{T_i}) P_{E_i^-} \end{aligned} \tag{3.12}$$

Clearly, given T, the upper-bound in (3.12) can be calculated in $O(|E(g^u)||V(g^u)|)$. Moreover, the upper-bound in (3.12) is not greater than the upper-bound $UB(g^u, T, n-1)$ of the probabilistic signature regarding the same T. In our implementation, for each g_i^u we use the same spanning tree to derive the two upper-bounds; thus, the upper-bound in (3.12) is always tighter. We follow the filtering framework in Algorithm 1 with the following differences:

- Instead of mining discriminative frequent subgraphs among g_i^{exc}s, we mine discriminative frequent subtrees among the spanning trees each of which is obtained by Algorithm 2 from a g_i^u.
- Replace $P_c(g^u)$ by its upper-bound.
- Replace $P_c(g^u) - P_c(f \subseteq g^u)$ by its upper-bound.

4 Verification Algorithms

We develop two novel algorithms Baseline and Synchronize to efficiently verify the candidates after filtering.

4.1 Baseline Algorithm

Basic Idea. Since each $g' \in \mathcal{W}_c(g^u)$ contains a spanning tree T of g^{exc}, we may use $T(g^{exc})$, the set of all spanning trees of g^{exc}, to partition $\mathcal{W}_c(g^u)$ into $|T(g^{exc})|$ disjoint sets $\mathcal{W}_T$ such that (1) for any $g' \in \mathcal{W}_T$, $T \subseteq g'$; and (2) $\bigcup_{T \in T(g^{exc})} \mathcal{W}_T = \mathcal{W}_c(g^u)$ We will present how to partition $\mathcal{W}_c(g^u)$ into such disjoint sets $\mathcal{W}_T$ later in this subsection.

The basic idea of Baseline is to first search the mappings from all $T \in T(g^{exc})$ to q (i.e., conducting subgraph isomorphism tests) and then extend the mappings of T to induce the matches of all $g' \in \mathcal{W}_T$ (i.e., linear-time extension regarding $|E(g') - E(T)|$). Baseline not only tests all PWGs with shared computation on T but also reduces the number of subgraph isomorphism tests from $O(|\mathcal{W}_c(g^u)|)$ to $O(|T(g^{exc})|)$.

Efficiently Generating $T(g^{exc})$. We first present our enumeration algorithm to generate $T(g^{exc})$ in a *depth first search* fashion from the lowest level ($h = 1$) to the highest level ($h = |V(g^u)| - 1$). It is outlined in Algorithm 3 with 2 phases: *go-down* phase (Lines 1-2) and *alternate* phase (Lines 3-8).

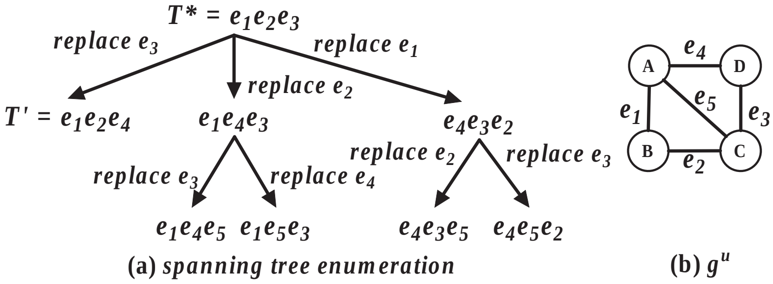
Algorithm 3: EnumTrees $(h, T, T.R, T(g^{exc}))$

```
  Input  : $h$: the current level, initially 1;
           $T$: the current spanning tree, initially $T^*$;
           $T.R$: the edges replaced to get $T$, initially $\emptyset$;
           $T(g^{exc})$: the set of all spanning trees;
1 if $h < n - 1$ then
2     EnumTrees $(h + 1, T, T.R, T(g^{exc}))$ ;
3 if ReCheck$(T_h)$ then
4     $e :=$ Relpace$(T_h)$ ;
5     $T' :=$ Reorder$((T - \{ T_h \}) \bigcup \{ e \})$ ;
6     $T'.R := T.R \bigcup \{ T_h \}$;
7     $T(g^{exc}) := T(g^{exc}) \bigcup \{ T' \}$;
8     EnumTrees $(h, T', T'.R, T(g^{exc}))$ ;
```

Algorithm 3 begins with an initial spanning tree T^* pre-loaded in $T(g^{exc})$. We will later discuss how to generate T^* for verification efficiency. The *go-down* phase corresponds to the depth first search fashion. For each enumerated T, the *edge exclusion set* $T.R$ records the set of edges iteratively replaced during the enumeration to get T from T^*. We use $T.R$ to generate $\mathcal{W}_T$ later. In the *alternate* phase at the h-th level of the current spanning tree T, if there is an edge $e \in (E(g^u) - T - T.R)$ which can replace T_h to form an alternative spanning tree T' together with the remaining edges in T, the function ReCheck (T_h) returns true (Line 3) and we enumerate such T'.

The function Relpace (T_h) (Line 4) returns an arbitrary edge $e \in (E(q) - T - T.R)$ to replace T_h and form T' with the remaining edges in T by fixing the prefix $T[1, h-1]$ (i.e, $T'_i = T_i$ for $1 \leq i \leq h - 1$). We may need to reorder the edges T'_i for $h \leq i \leq |V(g^u)| - 1$ (Line 5) to enforce that T'_i always connects $T'[1, i - 1]$.

Example 3 *We use Figure 4.1(a) to illustrate the enumeration process to generate $T(g^{exc})$ for the data graph g^u in Figure 4.1(b). The root in Figure 4.1(a) gives the initial spanning tree $T^* = e_1e_2e_3$ and $T^*.R = \emptyset$. We assume the root vertex T^*_0 of T^* is A. Consecutively conducting the go-down phase on T^*, we drill down to e_3 for executing the alternate phase and replace e_3 with e_4 to form the next spanning tree $T' = e_1e_2e_4$ and $T'.R = \{ e_3 \}$. Continuing the alternate phase on T' to replace e_4 of T' will fail since no edge in ($E(g^u) - T' - T'.R$) can connect the remaining edges in T'.*



(a) *spanning tree enumeration* (b) g^u

Figure 4.1: Spanning Tree Enumeration

Algorithm 3 may execute the alternate phase while drilling down to e_2 and e_1 in T^*, respectively. We depict the entire $T(g^{exc})$ in Figure 4.1(a). □

Partition $\mathcal{W}_c(g^u)$. We partition $\mathcal{W}_c(g^u)$ into a set of $|T(g^{exc})|$ disjoint sets $\mathcal{W}_T$ as follows. For each $T \in T(g^{exc})$, $\mathcal{W}_T = \{\, g' \mid g' \in \mathcal{W}_c(g^u) \wedge T \subseteq g' \wedge E(g') \cap T.R = \emptyset \,\}$. Intuitively, each $g' \in \mathcal{W}_T$ contains T but no edge in $T.R$. Theorem 5 gives the correctness of the partition.

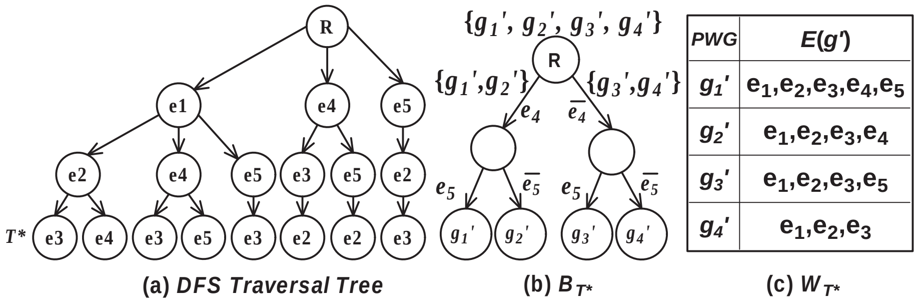
Theorem 5 *Given $T(g^{exc})$, (1) For any $T, T' \in T(g^{exc})$ such that $T \neq T'$, $\mathcal{W}_T \bigcap W_{T'} = \emptyset$. (2) For any $g' \in \mathcal{W}_c(g^u)$, there exists a $T \in T(g^{exc})$ such that $g' \in \mathcal{W}_T$.*

Proof 5 *Regarding (1), it is immediate that, given any two different spanning trees in $T(g^{exc})$, there must be an edge $e \in E(g^u)$ such that e occurs in one spanning tree T but occurs in the edge exclusion set $T'.R$ of another spanning tree T'. Thus for any $g'_1 \in \mathcal{W}_T$ and $g'_2 \in \mathcal{W}_T$, $g'_1 \neq g'_2$ since $e \in E(g'_1)$ and $e \notin E(g'_2)$.*

Regarding (2), consider any $g' \in \mathcal{W}_c(g^u)$. Let $E = E(g^u) - E(g')$ be the set of edges in g^u missed by g'. If $T^* \cap E = \emptyset$ for the initial spanning tree T^*, we have $g' \in \mathcal{W}_{T^*}$. Otherwise, assume $T^* \cap E = \{\, T^*_{i1}, ..., T^*_{ik} \,\}$ where $i_1 < .... < i_k$. We may drill down to T^*_{i1} and execute Relpace *(T^*_{i1}) in Algorithm 3 to form another spanning tree T'. Since T^* and T' have identical prefix (i.e., $T^*[1, i_1 - 1]$), after we reorder the remaining edges in T', $T^*_{i2}, ..., T^*_{ik}$ will only be ordered after T'_{i1}. We can keep drill down on T' and execute the alternate phase to replace the $T^*_{i2}, ..., T^*_{ik}$ until we have some T'' such that T'' does not contain any edge in $T^* \cap E$. It is clear that $g' \in \mathcal{W}_{T''}$.*

Effectively Storing $T(g^{exc})$ and $\mathcal{W}_T$. We present a data structure, DFS Traversal Tree, denoted by $\mathcal{T}$, to organize $T(g^{exc})$ for efficient prefix sharing search. The basic idea is that when enumerating T' from T by replacing T_h, we store the common prefix $T[1, h-1]$ of T and T' for only once in $\mathcal{T}$ and organize their remaining spanning edges as two different branches. In $\mathcal{T}$, each node N represents an edge T_h of a spanning tree $T \in T(g^{exc})$, while the root R represents the head vertex of the initial spanning tree T^*. We first load T^* as the left-most path of $\mathcal{T}$. In Algorithm 3, iteratively, if T_h is replaced by an edge T'_h to form T', T'_h is allocated as the right sibling next to T_h such that T and T' share the prefix $T[1, h-1]$. Clearly, $\mathcal{T}$ requires a space of $O(|T(g^{exc})||V(g^u)|)$.

Example 4 *Given the data graph g^u in Figure 4.1(b), Figure 4.2(a) shows the resulted $\mathcal{T}$ of $T(g^{exc})$. Here, a path from the root to a leaf gives a spanning tree. Note that the order of paths in $\mathcal{T}$ from left to right gives the enumeration order of all spanning trees in Algorithm 3.* □



PWG	E(g')
g1'	e1,e2,e3,e4,e5
g2'	e1,e2,e3,e4
g3'	e1,e2,e3,e5
g4'	e1,e2,e3

(a) *DFS Traversal Tree* (b) B_{T*} (c) W_{T*}

Figure 4.2: Effectively Storing $T(g^{exc})$ and $\mathcal{W}_T$

We organize each set $\mathcal{W}_T$ in a binary tree B_T by using the *extra edge set* $T.E = E(g^u) - T - T.R$ of T. All the edges in $T.E$ are ordered by their edge IDs. Intuitively, any PWG $g' \in \mathcal{W}_T$ may only contain T and a subset of edges in $T.E$. Assume $T.E = \{ e_{i1}, ..., e_{ik} \}$, we build B_T from the lowest level ($h = 1$) to the highest level ($h = k + 1$). In B_T, each internal node N represents a subset W of $\mathcal{W}_T$, while the root node represents $\mathcal{W}_T$. Assume N is an internal node at level-h, the left child of N stores all PWGs in W containing the h-th edge e_{ih} in $T.E$, while the right child of N stores the remaining PWGs in W. Finally, each leaf node in B_T gives a PWG $g' \in \mathcal{W}_T$.

Example 5 *Figure 4.2(b)-(c) depict B_{T^*} and $\mathcal{W}_{T^*}$ of the initial spanning tree T^* in Figure 4.2(a). Since $T^*.E = \{ e_4, e_5 \}$, we first use e_4 and then e_5 to partition $\mathcal{W}_{T^*}$. We show the set of PWGs represented by each node in the brackets. The 4 leaf nodes give the 4 PWGs.* □

Baseline Algorithm. Baseline searches $\mathcal{T}$ in a *depth first search* fashion to first generate extensible mappings $\mathcal{F}$ of all $T \in T(g^{exc})$ and then extend $\mathcal{F}$ to induce the matches of PWGs in $\mathcal{W}_T$. Iteratively, after extending $\mathcal{F}$ on the current spanning tree T from the spanning edge T_h, we direct the extension to the right sibling node of T_h for a prefix sharing search; that is, extend $\mathcal{F}$ from T'_h on the alternative spanning tree T' enumerated by replacing T_h. Note that we do not build the entire $\mathcal{T}$ (i.e., $T(g^{exc})$) before the search but build $\mathcal{T}$ on demand during search; that is, based on the current results. We present Baseline in Algorithm 4 with two phases. The *inducing* phase (Lines 3-6) induces the matches of PWGs in $\mathcal{W}_T$ based on a mapping $\mathcal{F}$ of T. The *drill-down* phase (Lines 8-18) iteratively drills down on the current or alternative spanning trees to generate extensible mappings $\mathcal{F}$.

Algorithm Sketch. In Algorithm 4, h is the current search depth (initially 0). T is the current spanning tree to be searched (initially T^*). A spanning edge T_h is a tuple (v', v) where $T_h.v'$ is the vertex already existing in $T[1, h-1]$ and $T_h.v$ is the new vertex introduced to $T[1, h]$ by T_h. T_0 is the head vertex of T. The current partial mapping $\mathcal{F}$ on $T[1, h-1]$ is a vector $\{ \mathcal{F}[1], ..., \mathcal{F}[h-1] \}$ where $\mathcal{F}[i]$ $(1 \leq i \leq h-1)$ is a vertex in q mapped from $T_i.v$. Initially, the DFS Traversal Tree $\mathcal{T}$ has only the left-most path (i.e., T^*). q and θ are the query and probability threshold, respectively. P is the sum of probabilities of all PWGs currently contained by q (initially 0). If $P \geq \theta$ after Baseline terminates, g^u is a result of q.

In Algorithm 4, Line 1-2 extends the partial mapping $\mathcal{F}$ on $T[1, h-1]$ to $T[1, h]$. We iteratively retrieve the next unmapped vertex u in the candidate set $C(T_h.v)$ to map the spanning edge T_h. $C(T_h.v)$ contains all the vertices in q with the same label as $T_h.v$. If $h = 0$, u can be any vertex in $C(T_0)$; otherwise, u is chosen from $C(T_h.v)$ to map T_h into q; that is, u must be a neighbor of the vertex in q mapped from $T_h.v'$.

Algorithm 4: Baseline $(h, T, \mathcal{F}, q, P, \theta, \mathcal{T})$

```
Input : $h$: the current search depth; $T$: current spanning tree;
        $\mathcal{F}$: the current partial mapping; $q$: the query;
        $P$: the current probability; $\theta$: the probability threshold;
        $\mathcal{T}$: the DFS Traversal Tree;
1  foreach $u \in$ NextCan$(T_h, \mathcal{F}, q)$ do
2      $\mathcal{F}[h] := u$ ;
3      if $h = |V(g^u)| - 1$ then
4          Inducing$(\mathcal{W}_T, \mathcal{F}, q, P, \theta)$ ;
5          if $P \geq \theta \lor$ CheckNode$(T_h)$ then
6              return ;
7      else
8          $T^c := T$ ;
9          do
10             if $\neg$CheckNode$(T^c_h)$ then
11                 Baseline$(h + 1, T^c, \mathcal{F}, q, P, \theta, \mathcal{T})$ ;
12             if $P \geq \theta \lor$ CheckChildren$(T^c_h)$ then
13                 return ;
14             if ReCheck$(T^c_{h+1})$ then
15                 $T' :=$ getNext$(T^c, T^c_{h+1}, \mathcal{T})$ ;
16             else
17                 $T' := nil$;
18             $T^c := T'$ ;
19         while $T^c \neq nil$;
```

inducing. If all spanning edges in T have been mapped (i.e., checked by Line 3), the function Inducing traverses the binary tree B_T in a breadth first search fashion to induce the matches of PWGs in $\mathcal{W}_T$ based on $\mathcal{F}$. From the root of B_T (level-1), we iteratively check if $\mathcal{F}$ can map the i-th edge $(u, v) \in T.E$ into q (i.e., if $(\mathcal{F}[u], \mathcal{F}[v]) \in E(q)$). If the i-th edge can be mapped, we traverse both child nodes of the current node; otherwise, we only traverse the right child of the current node. Such extension on each $g' \in \mathcal{W}_T$ runs in $O(|E(g')|)$ time. When we reach a leaf node N_l for the first time, we add $P(g')|_{g^u}$ to P for the PWG g' given by N_l. Note that we do not build the entire B_T before Inducing begins but built it during executing Inducing. Whenever we need to traverse the left and/or right child of a node N, we build them if they have not been built before.

If $P \geq \theta$ after executing Inducing, we immediately terminate to verify g^u as a result. Let N be the node in $\mathcal{T}$ representing T_h. CheckNode (T_h) checks if for each leaf node N_l in the subtree of $\mathcal{T}$ rooted at N, all PWGs in $\mathcal{W}_T$ given by each N_l are validated subgraphs of q. If CheckNode (T_h) returns true, we backtrack to avoid future extension on T_h. Efficiently executing CheckNode (T_h) will be given later.

drill-down. If there are still unmapped spanning edges in T, we first use T as the next spanning tree T^c to be searched. Iteratively, we first drill down on T^c_{h+1} to extend $\mathcal{F}$ if CheckNode (T^c_{h+1}) returns false (Line-10-13) and then enumerate an alternative T' by replacing T^c_{h+1} (Line-14-18). We iteratively alternate the search by using T' as T^c.

If $P \geq \theta$ after drilling-down on T^c_{h+1}, we immediately terminate and return g^u as a result. The function CheckChildren returns true if (1) we have enumerated all alternative T' by iteratively replacing the $(h + 1)$-th spanning edge starting form T. (2) CheckNode (T_{h+1}) and CheckNode (T'_{h+1}) return true for T and all alternative T'. If both conditions hold, we backtrack to avoid future extension on T_h. ReCheck

(T^c_{h+1}) returns true if we can enumerate T' from T^c by replacing T^c_{h+1}; otherwise, we end the iteration by setting $T' = nil$. If T' has already been generated in $\mathcal{T}$, the function getNext immediately returns T' to save the enumeration cost; otherwise, it enumerates T' in $\mathcal{T}$ and then returns T'.

Remark. We efficiently execute CheckNode (T_h) as follows. For each T_h represented by a node N in $\mathcal{T}$, we store a boolean flag $N.flag$ (initially false) and simply let CheckNode (T_h) return $N.flag$. After executing Inducing on a leaf node N_l representing T, if all PWGs in $\mathcal{W}_T$ have been validated as subgraphs of q, we set $N_l.flag$ to true. Besides, when CheckChildren (T_h) returns true, we set $N.flag$ to true for the node N representing T_h. This is because CheckChildren (T_h) = true implies that $N_c.flag = true$ for all child nodes N_c of N. (i.e., CheckNode (T'_{h+1}) = true for all alternative T' is enumerated from T in Algorithm 4).

Correctness of Baseline. The correctness of Algorithm 4 is immediate based on Theorem 5.

Search Ordering. To test subgraph isomorphism from a graph g to a graph q, [20] transforms g to a weighted graph and uses a minimum spanning tree T of g to order the search. It assigns a weight $\phi(v)$ ($\phi(u, v)$) to each vertex u (edge (u, v)) in g such that $\phi(v)$ ($\phi(u, v)$) is the occurrence of the vertices (edges) in q with the label $l(u)$ ($(l(u), l(v))$). [20] picks the vertex v with the minimum $\phi(v)$ as the root vertex T_0 and then iteratively selects the spanning edges by the PRIM algorithm [11]; that is, enforce the following PRIM order.

Definition 4 (PRIM Order) *A spanning tree T of g conforms to PRIM order if and only if for $1 \leq h \leq n-1$, T_h is the edge in $E(g) - T[1, h-1]$ with the smallest weight to connect $T[1, h-1]$.*

PRIM order aims to minimize the number of potential mappings for T_0 and each spanning edge in T. In Algorithm 3, we first generate the initial spanning tree T^* to conform the PRIM order. When generating the alternative T', we always replace T_h with the minimum weighted $e \in (E(g^u) - T[1, h] - T.R)$ to connect the remaining edges and reorder the edges to conform the PRIM order.

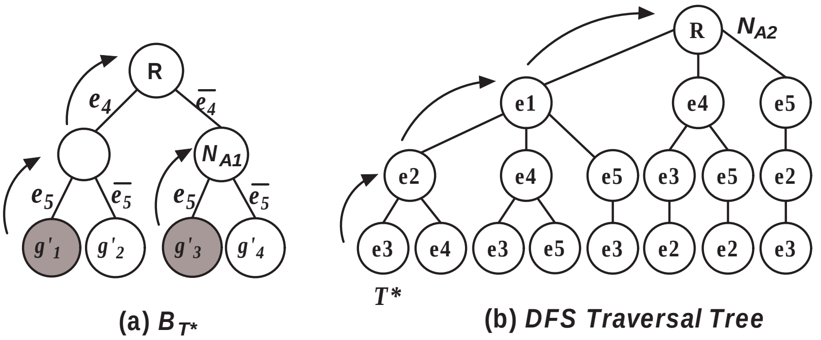
4.2 $Baseline^+$: Subgraph Validation

We next present an optimization technique to enhance our baseline algorithm. The enhanced algorithm is referred to as Baseline$^+$. The motivation is as follows. Given two PWGs $g', g'' \in \mathcal{W}_c(g^u)$ such that $g' \supseteq g''$, whenever we find a match $\mathcal{F}(g')$ of g' by extending the mapping $\mathcal{F}$ in Inducing, $\mathcal{F}(g')$ must contain a match $\mathcal{F}(g'')$ of g'' as a sub-match. To save the test on some subgraphs g'' of g', we propose a subgraph validation technique by efficiently traversing B_T and DFS Traversal Tree $\mathcal{T}$. The traversal consists of two phase, *look-up* and *span-down*.

look-up. Assume Inducing finds a match $\mathcal{F}(g')$ for a PWG $g' \in \mathcal{W}_T$. The look-up phase begins from the leaf node N in B_T representing g'. We look up to traverse all ancestors of N in B_T until we reach the first ancestor N_A which has a left sibling. If the root $B_T.R$ is reached, the look-up continues from the leaf node N in $\mathcal{T}$ representing T. If the root $\mathcal{T}.R$ is reached, $N_A = \mathcal{T}.R$.

span-down. If N_A is in B_T, beginning from N_A, we span down to find all leaf nodes N_l in the subtree of B_T rooted at N_A. Assume N_A is at level-h of B_T. For any PWG g'' given by N_l, it is immediate that $g' \supseteq g''$. We explain the observation as follows. Since g' and g'' both contain T and the same prefix from $B_T.R$ to N_A, they only differ on the edges in $T.E$ from the h-th edge. Besides, g' contains all the edges in $T.E$ from the $(h-1)$-th edge since we only traverse the left child nodes. Thus, we have $E(g'') \subseteq E(g')$.

If N_A is in $\mathcal{T}$, beginning from N_A and all the right siblings N of N_A, we span down to find all leaf nodes N_l in the subtrees of $\mathcal{T}$ rooted at N_A and N. Assume N_A is at level-h of $\mathcal{T}$. For all T' given by such leaf nodes N_l, it is immediate that $g' \supseteq g''$ for all PWGs $g'' \in W_{T'}$. We explain the observation as follows. Since the look-up reaches $B_T.R$ by the left-most path, we have $T.E \subseteq E(g')$ and $E(g') = E(g^u) - T.R$. Besides, $T.R' \supseteq T.R$ as T' is generated from T. Thus, we have $E(g'') \subseteq (E(g^u) - T'R) \subseteq E(g')$.



(a) B_{T*}

(b) *DFS Traversal Tree*

Figure 4.3: Subgraph Validation

Example 6 *Regarding the two shaded PWG g'_1 and g'_3 in the binary tree of the initial spanning tree T^* depicted in Figure 4.3(a). If $q \supseteq g'_3$, the look-up phase finds the ancestor N_{A1} in Figure 4.3(a) where the arrows correspond to the look-up. We can span down from N_{A1} to validate g'_4 as a subgraph of g'_3. If $q \supseteq g'_1$, the look-up phase finds the ancestor N_{A2} in Figure 4.3(b) (i.e., the root $\mathcal{T}.R$). Since all leaf nodes in $\mathcal{T}$ can be spanned from $\mathcal{T}.R$, we can validate all $g' \in \mathcal{W}_c(g^u)$ as subgraphs of g'_1 (i.e., g'_1 is g^{exc}).* □

4.3 Synchronization Algorithm

To terminate a non-promising search as early as possible, we develop a synchronization traversal strategy to conduct pruning based on the current search results.

Basic Idea. It searches the spanning trees T in $\mathcal{T}$ by their enumeration order in Algorithm 3, from the left-most path (i.e, T^*) to the right-most path. This means, we search the current T only after all the spanning tree T' to the left of T in $\mathcal{T}$ have been searched. Based on the current T, we partition $\mathcal{W}_c(g^u)$ into three disjoint groups: (1) $\mathcal{W}_T$; (2) $\mathcal{W}_L = \bigcup_{T'} \mathcal{W}_{T'}$ where $T' \in \mathcal{T}$ is to the left of T; (3) $\mathcal{W}_R = \bigcup_{T'} \mathcal{W}_{T'}$ where $T' \in \mathcal{T}$ is to the right of T. Since all T' to the left of T have been searched, we can record the sum P of the probabilities of all PWGs in $\mathcal{W}_L$ contained by q. Then we use an upper bound $UB(T,h)$ to bound the sum of probabilities of all PWGs in $\mathcal{W}_T \cup \mathcal{W}_R$. It is immediate that whenever $P + UB(T,h) < \theta$, we can terminate the search and safely prune g^u. We outline our algorithm in Algorithm 5 in a depth first search fashion.

Algorithm Sketch. Algorithm 5 synchronizes the extension of all partial mappings on $T[1,h-1]$ for the current T depth by depth. For each prefix $T[1,h-1]$ of T, we maintain all partial mapping on $T[1,h-1]$ on the node N representing T_{h-1} for share computation; that is, after we execute the drill down on T_h, we alternate the extension

```
Algorithm 5: Synchronize $(h, T, M, q, P, \theta, \mathcal{T}, f)$
Input : $h$: the current search depth; $T$: current spanning tree;
        $M$: the set of current partial mappings; $q$: the query;
        $P$: the current probability; $\theta$: the probability threshold;
        $\mathcal{T}$: the DFS Traversal Tree; $f$: a boolean flag;
1  $M_h :=$ SynchrMatch$(T_h, M, q)$ ;
2  if $P + UB(T, h) < \theta$ then
3      $f := true$;
4      return ;
5  if $h = |V(g^u)| - 1 \wedge M_h \neq \emptyset$ then
6      Inducing$(\mathcal{W}_T, q, M_h, P, \theta)$ ;
7      if $P \geq \theta$ then
8          return ;
9  else
10     do
11         Synchronize$(h+1, T, M_h, q, P, \theta, \mathcal{T}, f)$;
12         if $P \geq \theta \vee f$ then
13             return ;
14         if ReCheck$(T_{h+1})$ then
15             $T' :=$ getNext$(T, T_{h+1}, \mathcal{T})$ ;
16         else
17             $T' = nil$;
18         $T := T'$ ;
19     while $T \neq nil$ ;
```

to the spanning tree T' enumerated by replacing T_h with the all stored partial mapping on $T[1, h-1]$ (i.e., $T[1, h-1] = T'[1, h-1]$). In Algorithm 5, M is the set of all partial mappings $\mathcal{F}$ on $T[1, h-1]$ (initially $\emptyset$). f is a boolean flag and $f = true$ means g^u can be safely pruned. P is the sum of probabilities of all PWGs currently contained by q (initially 0). If $P \geq \theta$ after Synchronize terminates, g^u is a result of q. The other inputs are the same as in Algorithm 4.

inducing. The function SynchrMatch extends all partial mappings M on $T[1, h-1]$ to all partial mappings M_h on $T[1, h]$. If $h = 0$, SynchrMatch returns $C(T_0)$; otherwise, we extend each $\mathcal{F} \in M$ by the function NextCan in Algorithm 4. If $P + UB(T, h) < \theta$ (checked by Line 2) we set f to true immediately prune g^u. If all spanning edges in T have been mapped and $M_h \neq \emptyset$, Inducing extends the mappings in M_h to induce the matches of PWGs in $\mathcal{W}_T$. If $P \geq \theta$, the algorithm terminates and g^u is verified as a result.

drill-down. Algorithm 4 and 5 have similar drill-down phase. The only difference is that we check f to terminate earlier after executing drill-down.

Computing $UB(T, h)$. Line 2 in Algorithm 5 computes $UB(T, h)$ based on the current T and depth h. We use $UB(g^u, T, h)$ with the *edge exclusion constraint* as $UB(T, h)$; that is, we remove the sum of $P(g')|_{g^u}$ of all $g' \in \mathcal{W}_L$ from $UB(g^u, T, h)$ by using the edge exclusion set $T.R$ as follows. When computing $P_{E_i^-}$ at each level i ($1 \leq i \leq h$), we exclude all the edges in $T.R$ replaced before T_i from E_i^-. Clearly, for any spanning tree T'' containing $T[1, i-1]$ and at least one edge in $T.R$ replaced before T_i, T'' must be enumerated before T in Algorithm 3 and $\mathcal{W}_{T''} \subseteq \mathcal{W}_L$. Thus, the sum of $P(g')|_{g^u}$ for all $g' \in \mathcal{W}_{T''}$ is removed from $UB(g^u, T, h)$.

5 Performance Evaluation

We report the results of our performance evaluation in this section. The following filtering techniques are evaluated.

- PS: Filtering with the probabilistic signature $P_c(g^u)$ proposed in Section 3.1
- PSUR: Filtering with the upper bound $UB(g^u, T, h)$ proposed in Section 3.2 by using a random T of g^u.
- PSUG: Filtering with the upper bound $UB(g^u, T, h)$ proposed in Section 3.2 by generating T with Algorithm 2.
- PF: The feature based filtering with $P_c(g^u) - P_c(f \subseteq g^u)$ proposed in Section 3.1.
- PFU: The feature based filtering with the upper bound of $P_c(g^u) - P_c(f \subseteq g^u)$ proposed in Section 3.2
- PS+PF: Filtering with PS first and then PF.
- PSUG+PFU: Filtering with PSUG first and then PFU.

The following verification algorithms are evaluated.

- BASE: The Baseline algorithm proposed in Section 4.1.
- BASE$^+$: The Baseline$^+$ algorithm proposed in Section 4.2.
- SYNC: The Synchronize algorithm proposed in Section 4.3.

All algorithms are implemented in C++ and compiled with GNU GCC. All experiments are conducted on PCs running Debian Linux with Intel Xeon 2.40GHz CPU and 4GB memory . We evaluate all algorithms on both real and synthetic datasets.

Real Uncertain Dataset. Following [25], we use the STRING dataset as the real uncertain dataset. It is a popular benchmark for evaluating mining and querying techniques over uncertain graphs. We generate the vertex labels according to the COGs functions (Clusters of Orthologous Groups of proteins) [1], while we follow the previous works to ignore all edge labels. We extract the query graphs q and uncertain data graphs g^u from the STRING dataset by conforming the following **default** settings. (1) average number of vertices: $avg.|V(g^u)| = 20$, $avg.|V(q)| = 40$; (2) average vertex degree: $avg.deg(g^u) = 2.5$, $avg.deg(q) = 3$. When extracting the data graphs, we enforce the occurrence probability to fall in $[0.5, 1]$. General speaking, the higher the edge occurrence probability is, the more challenging our problem is (i.e., more candidates and answers). The *default query set* Q_R contains $2K$ query graphs. The *default database* D_R contains $10K$ uncertain data graphs. The occurrence probabilities all edges in q are ignored.

Besides, we also generate the following query sets and databases for evaluating the effect of varying query and database settings.

- **Varying** $avg.|V(q)|$: We extracts 4 sets of queries, $Q_{v=20}$, $Q_{v=40}$, $Q_{v=60}$ and $Q_{v=80}$. Each $Q_{v=i}$ has $2K$ queries with $avg.|V(q)| = i$ and $avg.deg(q) = 3$.

[1] http://www.ncbi.nlm.nih.gov/COG/

- **Varying** $avg.deg(q)$: We extracts 4 sets of queries, $Q_{d=2}$, $Q_{d=3}$, $Q_{d=4}$ and $Q_{d=5}$. Each $Q_{d=i}$ has $2K$ queries with $avg.deg(q) = i$ and $avg.|V(q)| = 40$.

- **Varying** $|D|$: We extracts 5 sets of data graphs, D_{2K}, D_{6K}, D_{10K}, D_{15K} and D_{20K}. Each D_{iK} has iK data graphs.

Synthetic Dataset. We use our graph generator to generate a synthetic dataset D_{SYN} of $10K$ graphs with an average size of 60 vertices and 240 edges. We follow uniform distribution to generate 25 vertex labels. The *default query set* Q_S has $2K$ queries q which are graphs randomly selected from D_{SYN}. The *default database* D_S has $10K$ data graphs g^u which are frequent subgraphs of the graphs in D_{SYN} with frequency ranging from $[0.05, 0.1]$. Note that $avg.|V(g^u)| = 20$ and $avg.deg(g^u) = 2.5$.

Besides, we also generate the following databases for evaluating the effect of varying database settings.

- **Varying** $avg.|V(g^u)|$: We generate 4 sets of data graphs, $D_{v=10}$, $D_{v=20}$, $D_{v=30}$ and $D_{v=40}$. Each $D_{v=i}$ has $10K$ data graphs and $avg.|V(g^u)| = i$, $avg.deg(g^u) = 2.5$.

- **Varying** $avg.deg(g^u)$: We generate 4 sets of data graphs, $D_{d=2}$, $D_{d=2.5}$, $D_{d=3}$ and $D_{d=3.5}$. Each $D_{d=i}$ has $10K$ data graphs and $avg.deg(g^u) = i$, $avg.|V(g^u)| = 20$.

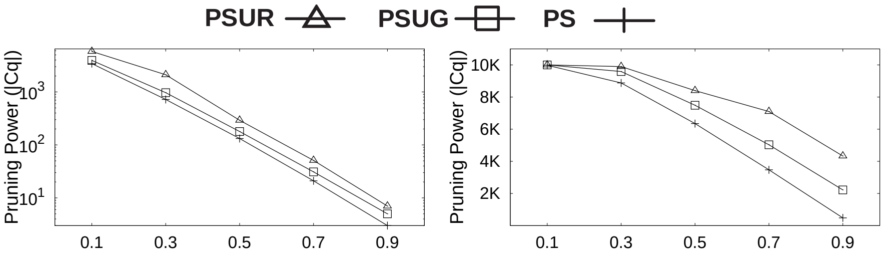
We use four different distributions to assign occurrence probabilities on the edges. These include random, normal, zipf [5], and power law [15] distributions. The **default** distribution is power law distribution. For all distributions, we force the occurrence probabilities to fall in $[0.5, 1]$. For random distribution, we randomly pick a value P for each edge. For normal distribution, the mean value is $\mu = 0.75$ and the standard deviation $\sigma = 0.1$. For zipf distribution, we first randomly pick 100 values P in $[0.5, 1]$ and the weight of any value P is $\frac{1}{R(P)}$ where $R(P)$ is the rank of P. The larger the value P, the lower the rank $R(P)$. For power law distribution, the exponent $k = 3$ and the weight of any value P is P^3. The average occurrence probabilities for all distributions are 0.76 (random), 0.79 (normal), 0.86 (power law) and 0.96 (zipf).

Default θ **Value.** The default probability threshold is $\theta = 0.5$.

5.1 Evaluating Pruning Power

The pruning power records the average number of candidates $|C_q|$ per query. Smaller $|C_q|$ indicates stronger pruning power.

Probabilistic Signature Based Filtering. Figure 5.1(a) and 5.1(b) report the pruning power of PSUR, PSUG and PS by varying θ on real and synthetic data graphs, respectively. Clearly, by generating the spanning tree T with the greedy Algorithm 2, PSUG can better tighten $UB(g^u, T, h)$ than PSUR and effectively approach PS. Thus, we exclude PSUR from further evaluation in the remainder of this paper.

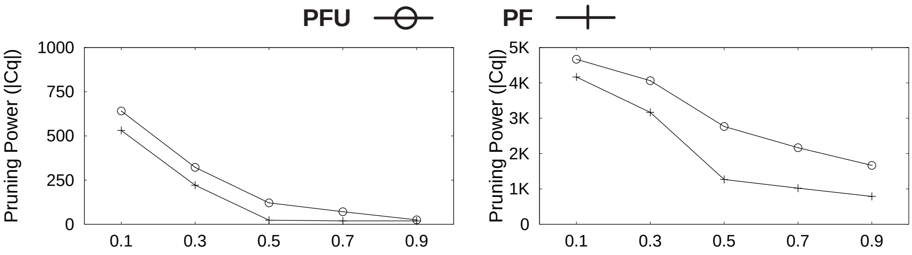


(a) Varying θ on D_R

(b) Varying θ on D_S

Figure 5.1: Probabilistic Signature Based Filtering

Feature Based Filtering. Figure 5.2(a) and 5.2(b) report the pruning power of PFU and PF by varying θ to query D_R and D_S, respectively. Note that PFU can significantly retain the pruning power of PF, while PFU performs better on D_R than on D_S. This is because D_R has more features (i.e., more pruning opportunities) than D_S due to the underlying distribution of biological substructures.

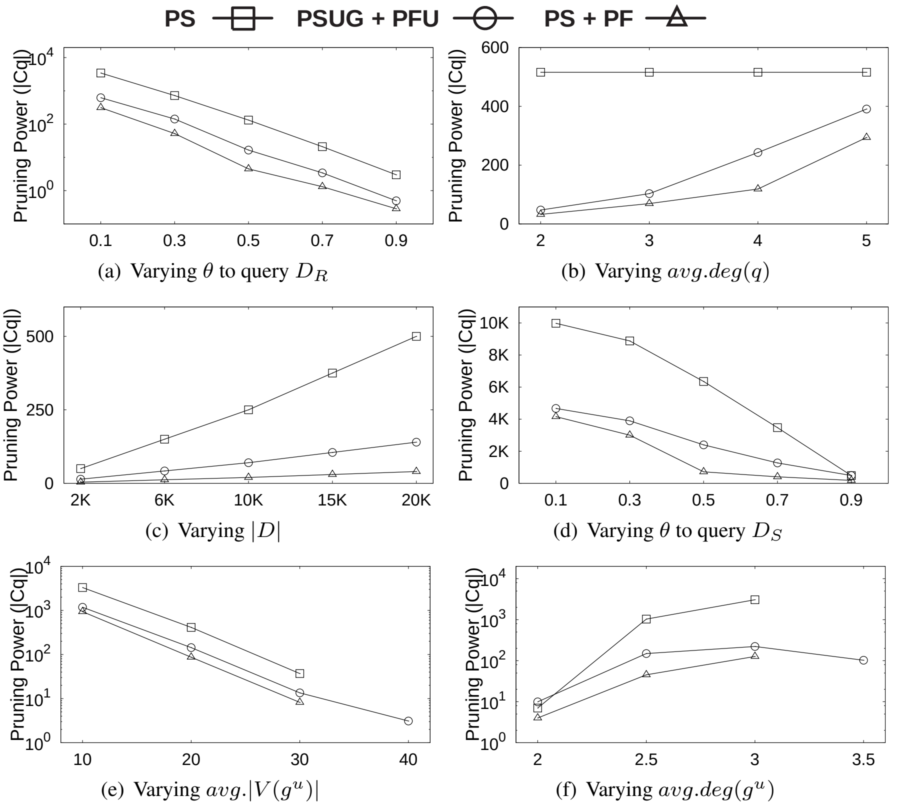


(a) Varying θ to query D_R

(b) Varying θ to query D_S

Figure 5.2: Feature Based Filtering

Total Pruning Power. Finally, we evaluate the pruning power of PS, PSUG+PFU and PS+PF. Figure 5.3(a)-5.3(c) report the results on real data graphs. PSUG+PFU can prune at least 95% of the false results captured by PS+PF and significantly improves PS with the pruning power of features. In Figure 5.3(b), the candidate numbers of all techniques increase with $avg.deg(q)$ as denser query graphs may contain more PWGs and thus have more answers. In Figure 5.3(c), the candidate numbers of all techniques increase proportionally with $|D|$. The intuition is the number of answers to each queries increase proportionally due to the similar distributions of biological substructures in all databases. Figure 5.3(d)-5.3(f) report the results on synthetic data graphs. Interestingly, when $avg.deg(g^u)$ increase, the pruning power of PSUG+PFU initially decreases but finally improves. The reason is that PSUG+PFU is initially dominated by the effect of PSUG on sparse data graphs but later dominated by the effect of PFU on dense data graphs. This shows that the pruning power of PFU and PSUG are complementary to each other. Besides, combining PFU and PSUG can better tighten the upper bounds, which gives very close pruning power as PS+PF. Note that we do not report the results of PS+PF and PS on $avg.|V(g^u)| = 40$ and $avg.deg(g^u) = 3.5$ as the index construction fail to terminate after 2 days.



(a) Varying θ to query D_R

(b) Varying $avg.deg(q)$

(c) Varying $|D|$

(d) Varying θ to query D_S

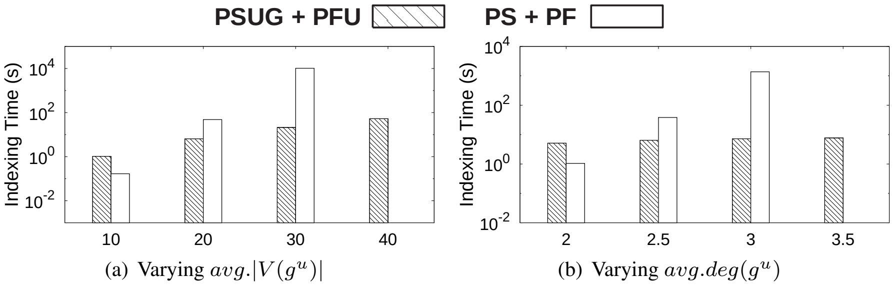
(e) Varying $avg.|V(g^u)|$

(f) Varying $avg.deg(g^u)$

Figure 5.3: Total Pruning Power

5.2 Evaluating Indexing Time

We next evaluate the indexing time of PSUG+PFU and PS+PF. Figure 5.4(a) and 5.4(b) report the results by varying $avg.|V(g^u)|$ and $avg.deg(g^u)$ on synthetic data graphs. As reported earlier, the index construction of PS + PF fails when $avg.deg(g^u) = 3.5$ and $avg.|V(g^u)| = 40$. Even excluding the failed cases, PSUG+PFU is up to 2 orders of magnitude more efficient on indexing time against PS+PF due to the $\#$P-Complete nature of $P_c(g^u)$ and $P_c(g^u) - P_c(f \subseteq g^u)$. Our experiments show that the index construction of PSUG+PFU runs efficiently and is more practical when dealing with larger and denser data graphs.



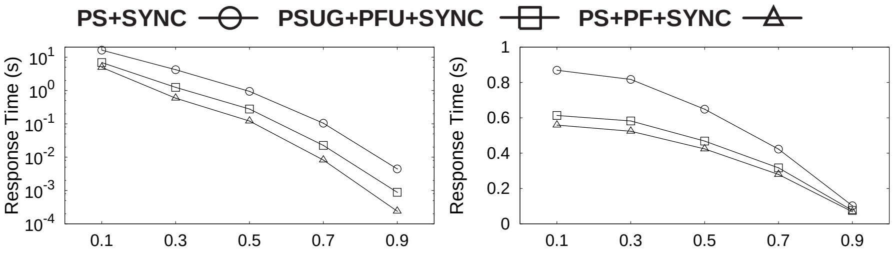
(a) Varying $avg.|V(g^u)|$

(b) Varying $avg.deg(g^u)$

Figure 5.4: Indexing Time

5.3 Evaluating Efficiency

Total Response Time. We equip PS, PSUG+PFU and PS+PF with our Synchronize algorithm to evaluate the total response time. Figure 5.5(a) and 5.5(b) report the results on real and synthetic data graphs, respectively. Note that PSUG+PFU outperforms PS by one order of magnitude on real data and is only slightly slower than PS+PF due to close pruning power. The results show that by combining our two proposed upper bounds, the query processing can be conducted very efficiently.

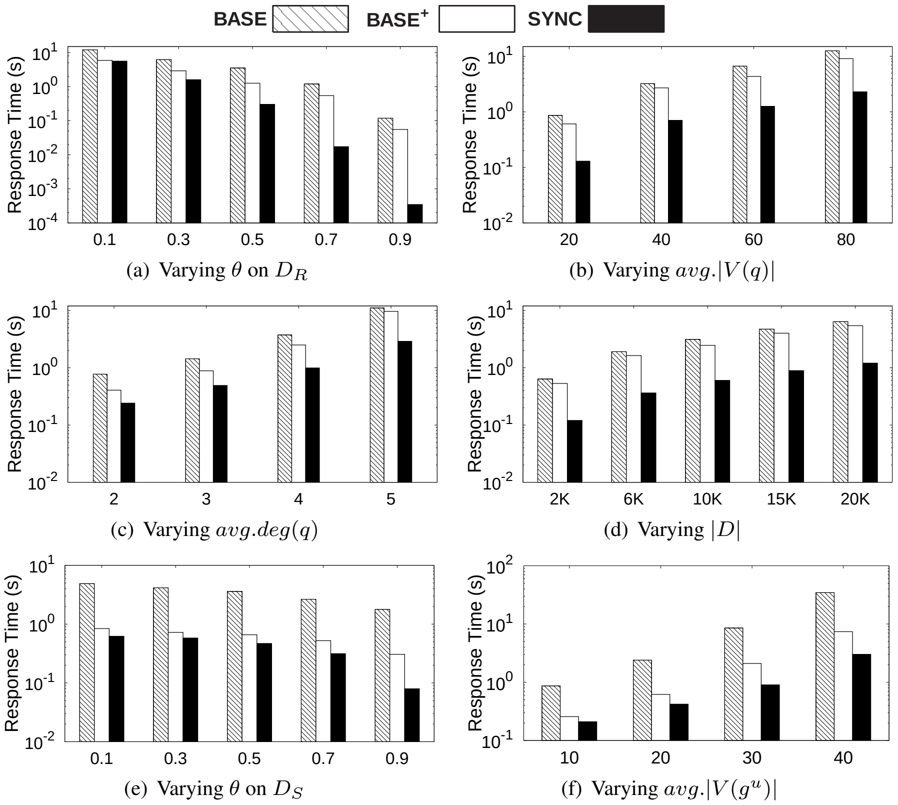


(a) Varying θ on D_R

(b) Varying θ on D_S

Figure 5.5: Total Response Time

Verification Time. We also evaluate the verification time of BASE, BASE^{+} and SYNC. We first filter false results with PSUG+PFU and then run our verification algorithms. Figure 5.6(a)-5.6(d) and Figure 5.6(e)-5.6(f) report the results on real and synthetic data graphs, respectively. BASE^{+} outperforms BASE for efficiently validating subgraphs of already contained PWGs. SYNC improves BASE^{+} by up to one order of magnitude. This demonstrates the effectiveness of our pruning technique employed by SYNC.



(a) Varying θ on D_R

(b) Varying $avg.|V(q)|$

(c) Varying $avg.deg(q)$

(d) Varying $|D|$

(e) Varying θ on D_S

(f) Varying $avg.|V(g^u)|$

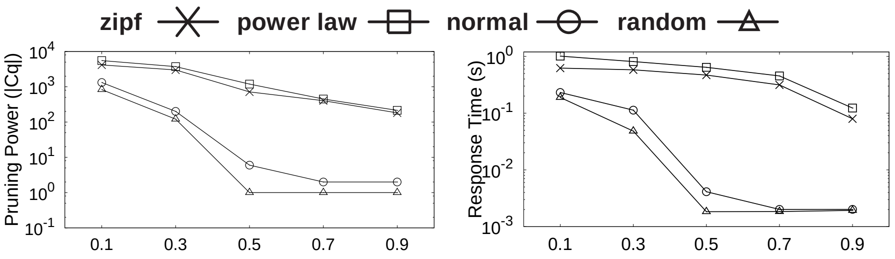
Figure 5.6: Verification Time

Filtering Time. We next evaluate the filtering time of PSUG+PFU by varying $avg.|V(q)|$ on real data. Table 5.1 reports the filtering time and the total response time by using SYNC to verify the candidates given by PSUG+PFU. Clearly, the total response time is dominated by the verification time, while our filtering algorithm executes very efficiently.

Table 5.1: Filtering Time

$avg.\|V(q)\|$	20	40	60	80
Filtering Time (millisecond)	1. 22	1. 25	1. 29	1. 36
Response Time (millisecond)	129	703	1261	2296

Varying Probability Distribution. We finally query D_S by varying probability distribution of the edges. Figure 5.7(a) and 5.7(b) report the total pruning power and response time, respectively. Due to higher average occurrence probabilities, D_S under zipf and power law distributions has more candidates than D_S under normal and random distributions. This aligns with the results in Figure 5.7(b) as D_S with zipf requests the longest response time. Clearly, even on high average occurrence probability, our proposed techniques is still applicable and efficient.



(a) Total Pruning Power

(b) Total Response Time

Figure 5.7: Varying Probability Distributions

6 Conclusions

In this paper, we investigate the problem of efficiently conducting probabilistic supergraph search. We propose two novel and effective pruning rules to exclude non-promising data graphs before conducting the costly verification. Seeing the hardness (i.e., #P-Complete) of computing $P_c(g^u)$ and $P_c(g^u) - P_c(f \subseteq g^u)$, we propose two PTIME based upper-bounds, respectively, to significantly reduce the pre-computation costs, while effectively retaining the pruning power. Our verification algorithms $\mathsf{Baseline}^+$ and $\mathsf{Synchronize}$ can efficiently test PWGs by sharing computation and excluding non-promising candidates as early as possible. The experimental results demonstrate that our proposed upper bounds have close pruning power to those of $P_c(g^u)$ and $P_c(g^u) - P_c(f \subseteq g^u)$, respectively, and our verification algorithms are very efficient in practice.

Bibliography

[1] S. Asthana, O. D. King, F. D. Gibbons, and F. P. Roth. Predicting protein complex membership using probabilistic network reliability. *Genome Res*, 14:1170–1175, 2004.

[2] J. S. Bader, A. Chaudhuri, J. M. Rothberg, and J. Chant. Gaining confidence in high-throughput protein interaction networks. *Nature Biotechnology*, 22(1):78–85, 2004.

[3] R. A. Bailey. *Surveys in Combinatorics*. Cambridge University Press, 1997.

[4] G. Blin, F. Sikora, and S. Vialette. Querying graphs in protein-protein interactions networks using feedback vertex set. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 7(4):628–635, 2010.

[5] H. S. C. D. Manning. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

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

[7] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. In *SIGMOD Conference*, pages 857–872, 2007.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[9] E. T. George Kollios, Michalis Potamias. Clustering large probabilistic graphs. *IEEE Transaction on Knowledge Discovery and Engineering*, 2012.

[10] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[11] P. Hell and J. Nesetril. *Graphs and Homomorphisms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, 2004.

[12] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*, pages 549–552, 2003.

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

[14] A. Mohajeri and D. Pouria. Evaluating the nature of chemical bonds based on probabilistic models. *International Journal of Modern Physics C : Computational Physics and Physical Computation*, 11(18):1795–1809, 2007.

[15] M. E. J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary Physics*, 46:323–351, 2005.

[16] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *KDD*, pages 647–652, 2004.

[17] O. Papapetrou, E. Ioannou, and D. Skoutas. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *EDBT*, pages 355–366, 2011.

[18] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. k-nearest neighbors in uncertain graphs. *Proc. VLDB Endow.*, 3(1-2):997–1008, 2010.

[19] H. Shang, X. Lin, Y. Zhang, J. X. Yu, and W. Wang. Connected substructure similarity search. In *SIGMOD*, pages 903–914, 2010.

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

[21] D. Shasha, J. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *PODS*, pages 39–52, 2002.

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

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

[24] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD Conference*, pages 335–346, 2004.

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

[26] S. Zhang, M. Hu, and J. Yang. Treepi: A novel graph indexing method. In *ICDE*, pages 966–975, 2007.

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

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

[29] S. Zhang, J. Yang, and W. Jin. Sapper: Subgraph indexing and approximate matching in large graphs. In *VLDB*, 2010.

[30] P. Zhao and J. Han. On graph query optimization in large networks. *PVLDB*, 3(1):340–351, 2010.

[31] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: Tree + delta >= graph. In *VLDB*, pages 938–949, 2007.

[32] M. T. Zhi Liang, Meng Xu and L. Niu. Comparison of protein interaction networks reveals species conservation and divergence. *BMC Bioinformatics*, 7(457):1–14, 2006.

[33] G. Zhu, X. Lin, K. Zhu, W. Zhang, and J. X. Yu. Treespan: efficiently computing similarity all-matching. In *SIGMOD Conference*, pages 529–540, 2012.

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