Efficient Discovery of Frequent Subgraph Patterns in Uncertain Graph Databases

Odysseas Papapetrou
L3S Research Center
Hannover, Germany
papapetrou@L3S.de

Ekaterini Ioannou
L3S Research Center
Hannover, Germany
ioannou@L3S.de

Dimitrios Skoutas
L3S Research Center
Hannover, Germany
skoutas@L3S.de

ABSTRACT

Mining frequent subgraph patterns in graph databases is a challenging and important problem with applications in several domains. Recently, there is a growing interest in generalizing the problem to *uncertain graphs*, which can model the inherent uncertainty in the data of many applications. The main difficulty in solving this problem results from the large number of candidate subgraph patterns to be examined and the large number of subgraph isomorphism tests required to find the graphs that contain a given pattern. The latter becomes even more challenging, when dealing with uncertain graphs. In this paper, we propose a method that uses an index of the uncertain graph database to reduce the number of comparisons needed to find frequent subgraph patterns. The proposed algorithm relies on the apriori property for enumerating candidate subgraph patterns efficiently. Then, the index is used to reduce the number of comparisons required for computing the expected support of each candidate pattern. It also enables additional optimizations with respect to scheduling and early termination, that further increase the efficiency of the method. The evaluation of our approach on three real-world datasets as well as on synthetic uncertain graph databases demonstrates the significant cost savings with respect to the state-of-the-art approach.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

General Terms

Theory, Algorithms

1. INTRODUCTION

Graphs constitute a generic data model with wide applicability in numerous domains and applications. Consequently, mining frequent subgraph patterns in graph databases has become an important method for obtaining interesting insights and discovering useful knowledge from the data, for example in bioinformatics, where graphs are used to represent protein interactions. Given a graph database D, the *support* of a graph G is defined as the portion of graphs in D that contain G as a subgraph. A frequent subgraph pattern is then a graph with support above a minimum specified threshold *minSup*. Discovering frequent subgraph patterns in a graph database is a challenging task due to two main reasons. First, any subgraph of a graph in the database constitutes a candidate frequent subgraph pattern. This results in an extremely large number of candidates that have to be enumerated and examined in an efficient manner. Existing approaches typically exploit the *apriori* property [1] to more efficiently enumerate candidate patterns. According to it, if a subgraph is frequent, then all of its subgraphs are also frequent. This allows for a systematic way to enumerate subgraph patterns. Still, there remains a second challenge, that is, computing the support of a candidate pattern in order to decide whether it is frequent. This requires testing for subgraph isomorphism with the graphs contained in the database, which is NP-complete.

Recently, there is an increasing need for the management of uncertain graphs, necessitated by the inherent uncertainty in the data of many applications. Uncertain graphs are a generalization of exact graphs, in which each edge is associated with a probability that indicates the belief we have that this edge exists. This additional expressiveness makes uncertain graphs a useful data model in numerous applications. For instance, uncertain graphs are often used in bioinformatics to model interactions between proteins, which are derived by experiments that are inevitably noisy and error-prone [2]. Edge probabilities are then used to express this uncertainty. Graphs are also used very often to represent communities of users in social networks, where probabilities can be assigned to edges to model the belief in the existence of the link or the degree of influence between two entities [19, 17]. Furthermore, in communication networks or road networks, edge probabilities are used to quantify the connectivity between nodes [7], or to take traffic uncertainty into consideration [12], respectively.

The problem of frequent subgraph pattern mining becomes even more challenging in the case of uncertain graphs. A k-edge uncertain graph implies a set of 2^k exact graphs, which are derived by sampling the edges of the uncertain graph according to their probabilities. Hence, for each uncertain graph, there are now 2^k subgraph isomorphism tests required. In particular, the significance of a subgraph pattern is now measured by its *expected support*, since its support is now a random variable over the support values of the pattern in all the exact graph databases that can be sampled from the initial uncertain graph database.

Existing approaches [16, 18, 8, 28, 21, 13, 25] solve the frequent subgraph pattern mining problem for exact graphs by performing two main operations: (a) generating candidate patterns to be examined, and (b) testing for subgraph isomorphism to determine which graphs in the database contain a given candidate pattern. An extension to the case of uncertain graphs has been recently presented in [32]. It relies on the same techniques as the previous methods for enumerating candidate patterns and focuses on the second task, where the basic idea is to trade off accuracy with efficiency when computing the expected support of a pattern. In particular, it proposes an approximate algorithm for computing the expected support of a subgraph pattern by transforming the problem to an instance of the DNF counting problem. However, although this can reduce the cost of the computation for a single uncertain graph, the overall cost still remains prohibitively high, even when dealing with moderate size databases containing a few hundreds of graphs.

In this paper, we propose an efficient algorithm called UGRAP to address this problem. UGRAP relies on an index of the uncertain graph database to significantly reduce the amount of computations required to determine the support of a candidate pattern. The index comprises two structures. The first is an inverted index on graph edges enhanced with edge probabilities, and the second is a structure providing summarized information regarding connectivity of graph nodes up to a specified path length. Similar to previous approaches, the algorithm efficiently enumerates candidate patterns based on the apriori property. Then, when the support of a candidate subgraph pattern needs to be computed, the index is used to identify a subset of the uncertain graphs in the database that may contain this pattern, thus avoiding a significant number of unnecessary subgraph isomorphism tests. In addition, we also propose optimizations to further increase the efficiency of the method, allowing for early termination and more effective scheduling of the graphs to be examined. Our extensive experimental evaluation on three real-world data sets from the bioinformatics domain, as well as on a synthetic uncertain graph database, demonstrates the significant reduction of the computational cost when compared to the state-of-the-art method for the same problem.

Summarizing, our main contributions are as follows:

- We introduce an index of an uncertain graph database comprising information on graph edges along with their probabilities and a summary of connectivity information between graph nodes.
- We propose an algorithm that uses the aforementioned index to efficiently solve the problem of frequent subgraph pattern mining in uncertain graphs, by pruning the search space when computing the expected support of candidate patterns.
- We further improve the efficiency of the algorithm by proposing additional optimizations for early termination and effective scheduling of graph comparisons.
- We demonstrate the efficiency of our method by conducting a comprehensive experimental evaluation on large real-world and synthetic datasets, showing that it significantly outperforms existing state-of-the-art solutions to the problem.

The rest of the paper is organized as follows. The next section presents related work. The data model and a formal problem definition are introduced in Section 3. Section 4 introduces the UGRAP index, while Section 5 explains the frequent subgraph pattern mining algorithm based on this index. Section 6 presents and discusses the results of our experimental evaluation on real and synthetic datasets. Finally, Section 7 concludes the paper.

2. RELATED WORK

In this section, we present related work considering both the case of exact and uncertain graphs.

2.1 Exact Graphs

Given the significance of the problem, a lot of research efforts have focused on mining frequent subgraph patterns in exact graphs, with first approaches dating back to the 1990's [26]. Existing methods are typically classified in two categories: *apriori-based* and *pattern growth*.

The approaches of the first category follow the main idea behind the Apriori algorithm [1] for mining frequent itemsets. More specifically, they rely on the *apriori property*, according to which all the subpatterns of a frequent subgraph pattern are also frequent. Thus, to enumerate candidate patterns, they apply breadth-first search to generate subgraphs of size $(k + 1)$, by joining two subgraphs of the previous level.

The main representatives of this category are AGM [16], FSG [18] and PM [8]. They mainly differ on the basic building block used to enumerate candidate patterns, which can be nodes [16], edges [18], or edge-disjoint paths [8]. AGM starts the search by examining graphs comprising a single vertex, and then it proceeds by generating larger candidates adding one extra vertex at each subsequent step. FSG uses edges, instead of vertices, as the primary building block for candidate generation. It limits the class of the frequent subgraphs to connected graphs and introduces several heuristics to increase the efficiency of computing the support of a pattern, using graph vertex invariants, such as the degree of each vertex in the graph. It also improves the efficiency of the candidate pattern generation by introducing the transaction ID method. PM also follows breadth-first enumeration for generating the candidate patterns; however, in contrast to the previous approaches which employ single vertices or edges as basic building blocks for pattern generation, it utilizes edge-disjoint paths. This reduces the required iterations, while it is proved that completeness is maintained.

To avoid the costly breadth-first based candidate pattern generation, which incurs heavy memory requirements, the methods in the second category adopt depth-first search, where patterns are grown directly from a single graph instead of joining two previous subgraphs. The main representative of this category is gSpan [28], which also relies on canonical labeling like previous approaches, but it uses a tree representation instead of an adjacency matrix as a coding scheme for the graph. Based on the assigned codes, candidate patterns are organized lexicographically in a tree hierarchy, which is then searched in a depth-first manner. In the same direction, GASTON [21] splits the discovery process into several phases to increase efficiency by first searching for frequent paths, then for frequent free trees, and finally for cyclic graphs. Efficiency is improved since these classes of structures are contained in each other. The basic idea is to store and reuse the embeddings instead of performing subgraph isomorphism tests. However, this has high space requirements and does not scale well to large graph databases.

Another approach is FFSM [13], which proposes a vertical search scheme within an algebraic graph framework. Relying on a graph canonical form, it introduces two new operations, FFSM-Join and FFSM-Extension, to improve the efficiency of pattern enumeration. An embedding set for each frequent subgraph is also maintained to avoid expensive subgraph isomorphism tests. Furthermore, an adjacency index structure, called ADI, is proposed in [25] to deal with the cases in which the graph database is too large to fit in main memory. It is also shown how the gSpan algorithm [28] can be adapted to use ADI.

Finally, to reduce the size of the output, more recent works have focused on mining only subgraph patterns that are closed [29], maximal [14], significant [27] or representative [31], or on summarizing subgraph patterns [20].

2.2 Uncertain Graphs

Recently, there has been a growing interest in using uncertain graphs as a data model in applications that need to deal with uncertainty. Thus, various problems for mining uncertain graphs have emerged. The problem of finding reliable subgraphs in uncertain graphs is studied in [11]. Given a graph that is subject to random edge failures, the goal is to find and remove a number of edges so that the probability of connecting a set of selected nodes in the remaining subgraph is maximized. Three novel types of probabilistic path queries have been defined in [12] for uncertain graphs representing road networks, where edge probabilities capture the uncertainty in traffic conditions. Also, both exact and approximation algorithms are introduced to answer such queries. A generalization of k-Nearest Neighbor queries in uncertain graphs is presented in [22], where a framework is proposed considering alternative ways to define the distance between nodes taking edge probabilities into account. All these works clearly show the increasing need and interest in mining uncertain graphs.

However, to the best of our knowledge, up to now only one work has dealt with the problem of frequent subgraph pattern mining in uncertain graphs [32]. The proposed method is an approximation algorithm, called MUSE, which allows for a tradeoff between accuracy and efficiency when computing the expected support of candidate subgraph patterns. In particular, given a support threshold *minSup* and a relative error tolerance $\varepsilon \in [0, 1]$, the algorithm returns all subgraph patterns with expected support at least *minSup*, allowing also for some false positives with expected support in the range $[(1 - \varepsilon)$ *minSup*, *minSup*$]$. Similar to corresponding methods for exact graphs, the solution addresses two main subtasks: (a) a method for enumerating candidate patterns, and (b) a method to compute the expected support of a pattern. Regarding the first task, the method proposed in gSpan [28] is adopted to construct a search tree of subgraph patterns. For the second task, two algorithms are proposed, an exact one for small instances of the problem (e.g., graphs with up to 30 edges) and an approximate one for larger instances. The main idea in both algorithms is to transform the problem to an instance of the DNF counting problem [24].

Although this algorithm makes it possible to approximate the expected support of a candidate pattern for an uncertain graph with a large number of edges, the computational cost is still quite high, and therefore the method does not scale well, even for moderate size databases with up to a few hundreds of uncertain graphs. In our approach, we remove this limitation, by constructing an index of the uncertain graph database, which significantly prunes the search space and enables for additional optimizations based on early termination and efficient scheduling to avoid the expensive subgraph isomorphism tests.

3. DATA MODEL & PROBLEM DEFINITION

In this section, we formally define uncertain graphs and the problem of frequent subgraph pattern mining in uncertain graph databases. For clarity of the presentation, we first introduce the problem of frequent subgraph pattern mining in exact graphs, and then we explain how it is generalized in uncertain graphs. The data model and definitions used in this paper are in line with previous approaches for mining frequent subgraph patterns in both exact and uncertain graphs (e.g., [28, 32]).

Definition 1 (Exact Graph). *An* exact graph *is a tuple $G = (V, E, \Sigma, L)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, Σ is a set of labels, and $L : V \cup E \rightarrow \Sigma$ is a function assigning labels to vertices and edges.*

The vertex set of a graph G is denoted by $V(G)$ and the edge set by $E(G)$. The size of a graph G, denoted as $|G|$, is defined by the number of edges it contains, i.e., $|E(G)|$. For simplicity, we assume that the graph is undirected, since this a more typical scenario in frequent subgraph pattern mining, e.g., in bioinformatics; however, it is straightforward to extend the proposed method in the case of directed graphs.

Definition 2 (Subgraph Isomorphism). *Given two exact graphs, $G = (V, E, \Sigma, L)$ and $G' = (V', E', \Sigma', L')$, a* subgraph isomorphism *from graph G to graph G' is an injective function $f : V \rightarrow V'$ such that:*

1. $\forall\, u \in V$, $f(u) \in V'$ and $L(u) = L'(f(u))$, and

2. $\forall\, (u, v) \in E$, $(f(u), f(v)) \in E'$ and $L(u, v) = L'(f(u), f(v))$.

If such a function f exists, then G is subgraph isomorphic to G', denoted as $G \sqsubseteq G'$. We also say that G' contains G. Moreover, the subgraph G'' of G' with vertex set $V'' = \{f(u) \mid u \in V\}$ and edge set $E'' = \{(f(u), f(v)) \mid (u, v) \in E\}$ is called the *embedding* of G in G' under f.

Based on the above, we can define the *support* or *frequency* of a subgraph pattern S in an exact graph database D as the portion of graphs in D to which S is subgraph isomorphic. Notice that we only consider connected graphs as subgraph patterns. Furthermore, a subgraph pattern is considered to be *frequent* in D, if its support exceeds a pre-defined threshold *minSup*. Formally, we define frequent subgraph patterns in exact graph databases as follows.

Definition 3 (Support). *Given a subgraph pattern S and an exact graph database D, the* support *of S in D is defined by*

$$sup(S, D) = \frac{|\{G \in D \mid S \sqsubseteq G\}|}{|D|} \tag{1}$$

If $sup(S, D) \geq minSup$, where minSup is a given support threshold within $[0, 1]$*, then S is a* frequent *subgraph pattern in D.*

In the following, we show how the above concepts generalize in the case of uncertain graphs. *Uncertain* or *probabilistic* graphs generalize exact graphs by associating to each edge a probability that it exists. Formally:

Definition 4 (Uncertain Graph). *An* uncertain *graph is a tuple $G^p = (V, E, \Sigma, L, P)$, where (V, E, Σ, L) is an exact graph defined as previously and $P : E \rightarrow (0, 1]$ is a function assigning to each edge a probability that it exists.*

An uncertain graph G^p implies a set of $2^{|E|}$ possible exact graphs. These are sampled from G^p according to the probabilities assigned by the function P. As in previous approaches, we assume independence among edges, which is a realistic assumption in many real-world applications. The probability of an exact graph G being implied by G^p, denoted as $G^p \Rightarrow G$, is computed based on the probability of each edge of G^p being included or excluded from G:

$$P(G^p \Rightarrow G) = \prod_{e \,\in\, E(G)} P(e) \prod_{e \,\in\, E(G^p) \setminus E(G)} (1 - P(e)) \tag{2}$$

Consequently, an uncertain graph database D^P implies a set of $\prod_{i=1}^{|D^P|} 2^{|E(G_i^p)|}$ exact graph databases. Assuming also independence among the uncertain graphs in the database, the probability of an exact graph database D being implied by D^P is:

$$P(D^P \Rightarrow D) = \prod_{i=1}^{|D^P|} P(G_i^p \Rightarrow G_i) \tag{3}$$

where G_i^p and G_i are the i-th graphs in D^P and D, respectively.

In an uncertain graph database D^P, the support of a subgraph pattern S is based on its support in the implied exact graph databases, taking also into consideration the corresponding probabilities of these databases. In particular, the support in this case is a random variable with probability distribution defined by:

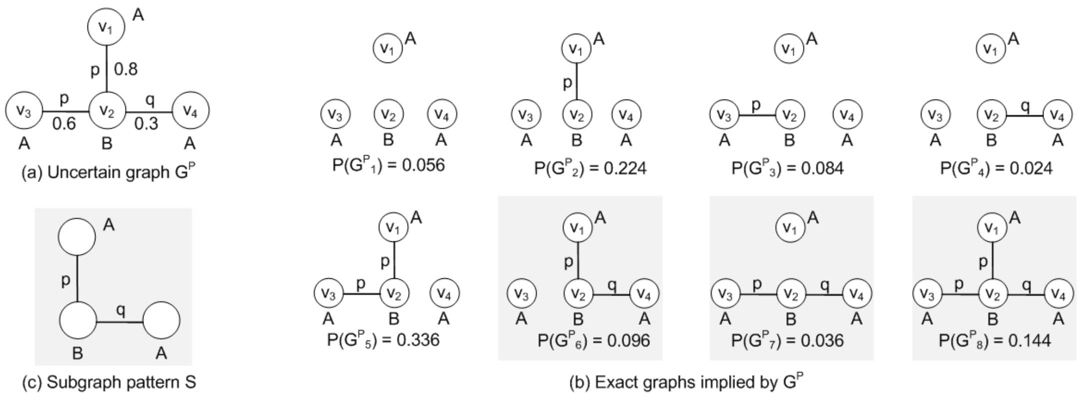

Figure 1: An illustrative example showing (a) an uncertain graph, (b) its implied exact graphs and (c) a subgraph pattern.

$$P(s_i) = \sum_{\{D \mid D^P \Rightarrow D \text{ and } sup(S,D) = s_i\}} P(D^P \Rightarrow D) \qquad (4)$$

Therefore, to define frequent subgraph patterns in uncertain graph databases, we use as measure the *expected support*, which is defined as follows.

Definition 5 (Expected Support). *The* expected support *of a subgraph pattern S in an uncertain graph database D^P is defined by:*

$$esup(S, D^P) = \sum_{\{D \mid D^P \Rightarrow D\}} P(D^P \Rightarrow D) \cdot sup(S, D) \qquad (5)$$

We can now formally define the problem of frequent subgraph pattern mining in uncertain graph databases.

Problem Definition. Given an uncertain graph database D^P and a minimum support threshold *minSup*, return all the subgraph patterns S with expected support greater than or equal to *minSup*, i.e., $esup(S, D^P) \geq minSup$.

Example 1. *An illustrative example is presented in Figure 1, comprising an uncertain graph G^P and a candidate subgraph pattern S. The labels of the vertices and edges denote their type, e.g., category of a protein or type of protein interaction. The figure also depicts the* 8 *exact graphs implied by G^P, together with their probabilities, computed according to Equation 2. As shown, the subgraph pattern S is contained in the implied graphs G^P_6, G^P_7 and G^P_8. Therefore, according to Equation 5, the expected support of S in G^P is* 0.276.

A straightforward algorithm for solving this problem works as follows: (a) enumerate all candidate subgraph patterns; (b) for each generated candidate pattern, and for each uncertain graph in the database, generate all the implied exact graphs and compute the expected support of the pattern. The cost of the first step is the same as in the case of exact graphs. Hence, one of the existing strategies, based on the apriori property, can be applied for enumerating candidate patterns more efficiently. In our method, we use the approach of gSpan [28]. However, the cost of the second step is significantly increased compared to the corresponding one for the case of exact graph databases. Recall that, each uncertain graph with k edges implies a set of 2^k exact graphs. Therefore, for each pair of a candidate pattern and a graph, it requires 2^k subgraph isomorphism tests when the graph is uncertain instead of a single one when the graph is exact. A first approach for dealing with this problem is proposed in [32], which replaces this computation with a more efficient but approximate algorithm that can estimate the expected support of a subgraph pattern in an uncertain graph, when dealing with large graphs (i.e., above 30 edges). However, even with this approximation, the cost remains prohibitively high even for moderate size databases (e.g., above 100 graphs). Therefore, reducing the uncertain graphs to be considered to only those that may contribute to making a pattern frequent, and especially avoiding large graphs in the computation, becomes crucial. In the next sections, we propose a solution to this problem, using an index and a summary of the uncertain graph database, with additional optimizations for early termination and effective scheduling of graph comparisons.

4. THE UGRAP INDEX

As explained above, our goal is to prune the search space when computing the expected support of candidate subgraph patterns, by limiting the number of uncertain graphs that need to be examined for containment. For this purpose, we construct an index of the uncertain graph database, containing graph edges and their probabilities. Furthermore, to achieve better pruning, taking into consideration the structure of each candidate pattern and each examined graph, we also construct a structure containing connectivity information between graph nodes. This information is summarized in order to reduce memory requirements when dealing with large databases and large graphs, especially in the case of dense graphs. In this section, we present how the UGRAP index is constructed and maintained.

4.1 Edge Index

The first component of the UGRAP index, denoted with I^E, is an inverted index on graph edges extended with information on edge probabilities in order to take uncertainty of edges into account. More specifically, the structure I^E is a map where:

- each key is a label triple of the form $t = (L_u,\ L_v,\ L_e)$, representing graph edges, and
- the value of each key is a list containing the identifiers of the graphs in which these edges appear, as well as the corresponding occurrence probability.

An edge (u, v) contained in an uncertain graph in the database is mapped to the key $T(u, v) = (L(u),\ L(v),\ L(u, v))$. The value of a key t is then a list of pairs of the form $(G^P, p^{G^P}_t)$, where $p^{G^P}_t$ is the probability that the graph G^P contains at least one edge e mapped to the key t. Only those graphs with non zero probability are stored in the index. Given the independence assumption between edges, this probability is computed by:

$$p_t^{G^P} = 1 - \prod_{e \in G^P \wedge T(e)=t} (1 - P(e)) \tag{6}$$

where the product denotes the probability that no edge mapped to t exists. Formally, the edge index I^E can be defined as follows.

Definition 6 (Edge Index). *Given an uncertain graph database D^P, the edge index I^E is a structure that returns, for any given triple $t = (L_u,\ L_v,\ L_e)$, a list of all the pairs $(G^P, p_t^{G^P})$, where G^P is an uncertain graph in D^P containing an edge (u, v) having probability $p_t^{G^P} > 0$, such that $L(u) = L_u$, $L(v) = L_v$, and $L(u, v) = L_e$.*

Constructing the I^E structure is straightforward. Each uncertain graph in the database can be processed independently, parsing its edges to identify the list of keys and their probabilities, using Equation 6. The results are then merged to create the map described above. The process is detailed in Algorithm 1.

Updating the index when an uncertain graph is added or removed from the database can be performed incrementally. The keys for this graph are computed and the corresponding entries in the index are updated accordingly, by appending or removing the corresponding item from the list of each of these keys. If the key is not already contained in the index, a new entry is created (or the entry is removed if the list of the key becomes empty). Finally, if an existing uncertain graph is updated, then the probabilities of all the affected keys need to be updated accordingly (which may also result in removing or adding keys).

Notice that, although more complex index structures have been proposed for querying graph databases, which aim at avoiding expensive subgraph isomorphism tests [5, 9, 30], these structures are not suitable for our problem for two reasons. First, they target exact graphs; hence, their adaptation to uncertain graph databases is an open issue. Second, and most importantly, more advanced index structures, such as the ones proposed in [5, 30], require first to compute the frequent subgraphs in the database, which are then used as features for the index. Instead, since our goal is to find such frequent subgraph patterns, the index can only rely on simpler features. As shown in Section 6, our index requires negligible memory and computational resources to be built, even for large uncertain graph databases.

Example 2. *The edge index I^E for a database containing only the uncertain graph illustrated in Figure 1 would contain two keys, (A, B, p) and (A, B, q), pointing to the lists $\{(G^P, 0.92)\}$ and $\{(G^P, 0.3)\}$, respectively.*

4.2 Connectivity Index

The second component of the UGRAP index, denoted by I^C, is a structure containing summarized information regarding connectivity of graph nodes. This additional structural information is useful when deciding which uncertain graphs may contain a candidate subgraph pattern with non-zero probability.

Intuitively, the purpose of this structure is to extend the edge index allowing paths of length $\ell > 1$. In particular, I^C provides information on whether there exists a path of length ℓ (for values of ℓ up to a maximum length ℓ_{max}) between two vertices u and v of a graph G^P with labels $L(u)$ and $L(v)$, respectively. Notice that, unlike the case of single edges, the independence assumption does not hold between paths, since two paths may contain common edges. Therefore, the probability that an uncertain graph G^P contains a path of length ℓ between two vertices with labels $L(u)$ and $L(v)$ cannot be computed in a straightforward way, i.e., similar to Equation 6 for edges. Instead, it requires applying the inclusion-exclusion principle, which involves finding all the possible paths between all pairs

Algorithm 1 Construction of the Edge Index I^E

```
Input  : An uncertain graph database $D^P$
Output : The edge index $I^E$
 1: Initialize $I^E$ to an empty map
 2: for all $G^P \in D^P$ do
 3:     Initialize $K$ to an empty map
 4:     for all $(u, v) \in E(G^P)$ do
 5:         $t \leftarrow (L(u),\ L(v),\ L(u, v))$
 6:         $K(t) \leftarrow K(t) \cup \{(u, v)\}$
 7:     end for
 8:     for all $t \in K$ do
 9:         $p_t \leftarrow 1 - \prod_{e \in K(t)} (1 - P(e))$
10:         $I^E(t) \leftarrow I^E(t) \cup \{(G^P, p_t)\}$
11:     end for
12: end for
13: return $I^E$
```

of vertices with labels $L(u)$ and $L(v)$ and identifying all the overlaps between any subset of these paths. Since this would make the construction and maintenance of the index an expensive and complex operation, we do not compute and store these probabilities; instead, we only store whether such a path exists with probability higher than zero or not.

Another issue that arises by allowing for paths of length $\ell > 1$ is that the size of the index is significantly increased, due to the exponential increase of the number of possible paths. To deal with this problem, we only maintain a summary of this information using Bloom filters [3]. A Bloom filter consists of an array of m bits and k independent hash functions $F = \{f_1, f_2, \ldots, f_k\}$, which hash elements of a universe U to an integer in the range of $[1, m]$. The m bits are initially set to 0 in an empty Bloom filter. An element $x \in U$ is inserted into the Bloom filter by setting all positions $f_i(x)$ of the bit array to 1, for all $f_i \in F$. Thus, an element x is contained in the original set only if all the positions $f_i(x)$ of the Bloom filter are set to 1. If at least one of these positions is set to 0, we can safely conclude that x is not present in the original set. However, due to hash collisions, there is also a small probability of false positives, $Pr_{fp} \approx (1 - e^{-kn/m})^k$, where n denotes the number of elements hashed in the Bloom filter. In our case, a high probability of false positives decreases the pruning power of the connectivity index, since we use Bloom filters to summarize all the paths of a given length contained in each graph. Each path inserted in the Bloom filter is represented by the labels of its start node and end node, sorted lexicographically. Formally, the connectivity index I^C is defined as follows.

Definition 7 (Connectivity Index). *Given an uncertain graph G^P, an integer $\ell \leq \ell_{max}$ and two labels L_u and L_v, the connectivity index I^C is a structure such that:*

- *if the uncertain graph G^P contains a path of length ℓ between two vertices u and v with labels $L(u) = L_u$ and $L(v) = L_v$, then $I^C(G^P, L_u, L_v, \ell) = 1$,*
- *otherwise, $I^C(G^P, L_u, L_v, \ell) = 0$ with probability at least $1-\varepsilon$, and $I^C(G^P, L_u, L_v, \ell) = 1$ with probability at most ε, for a fixed error probability threshold ε.*

The process of constructing the connectivity index I^C is described in detail in Algorithm 2. As with the edge index, I^C can also be built progressively, or maintained to reflect changes in the underlying graph database.

Note that there is no need to construct the index for $\ell = 1$, since the graphs containing single-edge paths can be efficiently retrieved from the edge index I^E. Therefore, we only consider $\ell \in [2, \ell_{max}]$ when constructing the connectivity index, as well as for deciding whether an uncertain graph contains a candidate subgraph pattern.

Algorithm 2 Construction of the Connectivity Index I^C

```
Input  : An uncertain graph database $D^P$; the maximum path length $\ell_{max}$
Output : The connectivity index $I^C$
 1: Initialize $I^C$ to an empty map which maps graphs to arrays of Bloom
    filters
 2: for all $G^P \in D^P$ do
 3:     for all $u \in V(G^P)$ do
 4:         Start depth-first search from $u$ to maximum depth $\ell_{max}$
 5:         for all visited node $v$ do
 6:             $\ell \leftarrow$ the current depth
 7:             if $\ell \geq 2$ then
 8:                 $I^C(G^P, \ell, L(u), L(v)) \leftarrow 1$
 9:             end if
10:         end for
11:     end for
12: end for
13: return $I^C$
```

The value of the maximum path length ℓ_{max} provides a trade-off between pruning effectiveness and space requirements, and its value can be selected empirically. In our experiments, we set ℓ_{max} to 3.

Example 3. *The graph G^P illustrated in Figure 1 would contribute to the connectivity index only the entry $I^C(G^P, A, A, 2) = 1$, since all the paths of length 2 are between vertices with label A and there are no paths with length more than 2.*

5. FREQUENT SUBGRAPH PATTERN MINING ALGORITHM

Similar to existing approaches for frequent subgraph pattern mining in either exact or uncertain graph databases, our method comprises two main parts. The first part is a process to generate candidate subgraph patterns for examination. For this purpose, we adopt the method proposed in gSpan [28], which is also followed in [32] for the same purpose. We briefly explain this process in Section 5.1. The second involves the process of evaluating whether a candidate pattern is frequent. This is performed efficiently using the UGRAP index structure described in Section 4 to prune candidate patterns with support lower than the required threshold. This process is explained in Sections 5.2 and 5.3.

5.1 Enumerating Candidate Patterns

The first part for finding frequent subgraph patterns is a process that generates candidate patterns to be examined. Every subgraph of a graph in the database constitutes a candidate pattern. Therefore, this process should enumerate all possible subgraph patterns in a systematic and efficient manner. The typical solution adopted by existing methods for mining both exact and uncertain graphs exploits the apriori property [1] as described below.

Let S and S' be two subgraph patterns. If $S \sqsubseteq S'$, then S is called a *subpattern* of S' (and S' is called a superpattern of S). If, in addition, $|E(S')| = |E(S)| + 1$, then S is called a *direct* subpattern of S'. It is easy to show that, if S is a subpattern of S', then $sup(S, D) \geq sup(S', D)$. This is referred to as the *apriori* property, and it can be easily seen that it also holds for the expected support of subgraph patterns in uncertain graph databases. Using this property, it is possible to avoid unnecessary tests when searching for frequent subgraph patterns in a database, since:

1. if S is a frequent subgraph pattern, then every subpattern of S is also frequent, and
2. if S is not a frequent subgraph pattern, then no superpattern of S can be frequent.

Consequently, the subgraph patterns occurring in the database D^P can be organized in a rooted directed acyclic graph (DAG), where the nodes represent candidate patterns (with the root being an empty pattern) and an edge from a pattern S_i to a pattern S_j denoting that S_i is a direct subpattern of S_j. This structure enumerates all the possible subgraph patterns by starting from patterns comprising a single edge and then expanding them by adding additional edges, so that all the patterns consisting of n edges can be found at level n, i.e., having a path from the root of length n. Each subgraph pattern typically has more than one direct superpatterns (since there are many possible edges that can be added to it). To avoid enumerating each subgraph pattern multiple times, a spanning tree of this DAG needs to be selected. The method proposed in gSpan [28] achieves this by imposing a lexicographic order among the patterns. Each pattern is assigned a unique canonical label, which is derived by a depth-first traversal of the pattern nodes combined with a lexicographic order. Then, by convention, patterns are allowed to "grow" by adding an edge only to vertices on the right-most path according to the considered ordering. This transforms the above graph to a tree hierarchy of patterns. The subgraph patterns are enumerated applying a depth-first search of this tree structure.

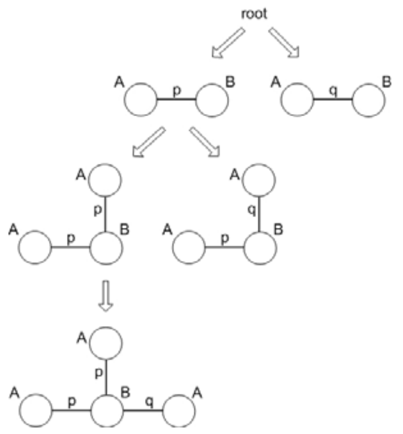

Figure 2: Search tree of candidate subgraph patterns.

Example 4. *Figure 2 depicts the tree hierarchy of candidate subgraph patterns for a database containing only the uncertain graph shown in Figure 1.*

5.2 Computing Expected Support

In the following, we introduce an algorithm that employs the constructed UGRAP index to efficiently evaluate candidate subgraph patterns for deciding whether the support of a pattern S exceeds the requested minimum support threshold *minSup*. The algorithm uses the information stored in the index to compute upper bounds of the expected support of a pattern, so that infrequent subgraph patterns can be identified and pruned early, without performing the computationally expensive subgraph isomorphism tests and calculations of the expected supports.

According to Definition 5, the expected support of a candidate pattern S in an uncertain graph G^P is above zero only if S is subgraph isomorphic to at least one of the exact graphs implied by G^P. Consequently, this may hold only if all the edges contained in S can be mapped to some edge in G^P. For each edge $e \in E(S)$, corresponding to a key $t = T(e)$, the index I^E returns a list $I^E(t)$ of those uncertain graphs in the database containing edges that e could be mapped to. Hence, the intersection of these lists, i.e.,

$$I_S = \bigcap_{e \in E(S)} I^E(T(e))$$

returns the set of uncertain graphs in the database to which S is potentially subgraph isomorphic. S can not be subgraph isomorphic to any of the remaining graphs, since there will be at least one edge of S that can not be mapped to some edge in those graphs. Consequently, assuming trivially that the expected support of S in each of the graphs in I_S is at most 1, the expected support of S in the database is at most as high as the size of this list, i.e.,

$$esup(S, D^P) \leq \frac{|I_S|}{|D^P|} \tag{7}$$

This already provides a first upper bound of the expected support of S in D^P. Subgraph patterns that do not satisfy the above condition can be immediately pruned and need not be involved in any subgraph isomorphism tests. Moreover, tighter bounds can be obtained by taking the uncertainty of edges into consideration. In particular, the expected support of S in an uncertain graph $G^P \in I_S$ is bounded by the probability that G^P implies an exact graph that contains all the edges to which the edges of S are mapped. This can be derived from the probabilities stored in the index for each pair of a key and a graph. Specifically,

$$esup(S, G^P) \leq \prod_{e \in E(S)} p^{G^P}_{T(e)} \tag{8}$$

where $p^{G^P}_{T(e)}$ is the probability provided by the index for the key $T(e)$ and the list entry of G^P, calculated according to Equation 6. Therefore, the following upper bound can be obtained for the expected support of the candidate pattern S in the database D^P:

$$esup(S, D^P) \leq \frac{1}{|D^P|} \sum_{G^P \in I_S} \prod_{e \in E(S)} p^{G^P}_{T(e)} = \overline{esup(S, D^P)} \tag{9}$$

If the upper bound calculated by Equation 9 is lower than the required support threshold *minSup*, i.e.,

$$\overline{esup(S, D^P)} < minSup \tag{10}$$

then the subgraph pattern S can be safely pruned. Notice that this pruning process of candidate patterns is very efficient for two reasons. First, it examines only a relatively small subset of the graphs in the database, i.e., those graphs contained in the list I_S. Second, it only uses the probabilities stored in the index, i.e., it does not involve any subgraph isomorphism tests.

Clearly, an important factor determining the cost savings is the size of the list I_S. This list contains each graph G^P in the database such that, for each edge $e \in E(S)$ there exists an edge in G^P to which e can be mapped. Although this is a necessary condition for S being subgraph isomorphic to at least one exact graph implied by G^P, it allows for many false positives, since it does not consider any structural information regarding how these edges are connected. For this purpose, we exploit the connectivity index I^C to perform an additional filtering step before computing the upper bound of the expected support in Equation 9 and testing the condition in Equation 10. In particular, we compute all the pairs (L_u, L_v) such that there exists two vertices u and v in S with labels L_u and L_v, respectively, and a path between them with length $\ell \leq \ell_{max}$. Then, for each graph $G^P \in I_S$, we test whether $I^C(G^P, \ell, L_u, L_v) = 1$ for each one of these pairs (L_u, L_v) with path length ℓ. If this does not hold, then G^P is removed from I_S, since S can not be subgraph-isomorphic to it. This reduces the size of the list I_S, allowing for a tighter bound to be computed in Equation 9. The above process is detailed in Algorithm 3.

Algorithm 3 UGRAP

```
Input : The UGRAP index of the uncertain graph database $D^P$; the minimum support threshold $minSup$
Output : The set of frequent subgraph patterns $Freq$
 1: Initialize $Freq \leftarrow \emptyset$
 2: Start enumerating candidate subgraph patterns
 3: while there are more subgraph patterns do
 4:     $S \leftarrow$ the next subgraph pattern to be examined
 5:     $T_S \leftarrow \bigcup_{e \in E(S)} T(e)$
 6:     // Graph pruning using $I^E$
 7:     $I_S \leftarrow \{G^P \mid G^P \in \bigcap_{t \in T_S} I^E(t)\}$
 8:     if $I_S = \emptyset$ or $|I_S|/|D^P| < minSup$ then
 9:         skip $S$
10:     end if
11:     for all $G^P \in I_S$ do
12:         $L_S \leftarrow$ label pairs of vertices in $S$ connected with path of length $\ell \in (1, \ell_{max}]$
13:         // Graph pruning using $I^C$
14:         for all $(L_u, L_v, \ell) \in L_S$ do
15:             if $I^C(G^P, L_u, L_v, \ell) = 0$ then
16:                 $I_S \leftarrow I_S \setminus \{G^P\}$
17:             end if
18:         end for
19:     end for
20:     $\overline{esup(S, D^P)} \leftarrow \sum_{G^P \in I_S} \prod_{e \in E(S)} p^{G^P}_{T(e)}$
21:     if $\overline{esup(S, D^P)}$ / $|D^P| < minSup$ then
22:         skip $S$
23:     end if
24:     // Compute the actual expected support for the remaining graphs
25:     $esup(S, D^P) \leftarrow \sum_{G^P \in I_S} esup(S, G^P)$
26:     if $esup(S, D^P)$ / $|D^P| \geq minSup$ then
27:         $Freq \leftarrow Freq \cup S$
28:     end if
29: end while
30: return $Freq$
```

Finally, if the pruning condition in Equation 10 is not satisfied, then the expected support of S needs to be calculated, or at least approximated to an extent that the pattern can be either identified as frequent or discarded as infrequent. This requires calculating the expected support of S in each of the uncertain graphs $G^P \in I_S$. We distinguish two cases. If G^P is relatively small, we compute the exact value of the expected support, as defined in Equation 5. Otherwise, we apply the approximation algorithm proposed in [32]. This algorithm transforms the problem to an instance of the DNF counting problem, by constructing a DNF formula based on the embeddings of S in G^P. Then, it approximates the satisfaction probability of this formula, which corresponds to the expected support of S in G^P, in an interval of width at most $\epsilon \cdot minSup$ with a probability $1 - \delta$, where ϵ and δ are two error tolerance parameters that determine the trade-off between the accuracy and the efficiency of the approximation.

For choosing between the exact and the approximate algorithm, the computation cost of each approach is calculated, and the approach with the smallest cost is selected. For a pattern S and graph G^P with X embeddings, the cost $Cost(G^P, S)$ is computed as follows (based on the theoretical results of [32]):

$$Cost(G^P, S) = \begin{cases} \ln(2/\delta)/(\epsilon \cdot minSup)^2 & \text{for approximation} \\ 2^{X-5}/X & \text{for exact} \end{cases} \tag{11}$$

5.3 Scheduling and Early Termination

Reducing the graphs to be examined only to the subset contained in the list I_S for a candidate pattern S, avoids a large number of subgraph isomorphism tests and therefore yields significant cost savings. However, as we explain in Section 6, depending on the characteristics of the dataset, the remaining cost may still be high for two reasons. First, if the number of distinct labels assigned to the vertices and edges of the graphs in the database is low, then the selectivity of the index is also low, since this means that a large number of edges is mapped to the same key. Consequently, the size of the list I_S is high, i.e., not many graphs are skipped. Second, as explained above, if the pruning condition in Equation 10 fails, then the expected support of S for the graphs in I_S needs to be computed, which is an expensive operation.

To address this issue, we introduce further optimizations in the algorithm which enable early termination in combination with efficient scheduling of the graphs to be examined. The purpose of these optimizations is to reach the situation where the pattern S can either be confirmed as frequent, or discarded as infrequent, with the minimal computation cost.

Observe that, when computing the expected support of a candidate pattern, the computational cost for each graph to be examined varies based on the characteristics of the graph (see Equation 11). Thus, one possible optimization strategy is to examine graphs in increasing order of cost, since it might often occur that a candidate pattern can be identified as frequent or infrequent before all the graphs are examined, hence saving the time for the most costly operations. However, the computation cost per graph is not a sufficient feature for scheduling the graphs. Indeed, some graphs may have a relatively low computation cost, but also provide a very small expected support, thereby not contributing much toward deciding whether the candidate pattern is frequent or not. Instead, we consider as a scheduling criterion the *cost per benefit* ratio (*CBR*) for the given graph and pattern. Clearly, to exactly compute CBR, we need the exact value of the expected support, which is unknown. Thus, we use instead as an approximation the upper bound of the expected support, computed according to Equation 8 and the information contained in the UGRAP index. More specifically, we approximate the cost per benefit ratio as follows:

$$CBR(G^P, S) = \frac{Cost(G^P, S)}{\prod_{e \in E(S)} p_{T(e)}^{G^P}} \tag{12}$$

where $Cost(G^P, S)$ is computed according to Equation 11. Note that the upper bound of the expected support per graph is already computed for all graphs contained in I_S, during the previous steps of the algorithm. Therefore, the additional cost for enabling graph scheduling is negligible.

Following, UGRAP considers the graphs in I_S in an increasing order based on their cost per benefit ratio, maintaining two values: (a) the actual expected support *esup*, which is the sum of the expected supports for all the graphs examined so far, initialized with 0, and, (b) the upper bound of the expected support for the remaining graphs *remSup*, initialized with $remSup = \overline{esup(S, I_S)}$, as per Equation 9. At each subsequent step, *remSup* is updated, calculating the value over the remaining graphs in I_S. The algorithm terminates when either (a) $esup \geq minSup$, i.e., the pattern is confirmed to be frequent, or (b) $esup + remSup \leq minSup$, in which case the pattern is discarded as infrequent. The above process is detailed in Algorithm 4.

As shown in the next section, this combination of the UGRAP index with the scheduling and early termination drastically reduces the number of required subgraph isomorphisms, as well as the number of required *esup* computations, thereby significantly reducing the execution time compared to the current state of the art approach.

Algorithm 4 Scheduling and Early Termination

```
Input  : A candidate subgraph pattern $S$; a list of uncertain graphs $I_S$;
Output : True, if the pattern has expected support higher than $minSup$
 1: $esup \leftarrow 0$
 2: $remSup \leftarrow \overline{esup(S, D^P)}$ (Equation 9)
 3: for all $G^P \in I_S$ do
 4:     Compute $CBR(G^P, S)$ using Equation 12
 5: end for
 6: Sort graphs in $I_S$ in increasing order of $CBR$
 7: while $esup < minSup$ and $esup + remSup > minSup$ do
 8:     $G^P \leftarrow$ remove first graph from $I_S$
 9:     $s \leftarrow$ compute the expected support of $S$ in $G^P$, using either approx-
        imation or exact approach
10:     $esup \leftarrow esup + s$
11:     $remSup \leftarrow remSup - \overline{esup(S, G^P)}$
12: end while
13: return true if ($esup \geq minSup$); false otherwise
```

6. EXPERIMENTAL EVALUATION

We have conducted an extensive experimental evaluation of the performance of the UGRAP algorithm for mining frequent subgraph patterns in uncertain graph databases. We have used three real-world datasets from the bioinformatics domain, as well as a set of synthetic datasets for examining in more detail the impact of different parameters in the performance of UGRAP. For comparison, we also implemented MUSE [32], which is currently the state-of-the-art approach for discovering frequent subgraph patterns in uncertain graphs, and we used it in the experiments to evaluate the performance of UGRAP.

All experiments were executed on a single core of an AMD Opteron 2.7 Ghz machine running CentOS Linux, kernel 2.6. Regarding the approximation method for computing expected support in MUSE, both parameters δ and ε were set to 0.4. Note that the same method is used by UGRAP, for approximating the expected support of patterns that are not pruned using the index, as explained in Section 5.2. Therefore, both UGRAP and MUSE have the same performance regarding precision and recall of the discovered patterns; hence, we do not present precision/recall results in this evaluation (quality results for MUSE can be found in [32]). Regarding the UGRAP index, unless otherwise noted, it was configured to construct 2 Bloom filters per graph, covering paths of length 2 and 3. For enumerating the candidate patterns, we have used a C implementation of the gSpan algorithm [28], which is available online[1]. For subgraph isomorphism testing, we have used the C++ implementation of VF2 [6], the state-of-the-art subgraph isomorphism algorithm, available from the VFLIB library[2]. The experiments were also repeated with the subgraph isomorphism algorithm from Ullmann [23], which is also very frequently used in the literature. The results with Ullmann's algorithm were almost identical with the ones obtained using VF2. Therefore, in the following we only show the results corresponding to VF2. All C/C++ code was compiled with gcc/g++ 4.1.2, using the second level of optimization.

In the next section we describe in detail the different datasets used in the experiments. Following, in Section 6.2 we present and discuss the results of our evaluation.

[1] http://wwwkramer.in.tum.de/research/data_mining/pattern_mining/graph_mining

[2] http://amalfi.dis.unina.it/graph/db/vflib-2.0/doc/vflib.html

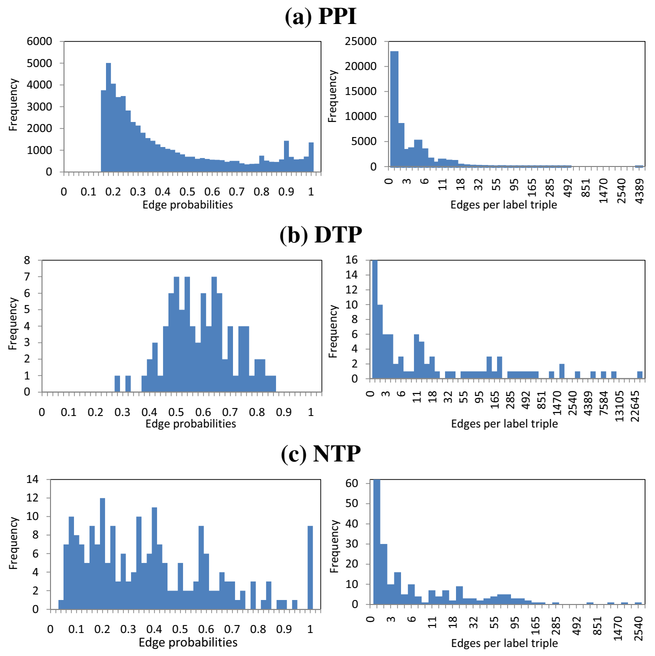

Figure 3: Distribution of edge probabilities and label triples.

6.1 Datasets

We have evaluated the performance of our approach on three real-world datasets from the bioinformatics domain, as well as on synthetic datasets. We describe these datasets below.

Protein-Protein Interactions (PPI). This is an uncertain graph database representing protein interactions for different organisms obtained from the STRING database[3]. Each graph represents protein-to-protein interactions for a specific organism. The labels represent COG functions, while edge probabilities represent confidence scores for each particular interaction.

Developmental Therapeutics Program (DTP). This is an AIDS antiviral screening dataset[4], containing chemical compounds that have been tested for evidence of anti-HIV activity. Node and edge labels correspond to types of atoms and bonds, respectively. We converted these graphs to uncertain graphs by assigning probabilities to the edges as follows. First, to each vertex or edge label L, we assigned randomly a probability $p_L \in (0, 1]$. Then, to each edge e=(u, v) with label L_e, we assigned randomly a probability in the interval $[\mu - \delta, \mu + \delta]$, where μ was set to the average value of the probabilities assigned to the labels $L(u)$, $L(v)$, and $L(e)$, and δ was set to 0.1. This process was followed so that the probability assigned to an edge was related to the type of the edge.

National Toxicology Program (NTP). This dataset was constructed from the National Toxicology Program and includes a set of chemical compounds that have been experimentally examined for carcinogenic effects in rodents [15]. The dataset is frequently used for evaluating pattern mining in graphs. In this work, we used the version provided as part of the Predictive Toxicology Evaluation Challenge [10], which includes 340 chemical compounds. To assign a probability to each edge $e = (u, v)$, we used the average of the probabilities of the vertices u and v as given in the dataset.

The characteristics of the aforementioned datasets are presented in Table 1. This table provides statistics regarding the number of graphs in each dataset, the total and average number of nodes and

[3]`http://string-db.org/`

[4]`http://dtp.nci.nih.gov/docs/aids/aids_data.html`

	PPI	DTP	NTP
number of graphs	587	1,572	340
number of nodes	61,764	54,485	9,189
with distinct labels	5,350	30	66
number of edges	214,314	58,184	9,317
with distinct labels	7	3	4
number of distinct label triples	54,134	90	207
average nodes per graph	105	35	27
with minimum	3	7	2
with maximum	818	259	214
average edges per graph	365	37	27
with minimum	3	5	1
with maximum	3,910	273	214

Table 1: Statistics of the real-world datasets.

edges, and the number of distinct labels. In addition, Figure 3 plots the distribution of the frequencies of edge probabilities and of the occurrences of label triples. Recall that the latter correspond to the keys used in the I^E index. As can be observed, each dataset has different characteristics. In particular, they differ in 3 main factors: the number of graphs contained in the dataset, the size of the graphs contained in the dataset, and the number of distinct labels. For instance, DTP contains the larger number of graphs among the three datasets; however, the graphs contained in PPI are, on average, much larger than those contained in the other two datasets. Moreover, PPI contains a large number of distinct label triples, namely 54,134 distinct label triples for a total of 214,314 edges (avg. ratio 0.25), whereas DTP contains only 90 distinct label triples for a total of 58,184 edges (avg. ratio 0.0015) and NTP has 207 distinct label triples for a total of 9,317 edges (avg. ratio 0.022). Finally, in NTP, edge probabilities are relatively more uniformly distributed, whereas DTP has higher frequencies on edge probabilities around 0.5, and PPI has higher frequencies on lower probabilities, e.g. around 0.2 and 0.3. In Section 6.2, we discuss the effect of these different characteristics on the performance of the frequent subgraph pattern mining algorithms.

Synthetic Datasets. For evaluating UGRAP with a significantly larger dataset, and for examining the effect of specific factors on the algorithm's performance, we have also created a set of synthetic datasets, using the graph generator from [4]. More details about the resulted synthetic datasets are provided in Section 6.2.2 along with the corresponding evaluation results.

6.2 Results

We now report the results of the performance evaluation of UGRAP and MUSE on the real-world and the synthetic datasets. Recall that UGRAP does not affect the quality of the retrieved results (i.e., precision in terms of patterns falsely identified as frequent, and recall); it derives exactly the same results as MUSE, whose quality is thoroughly evaluated in [32]. Therefore, we focus on the efficiency of the two approaches.

6.2.1 Real Datasets

We first examined the cost for constructing the UGRAP index for each dataset. This is an offline operation, performed only once. Regarding the memory cost, we measured the memory required for storing the edge index and the Bloom filters; the memory for storing the graphs is not included, since it is the same in both UGRAP and MUSE. Figure 4 shows the memory and time required for indexing the three real-world datasets. As shown, index construction is very efficient, requiring less than 15 seconds for the PPI dataset, and less than 4 seconds for the other two datasets. The memory cost is also negligible, reaching a maximum of 1.6 Mbytes for the PPI dataset, and less than 60 Kbytes for the other two datasets. Both the

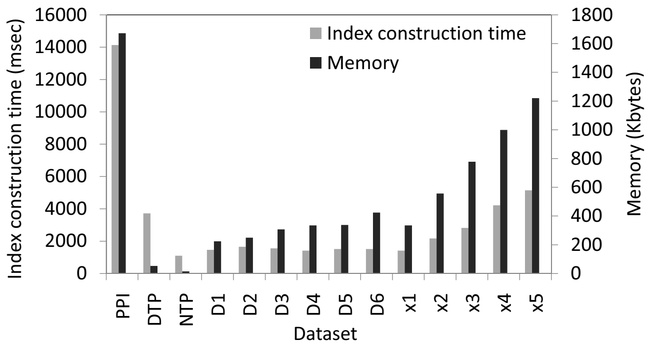

Figure 4: Index construction time and memory for UGRAP.

	minSup (%)	# frequent patterns	# SG Isomorphisms MUSE	UGRAP	Percent SG Isom.
PPI	4	97	3.53E7	30,027	0.085%
	6	50	3.38E7	16,954	0.050%
	8	30	3.29E7	6,124	0.019%
	10	11	3.22E7	1,603	0.005%
	12	2	3.19E7	150	0.000%
	14	0	3.18E7	0	0.000%
DTP	4	105	2,562,360	381,695	14.896%
	6	68	1,942,992	278,998	14.359%
	8	57	1,757,496	250,338	14.244%
	10	41	1,413,228	183,135	12.959%
	12	33	1,026,516	118,698	11.563%
	14	27	902,328	101,606	11.260%
NTP	4	27	150,620	4,476	2.972%
	6	17	132,600	3,679	2.775%
	8	10	101,660	2,538	2.497%
	10	9	100,300	2,262	2.255%
	12	7	99,280	1,884	1.898%
	14	5	93,840	1,579	1.683%

Table 2: Number of discovered frequent subgraph patterns and subgraph isomorphism tests performed by UGRAP and MUSE.

memory and indexing time are affected by the number and the size of the graphs in the database, as well as by the number of distinct label triples, which explains the higher cost for the PPI dataset. However, as will be shown later, this process scales easily to very large and complex datasets and does not constitute a bottleneck.

We then investigated the effect of the minimum support threshold *minSup* on the performance of UGRAP, comparing it to MUSE. For this purpose, we executed UGRAP and MUSE on the three real-world datasets described above, varying the value of *minSup* in the range of [0.04 − 0.14], and we measured the execution time for each case. Figure 5 plots the results for the three datasets. We see that the execution time of both UGRAP and MUSE decreases for higher *minSup* values. This is due to the fact that by increasing *minSup*, fewer subgraph patterns qualify as frequent (see Table 2), resulting to a faster termination of the traversal of the search tree. Notice however that the performance improvement in UGRAP is substantially higher compared to MUSE. This additional gain in performance is due to the prioritization with early termination, which becomes more effective in eliminating candidate graphs for higher *minSup* thresholds. More specifically, for higher *minSup* values, the information obtained from the index is sufficient for discarding patterns in most of the cases, without requiring expensive isomorphisms or probability approximations. The results are consistent for all datasets.

Moreover, UGRAP outperforms MUSE by one to three orders

	D_1	D_2	D_3	D_4	D_5	D_6
value of λ	0.02	0.04	0.08	0.12	0.16	0.20
label triples	1,770	3,723	7,451	10,665	14,861	18,426

Table 3: Number of distinct label triples for the synthetic datasets generated with varying λ.

of magnitude. This difference is due to two factors. First, the edge index substantially reduces the number of graphs that need to be examined for isomorphism for each candidate pattern. This large difference in the number of subgraph isomorphism tests performed by the two methods can be clearly seen in Table 2. The highest cost savings are observed for the PPI dataset, which has the highest number of distinct vertex and edge labels, and therefore the edge index I^E has the highest selectivity. In this case, the number of subgraph isomorphism tests performed by UGRAP is always less than 0.01% of the ones required by MUSE. This reduction becomes less in the case of the DTP dataset, which has the lowest number of distinct label triples. However, UGRAP still avoids up to 85% of the subgraph isomorphism tests. The second factor contributing to the performance difference of the two algorithms is the prioritization scheme with early termination. This substantially reduces the time required for computing expected supports, by prioritizing the graphs with a low cost per benefit ratio, and postponing the most costly graphs for the end, anticipating that they will not be processed due to the early termination condition. As explained, the effectiveness of both the index and the prioritization strategy grows with the *minSup* values, and therefore the performance gain of UGRAP compared to MUSE also grows.

6.2.2 Synthetic Datasets

The second series of experiments was conducted using a set of synthetic datasets, which were created in order to study the effect of specific factors in the performance of UGRAP and MUSE.

Varying the number of distinct label triples. As explained in Section 4, the keys used in our index are label triples of the form $t_i = (L_u, L_v, L_e)$. As such, both the size and the selectivity of the index are expected to grow with the number of distinct label triples in the database. To examine the effect of this parameter on the performance of UGRAP, we constructed six different synthetic datasets, each with a different number of distinct label triples. The datasets were constructed using as basis the 1572 graphs contained in DTP, the largest of the real datasets, and adding 428 graphs generated using the graph generator from [4]. These additional graphs for each of the six datasets were generated with a different number of label triples, controlled through the configuration parameters of the graph generator. All the resulting datasets had a total of 2000 graphs, with 64,413 nodes and 927,510 edges. For each dataset, the number of distinct label triples was equal to $\lambda{\cdot}\tau$, for $\lambda \in [0.02{-}0.2]$, and τ set to the number of total edges, i.e., 927,510. In other words, each dataset was created with a predetermined ratio λ of the number of distinct label triples to the total number of edges, as shown in Table 3. In all other aspects, the datasets were generated with the same characteristics (i.e., the same frequent patterns and the same complexity of the contained graphs); hence, the differences in execution time can only be attributed to the parameter λ.

Figure 6(a) shows the execution time of both algorithms for different values of λ. The results are shown in different plots for clarity, because of the large difference in the values of the y axis. Similar to the results for the real-world datasets, UGRAP outperforms MUSE by two to three orders of magnitude. Furthermore, we observe that increasing λ has a clear negative effect on the performance of MUSE, since it results to a larger number of *potentially*

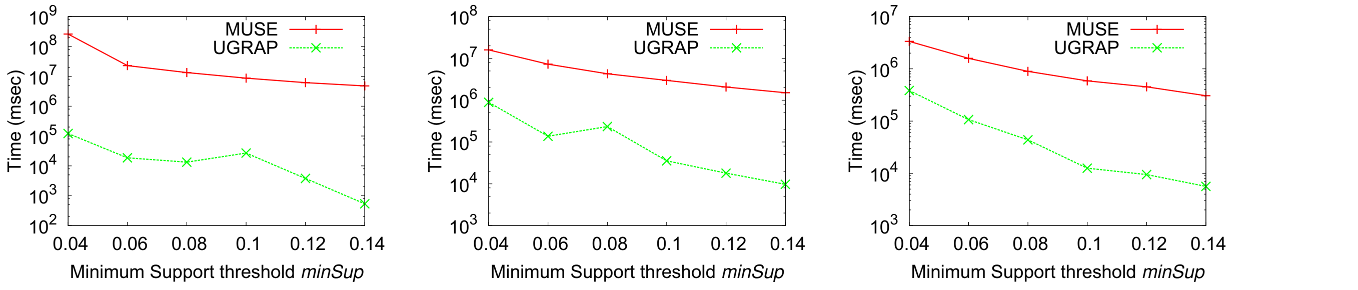

Figure 5: Execution time vs. support threshold *minSup* for the real-world datasets (a) PPI, (b) DTP, and (c) NPT.

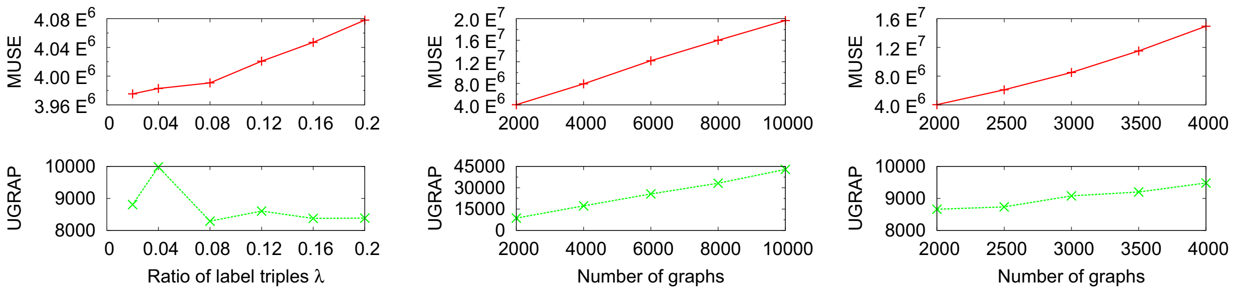

Figure 6: Execution time for synthetic datasets in milliseconds, when varying: (a) the ratio λ of distinct label triples to the total number of edges, (b) the number of uncertain graphs, by cloning, and (c) the number of uncertain graphs, by adding new graphs.

frequent patterns that need to be examined. On the contrary, in the case of UGRAP, this negative effect is canceled by the fact that the selectivity of the index is also increased with a higher λ, making the algorithm more effective in pruning infrequent patterns and in skipping graphs that do not contain the pattern in question. As such, UGRAP has approximately the same execution time on all datasets, independent of the value of λ.

We also measured the additional memory and time requirements of UGRAP for constructing the edge index and the Bloom filters. As shown in Figure 4, the additional requirements are negligible for all datasets (i.e., D_1-D_6). As expected, the required memory slightly increases with an increase of λ, since more label triples need to be indexed. However, this memory increase is very small, due to the compactness of both the index and especially the Bloom filters. Furthermore, the total memory required by the UGRAP index is in the range of a few hundreds of kilobytes. The time required for constructing the index and the Bloom filters is also very small, not exceeding two seconds for all scenarios.

Scalability. The next set of experiments evaluated the performance of UGRAP in terms of scalability. Similar to the scalability evaluation methodology of [32], we created larger datasets by duplicating the graphs of a single uncertain database. As a basis, we used the previously described synthetic collection of 2000 graphs with λ=0.12, replicated up to 5 times, to produce datasets of up to 10000 graphs. We denote these datasets by xR, where $R \in [1, 5]$ indicates the number of replications.

Figure 6(b) plots the execution time of the two algorithms with respect to the number of graphs. As expected, the execution time increases with the number of graphs. The increase is linear for both algorithms, and therefore UGRAP maintains the three orders of magnitude difference in performance, compared to MUSE. The cost for creating the index and Bloom filters for this experiment is also shown in Figure 4 (series x1 to x5). Both the time for the index construction and the memory requirements scale linearly with the number of graphs in the database.

As a second scalability test, we progressively increased the size of the uncertain graph database by adding new graphs, instead of replicating it as previously. The difference from the previous configuration is that by adding new graphs, the number of distinct label triples increases. As a basis, we used again the synthetic collection with λ=0.12, and we incremented it in steps of 500 graphs, to a maximum of 4000 graphs. The additional graphs were generated using the standard graph generator [4], which also allowed us to keep the value of λ constant. Note that the newly added graphs did not introduce any additional frequent patterns; the frequent patterns were all part of the initial set of 2000 graphs. To make the results across the datasets comparable, we fixed the set of frequent subgraphs of all databases by adjusting the value of *minSup* at each run accordingly. For example, having as a reference set the frequent patterns resulting from 2000 graphs with *minSup*=0.08, the same frequent patterns were retrieved for the database of 4000 graphs by adjusting *minSup* to 0.04. Due to this configuration, the resulting differences are attributed only to the database size, and not to the number of frequent subgraphs or other factors.

Figure 6(c) plots the execution time with respect to the number of graphs. Increasing the number of graphs causes a very small, sublinear increase on the execution time of UGRAP. The execution time difference between the datasets with 2000 and 4000 graphs is below 10%. This good scalability is attributed to the combination of the UGRAP index with the scheduling and early termination strategy, which minimize the effect of the additional graphs and patterns. In contrast, the execution time of MUSE grows superlinearly with the number of graphs, from 4×10^6 for 2000 graphs, to 1.6×10^7 for 4000 graphs. Two factors are responsible for this drastic increase: the additional graphs that need to be examined, and the additional candidate patterns, introduced by these new graphs. Since MUSE does not prune any infrequent patterns without examination, these additional patterns and graphs increase its execution time substantially. Finally, regarding the UGRAP index, the construction time and memory were again very small, not exceeding 3 seconds and 1.3 Mbytes respectively, for the largest dataset.

6.3 Evaluation Summary

Concluding, the experimental evaluation with three real-world databases and with an extensive set of synthetic databases shows that UGRAP outperforms MUSE, the current state-of-the-art approach for frequent subgraph pattern mining in uncertain graph databases, by two orders of magnitude for most configurations, and at least by one order of magnitude for all configurations. This significant performance increase does not have a negative influence on the precision or the recall of the detected frequent patterns. The scalability of the algorithm is also verified with databases of up to 10000 graphs. The additional time and memory requirements of UGRAP for constructing and maintaining the index are negligible, making the algorithm suitable for execution in any typical PC.

7. CONCLUSIONS

In this paper, we have presented UGRAP, an efficient algorithm for mining frequent subgraph patterns in uncertain graph databases. The algorithm relies on a compact probability-aware index on graph edges and paths, which allows for a drastic pruning of the search space when computing the expected support of candidate patterns. The index also enables an efficient scheduling and early termination strategy, which further improves the performance of our approach. A thorough experimental evaluation using three real-world datasets from the bioinformatics domain, as well as a set of synthetic datasets, demonstrated that UGRAP significantly outperforms the state-of-the-art solution to this problem. In particular, our results showed a performance gain between one and three orders of magnitude, depending on the dataset characteristics. Moreover, the additional time and memory requirements for constructing the index were shown to be negligible.

Our future work focuses on two main directions. The first involves an extension of the scheduling and early termination strategy, using probabilistic pruning to filter out candidate patterns more aggressively. The second aims at extending UGRAP to use the index for finding only the maximal or closed frequent subgraphs in the uncertain graph database.

Acknowledgments. This work was partially supported by the FP7 EU Project SYNC3 (contract no. 231854).

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.

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

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[4] J. Cheng, Y. Ke, and W. Ng. Graphgen: A synthetic graph generator. http://www.cse.ust.hk/graphgen/, 2006.

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

[6] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159, 2001.

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

[8] E. Gudes, S. E. Shimony, and N. Vanetik. Discovering frequent graph patterns using disjoint paths. *IEEE Trans. Knowl. Data Eng.*, 18(11):1441–1456, 2006.

[9] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, page 38, 2006.

[10] C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The predictive toxicology evaluation challenge 2000-2001. *Bioinformatics*, 17(1):107–108, 2001.

[11] P. Hintsanen and H. Toivonen. Finding reliable subgraphs from large probabilistic graphs. *Data Min. Knowl. Discov.*, 17(1):3–23, 2008.

[12] M. Hua and J. Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*, pages 347–358, 2010.

[13] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*, 2003.

[14] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. In *KDD*, pages 581–586, 2004.

[15] J. Huff and J. Haseman. Long-term chemical carcinogenesis experiments for identifying potential human cancer hazards. *Environmental Health Perspectives*, 96(3):23–31, 1991.

[16] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, pages 13–23, 2000.

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

[18] M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1038–1051, 2004.

[19] D. Liben-Nowell and J. M. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559, 2003.

[20] Y. Liu, J. Li, and H. Gao. Summarizing graph patterns. In *ICDE*, pages 903–912, 2008.

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

[22] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. k-nearest neighbors in uncertain graphs. In *PVLDB*, 2010.

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

[24] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[25] C. Wang, W. Wang, J. Pei, Y. Zhu, and B. Shi. Scalable mining of large disk-based graph databases. In *KDD*, pages 316–325, 2004.

[26] T. Washio and H. Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, 2003.

[27] X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining significant graph patterns by leap search. In *SIGMOD*, pages 433–444, 2008.

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

[29] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *KDD*, pages 286–295, 2003.

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

[31] S. Zhang, J. Yang, and S. Li. Ring: An integrated method for frequent representative subgraph mining. In *ICDM*, pages 1082–1087, 2009.

[32] Z. Zou, J. Li, H. Gao, and S. Zhang. Frequent subgraph pattern mining on uncertain graph data. In *CIKM*, pages 583–592, 2009.