Mining Uncertain Graphs: An Overview

Vasileios Kassiano, Anastasios Gounaris, Apostolos N. Papadopoulos, and Kostas Tsichlas

Department of Informatics
Aristotle University of Thessaloniki, Greece
{vkassiano,gounaria,papadopo,tsichlas}@csd.auth.gr

Abstract. Graphs play an important role in modern world, due to their widespread use for modeling, representing and organizing linked data. Taking into consideration that most of the "killer" applications require a graph-based representation (e.g., the Web, social network management, protein-protein interaction networks), efficient query processing and analysis techniques are required, not only because these graphs are massive but also because the operations that must be supported are complex, requiring significant computational resources. In many cases, each graph edge e is annotated by a probability value $p(e)$, expressing its *existential uncertainty*. This means that with probability $p(e)$ the edge will be present in the graph and with probability $1 - p(e)$ the edge will be absent. This gives rise to the concept of *probabilistic graphs* (also known as *uncertain graphs*). Formally, a probabilistic graph $\mathcal{G}$ is a triplet (V, E, p) where V is the set of nodes, E is the set of edges and $p : E \rightarrow (0, 1]$. The main challenge posed by this formulation is that problems that are relatively easy to solve in exact graphs become very difficult (or even intractable) in probabilistic graphs. In this paper, we perform an overview of the algorithmic techniques proposed in the literature for uncertain graph analysis. In particular, we center our focus on the following graph mining tasks: clustering, maximal cliques, k-nearest neighbors and core decomposition. We conclude the paper with a short discussion related to distributed mining of uncertain graphs which is expected to achieve significant performance improvements.

Keywords: graph mining, network analysis, uncertain graphs

1 Introduction

Graph mining is an important research area with a plethora of practical applications [1, 12]. The main reason for this is the fact that graphs are ubiquitous and, therefore, their efficient management and mining is necessary to guarantee fast and meaningful knowledge discovery.

In its simplest form, a graph $G(V, E)$, is composed of a set of nodes V, representing the entities (objects), and a set of edges E, representing the relationships among the entities. An edge $e_{u,v} = (u, v) \in E$ connects a pair of nodes u, v, denoting that these nodes are directly related in a meaningful manner. For example, if nodes represent authors, then an edge between two authors may denote

that they have collaborated in at least one paper. As another example, in a social network application, an edge may denote that two users are connected by a friendship relationship.

A special category of graphs, include graphs that introduce *uncertainty* with respect to the existence of nodes and edges. For example, an edge e between nodes u and v may exist or not. The existence of an edge depends on several factors depending on the particular application under consideration. For example, in a social network where the edge corresponds to a message exchange between two users, the message will be sent with some probability (i.e., it is not sure that user u will send a message to user v). As another example, consider a protein-protein interaction network, where each node corresponds to a protein and each edge denotes that two proteins are combined. In this case, we may realize that proteins u and v interact in 70% of the cases, which means that the edge $e_{u,v}$ will be present in the graph with a probability of 0.7.

Let $\mathcal{G} = (V, E, p)$ be an uncertain (a.k.a probabilistic) graph, where $p : E \rightarrow (0, 1]$ is a function that assigns probabilities to the edges of the graph[1]. A widely used approach to analyze uncertain graphs is the one of *possible worlds*, where each possible world constitutes a deterministic realization of $\mathcal{G}$. According to this model, an uncertain graph $\mathcal{G}$ is interpreted as a set $\{G = (V, E_G)\}_{E_G \subseteq E}$ of $2^{|E|}$ possible deterministic graphs [43, 44]. Let $G \sqsubseteq \mathcal{G}$ indicate that G is a possible world of $\mathcal{G}$. Then, the probability that $G = (V, E_G)$ is observed as a possible world of $\mathcal{G}$ is given by the following formula:

$$\Pr(G|\mathcal{G}) = \prod_{e \in E_G} p(e) \prod_{e \in E \setminus E_G} (1 - p(e)) \tag{1}$$

For instance, consider the probabilistic graph $\mathcal{G}$ shown in Figure 1(a). Two possible instances of $\mathcal{G}$ are given in Figure 1(b) and 1(c). Edges with high probability values are expected to show up more frequently in instances of $\mathcal{G}$. Consequently, triangles formed by high probability edges are more likely to be present in a random instance of $\mathcal{G}$. For example, it is not a surprise that the triangle formed by nodes v_4, v_5 and v_6 that has an existential probability of $0.9 \cdot 0.9 \cdot 0.9 = 0.729$ appears in both G_1 and G_2. In contrast, the triangle (v_1, v_4, v_5) has an existential probability of $0.5 \cdot 0.2 \cdot 0.9 = 0.09$ and its presence in a random instance of $\mathcal{G}$ is not very likely.

The annotation of edges with existential probabilities has significant impact on the algorithmic efficiency for particular problems. The uncertain existence of edges poses severe difficulties in solving problems whose counterparts in conventional (i.e., certain or deterministic) graphs can be easily addressed using polynomial-time algorithms. For example, assume that we are interested in determining the probability that nodes v and u are reachable from each other within a distance threshold. In a conventional (i.e., certain) undirected graph, the nodes will be either reachable if they belong to the same connected component or otherwise unreachable. Moreover, computing shortest path distances

[1] Although existential probabilities can be assigned to the vertices of the graph as well, in this paper we focus on edge probabilities only.

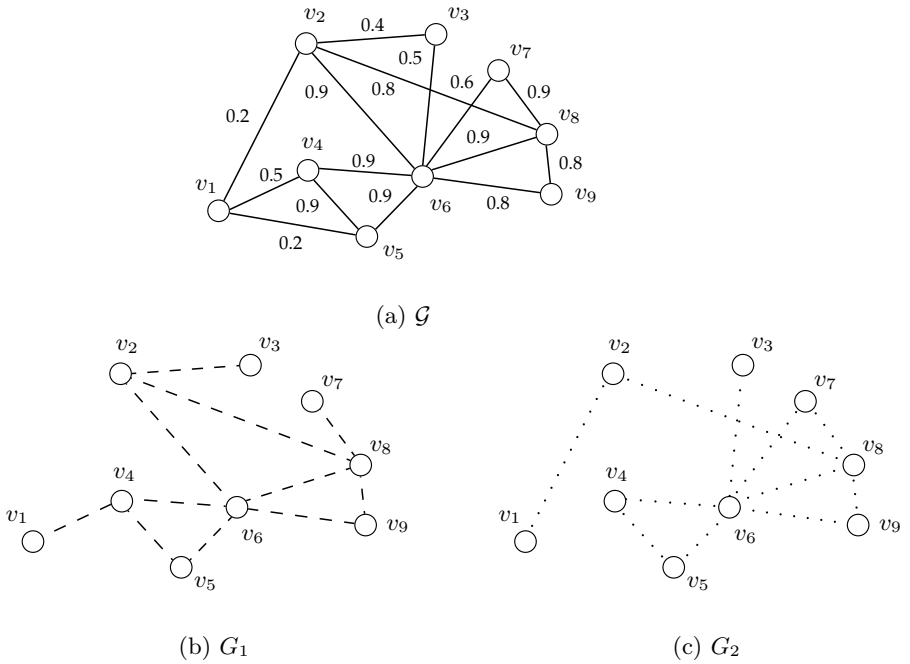

Fig. 1. A probabilistic graph $\mathcal{G}$ and two possible instances G_1 and G_2. The numbers near the edges denote existential probabilities.

is considered a common graph operation that can be solved in polynomial time using Dijkstra's shortest path algorithm. On the other hand, computing the probability that the nodes are reachable in a probabilistic graph requires significant effort, since the problem is known to be #P-complete. Similar difficulties appear in other problems such as finding nearest neighbors or computing shortest paths in a probabilistic setting.

Based on the possible worlds semantics, several graph analytic tasks have been considered recently under the uncertain graph model. Note that, a simplistic technique to work with uncertain graphs is to assume that edge probabilities are simple weights. However, this approach does not produce meaningful results in many cases, since edge probabilities have different semantics than simple edge weights. Moreover, additional difficulties may be posed when edges contain existential probabilities and weights. Therefore, the recent years specialized algorithmic techniques have been proposed. The problem of *k-nearest-neighbors* have been addressed in [48], where shortest paths are computed based on a probabilistic approach. Another important graph mining task is *clustering* where graph nodes must be assigned to clusters based on connectivity. This problem is studied in [26, 29]. Concerning the problem of mining *dense components* in uncertain graphs, [65] finds the densest induced subgraph in terms of the maximum ex-

pected density, that is the expected density value of an exact graph chosen at random. Another important concept is *reachability analysis* in uncertain graphs, where we are interested in determining if nodes are reachable given specific constraints [28]. Bonchi et al. [7] proposed an extension of the *core decomposition* for uncertain graphs. Very recently, a *triangle-based* extension of the core decomposition, namely the *truss decomposition*, was introduced for the uncertain graph model [25, 68]. Other related contributions include algorithms for *subgraph similarity search* [63], *centrality computation* [47], the discovery of *frequent subgraph patterns* in uncertain graph databases [42].

In this paper, we focus on a subset of the aforementioned algorithmic techniques. In particular, we address the following graph mining problems: 1) clustering the nodes of an uncertain graph, 2) computing k-nearest-neighbors, 3) finding maximal cliques and 4) computing the core decomposition. In addition, we provide a short discussion related to how uncertain graph mining could benefit from distributed computation in clusters and what are the basic challenges that must be addressed in such a setting.

2 Clustering Uncertain Graphs

2.1 Introduction

Clustering has a plethora of applications in many diverse fields and it is considered as one of the most important data mining tasks in general. It is defined as the problem of grouping data objects into clusters, such that objects with similar characteristics are assigned to the same cluster and objects that have dissimilar characteristics are assigned to different clusters. In many cases, the clustering problem in graphs is highly related to *community detection* [16] which has been studied thoroughly in network analysis. The problem becomes extremely challenging when probabilities are assigned to the edges of the graph.

The problem of clustering probabilistic graphs has been recently studied by [30]. Simple techniques and methods that are used for partitioning graphs into clusters cannot be applied to probabilistic graphs, due to their nature. The challenges that arise are related to the applicability of the standard algorithms and the time complexity associated with some already developed algorithms. Many studies for uncertain data management and graph mining have been conducted in the computer science community [9],[40]. These studies approach the probabilistic graph clustering problem as a deterministic one, by either considering the edge probabilities as weights or by leaving out probabilities smaller than a specific threshold. For the first approach the main problem is that it cannot solve the clustering problem for weighted probabilistic graphs, because once the probability is considered as weight, the actual weights cannot be encoded meaningfully onto the edges. For example, finding a mixed weight by multiplying the actual weight with the probability of the edge, gives a result with no possible real interpretation. For the second approach, the problem is that the threshold value cannot be computed in a principled, reliable way.

The probabilistic clustering problem has many important applications, with the most well-known being the discovery of complexes in protein-protein interaction networks and discovering communities in affiliation and social networks. The aforementioned problems contain uncertainty, so they are best represented using probabilistic graphs. In these cases, the nodes represent a protein or a user, and the edges represent the probability of an interaction between proteins or users.

The possible world semantics is being used to treat this type of graphs, meaning that every probabilistic graph $\mathcal{G}$ is treated as a generative model for deterministic graphs. This means that every possible instance of $\mathcal{G}$ represents a deterministic graph, or other words a possible world of $\mathcal{G}$. Figure 2 illustrates an example of a simple probabilistic graph $\mathcal{G}$ which consists of five nodes and eight edges.

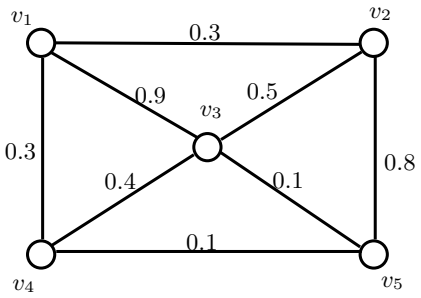

Fig. 2. A simple probabilistic graph $\mathcal{G}$ with 5 nodes and 8 edges

We call $D(G, C)$ the objective function that measures the cost of clustering the partitioning C of a deterministic graph $G = (V, E)$. Due to the possible world semantics, the expected value of this function D for clustering a probabilistic graph is the expected value of $D(\mathcal{G}, C)$ for all possible worlds of $\mathcal{G}$. By this definition, it is easily concluded that the computational cost of this function can be high, due to the number of possible worlds that are generated exponentially. There are $2^{|E|}$ possible graphs because each edge either will be part of the graph or not. Another problem with this approach is that there may be possible worlds with disconnected cliques, and therefore, it is very difficult to find a well-established clustering objective function, like the maximum cluster diameter or others [32], because the value will be infinite.

2.2 Algorithms

New definitions of the clustering problem in probabilistic graphs were proposed in [30]. The use of the edit distance as the optimization function is utilized for this definition. Considering a cluster graph C, i.e., a graph where there can be parts that are not connected with each other, as a clustering of a deterministic graph G,

the edit distance between C and G is the number of edges that need to be added and removed from G, to get C. To adapt this approach for finding the cluster graph C for probabilistic graphs, a generalization of the ClusterEdit problem by Shamir [55] was defined as the pClusterEdit problem. This problem can be defined as finding the clustering of $\mathcal{G}$, namely the cluster graph C, with the minimum expected edit distance from $\mathcal{G}$.

The advantages of this framework include its polynomial computational cost, the measurable output of the objective function and the independence of the clustering from factors other than the initial graph. This means that we do not need to evaluate each possible world of $\mathcal{G}$, so the output of the function will never be infinite because it is dependent on the node pairs within the graph. Furthermore, the variance of the output to any clustering is independent of the specific clustering but correlates only with the initial graph. Finally, the function does not require the specification of any free parameters, which means the number of clusters is part of the output.

Before getting to the algorithms, it is useful to explain the probabilistic graph model under consideration. The probabilistic graph $\mathcal{G}$ is defined as a tuple $\mathcal{G} = (V, P, W)$, where V is the set of nodes and $|V| = n$, P is the set consisting of every pair of nodes with a probability between 0 and 1; P_{uv} is the existential probability of the edge (u, v), and $|P| = m$. Finally, W is the sets of weights for all pairs of nodes. The algorithms presented below have been tested in undirected, unweighted probabilistic graphs and assume independence among edges, so $\mathcal{G} = (V, P)$. The complement of $\mathcal{G}$ is $\mathcal{G}'$ where $\mathcal{G}' = (V, 1 - P)$.

Given two deterministic graphs $G = (V, E_G)$ and $Q = (V, E_Q)$, the edit distance between G and Q is defined as the number of edges that need to be added to or deleted from G in order to be transformed into Q:

$$D(G, Q) = |EG \setminus EQ| + |EQ \setminus EG|. \tag{2}$$

By using the binary adjacency matrices $\mathbf{G}$ and $\mathbf{Q}$ of G and Q respectively we have:

$$(G, Q) = \sum_{u=1, v<u}^{n} \mathbf{G}(u, v)\mathbf{Q}(u, v)|. \tag{3}$$

This definition is extended for a probabilistic graph $\mathcal{G}$ and a deterministic graph Q as the expected edit distance between every cluster $G \in \mathcal{G}$ and Q:

$$D(G, Q) = \mathrm{E}_{G \subseteq \mathcal{G}}[D(G, Q)] = \sum_{G \subseteq \mathcal{G}} \Pr[G] D(G, Q). \tag{4}$$

Although this requires the calculation of all $2^{|E|}$ possible worlds, it is proven that the expected edit distance can be measured in polynomial time by using the following equation:

$$\mathrm{E}_{G \subseteq \mathcal{G}}[\sum_{u<v} X_{uv}] = \sum_{u<v} (\mathrm{E}_{G \subseteq \mathcal{G}} X_{uv}) = \sum_{\{u,v\} \in E_Q} (1 - P_{uv}) + \sum_{\{u,v\} \notin E_Q} P_{uv} \tag{5}$$

where X_{uv} denotes the random variable $|\mathbf{G}(u, v) - \mathbf{Q}(u, v)|$.

A *cluster graph* $C = (V, E_C)$ is a deterministic graph with the following properties: 1) C defines a partition of the nodes in V into k parts, $V = (V_1, \ldots, V_k)$ such that $V_i \cup V_j = \emptyset$. 2) For every $i \in (1, \ldots, k)$ and for every pair of nodes $v \in V_i$ and $v \in V_i$ we have that $(v, v) \in E_C$. 3) For every $i, j \in (1, \ldots, k)$ with $i \neq j$ and every pair of nodes v, v such that $v \in V_i$ and $v \in V_j$, $(v, v) \notin E_C$. The pClusterEdit clustering problem is defined as follows: Given a probabilistic graph $\mathcal{G} = (V, P)$ find the cluster graph $C = (V, E_C)$ such that $D(\mathcal{G}, C)$ is minimized.

In the *correlation clustering problem* [5], there is a positive $(+)$ or negative $(-)$ relation between any two pair of objects. The notation E^+ is used for the set of pairs that are positively related and E^- for the corresponding negative. The goal is to find a partition that covers all V and minimizes the number of disagreement pairs, i.e., $+$ pairs that are in different clusters and $-$ pairs that are in the same clusters. When weights exist, the cost of clustering is the sum of W^+_{uv} over all $\{u, v\}$ that are in different clusters plus the sum of W^-_{uv} for all the pairs that are in the same clusters. If all W are between 0 and 1 we have that $W^+_{uv} + W^-_{uv} = 1$ thus, they satisfy the probability constraint and the objective function is formulated as:

$$CC(\mathcal{P}) = \sum_{(u,v),P(u)=P(v)} W_{uv} + \sum_{(u,v),P(u)\neq P(v)} (1 - W_{uv}) \tag{6}$$

which is very similar to the objective function of the pClusterEdit problem. The only difference is that we need to replace $(1 - P_{uv})$ with W^+ and P with $(1 - W^-)$. The similarity of these problems showed that the approximation algorithm for the aforementioned problem can be utilized for the pClusterEdit problem as well.

Algorithm pKwikCluster The first algorithm that we present is the pKwikCluster algorithm that is originated from the KwikCluster algorithm [2] for the weighted correlation clustering problem. The algorithm starts by picking a random node u from the set of V. Then, it places u in the same cluster with all the nodes that are connected to it with probability higher than $0,5$. If u has not an edge with such a probability, it defines a singleton cluster, i.e., a cluster with only one node and no edges. Then the algorithm removes the nodes that belong to the newly formed cluster and repeats the process for the remaining nodes of the graph. The process is depicted in Algorithm 1.

As KwikCluster is a randomized 5-approximation algorithm for the correlation clustering problem we have that pKwikCluster algorithm is a randomized algorithm for the pClusterEdit problem. The algorithm has a time complexity of $O(n)$ since it only depends on the number of nodes and thus, it is linear and easily scalable. Figure 3 shows the result of one iteration of the algorithm for the example graph shown in Figure 2.

Algorithm Furthest The Furthest algorithm is a top-down algorithm that uses a center-based logic for the graph clustering. First, the algorithm assigns

Algorithm 1 pKwikCluster algorithm for probabilistic graph clustering.

repeat
  Choose $u \in V$ **randomly**
  $C(u) \leftarrow u$
  for all $v \in V$ such that $p(u, v) \geq 0.5$ **do**
    $C(u) \leftarrow C(u) \cup v$
  end for
  $V \leftarrow V - C(u)$
until $V = \emptyset$

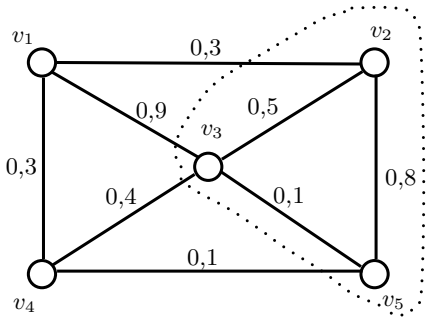

Fig. 3. The probabilistic graph after the first iteration of pKwikCluster. Node v_2 was picked randomly to be the center of the new cluster, and nodes v_5 and v_8, which they have an edge with probability equal or over 0.5 with v_2, were assigned to it.

every node into a single cluster. Then, it determines the pair of nodes that have the smallest probability of having an edge between them and marks these nodes as the centers of two new clusters. The remaining nodes are assigned to the cluster that are connected with the highest probability. This is an iterative process and at the end of each iteration i, a new cluster graph C_i is created with cost $D(\mathcal{G}, C_i)$. If $D(\mathcal{G}, C_i) < D(\mathcal{G}, C_i - 1)$ then the algorithm proceeds to the next iteration. Otherwise, it terminates and $C_i - 1$ is considered as the algorithm's output. The outline of this approach is depicted in Algorithm 2.

In each iteration, the algorithm computes the distance of each node to the existing cluster centers. If the output C consists of k clusters, then the complexity of the algorithm is $O(mk^2)$. However, the complexity can drop to $O(mk)$ if some distance caching is achieved in order not to recompute distances from previous pivots Figure 4 shows the result of the execution of one iteration of the algorithm on the graph shown in Figure 2.

Algorithm Agglomerative The Agglomerative algorithm is a bottom-up procedure for the pClusterEdit problem. First we must define a new term, called the *average edge probability*. The average edge probability between two

Algorithm 2 FURTHEST algorithm for probabilistic graph clustering.

repeat
  $\mathcal{C} \leftarrow \emptyset$, $\mathcal{C} \subset V$ is the set of nodes acting as cluster centers
  for all $u \in V$ **do**
    $C(u) \leftarrow u$
  end for
  First iteration:
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{c_1, c_2\}$, such that $c_1, c_2 \in V - \mathcal{C}$ and $p(c_1, c_2)$ is minimum
  ***i*-th iteration:**
  $\mathcal{C} \leftarrow \mathcal{C} \cup c_i$, such that $c_i \in V - \mathcal{C}$ and the probability between c_i and members of $\mathcal{C}$ is minimum
  for all $u \in V - \mathcal{C}$ **do**
    Assign u to the cluster with which it is more probable to share an edge
    $V \leftarrow V - \{u\}$
  end for
until $V \leftarrow \emptyset$

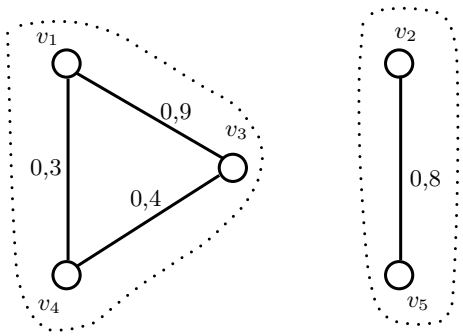

Fig. 4. The probabilistic graph after the first iteration of Furthest. Nodes v_4 and v_5 are picked as the new centers because they have the smallest probability between them. The remaining nodes are assigned to the center which they are closer to. The edges between the two clusters are removed.

clusters V_1 and V_2, is calculated by the formula:

$$\frac{1}{|Vi||Vj|} \sum_{u \in V_i, v \in V_j} P_{uv}. \tag{7}$$

The outline of this technique is depicted in Algorithm 3.

First, the algorithm turns every node into a singleton cluster. The algorithm is iterative. In each iteration i, it is checked If the largest average edge probability is more than $0,5$. If it is, it places the two nodes with the largest average edge probability between them in C_{i-1} into the same cluster, making the cluster graph C_i. If not, it stops and outputs the previous clustering C_{i-1}. The complexity of the algorithm using a naive method is $O(km^2)$, where k is the number of clusters in the final clustering. By using a data structure like a heap for retrieving the closest pair of clusters and placing every edge in it, the complexity reduces to

Algorithm 3 Agglomerative algorithm for probabilistic graph clustering.

```
repeat
    for all u ∈ V do
        u forms a singleton cluster
    end for
    for all pairs of clusters do
        Find pair with maximum average edge probability p_ae
        if p_ae ≥ 0.5 then
            Merge pair of clusters into one and continue
        else
            Stop and display current clustering
        end if
    end for
until p_ae < 0.5
```

$O(km \log m)$. Figure 5 shows the result of one iteration of the algorithm on the simple graph illustrated in Figure 2.

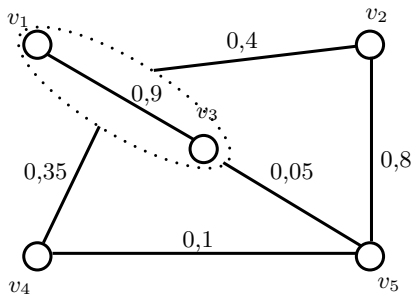

Fig. 5. The probabilistic graph after the first iteration of the Agglomerative algorithm. Nodes v_1 and v_3 are first selected to form a cluster as they have the largest edge probability between them. After that, the new probability is calculated for the newly formed cluster and nodes v_2 v_4 and v_5 using the average edge probability definition.

Based on the results provided AGGLOMERATIVE and FURTHEST do not scale well for large graphs. Also, the PKWIKCLUSTER, which is linear with respect to the number of edges, exhibits the problem of including only edges associated with a probability of existing that is more than 0.5, which is sometimes not desirable or practical. Also, the output clustering of PKWIKCLUSTER does not contain any path longer than two edges, which is usually not desirable. As a result of this limitations, it would be useful to conduct more studies to ensure better clustering quality using scalable algorithmic techniques.

Algorithm Balls The *Balls* algorithm was inspired by another algorithm introduced in [2], developed for the correlation clustering problem. The input of the

algorithm is the matrix of distances between pairs of edges $\mathcal{X}_{uv}$. The algorithm uses a parameter a that is set to a constant for a constant approximation ratio, but it can be changed accordingly depending on the problem at hand.

The Balls algorithm tries to find a set of nodes that are close to each other and far from other nodes. If a set with this characteristic is discovered, it removes it from the graph as a cluster and continues with the rest of the graph. To discover such a set of nodes is not easy, because every subset of nodes of the probabilistic graph $\mathcal{G}$ must be considered. To solve this problem, [20] used the triangle inequality method for distance X_{uv}. The algorithm tries to find clusters that are close (within a ball), to a node u, using the guarantee of triangle inequality, which concludes that if two nodes are close to u, then they are close to each other too. This algorithm tends to output ball-shaped clusters.

The steps of the algorithm are the following: first the nodes are placed in increasing order of the total sum of possibilities incident to each node. At every iteration, the algorithm selects the first unclustered node u of the ordered set. Then, it finds the set of nodes B that have probability a of 0.5 or more to have an edge with u. Then, the average distance $d(n, B)$ of the nodes in B from node u is calculated. If $d(n, B) < a$, then we get a cluster with all the nodes from B and node u. Else, u is a singleton cluster. The space complexity of the algorithm is $O(mn^2)$ for generating the table and its time complexity is $O(m^2)$.

To evaluate these algorithms, experiments were conducted that included the core PPI network. This dataset that we will refer to as COREPPI was provided by [33]. It contains 2708 nodes that represent proteins and 7123 edges that represent protein interactions. The edge probabilities show how likely it is that the interaction actually happens between two proteins. About 20% of the edges have probability over 0.98, while no edge has probability less than 0.27 in the graph. The remaining edge probabilities are uniformly distributed in the remaining range $[0.27, 0.98]$. Finally, the dataset is characterized by power-law degree distribution, short paths and high clustering coefficient.

The experiment that was conducted by [30], aimed to compare the performance of pKwikCluster, Agglomerative, Furthest and Balls algorithms regarding their running time and quality of output. The latter is measured in accordance with the objective function, which for this experiment was the edit distance of the output cluster graphs from the input probabilistic graphs, as we have explained before.

We present in Table 1 the results of the algorithms regarding their running time and the expected edit distance. The best value of the objective function for pKwikCluster is reported, after running it 100 times, since it is a randomized algorithm. Regarding the objective function, aka the expected edit distance we get the best result from Agglomerative with 3420, followed by pKwikCluster with 4194. Regarding running time, pKwikCluster is the fastest as expected with 0.005 seconds running time, since it is linear to the number of the input edges. It must be noted that although the dataset is small, the Furthest algorithm takes more than a minute to finish, showing its scalability problem. Actually, all algorithms except pKwikCluster cannot scale to more than a

few thousand nodes. The REFERENCE clustering has time complexity at least quadratic to the number of nodes because it uses matrix multiplications and the reported results are given from [33].

The known ground truth that was used to validate the results of the algorithms is the MIPS database [37]. MIPS complexes define relationships among proteins of the same complex and this knowledge is utilized only for the **Gcore** graph data and the result is 5380 pairs of proteins. The key difference is that this includes proteins that belong to more than one cluster, but the algorithms output cluster partitions, meaning that every protein can be only in one cluster.

Table 1 shows the number of non-singleton clusters that occurred as a result of each algorithm, and also the True Positive edges, i.e., edges that appear at the clustering and also in MIPS ground truth, the False Positive edges, i.e., edges that appear at the clustering but they do not exist in MIPS ground truth, and finally the False Negatives, i.e., edges that do not appear at the clustering, but they exist in MIPS ground truth. It can be observed that AGGLOMERATIVE and FURTHEST results for non-singleton clusters are the closest to REFERENCE. Also it can be noticed that every algorithm produces quite different results with different trade-offs. For example, PKWIKCLUSTER output includes only 838 TP edges, while REFERENCE contains 1791. However, PKWIKCLUSTER has better results than REFERENCE regarding the FPs.

Table 1. Summary of results.

algorithm	complexity	distance	runtime	# clusters	TP	FP	FN
REFERENCE	N/A	12230	N/A	547	1791	11635	3589
BALLS	$O(m^2)$	4960	8	757	1120	1734	4260
PKWIKCLUSTER	$O(n)$	4194	0.005	757	1120	1734	4260
AGGLOMERATIVE	$O(kmlogm)$	3420	10	542	946	1357	4434
FURTHEST	$O(mk^2)$	4612	150	619	894	2322	4486

2.3 Reliable Clustering

The possible worlds model also comprised the basis of the algorithm proposed in [35] for *reliable clustering* of probabilistic graphs. Reliability in terms of clustering involves the connectivity of the clusters over different possible worlds of the probabilistic graph, and two metrics were proposed for the measurement of the reliability of a clustering. A probabilistic alteration of the classic k-means was proposed based on the two new metrics, which is described below.

In deterministic (i.e., certain) graphs, the connectivity of a clustering is typically measured by the sum of the weights of the edges between each pair of clusters. Using this metric for probabilistic graph clustering where the probabilities of the edges are perceived as weights can be problematic, as it does not take into account the possible world semantics. Another problem raised is that

the connectivity of a cluster could be influenced by nodes belonging to different clusters, instead of being solely dependent on the nodes it contains. Thus, the criteria for probabilistic graph clustering should capture both local and global relationships between the nodes of the graph.

The *standard uncertain graph reliability* proposed in [11] poses such a criterion for reliable probabilistic graph clustering, but it can be very computationally expensive, which lead to the proposal of the *generalized reliability criterion*. In both cases, the reliability $R(C)$ of a set of vertices C, which comprises a cluster subgraph of $\mathcal{G}$, is defined as follows:

$$R(C) = \sum_{G_i \sqsubseteq \mathcal{G}} \Pr(G_i)\mathcal{I}(C, G_i) \tag{8}$$

where $\mathcal{I}(C, G_i)$ is equal to 1 if the cluster C is contained in a connected component in G_i, and 0 otherwise, and G_i is a possible world derived from $\mathcal{G}$.

In this sense, the reliability metric extends the concept of connectivity from deterministic graphs to probabilistic graphs and measures the probability of a probabilistic clustering to maintain its vertices connected in some possible world.

The two basic intuitions behind the generalized reliability criterion are described as *purity* and *size balance*. Given a possible world G of $\mathcal{G}$, purity imposes the constraint that in each connected component of G the number of different clusters should be minimal and one of the clusters should dominate the component. In other words, each connected component in this possible world should be pure, in the sense that the nodes contained in it should exhibit similar characteristics. The purity metric can then be formulated using the cluster label entropy:

$$\mathcal{F}_p = \sum_{G_i \sqsubseteq \mathcal{G}} \Pr(G_i) \sum_{j=1}^{L_i} |CC^{i,j}| H(\bigcup_k CC_k^{i,j}) \tag{9}$$

where $CC^{i,j}$ denote the L_i connected components in G_i, $CC_k^{i,j}$ denote the nodes belonging to the k-th cluster and $H(\bigcup_k CC_k^{i,j})$ is the entropy of cluster labels for the j-th connected component of G_i, defined as:

$$H(\bigcup_k CC_k^{i,j}) = -\sum_{k=1}^{K} \frac{|CC_k^{i,j}|}{|CC^{i,j}|} \log \frac{|CC_k^{i,j}|}{|CC^{i,j}|}. \tag{10}$$

The above criterion biases the clustering algorithm towards the selection of a single clustering containing all or most of the nodes. This is the reason behind the imposition of the second metric, which is described as size balance. The size balance metric is formulated as follows:

$$\mathcal{F}_e = \sum_{G_i \sqsubseteq \mathcal{G}} \Pr(G_i)|V|H(\bigcup_k C_k) = |V|H(\bigcup_k C_k) \tag{11}$$

where $H(\bigcup_k C_k)$ is the entropy of cluster size, defined as:

$$H(\bigcup_k C_k) = -\sum_{k=1}^{K} \frac{|C_k|}{|V|} \log \frac{|C_k|}{|V|} \tag{12}$$

Given the above definitions, the generalized reliability criterion guides the clustering algorithm towards the minimization of $\mathcal{F} = \mathcal{F}_p - \mathcal{F}_e$. This objective is equivalent to the minimization of the following equation:

$$\mathcal{F} = -\sum_{G_i \sqsubseteq \mathcal{G}} \Pr(G_i) \sum_{j=1}^{L_i} \sum_{k=1}^{K} |C_k^{i,j}| \log(\frac{|C_k^{i,j}|}{|C_k|}) \tag{13}$$

Algorithm 4 Coded k-means algorithm for probabilistic graph clustering.

Require: $CC^{i,j}$: connected components, K: number of clusters
  $C_k \leftarrow \emptyset, \forall k$
  for all $v \in V$ **do**
    $C_k \leftarrow C_k \cup v$, k is chosen randomly from $\{1, \ldots, K\}$
  end for
  while $\mathcal{F}$ hasn't converged **do**
    for all i **do**
      for all k **do**
        $c(CC_k^{i,j}) \leftarrow$ *HuffmanCoding*$(CC_k^{i,j})$
      end for
    end for
    $C_k = \emptyset$
    for all $v \in V$ **do**
      for all i **do**
        for all k **do** *CodeLength*$_k(v) += |c(CC_k^{i,f(v,i)})$
        end for
      end for
    end for
    $C_{k^*} \leftarrow C_{k^*} \cup \{v\}$, where k^* minimizes the coding cost *CodeLength*$_k(v)$
  end while
  return $C_k, k = 1, \ldots, K$

A Monte-Carlo sampling technique is then used to create N possible worlds of $\mathcal{G}$, denoted by $G_i, i = 1, \ldots, N$, in order to estimate the reliability criterion. In this fashion, the generalized reliability criterion can be reformulated as follows:

$$\mathcal{F}_s = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} |C_k| H(\bigcup_j CC_k^{i,j}) \tag{14}$$

where $\mathcal{F}_s$ is an unbiased estimator of $\mathcal{F}$.

Moreover, an auxiliary cluster table $\mathcal{T}_i$ is defined for each possible world G_i, in which each row represents a cluster and each column a connected component. Then, $\mathcal{T}_i(k, j)$ contains the $CC_k^{i,j}$ set of vertices, as defined above. The definition of $\mathcal{T}_i$ allows for the utilization of various coding algorithms for the purpose of encoding each table corresponding to each possible world.

With the rows of the auxiliary table representing each of the k clusters and using a fixed coding, each vertex v can be assigned to the row-cluster which reduces its coding cost. The algorithm begins by assigning each sample randomly to one of the clusters. Then, the coding for each cluster distribution is computed, using the Huffman coding or any other type of coding algorithm. After this computation, each vertex v is assigned to the k^*-th cluster, which minimizes its coding cost over all possible worlds. It has been proven that through this iterative process $\mathcal{F}$ converges to a (local) minimum. The outline of the process is depicted in Algorithm 4.

Experiments conducted using the DBLP dataset [49] and the PPI dataset [67] showed that the proposed k-means alteration improved performance in terms of the average coding length per vertex, the average vertex pairwise reliability and the average cluster reliability.

3 Clique Discovery in Uncertain Graphs

3.1 Introduction

One fundamental problem in graph mining is discovering vertices that are densely connected. A *clique* is a set C of vertices, where every vertice is connected by an edge, i.e. for every two vertices u and $v \in C$, there is an edge that connects the two vertices. A *maximal clique* is a clique that is not contained in any other clique. Cliques or maximal cliques can often be considered as cores of structures in graphs.

Discovering cliques in graphs is a fundamental task with many applications such as community detection in social or biological networks [41], genetics study under different circumstances [50] and genome mapping data integration [23].

The problem of finding top-k maximal cliques in a probabilistic graph has been studied in [66]. An application of this problem is found in protein-protein interaction networks, where studies has shown that cliques usually represent cores of protein complexes [4]. Because of the existence of probabilities on the edges of graphs, a set of vertices may not form a maximal clique in all possible worlds. The term *maximal clique probability* is used to describe the possibility that a set of vertices is a maximal clique in all possible worlds of the uncertain graph. A collection of k sets with the largest maximal-clique probabilities is defined as the top-k maximal cliques. Due to the fact that cliques which contain a small number of vertices do not give us useful information, we choose to discover cliques that contain at least s vertices.

The probability of a set of vertices C to be a maximal clique for all the possible worlds of an uncertain graph G, i.e. the *maximal-clique probability* of

C is given by $\sum_{G' \in \Omega} \Pr(G \Rightarrow |G')$ where Ω is the set of graphs for all possible worlds and G' is a world in which C is a maximal clique.

The problem of finding the top-k maximal cliques is NP-hard because it contains the maximal cliques enumeration problem [38], where $k = \infty$, s=1 and G is an exact graph. The problem is defined for an uncertain graph $\mathcal{G}$ and two positive integers k and s. The final output of the problem is a collection F of k sets of vertices of $\mathcal{G}$ with the following qualities: 1) the size of each set in F is equal to or more than s and 2) for any set of cliques $C \in F$, and any other set $C' \notin F$, the maximal-clique probability of C is not less than that of C'.

3.2 Algorithmic Techniques

It has been proven in [66] that the maximal-clique probability of a set of vertices can be found in polynomial time. Considering this, we present the BranchAndBound algorithm that was proposed to find top-k maximal cliques in an uncertain graph.

Given a probabilistic graph $\mathcal{G}$ and G the deterministic graph that occurs if we remove the edge probabilities from $\mathcal{G}$, we consider $\prec$ to be the ascending order of indices of the vertices in $\mathcal{G}$. All of the cliques in G can be organized into a search tree, where the root represents a clique with no vertices, each node contains a unique clique in G and the parent of each non-root node represents another clique C' which has the following properties: 1) $C' \subset C$, 2) $|C| = |C'|+1$ and 3) the only vertex $v \in C - C'$ satisfies $u \prec v$ for each vertex $v \in C$. So the problem of finding top-k maximal cliques in G can be approached as a tree searching problem, by trying to discover k nodes in the search tree that their maximal-clique probability is no less than that of any other node.

The BranchAndBound algorithm utilizes a min-heap H_{topk} of size k which stores the top-k nodes that have been found so far. The criterion that the nodes in H_{topk} are chosen is their maximal-clique probability, $\Pr(mcliq(C))$. The algorithm is initialized by storing a variable γ, which holds the maximal-clique probability of the root and its value is set to zero before the beginning of the algorithm. Also, a max-heap H_{ext} is used for the nodes that have not been examined yet. The nodes are stored in H_{ext} based on their clique probabilities $\Pr(cliq(C))$. At first, the singleton set u is stored into H_{ext}, for every v in G. Finally, the key of the singleton set v in the heap is $\Pr(cliq(v)) = P_V(v)$.

BranchAndBound in its basic form performs a simple best-first branch-and-bound search on the search tree. In each iteration, the algorithm takes the top node of H_{ext}, i.e., the node with the largest clique probability. For this node C, the algorithm applies 4 steps: **pruning**, a **computing**, an **expanding** and **updating**.

- During the **pruning step**, the algorithm checks if the H_{topk} is full and if $\Pr(cliq(C)) \leq \gamma$. If these do not hold true, then there might be top-k maximal cliques in the subtree that has as root C and the following steps of the algorithm must be performed. Otherwise, the aforementioned subtree can be pruned because the maximal-clique probability of each of its nodes is

for sure less than $\Pr(cliq(C))$, so $\Pr(mcliq(C)) \leq \gamma$. This means that C' is not a top-k maximal clique and the algorithm can skip the following steps.
- In the **computing step**, the algorithm computes the maximal-clique probability of C. At first, it finds the set of vertices adjacent to all vertices in C, called $N(C)$. For each vertex $v \in N(C)$, the algorithm gets a new clique $C' = C \cup v$ and computes its $\Pr(cliq(C'))$. After that, it finds the maximal clique probability of C.
- In the **expanding step**, the algorithm checks if C' is a child of C in the search tree. If this holds true and if H_topk is full with $\Pr(cliq(C')) \leq \gamma$, then the subtree at C' can be pruned, otherwise the algorithm inserts C' into H_{ext} as a possible candidate for future searching.
- In the **updating step**, if $|C| > s$, then H_{topk} is updated in two possible ways: 1) If the heap is not full, C is inserted and 2) if it is full and $\Pr(mcliq(C)) > \gamma$, then the root of the heap is removed and C is inserted because the root of H_{topk} cannot be a top-k maximal clique.

The algorithm terminates when either H_{ext} is empty or H_{topk} is full and $\Pr(cliq(C)) \leq \gamma$. The first condition is true when the whole non-pruned tree has been searched and the second condition is true when all subtrees below the nodes of H_{ext} can be pruned safely. The output of the algorithm consists of all the cliques in H_{topk}, which are the maximal cliques of $\mathcal{G}$. The outline of this technique is illustrated in Algorithm 5.

There is also an optimized version of the algorithm which uses techniques to improve efficiency. These techniques are: 1) size-based pruning, 2) look-ahead pruning and 3) anti-monotonicity-based Pruning. The optimized algorithm consists of two phases and can be further studied at [66].

Another approach for solving the maximal clique problem in uncertain graphs has been studied by [62]. The problem is tackled with the use of three metaheuristics, which are based on the popular tabu search (TS) [21], tabu's variation for discovering stable sets called STABULUS [17] and GRASP [14] algotithms. TS prevents getting stuck in local optima by using non-improving moves and the so-called *tabu list* and *tabu tenures* that forbid getting repetitive solutions. STABULUS differs from TS in the sense that it performs the local search on partially impracticable solutions. When the solution is found, the algorithm starts over to discover a better solution. The GRASP algorithm is a multi-start algorithm, in which the solution is found in a randomized greedy way in each iteration and then local search techniques are used to improve the current solution. The greedy part of the algorithm refers to the construction of the *restricted candidate list* which contains some of the best candidates. The algorithms developed using these techniques consider the CVAR (Conditional Value-at-Risk) [51] verification procedure, the robustness during local search moves and the bounds on the solution size.

3.3 Enumerating Maximal Cliques

Another problem, similar to finding the top-k maximal cliques in an uncertain graph, is called *maximal clique enumeration*. This problem has many applications

Algorithm 5 BranchAndBound algorithm for top-k maximal clique search in probabilistic graphs.

repeat
  if $\mathcal{H}_{topk}$ is not full **and** $\Pr(cliq(C)) \leq \tau$ **then**
    for all $v \in V$ **do**
      $\mathcal{H}_{ext} \leftarrow \{v\}, key_{\mathcal{H}_{ext}}(v) \leftarrow \Pr(cliq(v))$
    end for
    $C \leftarrow$ root of $\mathcal{H}_{ext}$
    if $\mathcal{H}_{topk}$ is not full **and** $\Pr(cliq(C)) \leq \tau$ **then**
      $N(C) = \{v : \forall u \in C, (u, v) \in E\}$
      for all $v \in N(C)$ **do**
        $C' \leftarrow C \cup v$
        $\Pr(cliq(C')) \leftarrow \Pr(cliq(C)) * P_V(v) \prod_{u \in C} P_E((u, v)|(u, v)$
      end for
      Compute $\Pr(mcliq(C))$
      for all $v \in N(C)$ **do**
        if $u \prec v, \forall u \in C$ **then**
          C' is a child of C in the search tree
          if $\mathcal{H}_{topk}$ is full **and** $\Pr(cliq(C')) \leq \tau$ **then**
            Prune subtree rooted at C'
          else
            $\mathcal{H}_{ext} \leftarrow \mathcal{H}_{ext} \cup C'$
          end if
        end if
      end for
      if $|C| \geq s$ **then**
        if $\mathcal{H}_{topk}$ is not full **then**
          Insert C into $\mathcal{H}_{topk}$
        else if $\mathcal{H}_{topk}$ is full **and** $\Pr(mcliq(C)) > \tau$ **then**
          Remove root of $\mathcal{H}_{topk}$
          Insert C into $\mathcal{H}_{topk}$
        end if
      end if
    end if
  else
    Prune subtree rooted at C
  end if
until $\mathcal{H}_{ext}$ is empty **or** $\mathcal{H}_{topk}$ is full and $\Pr(cliq(C)) \leq \tau$

in areas where data are more accurately represented by uncertain graphs like social networks [36, 46], email networks [45], protein-protein interaction networks [18] and bioinformatics [64].

The problem of enumerating maximal cliques in uncertain graphs has been studied in [38] [39]. It is used to find robust communities in graphs that contain probabilities. To solve this enumeration problem, an upper and lower bound for the largest number of maximal-cliques within a graph must be discovered.

First, the term a-maximal-clique is defined. An a-maximal clique is a maximal clique with probability at least a (where $0 \leq a \leq 1$)). If W is a set of vertices that form a clique with probability at least a, there is no other W' such that $W \in W'$, where W' a clique with probability at least a.

It is shown [38] that the maximum number of a-maximal-cliques in an uncertain graph with n vertices is $\binom{n}{\lfloor n/2 \rfloor}$. This means that there is an uncertain graph with $\binom{n}{\lfloor n/2 \rfloor}$ uncertain maximal cliques and no uncertain graph can have more than $\binom{n}{\lfloor n/2 \rfloor}$ a-maximal cliques.

The algorithm that is used for enumerating all a-maximal cliques is called MULE (Maximal Uncertain Clique Enumeration). The MULE algorithm is performing a depth-first-search (DFS) of the graph, utilizing optimization techniques for limiting search space and incremental computation of clique probabilities techniques for maximality check. The worst-case running cost of MULE on a graph with n vertices is $O(n2^n)$. However, this cost only appears when the graph is very dense, with much better results in typical graphs. It is proven that the algorithm does not perform an exhaustive search of the graph space and it can be optimized to find only large maximal cliques.

A simplistic approach for a-maximal cliques enumeration in a probabilistic graph $\mathcal{G}$ is to perform DFS with backtracking. At first, there is an empty set of vertices C that is an a-clique and the algorithm starts to add vertices to C, with the restriction that C must always be an a-clique, until no other vertices can be added. When that happens, we have an a-maximal-clique. Then, the algorithm backtracks to add any other possible vertices in C, until all possible search paths are explored.

The MULE algorithm improves this approach of DFS in several ways. First, the additional vertices that can be put in a current a-clique C have the property that are already connected with every vertex of C. So, it is efficient to try to discover this kind of vertices while the algorithm progresses, making the process quicker by not checking if a new vertex can be added to C. As a result, an incremental track of vertices is performed for the extension of C.

Another improvement of the MULE algorithm refers to the clique probability. If a vertex extends C into a clique, it does not mean that it extends C into an a-clique, too. The clique probability of C is decreased by a factor equal to the product of the edge probabilities between v and every vertex in C, when v is added in C. The algorithm calculates the factor by which the clique probability will fall, in $O(1)$ time, by incrementally maintaining the factor for each vertex v under consideration.

The last improvement that the MULE algorithm adds to the DFS approach is the reduction of maximality check. This is achieved by maintaining the set of vertices that can extend C, but will be explored in a different search path. This reduces the time cost from $\theta(n^2)$ to $\theta(n)$.

4 Nearest-Neighbor Search

4.1 Introduction

Another problem that has many applications regarding uncertain graphs is the *nearest-neighbor search* problem. Some of these applications include identifying protein neighbors in protein-protein interaction networks that is useful for possible co-complex membership predictions [33] and possible new interactions [54]. Another example refers to social networks, due to the uncertainty of their nature [54]. In social networks, it is useful to extract information from queries like how many people does a specific person influences the most with their on-line actions. In mobile ad-hoc networks, k-nearest neighbor queries can be used for connectivity applications [6] or for the *probabilistic-routing problem* [19]. The fundamental problem of computing distance functions and processing k-NN queries has long been studied for standard graphs, and it is also very important for the probabilistic graphs.

One way to compute the distance between two nodes v and u in a probabilistic graph is to consider the length of the MOST PROBABLE PATH (MPP). This distance is defined as the length of the path with the highest probability. Given two nodes v and u in a probabilistic graph $\mathcal{G}$, we can consider two alternatives as indicators for the closeness of the nodes. The first is the length of the most probable path and the second is the probability that at least one path exist.

The MPP distance can be easily computed in a probabilistic graph by assuming that the graph is certain and by running the Dijkstra shortest-path algorithm. However, this approach presents several problems. The probability of such a path may be extremely small, and even if it is large, the probability that it is indeed the shortest path can be itself very small. Figure 6 illustrates an example where the lower path which has an arbitrary length n is the most probable path, whereas the direct path between v and u is a little less probable. These problems are solved in [49] by using statistics of the *shortest path distribution*.

4.2 Distance Measures

Considering a possible world G from $\mathcal{G}$, let $d_G(v, u)$ be the shortest path distance between v and u. The distribution $\mathbf{p}_{v,u}$ of the shortest path distance is defined as:

$$\mathbf{p}_{v,u}(d) = \sum_{G|d_G(v,u)=d} \Pr[G] \tag{15}$$

This is the sum of all the possible worlds in which the shortest path distance between v and u is equal to d. Because of the nature of the probabilistic graphs,

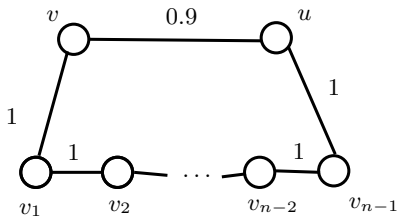

Fig. 6. Example of an arbitrarily long most probable path.

sometimes there are worlds where v and u are belong to disconnected parts of the graph, thus making this distance equal to ∞. So $p_{v,u}(\infty)$ is defined as the total probability of all the worlds that v and u are disconnected.

Below we present different types of distances that will be used for different ways of k-NN pruning, studied in [66]. For these distances, the probabilistic graph $G = (V, E, P, W)$ and two any two nodes v and u will be used to define them.

Median Distance : The *Median Distance* $d_M(v, u)$ is the shortest path distance among all possible worlds. The median distance can be infinite for some pairs v and u and k-th order statistic are held as well. Formally:

$$d_M(v,u) = \arg\max_D \left\{ \sum_{d=0}^{D} \mathbf{p}_{v,u}(d) \leq \frac{1}{2} \right\} \tag{16}$$

The exact calculation of this distance is difficult to be executed, since it involves executing a point-to-point shortest-path algorithm in every possible world and taking the median. To solve this problem, we approximate the value of the median distance by sampling, using the Chernoff bound [49].

Majority Distance : The *Majority Distance* $d_J(v, u)$ is defined as the most probable shortest path distance. For weighted graphs, this distance has meaning if the weights come from a discrete domain. Formally:

$$d_J(v,u) = \arg\max_d \mathbf{p}_{v,u}(d) \tag{17}$$

Expected Reliable Distance : For this distance, only the possible worlds containing a path between v and u are considered. The *Expected Reliable Distance* $d_{ER}(v, u)$ is defined as the shortest path distance in all worlds in which there exists a path between v and u. This can be described formally as:

$$d_{ER}(v,u) = \sum_{d|d<\infty} \frac{\mathbf{p}_{v,u}(d)}{1 - \mathbf{p}_{v,u}(\infty)} \tag{18}$$

Computing this distance is considered a $\#P$-hard problem, as a generalization of the reliability problem [60].

Another promising distance function has been defined, which is based on the concept of *random walks*. The difference between this approach and the shortest

path approach is that it considers all possible paths, whilst the shortest path distance relies on one path only. Also, the random walk approach, as its name suggests, uses random instead of optimal choices. In standard graphs, random walks have been studied in [52]. The random walk distance function is inspired by the *Individual Page Rank* (IPR) concept [15]. For deterministic graphs, in a IPR walk that starts from a node v, it always teleports back to v, instead of teleporting to any node in the graph. For probabilistic graphs, IPR is defined considering a weighted probabilistic graph $\mathcal{G} = (V, E, W, P)$, where W represents the *proximity* between nodes in the graph.

The parameterization of the random walk is based on a source node s and a teleportation probability a. The walk starts at source node s and possible world G_0, which is sampled online according to P. At step i, we are at the node u_i and in the world G_i. At this step there are 2 choices: either follow an active edge with probability $1 - a$, or teleport to s with probability a. If there are no outgoing edges we stay at the same node. This process is called probabilistic random walk and the *random walk distance* is defined as the inverse of the stationary probability of u of the probabilistic random walk with starting node s.

4.3 Algorithms

Different k-NN algorithms have been developed based on the distance functions described above. First, let us present the definition of the k-NN problem. Given $\mathcal{G} = (V, E, P, W)$ a source node s, a probabilistic distance d_P, and a positive integer k corresponding to the number of neighbors, find the set of nodes $T_k(s) = (t_1, ..., t_k)$ for which the distance $d_P(s, t_i)$ is less or equal to the distance $d_P(s, t)$ for any other node $t \in V \setminus T_k(s)$. The problem lies in the computation of the set $T_k(s)$ without having to compute the distance $d_P(s, t)$ for all nodes $t \in V$, as this can be computationally expensive especially for large graphs.

Using the median distance as defined above, involves truncating the distribution $\mathbf{p}_{v,u}$ so as to contain distance values smaller than a given value D. The remaining values (for $d > D$ in the original distribution) are redistributed and concentrated exactly at $d = D$. Let the D-truncated distribution be $\mathbf{p}_{D,v,u}(d)$, $d_{D,M}(v, u)$ be the median distance obtained from it and $d_M(v, u)$ be the actual median distance obtained from the original non-truncated distribution. It has been proven that if $d_{D,M}(v, u_1) < d_{D,M}(v, u_2)$ for any two nodes u_1, u_2, then $d_M(v, u_1) < d_M(v, u_2)$. In other words, the median distance obtained from the truncated distribution preserves distance relationships between the nodes of the graph. Thus, finding the set of nodes $T_k(v) = u_1, \ldots, u_k$ for which $d_{D,M}(v, u_i) \leq d_{D,M}(v, u)$ for $i = 1, \ldots, k$ and $v \in V - T_k(v)$ comprises a solution to the k-NN query for v.

The probability distribution is then approximated using sampling techniques. The algorithm based on the median distance begins by applying Dijkstra's algorithm with v as the starting node. Once a node gets traversed it is never been visited again, and to visit a new node, a sample of its outgoing nodes is generated. The algorithm terminates when it reaches a node whose distance is greater than D. Then, the sampled distribution of all the visited nodes is updated or

instantiated. Performing the above steps r times yields r samples of the distribution. If the distribution of a node u reaches half of its mass, the node is added to the k-NN query result. Algorithm 6 presents the outline of the median-based variation described above.

Algorithm 6 Median distance k-NN algorithm for nearest-neighbor queries in probabilistic graphs.

Require: $s \in V$: starting node, r: number of samples, k: number of neighbors, γ: distance increment
  $T_k \leftarrow \emptyset$
  $D \leftarrow 0$
  Run Dijkstra algorithm r times, from s
  while $|T_k| < j$ **do**
    $D \leftarrow D + \gamma$
    for $i \leftarrow 1 : r$ **do**
      Visit nodes in the i-th Dijkstra's execution until distance D is reached
      for all $t \in V$ visited by Dijkstra **do**
        update $\tilde{\mathbf{p}}_{D,s,t}$
      end for
    end for
    for all $t \notin T_k : \tilde{\mathbf{p}}_{D,s,t}$ exists **do**
      if $\tilde{\mathbf{p}}_{D,s,t} < D$ **then**
        $T_k \leftarrow T_k \cup \{t\}$
      end if
    end for
  end while
  return T_k

Using the majority distance defined previously, a variation of the k-NN algorithm similar to the one described above is obtained. One difference lies in the way the k neighbors are collected. In the median distance version, once the truncated distribution of a node u reaches 50% of its mass, it is added to the solution. In the majority distance alternative, the condition ensuring that d_1 will be the majority distance is:

$$\tilde{\mathbf{p}}_{D,v,u}(d_1) \geq \frac{r - r_u}{r} \tag{19}$$

where d_1 is the current majority value in the sampled distribution $\tilde{\mathbf{p}}_{D,v,u}(d)$ and r_u is the number of times the node u has been visited during the r Dijkstra traversals.

Experiments conducted on the PPI and DBLP datasets showed that the above proposed algorithms showed improved performance in terms of true positives versus false positive rates, in comparison to random walk and reliability-based algorithms.

5 Core Decomposition

The *core decomposition* is a useful tool for performing a wide range of graph mining tasks. One of its advantages against other methods is that it can be computed efficiently (linearly) in the size of the input graph. The core decomposition is related to the problem of discovering *dense subgraphs*, like *cliques*, *lambda-sets*, etc, most of which are NP-hard or they have high computational complexity and therefore, the ability to solve this problem in linear time is very appealing.

The k-core of a graph is defined as a maximal subgraph in which every vertex is connected to at least k other vertices within that subgraph [8]. The set of all these k-cores in a graph G forms the *core decomposition* of G [53]. While this problem is solved in linear time in deterministic graphs, it does not mean that it is solved this efficiently in probabilistic ones, due to their nature.

Core decomposition has been used to speed-up the computation of other problems with purpose of finding dense subgraphs in deterministic graphs. Some examples of this contribution are in the maximal-clique discovery problem [13], the *densest subgraph problem* [31] and the *densest at-least-k-subgraph* problem [3]. A core-decomposition solution developed for uncertain graphs would provide a natural extension of the aforementioned applications. Some applications of core decomposition in probabilistic graphs include *influence maximization* [61] and *task-driven-team-formation* [24].

In *influence maximization*, we have edges with probabilities that represent influence between nodes, and the goal is to find the nodes, e.g., users in a social network, that have the highest influence over a large number of users. To solve this problem, a greedy algorithm [22] has been proposed that requires a number of Monte Carlo simulations, which is computationally expensive. By using the core decomposition technique, this process can be executed more efficiently.

In *task-driven team formation*, the input is a collaboration graph $G = (V, E, r)$, where vertices represent individuals and edges represent topic(s) of past collaboration with the use of the probabilistic model r. Given a query $Q = (T, V)$, where T is a set of terms used to describe a new task and V is a set of individuals represented by vertices, the goal is to find the best set of vertices $A \subset Q$, to perform the task T. Due to the nature of the problem, we have a single probability for each edge pair $(v, u) \in E$, that (v, u) collaborate on task T. That gives a great opportunity for applying the core decomposition in order to determine the best set A.

Using the possible worlds model, the core decomposition concept can be applied to probabilistic graphs as well. Each vertex v has a probability of being a part of a k-core $\mathcal{H}$ which is defined as the probability that v has a degree greater or equal to k in $\mathcal{H}$. Then, a threshold η is applied to determine which vertices actually belong to the k-core based on this probability.

Therefore, given a probabilistic graph and a value for η, the (k, η)-core of $\mathcal{G}$ is defined as the maximal subgraph $\mathcal{H}$ such that the probability that each vertex belonging to it has a degree greater than or equal to k is greater than or equal to η. Thus, the (k, η)-core decomposition of a probabilistic graph is defined as the problem of finding the set of all (k, η)-cores of the graph. The η-degree of

Algorithm 7 (k, η)-cores algorithm for core decomposition of probabilistic graphs.

Require: η
  for all $v \in V$ **do**
    compute the η-degree of v (Equation 20)
  end for
  $\mathbf{c} \leftarrow \emptyset, \mathbf{d} \leftarrow \emptyset, \mathbf{D} \leftarrow [\emptyset, \ldots, \emptyset]$
  for all $v \in V$ **do**
    $\mathbf{d}[v] \leftarrow \eta - deg(v)$
    $\mathbf{D}[\eta - deg(v)] \leftarrow \mathbf{D}[\eta - deg(v)] \cup \{v\}$
  end for
  for all $k = 0, 1, \ldots, n$ **do**
    while $\mathbf{D}[k] \neq \emptyset$ **do**
      $\mathbf{D}[k] \leftarrow \mathbf{D}[k] - \{v\}$, random $v \in \mathbf{D}[k]$
      $\mathbf{c}[v] \leftarrow k$
      for all $u : (u, v) \in E, \mathbf{d}[u] > k$ **do**
        recompute η-$deg(u)$
        $\mathbf{D}[\mathbf{d}[u]] \leftarrow \mathbf{D}[\mathbf{d}[u]] - \{u\}$
        $\mathbf{D}[\eta\text{-}deg(u)] \leftarrow \mathbf{D}[\eta\text{-}deg(u)] \cup \{u\}$
        $\mathbf{d}[u] \leftarrow \eta$-$deg(u)$
      end for
      $V \leftarrow V - \{v\}$
    end while
  end for
  return c, n-dimensional vector containing the η-core number of each vertex in $\mathcal{G}$

each vertex in $\mathcal{G}$ is defined as:

$$\eta - deg(v) = \max\{k \in [0, \ldots, d_v] | Pr[deg(v) \geq k] \geq \eta\} \tag{20}$$

The outline of the (k, η)-core technique is outlined in Algorithm 7.

6 Distributed Mining of Uncertain Graphs

Having described some of the most important research contributions in uncertain graph mining, we center our focus on interesting issues related to uncertain graph mining in a distributed setting. Based on our previous discussion, it is evident that mining uncertain graphs is not trivial. In fact, it is considerably harder than mining conventional (i.e., certain) graphs. Therefore, efficient techniques are required to enable fast processing and provide meaningful results. A promising research direction with practical importance is the use of distributed engines that can utilize multiple resources (e.g., processors, memory, disks) aiming at mining massive datasets requiring significant space requirements. Nowadays, clusters running Spark or Hadoop are used consistently to provide the necessary functionality and performance.

Although many graph mining tasks have been parallelized for the case of conventional graphs, to the best of the authors' knowledge distributed algorithms

for uncertain graphs have appeared only recently and for a limited set of problems. More specifically, in [10] the authors study reachability query processing in large uncertain graphs using distributed algorithms. However, there is a large set of mining tasks that can benefit from distributed computing. In the sequel we discuss briefly interesting problems related to uncertain graph mining in association with the corresponding difficulties that are raised due to the distributed setting.

Assume that the input data represents a single massive graph $G(V, E, p)$ with edge uncertainty. To facilitate distributed processing the first step is to split the input graph into several partitions. Partitions are distributed across cluster nodes to enable parallel execution of specific tasks. Graph partitioning [34] is an interesting problem on its own, and many algorithms have been proposed. For example, METIS [27] and FENNEL [58] are two very promising graph partitioning algorithms. Existing algorithms either work on unweighted or unweighted graphs. However, these techniques are not equipped with tools to handle uncertain graphs. Moreover, graph partitioning algorithms supported by distributed engines such as Spark, apply a hash-based approach where edges are distributed across machines in random order. However, many graph mining algorithms can benefit significantly by sophisticated partitioning algorithms. Therefore, there is a need for effective uncertain graph partitioning, taking into account the existential probabilities assigned to the edges of the graph. The way graph partitions are defined has a significant impact on mining tasks like node reachability and community detection.

Graph summarization [57] is a very interesting directions aiming at reducing the space requirements of massive graphs and at the same time keeping graph properties in order to provide answers to mining tasks using the summary. Graph summaries are either lossless (no information loss) or lossy (some information is lost having an impact on accuracy). To the best of the authors' knowledge the problem of uncertain graph summarization has not been addressed yet. Evidently, there is a need to compute graph summaries as fast as possible which means that distributed techniques may offer significant advantage over centralized approaches.

An important graph mining task is the discovery of the set of triangles. A triangle among nodes u, v and w is formed if all edges (u, v), (v, w) and (u, w) are present in the graph. It has been shown that triangles are important primitive structures that are essential in community formation [59] and many efficient algorithms have been proposed. To speed up processing, parallel algorithms for triangle discovery have been proposed [56]. However, in an uncertain graph it is natural to ask for the triangles with the maximum existential probability. In case of edge independence, the existential probability of a triangle is equal to the product of the probabilities of the corresponding edges. This corresponds to a top-k query, where k is the desired number of triangles. It is expected that distributed algorithms for top-k triangle discovery will offer significant performance improvement over centralized approaches.

Finally, a different set of problems may be defined in the case of a massive collection of small probabilistic graphs [42]. Instead of working with a single massive graph, a very large collection of small graphs is given as input. In such a case, there is a need to apply graph mining techniques across graphs. For example, clustering in this scenario involves the grouping of graphs in clusters, which means that meaningful similarity measures for graphs must be used. Moreover, the frequent pattern mining becomes an extremely difficult problem since not only we have to deal with *subgraph isomorphism*, but we are facing additional challenges due to the probabilistic nature of the graphs. It is expected that distributed techniques will have a significant impact on the performance of the algorithms that solve similar problems, since graphs may be distributed in several partitions and operations may be parallelized to enable more efficient execution of mining tasks.

7 Conclusions

In many real-life applications, graphs are annotated with existential probabilities on the edges, leading to the concept of uncertain or probabilistic graph. This means that each edge appears in the graph with a specific probability. Although this extension seems quite simple, it poses significant computational challenges in graph mining tasks that are easy to apply on conventional (deterministic) graphs.

The possible world semantics has proven to be a useful tool to mine probabilistic graphs, allowing efficient solutions to be developed in problems like graph clustering, clique discovery, k-nearest neighbor discovery and core decomposition. Although the possible world model implicates the existence and computation of $2^{|E|}$ deterministic graphs (in the worst case) for a single probabilistic graph $\mathcal{G} = (V, E, P)$, the algorithms proposed in the literature have been shown to produce effective results in their respective areas, while preserving the notion of uncertainty.

In this survey, we covered some of these methods focusing on the basic concepts and their associated algorithmic techniques. Taking into account that graph mining tasks applied to probabilistic graphs are characterized by increased complexity, it is natural to use distributed architectures towards more efficient processing. However, applying distributed processing to probabilistic graphs seems to be a challenging task by itself, and we argue that such techniques should be applied in order to facilitate faster execution times and to enable scalable data mining that could be applied to massive amounts of probabilistic graph data.

References

1. C. C. Aggarwal and H. Wang. *Managing and Mining Graph Data.* Springer, 2010.
2. N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.

3. R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 25–37. Springer, 2009.
4. G. D. Bader and C. W. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC bioinformatics*, 4(1):2, 2003.
5. N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.
6. S. Biswas and R. Morris. Exor: opportunistic multi-hop routing for wireless networks. *ACM SIGCOMM Computer Communication Review*, 35(4):133–144, 2005.
7. F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *KDD*, pages 1316–1325, 2014.
8. F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1316–1325. ACM, 2014.
9. U. Brandes, M. Gaertler, and D. Wagner. Engineering graph clustering: Models and experimental evaluation. *ACM Journal of Experimental Algorithmics*, 12(1.1):1–26, 2007.
10. Y. Cheng, Y. Yuan, L. Chen, G. Wang, C. Giraud-Carrier, and Y. Sun. Distr: A distributed method for the reachability query over large uncertain graphs. *IEEE Trans. Parallel Distrib. Syst.*, 27(11):3172–3185, Nov. 2016.
11. C. J. Colbourn and C. Colbourn. *The combinatorics of network reliability*, volume 200. Oxford University Press New York, 1987.
12. D. J. Cook and L. B. Holder. *Mining Graph Data*. John Wiley & Sons, 2006.
13. D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation*, pages 403–414. Springer, 2010.
14. T. A. Feo and M. G. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71, 1989.
15. D. Fogaras and B. Rácz. Towards scaling fully personalized pagerank. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 105–117. Springer, 2004.
16. S. Fortunato. Community detection in graphs. *Physics Reports*, 483(3):75–174, 2010.
17. C. Friden, A. Hertz, and D. de Werra. Stabulus: a technique for finding stable sets in large graphs with tabu search. *Computing*, 42(1):35–44, 1989.
18. A.-C. Gavin, M. Bösche, R. Krause, P. Grandi, M. Marzioch, A. Bauer, J. Schultz, J. M. Rick, A.-M. Michon, C.-M. Cruciat, et al. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature*, 415(6868):141–147, 2002.
19. 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 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1721–1729. IEEE, 2007.
20. A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):4, 2007.
21. F. Glover. Tabu searchpart ii. *ORSA Journal on computing*, 2(1):4–32, 1990.
22. A. Goyal, W. Lu, and L. V. Lakshmanan. Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web*, pages 47–48. ACM, 2011.
23. E. Harley, A. Bonner, and N. Goodman. Uniform integration of genome mapping data using intersection graphs. *Bioinformatics*, 17(6):487–494, 2001.

24. X. Huang, H. Cheng, and J. X. Yu. Attributed community analysis: Global and ego-centric views. *Data Engineering*, page 29, 2016.
25. X. Huang, W. Lu, and L. V. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *SIGMOD*, pages 77–90, 2016.
26. R. Jin, L. Liu, C. Aggarwal, and Y. Shen. Reliable clustering on uncertain graphs. In *ICDM*, pages 459–468, 2012.
27. G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, Supercomputing '96, Washington, DC, USA, 1996. IEEE Computer Society.
28. A. Khan, F. Bonchi, A. Gionis, and F. Gullo. Fast reliability search in uncertain graphs. In *EDBT*, pages 535–546, 2014.
29. G. Kollios, M. Potamias, and E. Terzi. Clustering large probabilistic graphs. *IEEE Trans. on Knowl. and Data Eng.*, 25(2):325–336, Feb. 2013.
30. G. Kollios, M. Potamias, and E. Terzi. Clustering large probabilistic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 25(2):325–336, 2013.
31. G. Kortsarz and D. Peleg. Generating sparse 2-spanners. *Journal of Algorithms*, 17(2):222–236, 1994.
32. F. Kovács, C. Legány, and A. Babos. Cluster validity measurement techniques. In *6th International symposium of hungarian researchers on computational intelligence*. Citeseer, 2005.
33. N. J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, S. Pu, N. Datta, A. P. Tikuisis, et al. Global landscape of protein complexes in the yeast saccharomyces cerevisiae. *Nature*, 440(7084):637–643, 2006.
34. D. LaSalle, M. M. A. Patwary, N. Satish, N. Sundaram, P. Dubey, and G. Karypis. Improving graph partitioning for modern graphs and architectures. In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, IA3 '15, pages 14:1–14:4, New York, NY, USA, 2015. ACM.
35. L. Liu, R. Jin, C. Aggarwal, and Y. Shen. Reliable clustering on uncertain graphs. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 459–468. IEEE, 2012.
36. J. Mcauley and J. Leskovec. Discovering social circles in ego networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(1):4, 2014.
37. H.-W. Mewes, C. Amid, R. Arnold, D. Frishman, U. Güldener, G. Mannhaupt, M. Münsterkötter, P. Pagel, N. Strack, V. Stümpflen, et al. Mips: analysis and annotation of proteins from whole genomes. *Nucleic acids research*, 32(suppl 1):D41–D44, 2004.
38. A. Mukherjee, P. Xu, and S. Tirthapura. Enumeration of maximal cliques from an uncertain graph. *IEEE Transactions on Knowledge and Data Engineering*, 2016.
39. A. P. Mukherjee, P. Xu, and S. Tirthapura. Mining maximal cliques from an uncertain graph. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 243–254. IEEE, 2015.
40. M. E. Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.
41. G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.
42. O. Papapetrou, E. Ioannou, and D. Skoutas. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *Procedings of EDBT*, pages 355–366, 2011.

43. P. Parchas, F. Gullo, D. Papadias, and F. Bonchi. The pursuit of a good possible world: Extracting representative instances of uncertain graphs. In *SIGMOD*, pages 967–978, 2014.
44. P. Parchas, F. Gullo, D. Papadias, and F. Bonchi. Uncertain graph processing through representative instances. *ACM Trans. Database Syst.*, 40(3):20:1–20:39, 2015.
45. N. Pathak, S. Mane, and J. Srivastava. Who thinks who knows who? socio-cognitive analysis of email networks. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 466–477. IEEE, 2006.
46. J. Pattillo, N. Youssef, and S. Butenko. Clique relaxation models in social network analysis. In *Handbook of Optimization in Complex Networks*, pages 143–162. Springer, 2012.
47. J. Pfeiffer and J. Neville. Methods to determine node centrality and clustering in graphs with uncertain structure. In *ICWSM*, 2011.
48. M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest neighbors in uncertain graphs. *Proc. VLDB Endow.*, pages 997–1008, 2010.
49. M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest neighbors in uncertain graphs. *Proceedings of the VLDB Endowment*, 3(1-2):997–1008, 2010.
50. O. Rokhlenko, Y. Wexler, and Z. Yakhini. Similarities and differences of gene expression in yeast stress conditions. *Bioinformatics*, 23(2):e184–e190, 2007.
51. M. Rysz, M. Mirghorbani, P. Krokhmal, and E. L. Pasiliao. On risk-averse maximum weighted subgraph problems. *Journal of Combinatorial Optimization*, 28(1):167–185, 2014.
52. P. Sarkar, A. W. Moore, and A. Prakash. Fast incremental proximity search in large graphs. In *Proceedings of the 25th international conference on Machine learning*, pages 896–903. ACM, 2008.
53. S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.
54. P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link discovery in graphs derived from biological databases. In *International Workshop on Data Integration in the Life Sciences*, pages 35–49. Springer, 2006.
55. R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1):173–182, 2004.
56. K. Tangwongsan, A. Pavan, and S. Tirthapura. Parallel triangle counting in massive streaming graphs. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 781–786, New York, NY, USA, 2013. ACM.
57. Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 567–580, New York, NY, USA, 2008. ACM.
58. C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic. Fennel: Streaming graph partitioning for massive scale graphs. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 333–342, New York, NY, USA, 2014. ACM.
59. C. E. Tsourakakis. A novel approach to finding near-cliques: The triangle-densest subgraph problem. *CoRR*, abs/1405.1477, 2014.
60. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

61. Y. Wu, Y. Yang, F. Jiang, S. Jin, and J. Xu. Coritivity-based influence maximization in social networks. *Physica A: Statistical Mechanics and its Applications*, 416:467–480, 2014.
62. O. Yezerska, S. Butenko, and V. L. Boginski. Detecting robust cliques in graphs subject to uncertain edge failures. *Annals of Operations Research*, pages 1–24, 2016.
63. Y. Yuan, G. Wang, L. Chen, and H. Wang. Efficient subgraph similarity search on large probabilistic graph databases. *Proc. VLDB Endow.*, pages 800–811, 2012.
64. B. Zhang, B.-H. Park, T. Karpinets, and N. F. Samatova. From pull-down data to protein interaction networks and complexes with biological relevance. *Bioinformatics*, 24(7):979–986, 2008.
65. Z. Zou. Polynomial-time algorithm for finding densest subgraphs in uncertain graphs. In *Procedings of MLG Workshop*, 2013.
66. Z. Zou, J. Li, H. Gao, and S. Zhang. Finding top-k maximal cliques in an uncertain graph. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 649–652. IEEE, 2010.
67. Z. Zou, J. Li, H. Gao, and S. Zhang. Mining frequent subgraph patterns from uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering*, 22(9):1203–1218, 2010.
68. Z. Zou and R. Zhu. Truss decomposition of uncertain graphs. *Knowledge and Information Systems*, pages 1–34, 2016.