Improved Algorithms for Maximal Clique Search in Uncertain Networks

Rong-Hua Li[†*], Qiangqiang Dai[§*], Guoren Wang[†], Zhong Ming[§*], Lu Qin[‡], Jeffrey Xu Yu[#]

[†]*Beijing Institute of Technology, Beijing, China;* [§]*Shenzhen University, Shenzhen, China;*
[*]*National Engineering Laboratory for Big Data System Computing Technology;*
[‡]*University of Technology, Sydney, Australia;* [#]*The Chinese University of Hong Kong, Hong Kong, China;*
lironghuascut@gmail.com; qiang56734@163.com; wanggrbit@126.com;
mingz@szu.edu.cn; Lu.Qin@uts.edu.au; yu@se.cuhk.edu.hk

***Abstract*—Enumerating maximal cliques from an uncertain graph is a fundamental problem in uncertain graph analysis. Given an uncertain graph $\mathcal{G}$, a set of nodes C in $\mathcal{G}$ is a maximal (k,τ)-clique if (1) $|C| > k$ and C is a clique with probability at least τ, and (2) C is a maximal node set meeting (1). The state-of-the-art algorithm for enumerating all maximal (k,τ)-cliques is very costly when handling large uncertain graphs, as its time complexity is proportional to 2^n where n is the number of nodes in the uncertain graph. To overcome this issue, we propose two new core-based pruning algorithms to reduce the uncertain graph size without missing any maximal (k,τ)-clique. We also develop a novel cut-based optimization technique to further improve the pruning performance of the core-based pruning algorithms. Based on these pruning techniques, we propose an improved algorithm to enumerate all maximal (k,τ)-cliques, and a new algorithm with several novel upper-bounding techniques to compute one of maximum (k,τ)-cliques from the pruned uncertain graph. The results of extensive experiments on six real-world datasets demonstrate the efficiency and effectiveness of the proposed algorithms.**

I. Introduction

Real-world networks, such as social networks, biological networks, and communication networks often consist of cohesive subgraph structures. Mining cohesive subgraphs from a network is a fundamental problem in network analysis which has attracted much attention in the database and data mining communities [1], [2], [3], [4], [5], [6]. Perhaps the most elementary and widely-used cohesive subgraph model is the maximal clique model which has also been extensively studied in the literature [7], [8], [9]. Enumerating all maximal cliques from a network has numerous applications, including finding overlapping communities from social networks [10], detecting social hierarchy [11], and identifying protein complexes in protein-protein interaction (PPI) network [12].

Many real-world networks, however, are uncertain in nature where each edge is associated with a probability, representing the likelihood of the existence of the edge. The uncertain graph has been widely used in many applications to model or express the inferred relationship of the nodes in a network. Examples of such networks consist of PPI networks with experimentally inferred links [13], social networks with inferred influence [14], and sensor networks with uncertain connectivity links [15]. Many cohesive subgraph mining problems have recently been studied in the context of uncertain graphs. Notable examples include the core decomposition problem [16], the truss decomposition problem [17], and the maximal clique enumeration problem [18], [19].

In this paper, we focus on the problem of mining maximal cliques from an uncertain graph. In [18], Mukherjee et al. proposed a maximal (k,τ)-clique model to represent a maximal clique in an uncertain graph. Specifically, for an uncertain graph $\mathcal{G}$, a set of nodes C is called a maximal (k,τ)-clique if (1) $|C| > k$ and C is a clique with probability at least τ, and (2) C is a maximal node set satisfying (1). As shown in [18], the number of maximal (k,τ)-cliques in an uncertain graph can be exponentially large, thus the problem of enumerating all maximal (k,τ)-cliques is intractable. To enumerate all maximal (k,τ)-cliques, Mukherjee et al. developed a backtracking enumeration algorithm based on a classic set enumeration technique [20]. The main defect of their algorithm is that its time complexity is proportional to 2^n where n is the number of nodes in $\mathcal{G}$, thus it is very costly when handling large uncertain graphs. To speed up their algorithm, we develop a new algorithm to enumerate all maximal (k,τ)-cliques based on several non-trivial and powerful pruning techniques. The appealing feature of our algorithm is that its time complexity depends on $2^{n'}$ where $n' < n$ denotes the number of nodes in a *small subgraph* of $\mathcal{G}$, thus it can be used to handle large uncertain graphs. We also present an algorithm to compute one of maximum (k,τ)-cliques in $\mathcal{G}$ based on several novel upper bounds. More specifically, we make the following contributions.

New DP algorithm. To efficiently enumerate all maximal (k,τ)-cliques, we first apply an existing cohesive subgraph model in uncertain graph called (k,τ)-core [16] to prune the uncertain graph without missing any maximal (k,τ)-clique. To compute the (k,τ)-core, we propose a new dynamic programming (DP) algorithm which reduces the time complexity of the state-of-the-art DP algorithm [16] from $O(md_{\max})$ to $O(m\delta)$, where m, $d_{\max}$ and δ denote the number of edges, the maximum degree and the degeneracy [9] of the uncertain graph respectively. Note that δ is often much smaller than $d_{\max}$ in most real-world graphs [9], thus our new DP algorithm is faster than the state-of-the-art algorithm in practice.

Novel pruning techniques. Besides the (k,τ)-core pruning rule, we also propose a new notion called (Top_k,τ)-core to prune the uncertain graph without missing any maximal (k, τ)-clique. We show that the (Top_k, τ)-core pruning is more effective than the (k, τ)-core pruning rule. We develop an efficient algorithm with $O(m \log_2(d_{\max}))$ time complexity to calculate the (Top_k, τ)-core. In addition, we also propose a novel cut-based optimization technique to further improve the pruning performance of the core-based pruning rules.

New maximal clique algorithms. Equipped with the above pruning techniques, we develop an improved algorithm to enumerate all maximal (k, τ)-cliques. Unlike the existing algorithm [18], the worst-case time complexity of our algorithm is proportional to $2^{n'}$ where n' is no larger than the number of nodes in the (Top_k, τ)-core, which is typically much smaller than the size of the original uncertain graph. We also propose a new algorithm with several carefully-designed upper-bounding techniques to compute one of maximum (k, τ)-cliques in an uncertain graph.

Experimental evaluation. We conduct extensive experiments to evaluate the proposed algorithms using six real-world graphs. The results show that our new DP algorithm is up to three orders of magnitude faster than the state-of-the-art DP algorithm [16] for computing the (k, τ)-core on large uncertain graphs. We show that the proposed maximal (k, τ)-clique enumeration algorithm significantly outperforms the state-of-the-art algorithm [18] for enumerating all maximal (k, τ)-cliques. For example, on the DBLP[1] dataset (1,843,614 nodes and 8,350,259 edges), our algorithm takes less than 100 seconds to enumerate all maximal (k, τ)-cliques with all parameter settings, while the state-of-the-art algorithm consumes nearly 1,000 seconds. To compute one of maximum (k, τ)-cliques, the proposed algorithm is two orders of magnitude faster than the state-of-the-art algorithm [21] on large uncertain graphs. In addition, we also conduct a case study on a real-world PPI network to evaluate the effectiveness of our maximal (k, τ)-clique enumeration algorithm. The results indicate that our algorithm is much more effective than the state-of-the-art algorithm to detect protein complexes in PPI networks.

Organization. Section II formulates the maximal (k, τ)-clique search problems. The (k, τ)-core and (Top_k, τ)-core pruning algorithms, as well as the cut-based optimization technique are proposed in Section III. Section IV presents the maximal (k, τ)-clique enumeration algorithm. The maximum (k, τ)-clique search algorithm is shown in Section V. The experimental results are reported in Section VI. We review the related work in Section VII, and conclude this work in Section VIII.

II. Problem Definition

Let $\mathcal{G} = (V, E, p)$ be an uncertain graph, where V denotes the set of nodes, E is the set of edges, and $p : E \to (0, 1]$ is a function that assigns a probability of existence to each edge $e \in E$. We denote by $n = |V|$ and $m = |E|$ the number of nodes and edges respectively. For a node subset $C \subseteq V$, $\mathcal{G}_C = (C, E_C, p)$ is called an induced uncertain subgraph if $E_C = \{(u, v) | u \neq v, (u, v) \in E, u, v \in C\}$.

Denote by $\tilde{G} = (V, E)$ a deterministic graph of $\mathcal{G}$ which is obtained by ignoring all probabilities associated with the edges in $\mathcal{G}$. Let $N_u(\tilde{G})$ be the set of neighbors of u in $\tilde{G}$, and $d_u(\tilde{G}) \triangleq |N_u(\tilde{G})|$ be the degree of u in $\tilde{G}$. In the deterministic graph $\tilde{G}$, the k-core is the maximum subgraph of $\tilde{G}$ in which each node has degree no less than k [22]. The core number of a node u in $\tilde{G}$, denoted by c_u, is the largest k such that there is a k-core containing u [22], [1]. The maximum core number among all nodes in $\tilde{G}$, denoted by δ, is also referred to as the degeneracy of $\tilde{G}$ [9].

Following the standard uncertain graph model [23], [24], [25], [18], we assume that the existence of different edges are mutually independent events. Based on this assumption, the widely-used *possible world* semantics can be applied to analyze uncertain graphs [23], [25], [26]. Specifically, a possible world of the uncertain graph $\mathcal{G}$ is a deterministic graph that contains all nodes in V and a set of edges sampling from E based on the probability function p.

Let $G = (V, E_G)$ be a possible world of G, where $E_G \subseteq E$. The probability of observing a possible world G is defined as

$$\mathsf{Pr}(G) \triangleq \prod\nolimits_{e \in E_G} p_e \prod\nolimits_{e \in E \setminus E_G} (1 - p_e) \tag{1}$$

For convenience, a notion $G \sqsubseteq \mathcal{G}$ means that G is a possible world of $\mathcal{G}$. Following [19], we give a definition of maximal clique in uncertain graphs as follows.

In a possible world graph G, a clique C in $G = (V, E_G)$ is a complete subgraph, where each pair of nodes in C is connected by an edge in E_G. Based on the concept of clique, we define the clique probability in an uncertain graph below.

Definition 1 (**Clique probability** [18])**.** *In an uncertain graph $\mathcal{G}$, for a set of nodes $C \subseteq V$, the clique probability of C, denoted by $\mathsf{CPr}(C, \mathcal{G})$, is defined as*

$$\mathsf{CPr}(C, \mathcal{G}) = \prod\nolimits_{e \in E_C} p_e, \tag{2}$$

where E_C denotes the set of edges connecting nodes in C, i.e., $E_C \triangleq \{(u, v) | (u, v) \in E, u, v \in C, u \neq v\}$.

Based on Definition 1, the node set $C \subseteq V$ is referred to as a τ-clique if $\mathsf{CPr}(C, \mathcal{G}) \geq \tau$ [18]. For a typical uncertain graph, many τ-cliques are small and may be of no practical use. Thus, it will be more useful to compute large τ-cliques for practical applications. To this end, we introduce a concept called (k, τ)-clique as follows, which requires the size of a τ-clique strictly larger than a constant k.

Definition 2 (**(k, τ)-clique**)**.** *In an uncertain graph $\mathcal{G}$, a set of nodes $C \subseteq V$ is called a (k, τ)-clique if it satisfies: (1) $|C| > k$, and (2) $\mathsf{CPr}(C, \mathcal{G}) \geq \tau$.*

Clearly, by Definition 2, the nodes in a (k, τ)-clique form a clique in the deterministic graph $\tilde{G}$ if $\tau > 0$. Based on Definition 2, a maximal (k, τ)-clique is defined as follows.

Definition 3 (**maximal (k, τ)-clique**)**.** *In an uncertain graph $\mathcal{G}$, a set of nodes C is a maximal (k, τ)-clique if C is a (k, τ)-clique and there is no (k, τ)-clique C' in $\mathcal{G}$ containing C.*

[1]In DBLP, we generate an probability for each edge based on an independent exponential distribution, i.e., $p_{uv} = 1 - \exp(-w_{uv}/2)$ for an edge (u, v), where w_{uv} is the number of papers coauthored by u and v.

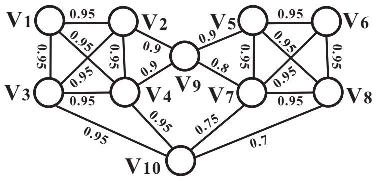


Fig. 1. Running example

Example 1. *Consider an uncertain graph $\mathcal{G}$ shown in Fig. 1. Let $k = 3$ and $\tau = 0.72$. Then, we can see that there are two maximal (k, τ)-cliques in $\mathcal{G}$ which are $C_1 = \{v_1, \cdots, v_4\}$ and $C_2 = \{v_5, \cdots, v_8\}$. This is because the clique probabilities of C_1 and C_2 are $\mathsf{CPr}(C_1, \mathcal{G}) = \mathsf{CPr}(C_2, \mathcal{G}) = 0.735$ which are larger than τ. Moreover, there does not exist a node set C' containing C_1 (or C_2) that satisfies $\mathsf{CPr}(C', \mathcal{G}) \geq \tau$.*

Let $\mathcal{C}$ be the set of all maximal (k, τ)-cliques in an uncertain graph $\mathcal{G}$. A maximal (k, τ)-clique in $\mathcal{C}$ with the largest size is called a maximum (k, τ)-clique. Note that there may exist several maximum (k, τ)-cliques with the same size in $\mathcal{G}$. In this paper, we aim at enumerating all maximal (k, τ)-cliques and finding one of maximum (k, τ)-cliques in $\mathcal{G}$. Specifically, we formulate our problems below.

Problem statement. Given an uncertain graph $\mathcal{G}$ and two parameters k and τ, our goal is to develop fast solutions to solve the following two fundamental problems: (1) enumerate all maximal (k, τ)-cliques in $\mathcal{G}$; and (2) find one of maximum (k, τ)-cliques in $\mathcal{G}$.

As shown in [18], [19], the number of maximal (k, τ)-cliques in an uncertain graph $\mathcal{G}$ is much larger than the number of traditional maximal cliques in the corresponding deterministic graph of $\mathcal{G}$ (i.e., $\tilde{G}$). As a result, the problem of enumerating all maximal (k, τ)-cliques is much harder than the problem of enumerating traditional maximal cliques. In [18], [19], Mukherjee et. al. developed a backtracking algorithm to enumerate all maximal (k, τ)-cliques based on a classic set enumeration technique [20]. Their algorithm, however, is very costly for large uncertain graphs, because its worst-case time complexity is $O(n2^n)$. To speed up their algorithm, we will develop several powerful pruning techniques to substantially prune unpromising nodes in the uncertain graph, thus significantly reducing the worst-case time complexity of the backtracking enumeration algorithm. We will also propose a maximum (k, τ)-clique search algorithm based on several novel upper-bounding techniques.

III. Core-based Pruning Algorithms

In this section, we develop two different core-based pruning techniques to prune the nodes in an uncertain graph $\mathcal{G}$ that are not contained in any maximal (k, τ)-clique. The first core-based pruning algorithm is based on the (k, τ)-core which was proposed in [16]. We show that any maximal (k, τ)-clique must be contained in the (k, τ)-core. To efficiently implement this pruning rule, we devise a new dynamic programming (DP) based algorithm to compute the (k, τ)-core. Compared to the algorithm proposed in [16], the new DP-based algorithm reduces the time complexity for computing the (k, τ)-core from $O(md_{\max})$ to $O(m\delta)$, where $d_{\max}$ is the maximum degree of the nodes in the deterministic graph $\tilde{G}$ of $\mathcal{G}$ and δ ($\delta \leq d_{\max}$) is the degeneracy of $\tilde{G}$. The degeneracy δ is typically much smaller than the maximum degree $d_{\max}$ in real-world uncertain graphs, thus our algorithm is faster than the algorithm proposed in [16]. The second pruning technique is based on a newly-proposed concept, called (Top_k, τ)-core. We develop an efficient algorithm to calculate the (Top_k, τ)-core. We also propose a novel cut-based optimization technique to further improve the above two core-based pruning rules. Below, we detail these pruning techniques.

A. The (k, τ)-core pruning technique

Let $\mathcal{G}_u^{\geq r}$ be the set of all possible worlds sampled from $\mathcal{G}$ where u has degree no less than r, i.e., $\mathcal{G}_u^{\geq r} \triangleq \{G | G \sqsubseteq \mathcal{G}, d_u(G) \geq r\}$. Then, we define the probability $\mathsf{Pr}(d_u(\mathcal{G}) \geq r) \triangleq \sum_{G \in \mathcal{G}_u^{\geq r}} \mathsf{Pr}(G)$ [16]. Based on this probability, the τ-degree of a node [16] is defined as follows.

Definition 4 (**τ-degree** [16])**.** *Given an uncertain graph $\mathcal{G} = (V, E, p)$ and a threshold $\tau \in (0, 1]$, the τ-degree of a node u in V, denoted by τ-$\mathsf{deg}(u, \mathcal{G})$, is defined as the largest integer r that meets $\mathsf{Pr}(d_u(\mathcal{G}) \geq r) \geq \tau$, i.e., τ-$\mathsf{deg}(u, \mathcal{G}) \triangleq$ $\max\{r | \mathsf{Pr}(d_u(\mathcal{G}) \geq r) \geq \tau, r \in [0, \cdots, d_u(\tilde{G})]\}$.*

Based on Definition 4, the (k, τ)-core is defined below.

Definition 5 (**(k, τ)-core** [16])**.** *Given an uncertain graph $\mathcal{G} = (V, E, p)$ and two parameters k and τ, a set of nodes $C \subseteq V$ is called a (k, τ)-core if it meets: (1) τ-$\mathsf{deg}(u, \mathcal{G}_C) \geq k$ for each $u \in C$, and (2) there does not exist a node set $C' \subseteq V$ that satisfies both (1) and $C \subset C'$.*

By Definition 5, we can easily derive that there is only one (k, τ)-core in $\mathcal{G}$ (the (k, τ)-core is not necessarily connected). Below, we show that any maximal (k, τ)-clique must be contained in the (k, τ)-core.

Lemma 1. *Given an uncertain graph $\mathcal{G} = (V, E, p)$ and two parameters k and τ, all maximal (k, τ)-cliques in $\mathcal{G}$ are contained in the (k, τ)-core of $\mathcal{G}$.*

Proof. Let C be a maximal (k, τ)-clique, and Ω_C be the set of possible worlds where C is a clique ($G \in \Omega_C$ if C is a clique in G). By Definitions 1 and 3, we have $\mathsf{CPr}(C, \mathcal{G}) = \sum_{G \in \Omega_C} \mathsf{Pr}(G) \geq \tau$. Since $|C| > k$, any node $u \in C$ must have degree no less than k in the possible world $G \in \Omega_C$. Therefore, for each node $u \in C$, Ω_C is a subset of $\mathcal{G}_u^{\geq k} = \{G | G \sqsubseteq \mathcal{G}, d_u(G) \geq k\}$. Based on this, we can derive that $\mathsf{Pr}(d_u(\mathcal{G}_C) \geq k) \geq \sum_{G \in \Omega_C} \mathsf{Pr}(G) \geq \tau$. As a result, we have τ-$\mathsf{deg}(u, \mathcal{G}_C) \geq k$ for every $u \in C$. Since the (k, τ)-core $\hat{C}$ of $\mathcal{G}$ is the maximum node set in which every node u satisfies τ-$\mathsf{deg}(u, \mathcal{G}_{\hat{C}}) \geq k$, C must be contained in $\hat{C}$. $\square$

Example 2. *Reconsider the uncertain graph $\mathcal{G}$ shown in Fig. 1. Let $k = 3$ and $\tau = 0.72$. Then, we can derive that the τ-degrees for nodes $\{v_1, v_2, \cdots, v_{10}\}$ are $\{3, 4, 4, 5, 4, 3, 4, 3, 3, 3\}$ respectively. Thus, all nodes in $\mathcal{G}$*

have τ-degrees no smaller than k, indicating that $\mathcal{G}$ is a (k,τ)-core. We can see that in this example, the two maximal (k,τ)-cliques are contained in the (k,τ)-core which confirms the result shown in Lemma 1.

Based on Lemma 1, we can first compute the (k,τ)-core of $\mathcal{G}$, and then invoke the backtracking enumeration algorithm proposed in [18] on the (k,τ)-core to find all maximal (k,τ)-cliques. Since the size of the (k,τ)-core is often much smaller than the size of the original uncertain graph $\mathcal{G}$, our algorithm is much more efficient than the algorithm proposed in [18] when handling large uncertain graphs. The remaining question is how can we efficiently compute the (k,τ)-core in an uncertain graph $\mathcal{G}$. Similar to the traditional k-core [22], the (k,τ)-core can be obtained by iteratively peeling the nodes that have τ-degrees smaller than k. The key step of this peeling algorithm is to calculate (and update) the τ-degrees for all nodes in $\mathcal{G}$. In [16], Bonchi et. al. proposed a DP algorithm to compute $\tau\text{-deg}(u)$ for each node $u \in \mathcal{G}$. The detailed description of this DP algorithm is shown as follows.

The DP algorithm. It is easy to see that $\mathsf{Pr}(d_u(\mathcal{G}) \geq k) = \sum_{i=k}^{d_u(\tilde{G})} \mathsf{Pr}(d_u(\mathcal{G}) = i) = 1 - \sum_{i=0}^{k-1} \mathsf{Pr}(d_u(\mathcal{G}) = i)$. Thus, to calculate $\tau\text{-deg}(u)$, the key step is to compute $\mathsf{Pr}(d_u(\mathcal{G}) = i)$. Once we have $\mathsf{Pr}(d_u(\mathcal{G}) = i)$ for each $i \in \{0, \cdots, d_u(\tilde{G})\}$, we can easily derive $\tau\text{-deg}(u)$ by the following procedure. First, we have $\mathsf{Pr}(d_u(\mathcal{G}) \geq 0) = 1$. Then, we can iteratively apply the fact $\mathsf{Pr}(d_u(\mathcal{G}) \geq i+1) = \mathsf{Pr}(d_u(\mathcal{G}) \geq i) - \mathsf{Pr}(d_u(\mathcal{G}) = i)$ to compute $\mathsf{Pr}(d_u(\mathcal{G}) \geq i+1)$ from $i = 0$ to $i = d_u(\tilde{G}) - 1$. Once $\mathsf{Pr}(d_u(\mathcal{G}) \geq i+1) < \tau$, the iterative procedure terminates and we can derive that $\tau\text{-deg}(u) = i$.

To compute $\mathsf{Pr}(d_u(\mathcal{G}) = i)$, Bonchi et. al. [16] devised a DP algorithm based on the following observation. Let $E_u(\mathcal{G}) = \{e_1, e_2, \cdots, e_{d_u(\tilde{G})}\}$ be the set of edges incident to u in $\mathcal{G}$, and $E_u^h(\mathcal{G}) = \{e_1, e_2, \cdots, e_h\}$ ($h \leq d_u(\tilde{G})$) be a subset of $E_u(\mathcal{G})$ (the first h edges in $E_u(\mathcal{G})$). Let $\mathcal{G}_h = (V, E \setminus (E_u(\mathcal{G}) \setminus E_u^h(\mathcal{G})), p)$ be the uncertain subgraph of $\mathcal{G}$ obtained by removing all edges in $E_u(\mathcal{G}) \setminus E_u^h(\mathcal{G})$, and $X_u(h,i) \triangleq \mathsf{Pr}(d_u(\mathcal{G}_h) = i)$. Then, for each $h \in [1, d_u(\tilde{G})]$ and $i \in [0, h]$, we have

$$X_u(h,i) = p_{e_h} X_u(h-1, i-1) + (1 - p_{e_h}) X_u(h-1, i). \quad (3)$$

Initially, it is easy to derive that $X_u(0,0) = 1$, $X_u(h,-1) = 0$ for all $h \in [0, d_u(\tilde{G})]$, and $X_u(h,i) = 0$ for all $h \in [0, d_u(\tilde{G})]$ and $i \in [h+1, d_u(\tilde{G})]$. Based on Eq. (3), Bonchi et. al. [16] developed a DP algorithm to compute $\mathsf{Pr}(d_u(\mathcal{G}) = i)$ (i.e., $X_u(d_u(\tilde{G}), i)$) for each $i \in [0, d_u(\tilde{G})]$. As shown in [16], the time complexity of the DP algorithm to calculate $\tau\text{-deg}(u)$ for a node u is $O(\tau\text{-deg}(u) \times d_u(\tilde{G}))$. Since $\tau\text{-deg}(u) \leq d_{\max}$, the DP algorithm takes $O(md_{\max})$ time to compute the τ-degrees for all nodes in $\mathcal{G}$.

Recall that to compute the (k,τ)-core, the peeling algorithm needs to update the τ-degrees of u's neighbors when deleting a node u [16]. Note that a node deletion is equivalent to a set of edge deletions, thus we focus mainly on updating $\tau\text{-deg}(v)$ when deleting an edge (u,v). Let $\mathcal{G}_{\neg e} = (V, E \setminus \{e\}, p)$ be the uncertain subgraph of $\mathcal{G}$ after deleting an edge e. If $e \in E_u(\mathcal{G})$, we have

$$\mathsf{Pr}(d_u(\mathcal{G}_{\neg e}) = i) = \frac{\mathsf{Pr}(d_u(\mathcal{G}) = i) - p_e \mathsf{Pr}(d_u(\mathcal{G}_{\neg e}) = i-1)}{1 - p_e}. \quad (4)$$

By setting $i = 0$ in Eq. (4), we have $\mathsf{Pr}(d_u(\mathcal{G}_{\neg e}) = 0) = \mathsf{Pr}(d_u(\mathcal{G}) = 0)/(1 - p_e)$. Then, we can apply Eq. (4) to iteratively compute the remaining $\mathsf{Pr}(d_u(\mathcal{G}_{\neg e}) = i)$ values for all $i \in [1, \cdots, \tau\text{-deg}(u)]$ in $O(\tau\text{-deg}(u))$ time. Since each edge is deleted at most once in computing the (k,τ)-core, the total updating cost for a node u is bounded by $O(d_u(\tilde{G}) \times \tau\text{-deg}(u))$. As a result, the total time complexity of the peeling algorithm is $O(md_{\max})$ [16]. Since $d_{\max}$ is often very large in real-life graphs, the DP algorithm is costly. Below, we propose a more efficient algorithm to compute the (k,τ)-core.

New DP-based (k,τ)-core computation. Before proceeding further, we define a notion, called τ-core number, for each node $u \in \mathcal{G}$ as follows.

Definition 6 (τ-core number). *Given an uncertain graph $\mathcal{G}$ and parameter $\tau \in (0,1]$, the τ-core number of a node $u \in \mathcal{G}$, denoted by ξ_u, is defined as the largest k such that there is a (k,τ)-core containing u.*

By Definition 6, we can easily derive that $\xi_u \leq c_u$ for each $u \in \mathcal{G}$, where c_u is the core number of u in the deterministic graph $\tilde{G}$. Therefore, we can first prune all the nodes in $\mathcal{G}$ that have core numbers in $\tilde{G}$ smaller than k, and then compute the (k,τ)-core in the remaining uncertain graph. To efficiently compute the (k,τ)-core in the remaining graph, we propose a new concept called truncated τ-degree as follows.

Definition 7 (truncated τ-degree). *Given an uncertain graph $\mathcal{G}$, the truncated τ-degree of a node u is defined as $\bar{\tau}\text{-deg}(u,\mathcal{G}) = \min\{c_u, \tau\text{-deg}(u,\mathcal{G})\}$, where c_u is the core number of u in the deterministic graph $\tilde{G}$ of $\mathcal{G}$.*

By Definition 7, we can see that the truncated τ-degree of a node u is equal to the τ-degree of u *truncated* by its core number. The following lemma shows that the (k,τ)-core can also be computed based on the truncated τ-degree.

Lemma 2. *Given an uncertain graph $\mathcal{G} = (V,E,p)$ and two parameters k and τ, the (k,τ)-core is the maximum node set $C \subseteq V$ such that every node $u \in C$ has $\bar{\tau}\text{-deg}(u,\mathcal{G}_C) \geq k$.*

Proof. Let C' be the maximum node set such that $\tau\text{-deg}(u,\mathcal{G}_{C'}) \geq k$ for each $u \in C'$. To prove the lemma, we need to show $C = C'$. First, we prove that $C \subseteq C'$. For any node $u \in C$, we have $\bar{\tau}\text{-deg}(u,\mathcal{G}_C) = \min\{c_u, \tau\text{-deg}(u,\mathcal{G}_C)\} \geq k$ by definition, thus $\tau\text{-deg}(u,\mathcal{G}_C) \geq k$ holds. Since C' is the maximum set in which every node v satisfies $\tau\text{-deg}(v,\mathcal{G}_{C'}) \geq k$, we have $u \in C'$. Second, we show that $C' \subseteq C$. For any node $u \in C'$, we have $\tau\text{-deg}(u,\mathcal{G}'_C) \geq k$ by definition. Since C' is the (k,τ)-core, the τ-core number of u (i.e., ξ_u) is no less than k. Based on this, we have $c_u \geq \xi_u \geq k$, and thus $\bar{\tau}\text{-deg}(u,\mathcal{G}_{C'}) \geq k$. Since C is the maximum set where each node v meets $\bar{\tau}\text{-deg}(v,\mathcal{G}_C) \geq k$, u is also included in C. This completes the proof. $\square$

Algorithm 1: NewDP($\mathcal{G} = (V, E, p), u, c_u, \tau$)

```
Input: Uncertain graph $\mathcal{G}$, node $u$, core number $c_u$, and parameter $\tau$
Output: $\bar{\tau}$-deg($u$)
1 Set $Y_u(0, i) \leftarrow 0$ for each $i \in [1, c_u]$;
2 Set $Y_u(h, 0) \leftarrow 1$ for each $h \in [0, d_u(\tilde{G})]$;
3 Let $E_u(\mathcal{G}) \triangleq \{e_1, e_2, \cdots, e_{d_u(\tilde{G})}\}$ be the set of edges incident to $u$;
4 for $i = 1$ to $c_u$ do
5     for $h = 0$ to $d_u(\tilde{G}) - 1$ do
6         $Y_u(h + 1, i) \leftarrow p_{e_{h+1}} Y_u(h, i - 1) + (1 - p_{e_{h+1}})Y_u(h, i)$;
7     if $Y_u(d_u(\tilde{G}), i) < \tau$ then return $i - 1$;
8 return $c_u$;
```

By Lemma 2, we can iteratively delete the nodes with truncated τ-degree smaller than k to determine the (k, τ)-core. The remaining question is how can we efficiently calculate the truncated τ-degrees for all nodes. Clearly, to compute the truncated τ-degrees, we can first compute the core numbers using a classic core-decomposition algorithm [27], and then calculate the τ-degrees using the DP algorithm proposed in [16]. It is easy to see that the time complexity of this algorithm is $O(md_{\max})$. To reduce the time complexity, we develop a new DP algorithm to compute the truncated τ-degrees for all nodes in $O(m\delta)$, where $\delta \leq d_{\max}$ is the degeneracy of $\tilde{G}$.

Unlike the DP algorithm to compute τ-deg(u) [16], our new DP algorithm directly calculates $\mathsf{Pr}(d_u(\mathcal{G}) \geq i)$ to determine $\bar{\tau}$-deg(u). Let $Y_u(h, i) \triangleq \mathsf{Pr}(d_u(\mathcal{G}_h) \geq i)$. Then, we have the following recursive equation

$$Y_u(h, i) = p_{e_h} Y_u(h - 1, i - 1) + (1 - p_{e_h})Y_u(h - 1, i). \quad (5)$$

The rationale of Eq. (5) is that the probability of $d_u(\mathcal{G}_h) \geq k$ can be derived by two different cases: (1) the edge $e_h \in E_u(\mathcal{G})$ definitely appears in the uncertain graph $\mathcal{G}$, and (2) e_h does not appear in $\mathcal{G}$. In the first case, the probability is equal to $p_{e_h} Y_u(h - 1, i - 1)$, while the probability of the second case equals $(1 - p_{e_h})Y_u(h - 1, i)$. Therefore, $Y_u(h, i)$ is equal to the sum of these two probabilities. It is important to note that although Eq. (5) is very similar to Eq. (3), the semantics of these two equations are totally different. The initial states of $Y_u(h, i)$ in our DP algorithm are also quite different from the initial states of the previous DP algorithm. Specifically, we initially set $Y_u(0, i) = 0$ for each $i \in [1, c_u]$. This is because when $h = 0$, no edge incident to u is considered, thus the probability $\mathsf{Pr}(d_u(\mathcal{G}_0) \geq i) = 0$. Moreover, based on Definition 7, we can set $i \leq c_u$. This is because if $i > c_u$ and $\mathsf{Pr}(d_u(\mathcal{G}_h) \geq i) \geq \tau$, we have $\bar{\tau}$-deg$(u) = c_u$ and thus we do not need to compute $\bar{\tau}$-deg(u) in this case using the DP algorithm. Additionally, we set $Y_u(h, 0) = 1$ for each $h \in [0, d_u(\tilde{G})]$. The reason is that the inequality $d_u(\mathcal{G}_h) \geq 0$ always holds, thus $Y_u(h, 0) = \mathsf{Pr}(d_u(\mathcal{G}_h) \geq 0) = 1$. Based on these initial states, we can easily devise a DP algorithm to compute all $Y_u(h, i)$. The pseudo code of our DP algorithm to compute $\bar{\tau}$-deg(u) is shown in Algorithm 1.

It is worth noting that in line 7 of Algorithm 1, the algorithm can early terminate when τ-deg$(u) < c_u$ (in this case $\bar{\tau}$-deg(u) $=\tau$-deg$(u) < c_u$). The correctness of Algorithm 1 can be guaranteed by Definition 7 and Eq. (5). In addition, we can easily derive that the time complexity of Algorithm 1 is $O(d_u(\tilde{G}) \times \bar{\tau}$-deg$(u))$. Since $\bar{\tau}$-deg$(u) \leq \delta$ for every node u, the total time complexity of Algorithm 1 for computing the truncated τ-degrees for all nodes is $O(m\delta)$.

Algorithm 2: NewDPCore($\mathcal{G} = (V, E, p), k, \tau$)

```
Input: Uncertain graph $\mathcal{G}$, parameters $k$ and $\tau$
Output: The $(k, \tau)$-core
1 Let $c_u$ be the core number of node $u$ in the deterministic graph $\tilde{G}$ of $\mathcal{G}$;
2 Compute $c_u$ for each $u \in V$ using a traditional core decomposition algorithm;
3 $\mathcal{G}' = (V', E', p) \leftarrow$ prune all nodes in $\mathcal{G}$ with core numbers smaller than $k$;
4 $Q \leftarrow \emptyset$;
5 for each $u \in V'$ do
6     $\bar{\tau}$-deg$(u) \leftarrow$ NewDP$(\mathcal{G}, u, c_u, \tau)$;
7     if $\bar{\tau}$-deg$(u) < k$ then $Q.push(u)$;
8 while $Q \neq \emptyset$ do
9     $u \leftarrow Q.pop()$;
10    for each $v \in N_u(\tilde{G}')$ do
11        Delete $(u, v)$ from $\mathcal{G}'$;
12        $\bar{\tau}$-deg$(v) \leftarrow$Update$(u, v, \tau)$;
13        if $\bar{\tau}$-deg$(v) < k$ then $Q.push(v)$ ;
14    $V' \leftarrow V' \setminus \{u\}$;
15 return $V'$;
16 Procedure Update$(u, v, \tau)$;
17 Let $e = (u, v)$ be the deleted edge;
18 for $i = 1$ to $\bar{\tau}$-deg$(v)$ do
19     $Y_v(d_v(\tilde{G}') - 1, i) \leftarrow \frac{Y_v(d_v(\tilde{G}'), i) - p_e \times Y_v(d_v(\tilde{G}') - 1, i - 1)}{(1 - p_e)}$;
20     if $Y_v(d_v(\tilde{G}') - 1, i) < \tau$ then return $i - 1$;
21 $d_v(\tilde{G}') \leftarrow d_v(\tilde{G}') - 1$;
22 return $\bar{\tau}$-deg$(v)$ ;
```

To compute the (k, τ)-core, we also need to maintain the truncated τ-degrees of u's neighbors when peeling a node u. The key step to update $\bar{\tau}$-deg(v) is to update $Y_v(d_v(\tilde{G}), i)$ (i.e., $\mathsf{Pr}(d_u(\mathcal{G}) \geq i)$). Note that after deleting an edge $e = (u, v)$, $d_v(\tilde{G})$ decreases by 1. Thus, our goal is to update $Y_v(d_v(\tilde{G}) - 1, i)$ using the values of $Y_v(d_v(\tilde{G}), i)$ for all $i \in [1,\bar{\tau}$-deg$(v)]$. Initially, we have $Y_v(d_v(\tilde{G}) - 1, 0) = 1$. Then, we can iteratively use the following recursive equation to compute $Y_v(d_v(\tilde{G}) - 1, i)$ from $i = 1$ to $i =\bar{\tau}$-deg(v):

$$Y_v(d_v(\tilde{G}) - 1, i) = \frac{Y_v(d_v(\tilde{G}), i) - p_e \times Y_v(d_v(\tilde{G}) - 1, i - 1)}{1 - p_e}. \quad (6)$$

Armed with Algorithm 1 and the above updating procedure, we can easily devise the (k, τ)-core computation algorithm which is shown in Algorithm 2. Specifically, Algorithm 2 first prunes the nodes using the k-core of the deterministic graph $\tilde{G}$ (lines 1-3). Then, the algorithm uses a queue $\mathcal{Q}$ to maintain the nodes that have truncated τ-degrees smaller than k (lines 5-7). After that, the algorithm iteratively deletes the nodes in $\mathcal{Q}$ until no node can be removed (lines 8-15). When deleting a node u, the algorithm invokes an updating procedure (lines 16-22) based on Eq. (6) to update the truncated τ-degrees of u's neighbor nodes (lines 10-13). The correctness of Algorithm 2 can be guaranteed by Lemma 2. Below, we analyze the time and space complexity of Algorithm 2.

Theorem 1. *The time and space complexity of Algorithm 2 is $O(m\delta)$ and $O(m+n)$ respectively, where δ is the degeneracy of the deterministic graph $\tilde{G}$ of $\mathcal{G}$.*

Proof. First, in lines 1-2, the algorithm takes $O(m + n)$ time to compute the core numbers of the nodes in $\tilde{G}$. Second, in line 5-7, the algorithm consumes $O(m\delta)$ time to compute the truncated τ-degrees for all nodes. Third, in lines 8-14, the algorithm takes $O(m\delta)$ time to iteratively compute the (k,τ)-core. This is because each edge is deleted at most once, the time cost to update the truncated τ-degree of a node v is $O(d_v(\tilde{G})\times\bar{\tau}\text{-deg}(v))$ by Eq. (6). Since $\bar{\tau}\text{-deg}(v)$ is bounded by δ, the total time cost taken in lines 8-14 is $O(m\delta)$. Putting it all together, the time complexity of Algorithm 2 is $O(m\delta)$. Note that the space usage of Algorithm 1 to compute $\bar{\tau}\text{-deg}(u)$ is $O(d_u(\tilde{G}))$, because the recursive equation can be implemented by two $d_u(\tilde{G})$-length arrays. For each u, the algorithm takes $O(\bar{\tau}\text{-deg}(u))$ space to maintain $\bar{\tau}\text{-deg}(u)$, which is bounded by $O(d_u(\tilde{G}))$. Therefore, the total space usage of Algorithm 2 is $O(m+n)$. □

Since the degeneracy δ is often much smaller than the maximum degree $d_{\max}$ in real-life graphs, Algorithm 2 is faster than the algorithm proposed in [16] for computing the (k,τ)-core, which is confirmed in our experiments.

B. The (Top_k,τ)*-core pruning technique*

In this subsection, we develop a simple but more effective pruning technique for maximal (k,τ)-clique enumeration based on a novel concept, called (Top_k,τ)-core. Again, we let $E_u(\mathcal{G})=\{e_1,e_2,\cdots,e_{d_u(\tilde{G})}\}$ be the set of edges incident to u in $\mathcal{G}$. Suppose without loss of generality that the edges in $E_u(\mathcal{G})$ are sorted in non-increasing order based on the probabilities, i.e., $p_{e_1}\geq p_{e_2}\geq\cdots\geq p_{e_{d_u(\tilde{G})}}$. Denote by $N_u^k(\mathcal{G})$ the set of top-k edges in $N_u(\mathcal{G})$ with the highest probabilities, i.e., $N_u^k(\mathcal{G})\triangleq\{e_1,e_2,\cdots,e_k\}$ if $d_u(\tilde{G})\geq k$, and $N_u^k(\mathcal{G})\triangleq\emptyset$ otherwise. Based on $N_u^k(\mathcal{G})$, we introduce a definition called *top-k product probability*, denoted by $\pi_k(u,\mathcal{G})$, for each node $u\in\mathcal{G}$.

Definition 8 (top-k product probability). *Given an uncertain graph $\mathcal{G}$ and an integer k, the top-k product probability of a node $u\in\mathcal{G}$ is defined as $\pi_k(u,\mathcal{G})\triangleq\prod_{e\in N_u^k(\mathcal{G})}p_e$.*

By Definition 8, we can easily show that the top-k product probability for any node meets a *monotonic* property.

Lemma 3. *For two uncertain graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, we have $\pi_k(u,\mathcal{G}_1)\geq\pi_k(u,\mathcal{G}_2)$ for any u if $\mathcal{G}_2$ is a subgraph of $\mathcal{G}_1$.*

Based on Definition 8, we define the (Top_k,τ)-core below.

Definition 9 ((Top_k,τ)-core). *Given an uncertain graph $\mathcal{G}=(V,E,p)$ and two parameters k and τ, a set of nodes $C\subseteq V$ is called a (Top_k,τ)-core if it satisfies: (1) $\pi_k(u,\mathcal{G}_C)\geq\tau$ for each $u\in C$, and (2) there does not exist a node set $C'\subseteq V$ that meets both (1) and $C\subset C'$.*

By Definition 9, we can easily obtain that there is only one (Top_k,τ)-core in $\mathcal{G}$. Below, we show that all maximal (k,τ)-cliques are contained in the (Top_k,τ)-core.

Lemma 4. *Given an uncertain graph $\mathcal{G}=(V,E,p)$ and two parameters k and τ, all maximal (k,τ)-cliques in $\mathcal{G}$ are contained in the (Top_k,τ)-core of $\mathcal{G}$.*

Algorithm 3: $\mathsf{TopKCore}(\mathcal{G}=(V,E,p),V_I,k,\tau)$

```
Input: Uncertain graph $\mathcal{G}$, a fixed node set $V_I$, parameters $k$ and $\tau$
Output: The $(\mathsf{Top}_k,\tau)$-core and a boolean constant
1  $Q\leftarrow\emptyset$;
2  for each $u\in V$ do
3      $\pi_k(u)\leftarrow$ compute $\pi_k(u,\mathcal{G})$ based on Definition 8;
4      if $\pi_k(u)<\tau$ then
5          if $u\in V_I$ then return $(\emptyset,0)$;
6          $Q.push(u)$;
7  while $Q\neq\emptyset$ do
8      $u\leftarrow Q.pop()$;
9      for each $v\in N_u(\tilde{G})$ do
10         Delete $(u,v)$ from $\mathcal{G}$;
11         $d_v(\tilde{G})\leftarrow d_v(\tilde{G})-1$;
12         Update $\pi_k(v)$ after deleting $(u,v)$;
13         if $\pi_k(v)<\tau$ then
14             if $v\in V_I$ then return $(\emptyset,0)$;
15             $Q.push(v)$;
16     $V\leftarrow V\setminus\{u\}$;
17 return $(V,1)$;
```

Proof. Let C be a maximal (k,τ)-clique. By Eq. (2) and Definition 3, we have $\mathsf{CPr}(C,\mathcal{G})=\prod_{e\in E_C}p_e\geq\tau$. Since $|C|\geq k$, we have $\pi_k(u,\mathcal{G}_C)=\prod_{e\in N_u^k(\mathcal{G}_C)}p_e\geq\prod_{e\in N_u(\mathcal{G}_C)}p_e$ for any node $u\in C$. Note that all the edges in $N_u(\mathcal{G}_C)$ are contained in E_C, thus $\prod_{e\in N_u(\mathcal{G}_C)}p_e\geq\prod_{e\in E_C}p_e\geq\tau$. As a result, we have $\pi_k(u,\mathcal{G}_C)\geq\tau$ for each $u\in C$. Since the (Top_k,τ)-core $\hat{C}$ of $\mathcal{G}$ is the maximum node set in which every node u meets $\pi_k(u,\mathcal{G}'_C)\geq\tau$, C must be contained in $\hat{C}$. □

Example 3. *Reconsider the uncertain graph $\mathcal{G}$ in Fig. 1. Let $k=3$ and $\tau=0.72$. It is easy to check that the top-k product probabilities for all nodes in $C=\{v_1,v_2,\cdots,v_9\}$ are no smaller than τ, while the top-k product probabilities for the node v_{10} is 0.68 which is smaller than τ. Clearly, v_{10} is not contained in the (Top_k,τ)-core by Definition 9. In this example, we can easily derive that C is a (Top_k,τ)-core. As desired, the two maximal (k,τ)-cliques in $\mathcal{G}$ (see Example 1) are contained in C which confirms the result shown in Lemma 4. Compared to the (k,τ)-core, the (Top_k,τ)-core pruning technique can further prune the node v_{10} in this example, indicating that the pruning performance of the (Top_k,τ)-core is better than that of the (k,τ)-core (see Corollary 1).*

By Lemma 4, we are capable of using the (Top_k,τ)-core to prune the nodes in $\mathcal{G}$ that are not contained in any maximal (k,τ)-clique. Since the top-k product probability for each node satisfies a *monotonic* property (Lemma 3), we can devise an iteratively peeling algorithm to calcualte the (Top_k,τ)-core based on the results shown in [28]. In particular, the (Top_k,τ)-core can be obtained by iteratively removing the nodes with top-k product probabilities smaller than τ. The pseudo code of this algorithm is shown in Algorithm 3.

Note that in Algorithm 3, the parameter V_I denotes a set of fixed node which will be used in our maximal (k,τ)-clique enumeration algorithm. If the (Top_k,τ)-core in $\mathcal{G}$ does not include V_I, Algorithm 3 will return an empty set. Obviously, we can set $V_I=\emptyset$ and invoke $\mathsf{TopKCore}(\mathcal{G},\emptyset,k,\tau)$ to compute the (Top_k,τ)-core in $\mathcal{G}$. The correctness of Algorithm 3 can be guaranteed by Lemma 3 and the generalized core theory established in [28]. Below, we analyze the time and space complexity of Algorithm 3.

Theorem 2. *The time and space complexity of Algorithm 3 is $O(m \log_2(d_{\max}))$ and $O(m+n)$ respectively.*

Proof. To compute $\pi_k(u, \mathcal{G})$, the algorithm needs to sort the edges in $E_u(\mathcal{G})$ which takes $O(d_u(\tilde{G}) \times \log_2(d_u(\tilde{G})))$ time. Thus, the total cost in lines 2-6 is $O(\sum_{u \in V}(d_u(\tilde{G}) \times \log_2(d_u(\tilde{G}))))$ which is bounded by $O(m \log_2(d_{\max}))$. Since the edges in $E_v(\mathcal{G})$ have been sorted for each $v \in V$, we can easily update $\pi_k(v, \mathcal{G})$ after deleting an edge (u, v) in constant time. Thus, the total cost in lines 7-16 is $O(m+n)$, because each edge is removed at most once by the algorithm. Putting it all together, the time complexity of Algorithm 3 is $O(m \log_2(d_{\max}))$. For the space complexity, the algorithm needs to maintain the uncertain graph, a queue $\mathcal{Q}$, and $\pi_k(u, \mathcal{G})$ for each u which consume $O(m+n)$ space in total. □

As shown in Theorem 2, the (Top_k, τ)-core can be computed in near-linear time with respect to (w.r.t.) the size of the uncertain graph. Note that the worst-case time complexity of Algorithm 3 can be higher than that of Algorithm 2, as $\log_2(d_{\max})$ may be larger than the degeneracy δ in some real-life graphs. Below, we show that the (Top_k, τ)-core pruning rule is more effective than the (k, τ)-core pruning rule.

Theorem 3. *For each node $u \in V$, if $\pi_k(u, \mathcal{G}) \geq \tau$, we have τ-$\mathsf{deg}(u, \mathcal{G}) \geq k$.*

Proof. Since $\pi_k(u, \mathcal{G}) \geq \tau$, we have $\prod_{e \in N_u^k(\mathcal{G})} p_e \geq \tau$ by definition. Let Ω_k be the set of all possible worlds drawn from $\mathcal{G}$ where each possible world $G \in \Omega_k$ contains all the edges in $N_u^k(\mathcal{G})$. Then, we have $\mathsf{Pr}(\Omega_k) \triangleq \sum_{G \in \Omega_k} \mathsf{Pr}(G) = \prod_{e \in N_u^k(\mathcal{G})} p_e \geq \tau$. Let $\mathcal{G}_u^{\geq k}$ be the set of all possible worlds drawn from $\mathcal{G}$ where u has a degree no less than k. Since $d_u(G) \geq k$ for each $G \in \Omega_k$, Ω_k is a subset of $\mathcal{G}_u^{\geq k}$. As a result, $\mathsf{Pr}(d_u(\mathcal{G}) \geq k) \triangleq \sum_{G \in \mathcal{G}_u^{\geq k}} \mathsf{Pr}(G) \geq \mathsf{Pr}(\Omega_k) \geq \tau$. By Definition 4, we can obtain that τ-$\mathsf{deg}(u, \mathcal{G}) \geq k$. □

The following corollary can be easily obtained by Theorem 3.

Corollary 1. *The (Top_k, τ)-core of an uncertain graph $\mathcal{G}$ is contained in the (k, τ)-core of $\mathcal{G}$.*

Corollary 1 indicates that the (Top_k, τ)-core is more effective than the (k, τ)-core for pruning in enumerating all maximal (k, τ)-cliques. Below, we develop a novel cut-based optimization technique to further improve the pruning performance of two core-based pruning rules.

C. Cut-based optimization

Let C be the (Top_k, τ)-core (or the (k, τ)-core), and $\mathcal{G}_C$ be the uncertain subgraph induced by C. Assume without loss of generality that $\mathcal{G}_C$ is connected. Note that if $\mathcal{G}_C$ is disconnected, we can enumerate all maximal (k, τ)-cliques in each connected component of $\mathcal{G}_C$ respectively. Below, we develop a technique to improve the pruning performance of the core-based pruning rules.

Let H be a maximal (k, τ)-clique and $\mathcal{G}_H = (V_H, E_H, p)$ be the uncertain subgraph induced by H. For convenience, we refer to $\mathcal{G}_H$ as a maximal (k, τ)-clique subgraph in the rest of this paper. A cut $\chi = (S, T)$ is a partition of V_C of an uncertain graph $\mathcal{G}_C = (V_C, E_C, p)$ into two disjoint subsets S and T. The cut set $E_\chi \triangleq \{(u, v) \in E_C | u \in S, v \in V\} = \{e_1, e_2, \cdots, e_{|E_\chi|}\}$ of $\mathcal{G}_C$ denotes a set of edges that have one endpoint in each subset of the partition. Clearly, if no maximal (k, τ)-clique subgraph contains the edges in E_χ, all edges in E_χ can be deleted. Since the deletions of the edges in E_χ will partition $\mathcal{G}_C$ into several small connected uncertain subgraphs, the computational costs for finding all maximal (k, τ)-cliques can be significantly reduced.

Suppose without loss of generality that the edges in the cut set E_χ are sorted in a non-increasing order based on the probabilities, i.e, $p_{e_1} \geq p_{e_2} \geq \cdots \geq p_{e_{|E_\chi|}}$. Let $E_\chi^k = \{p_{e_1} \geq p_{e_2} \geq \cdots \geq p_{e_k}\}$ be the set of top-k edges in E_χ with the highest probabilities if $|E_\chi| \geq k$, and $E_\chi^k = \emptyset$ otherwise. The top-k product probability of a cut set E_χ, denoted by $\pi_k(E_\chi)$, is defined as

$$\pi_k(E_\chi) \triangleq \begin{cases} \prod_{e \in E_\chi^k} p_e, & if\ |E_\chi| \geq k, \\ 0, & otherwise. \end{cases} \tag{7}$$

Based on Eq. (7), we define a notion called *low-probability cut set* as follows.

Definition 10. *For a given parameter τ, a cut set E_χ is called a low-probability cut set if $\pi_k(E_\chi) < \tau$.*

Note that by Eq. (7) and Definition 10, any cut set E_χ with $|E_\chi|$ smaller than k is a low-probability cut set. Below, we show that any maximal (k, τ)-clique subgraph cannot consist of any edge in a low-probability cut set.

Lemma 5. *For any low-probability cut set E_χ, there is no maximal (k, τ)-clique subgraph containing any edge in E_χ.*

Proof. Suppose without loss of generality that E_χ divides the connected uncertain graph $\mathcal{G}_C = (V_C, E_C, p)$ into two connected components $\mathcal{G}_1 = (V_1, E_1, p)$ and $\mathcal{G}_2 = (V_2, E_2, p)$ such that $V_C = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$. First, we show that if $|E_\chi| < k$, the lemma holds. Let H be a maximal (k, τ)-clique, and $\mathcal{G}_H = (V_H, E_H, p)$ be the uncertain subgraph induced by H. Assume that $\mathcal{G}_H$ contains some edges in a cut set E_χ with $|E_\chi| < k$. Then, H can be divided into two disjoint sets, denoted by H_1 and H_2, by the cut set E_χ. Since H is a maximal (k, τ)-clique, H must form a clique in the deterministic graph $\tilde{G}$ and $|H| \geq k$. As a result, the number of edges between H_1 and H_2 is no smaller than k. Note that the edges between H_1 and H_2 must be included in E_χ, thus we have $|E_\chi| \geq k$, which is a contradiction.

Second, we prove the lemma when $|E_\chi| \geq k$. Since E_χ is a low-probability cut set, we have $\prod_{e \in E_\chi^k} p_e < \tau$. Suppose that there is a maximal (k, τ)-clique subgraph $\mathcal{G}_H$ containing some edges in E_χ. Then, the cut set E_χ can partition H into two disjoint parts H_1 and H_2. Let $E_{1,2} \triangleq \{(u, v) \in E_H | u \in H_1, v \in H_2\}$ be the set of edges that have one endpoint in each part. Then, by definition, we have $E_{1,2} \subseteq E_\chi$. Therefore, we can easily derive that $\prod_{e \in E_{1,2}^k} p_e \leq \prod_{e \in E_\chi^k} p_e < \tau$. Since H is a maximal (k,τ)-clique, we have $\prod_{e \in E_{1,2}^k} p_e \geq \prod_{e \in E_H} p_e \geq \tau$, which is a contradiction. $\square$

Example 4. *Reconsider the uncertain graph $\mathcal{G}$ in Fig. 1. Again, we let $k=3$ and $\tau = 0.72$. In this example, there are two low-probability cut sets in the (Top_k, τ)-core C ($\{v_1, v_2, \cdots, v_9\}$) which are $E_{\chi_1} = \{(v_9, v_2), (v_9, v_4)\}$ and $E_{\chi_2} = \{(v_9, v_5), (v_9, v_7)\}$. Clearly, no maximal (k,τ)-clique contains an edge in E_{χ_1} or E_{χ_2} which confirms Lemma 5. Compared to the (Top_k, τ)-core pruning, we can see that the cut-based optimization technique can further prune the node v_9 in this example.*

By Lemma 5, if we find a low-probability cut set E_χ in the uncertain subgraph $\mathcal{G}_C$, then we can safely drop all the edges in E_χ without missing any maximal (k,τ)-clique. Clearly, this will divide $\mathcal{G}_C$ into several small connected subgraphs, thus reducing the cost for enumerating all maximal (k,τ)-cliques. However, finding all low-probability cut sets in $\mathcal{G}_C$ is very costly. In this paper, we make use of a simple min-cut algorithm [29] to find a part of low-probability cut sets. Specifically, the algorithm first selects an arbitrary node in $\mathcal{G}_C$, and then iteratively picks a node that is most tightly connected to the current set of selected nodes S and adds it into S. At this point, the algorithm verifies whether the cut set between S and $C \setminus S$ is a low-probability cut set or not. If so, the algorithm drops all edges in the cut set, and then performs the same iterative procedure in the node set $C \setminus S$. Otherwise, the algorithm continues to choose the next node until all nodes have been selected.

The worst-case time complexity of the cut-based optimization technique is $O(|V_C||E_C| \times \log_2(|E_C|))$. This is because in each iteration, the algorithm needs to sort the edges in the cut set E_χ to determine whether E_χ is a low-probability cut set, which consumes $O(|E_C| \times \log_2(|E_C|)$ time. In addition, the algorithm takes at most $O(|E_C|)$ time to select a node in each iteration. Since there are $O(|V_C|)$ iterations, the time complexity of the cut-based optimization technique is $O(|V_C||E_C| \times \log_2(|E_C|))$. Note that the size $(|V_C| + |E_C|)$ of the (Top_k, τ)-core (or (k,τ)-core) is often very small, thus the cut-based optimization approach is very efficient in practice.

Remark. We note that the cut-based optimization technique can be considered as a generalized (Top_k, τ)-core pruning technique. This is because the set of all edges incident to a node u is a *special* cut set which partitions the node set V into two disjoint sets u and $V \setminus \{u\}$. It is easy to see that the (Top_k, τ)-core pruning technique recursively deletes such a *special* cut set with top-k product probabilities smaller than τ. In the cut-based optimization technique, we generalize this *special* cut to a general low-probability cut set based on the result established in Lemma 5.

IV. Enumerating all maximal uncertain cliques

We present a new maximal (k,τ)-clique enumeration algorithm, called MUCE, by integrating our core-based pruning techniques into the backtracking enumeration algorithm proposed in [18]. The MUCE algorithm first invokes Algorithm 3 (or Algorithm 2) to prune unpromising nodes in the uncertain graph $\mathcal{G}$, and then performs a cut-based optimization on the (Top_k, τ)-core (or (k,τ)-core) to further reduce the graph size. Subsequently, the algorithm enumerates all maximal (k,τ)-cliques on the reduced uncertain graph using a backtracking enumeration algorithm. Note that unlike the backtracking enumeration algorithm proposed in [18], we also integrates the (Top_k, τ)-core pruning technique into the backtracking enumeration algorithm to prune unpromising search branches. The details of our algorithm is shown in Algorithm 4.

In Algorithm 4, the core-based pruning techniques are shown in lines 1-6. After performing the cut-based optimization, we are able to obtain a set of connected parts of the reduced uncertain graph (lines 3-4). Then, in each connected part, the algorithm calls the backtracking enumeration algorithm MUC to find all maximal (k,τ)-cliques (lines 7-22). MUC admits five parameters (R, C, X, k, τ). R denotes a τ-clique which may be expanded to a maximal (k,τ)-clique. C is the set of candidate nodes that is used to expand the current τ-clique R. X denotes a set of nodes that can expand the current τ-clique R, but have already been explored in a different search path by the algorithm. The MUC algorithm first checks whether the current τ-clique R is a maximal (k,τ)-clique or not (line 8). If so, the algorithm outputs R and terminates the current search path (lines 8-10). Then, the algorithm invokes Algorithm 3 to compute the (Top_k, τ)-core in the set $R \cup C$ (lines 12-13). If there is a (Top_k, τ)-core S' containing R with $|S'| \geq k$, then the algorithm updates the candidate set C by $S' \setminus R$ (line 15). This is because any maximal (k,τ)-clique in $R \cup C$ must be contained in S', thus the algorithm can prune the candidate set using the (Top_k, τ)-core S'. Otherwise, the algorithm terminates the current search path (line 14), because there is no maximal (k,τ)-clique contained in $R \cup C$.

After pruning the candidate set C, the algorithm iteratively selects a node in C to expand the current τ-clique R. To avoid repeatedly enumerating the same maximal (k,τ)-clique, the nodes in C are selected following a lexicographical ordering (line 16). When adding a node u into the current τ-clique R, the algorithm generates the candidate set C' for the expanded τ-clique $R' = R \cup \{u\}$ using a technique proposed in [18] (lines 17-18). Obviously, if $|R'| + |C'| < k$, the algorithm completes the current search path, and continues to process the next nodes (line 19). Also, the algorithm applies the technique proposed in [18] to determine the set X for the expanded τ-clique R' (line 20). Subsequently, the algorithm recursively calls the same procedure to expand the τ-clique R' (line 21). After processing a node u, the algorithm adds it into X, because u has already been processed in the current search path which cannot be explored in the following recursions. Below, we analyze the time complexity of our algorithm.

Algorithm 4: $\mathsf{MUCE}(\mathcal{G}=(V,E,p),k,\tau)$

```
Input: Uncertain graph $\mathcal{G}$, parameters $k$ and $\tau$
Output: All maximal $(k,\tau)$-cliques
/* $C \leftarrow$ NewDPCore($\mathcal{G}, k, \tau$) */;
$C \leftarrow$TopKCore($\mathcal{G}, \emptyset, k, \tau$);
$C' \leftarrow$ perform the cut-based optimization technique on $C$;
Let $\mathcal{H}$ be the set of connected components obtained by $C'$;
for each $H \in \mathcal{H}$ do
    MUC($\emptyset, H, \emptyset, k, \tau$);

Procedure MUC($R$, $C$, $X$, $k$, $\tau$);
if $C \cup X = \emptyset$ and $|R| \geq k$ then
    Output $R$ as a maximal $(k,\tau)$-clique;
    return;
/* Lines 12-14: prune the candidate set by the $(\mathsf{Top}_k, \tau)$-core */;
$S \leftarrow R \cup C$;
if $|R| < k$ then $(S', b) \leftarrow$TopKCore($\mathcal{G}_S, R, k, \tau$);
if $b = 0$ or $|S'| < k$ then return;
$C \leftarrow S' \setminus R$;
for each $u \in C$ considered in lexicographical ordering do
    $R' \leftarrow R \cup \{u\}$;
    $C' \leftarrow$ generate the candidate set for $R'$ using the algorithm in [19];
    if $|R'| + |C'| < k$ then continue;
    $X' \leftarrow$ generate the set $X$ for $R'$ using the algorithm in [19];
    MUC($R'$, $C'$, $X'$, $k$, $\tau$);
    $X \leftarrow X \cup \{u\}$;
```

Theorem 4. *The time complexity of Algorithm 4 is $O(2^{n'}(m'+n'))$, where n' is the size of the largest connected component of the (Top_k, τ)-core.*

Proof. Since the search space of Algorithm 4 can be represented as a classic set enumeration tree [20], [18], the number of vertices in the enumeration tree is $O(2^{n'})$. In each vertex of the set enumeration tree, the most time-consuming step is to perform the (Top_k, τ)-core pruning which takes $O(m'+n')$ time. This is because the neighbors of a node u in $\mathcal{G}$ have been sorted when computing the (Top_k, τ)-core of $\mathcal{G}$, thus the algorithm can compute the (Top_k, τ)-core in linear time (w.r.t. $R \cup C$) in each recursion. Since there are at most $O(2^{n'})$ vertices, the total time complexity of Algorithm 4 is $O(2^{n'}(m'+n'))$. □

Note that since n' and m' are often much smaller than the size of the original uncertain graph, Algorithm 4 is faster than the algorithm proposed in [18], [19], which is confirmed in our experiments. In addition, it is easy to show that the space complexity of Algorithm 4 is $O(m+n)$ which is linear w.r.t. the uncertain graph size.

V. Maximum uncertain clique search

Algorithm 4 can be slightly modified to compute one of maximum (k,τ)-cliques. Specifically, when obtaining a maximal (k,τ)-clique (line 9), we maintain the size of the largest maximal (k,τ)-clique C^* found so far, denoted by σ ($|C^*| = \sigma$). Note that in Algorithm 4, each search subspace can be represented by a pair of node sets (R, C), where R denotes a τ-clique and C is the set of candidate nodes for R. If the size of the candidate set C is smaller than $\sigma - |R|$ (or $|R \cup C| < \sigma$), the algorithm can early terminate, because all maximal (k,τ)-cliques in the current search subspace are no larger than σ. Clearly, the key step of this maximum (k,τ)-clique search algorithm is to develop a tight upper bound for the size of maximal (k,τ)-cliques contained in a search subspace (R, C). However, the upper bound based on the candidate set size $|C|$ is not very tight, because $|R \cup C|$ is often larger than σ. To speed up the algorithm, we first propose a basic upper bound based on a classic coloring technique [30]. Such a color-based upper bound is shown to be tighter than the candidate set size based upper bound. Then, we develop two nontrivial and more effective color-based upper bounds to further improve the pruning performance.

A basic color-based upper bound. A basic bound can be easily derived by a classic coloring algorithm. In particular, we assign a color to each node in $\tilde{G}$ using a degree-ordering based greedy coloring algorithm [30] so that no two adjacent nodes have the same color. Since each τ-clique R is a clique in $\tilde{G}$, the colors of the nodes in R must be different. Therefore, the number of colors of the nodes in $R \cup C$ is an upper bound of the size of a maximum (k,τ)-clique contained in $R \cup C$. Let $\mathsf{col}(C)$ be the number of colors of the candidate nodes in C. Then, $|R| + \mathsf{col}(C)$ is an upper bound of a maximum (k,τ)-clique contained in $R \cup C$. For an efficient implement, we only invoke the greedy coloring algorithm once, and compute the upper bound $|R| + \mathsf{col}(C)$ in each search subspace (R, C) based on the same coloring result. Note that the greedy coloring algorithm can be implemented in linear time w.r.t. the uncertain graph size [30] and compute the upper bound $|R| + \mathsf{col}(C)$ in each search subspace (R, C) can be done in $O(|C|)$ time, thus such a basic color-based pruning technique is very efficient.

Advanced color-based upper bound I: The basic color-based upper bound only considers the clique size constraint which ignores the clique probability constraint. Here we develop a tighter color-based upper bound based on both the clique size and clique probability constraints. Again, we assign a color to each node in $\tilde{G}$ using a greedy coloring algorithm [30]. Then, for a search subspace (R, C), we let r be the number of colors of the candidate nodes in C, i.e., $r = \mathsf{col}(C)$. Clearly, the nodes in C can be classified into r color groups $\{C_1, C_2, \cdots, C_r\}$ where the nodes in a group have the same color. For each candidate node $v \in C$, we define $\pi_v(R)$ as $\pi_v(R) \triangleq \prod_{u \in R} p_{uv}$. Let v_i^* be the node in C_i that has the largest $\pi_v(R)$ value, i.e., $v_i^* = \arg\max_{v \in C_i}\{\pi_v(R)\}$. Then, we sort the nodes $\{v_1^*, v_2^*, \cdots, v_r^*\}$ in a non-increasing order based on the $\pi_{v_i^*}(R)$ value. Assume without loss of generality that $\pi_{v_1^*}(R) \geq \pi_{v_2^*}(R) \geq \cdots \geq \pi_{v_r^*}(R)$. Let $f_i \triangleq \prod_{j=1}^{i} \pi_{v_j^*}(R)$. Then, we define $\bar{r}$ as

$$\bar{r} \triangleq \arg\max_{i \in \{1,\cdots,r\}} \{f_i \times \mathsf{CPr}(R, \mathcal{G}) \geq \tau\}. \quad (8)$$

By Eq. (8), we have the following result.

Lemma 6. *For any search subspace (R, C), $|R| + \bar{r}$ is an upper bound of the size of a maximum (k,τ)-clique contained in (R, C).*

Proof. The lemma can be proved by a contradiction. Suppose that there is a (k,τ)-clique H in (R, I) with $|H| > |R| + \bar{r}$.

Since H contains R, we have $|H \setminus R| > \bar{r}$. Note that H is a (k,τ)-clique, thus we have $\prod_{v \in H \setminus R} \pi_v(R) \times \mathsf{CPr}(R, \mathcal{G}) \geq \mathsf{CPr}(H, \mathcal{G}) \geq \tau$. Since H is a clique, the nodes in $H \setminus R$ have different colors. Based on this, we can derive that $\prod_{v \in H \setminus R} \pi_v(R) \leq \prod_{j=1}^{|H \setminus R|} \pi_{v_j^*}(R) = f_{|H \setminus R|}$. As a result, we have $f_{|H \setminus R|} \times \mathsf{CPr}(R, \mathcal{G}) \geq \tau$. Note that by Eq. (8), we have $\bar{r} \geq |H \setminus R|$ which is a contradiction. □

Clearly, the upper bound $|R| + \bar{r}$ is tighter than the basic color-based upper bound $|R| + \mathsf{col}(C)$, as $\bar{r} \leq \mathsf{col}(C)$ by Eq. (8). Below, we analyze the time complexity for computing $\bar{r}$ in each search subspace (R, C). First, for all nodes in C, the total cost for computing $\pi_v(R)$ $(v \in C)$ is bounded by $O(\sum_{v \in C} d_v(\tilde{G}))$. Second, it takes $O(|r| \log_2 |r|)$ time to sort the nodes in C. Thus, the time overhead for calculating $\bar{r}$ is $O(|r| \log_2 |r| + \sum_{v \in C} d_v(\tilde{G}))$. Since both r and $|C|$ are not very large, this *advanced* color-based upper bound can also be efficiently computed.

Advanced color-based upper bound II: Here we develop a different type of color-based upper bound which is also tighter than the basic color-based upper bound. Again, we let $\{C_1, C_2, \cdots, C_r\}$ be the r color groups of the nodes in C. For each $u \in R$, we define $\pi_i^*(u) \triangleq \max\limits_{v \in C_i, e=(u,v)} \{p_e\}$ for $i \in \{1, 2, \cdots, r\}$. Then, we sort $\{\pi_1^*(u), \pi_2^*(u), \cdots, \pi_r^*(u)\}$ in a non-increasing order. Assume without loss of generality that $\pi_1^*(u) \geq \pi_2^*(u) \geq \cdots \geq \pi_r^*(u)$. Let $g_i^u \triangleq \prod_{j=1}^{i} \pi_j^*(u)$. Then, we define r_u as

$$r_u \triangleq \arg \max_{i \in \{1, \cdots, r\}} \{g_i^u \times \mathsf{CPr}(R, \mathcal{G}) \geq \tau\}. \tag{9}$$

Based on Eq. (9), we define $\bar{s} \triangleq \min\limits_{u \in R} \{r_u\}$. Then, we have the following result.

Lemma 7. *For any search subspace (R, C), $|R| + \bar{s}$ is an upper bound of the size of a maximum (k,τ)-clique contained in (R, C).*

Proof. We prove this lemma by a contradiction. Suppose to the contrary that there is a (k,τ)-clique H contained in the search subspace (R, C) with $|H| > |R| + \bar{s}$. Then, we have $|H \setminus R| > \bar{s}$. Since H is a clique in $\tilde{G}$, the nodes in $H \setminus R$ have different colors. Consider a node $u \in R$ and $v \in H \setminus R$. Assume without loss of generality that $v \in C_i$. Let $p_i(u)$ be the probability of the edge (u, v) where $v \in C_i$. Clearly, by definition, we have $p_i(u) \leq \pi_i^*(u)$. Thus, we have $\prod_{j=1}^{|H \setminus R|} p_j(u) \leq \prod_{j=1}^{|H \setminus R|} \pi_j^*(u) = g_{|H \setminus R|}^u$. Since H is a (k,τ)-clique, we have $\prod_{j=1}^{|H \setminus R|} p_j(u) \times \mathsf{CPr}(R, \mathcal{G}) \geq \mathsf{CPr}(H, \mathcal{G}) \geq \tau$. Therefore, by Eq. (9), we can obtain that $r_u \geq |H \setminus R|$. The above argument holds for any $u \in R$, thus we have $\bar{s} \triangleq \min\limits_{u \in R} \{r_u\} \geq |H \setminus R|$, which is a contradiction. □

The upper bound $|R| + \bar{s}$ is tighter than the basic color-based upper bound, because $\bar{s} \leq \mathsf{col}(C)$ by Eq. (9). We analyze the time complexity for computing $\bar{s}$ in each search subspace (R, C) as follows. First, the time cost for calculating $\pi_i^*(u)$ for all $u \in R$ is bounded by $O(\sum_{u \in R} d_u(\tilde{G}))$. Second, the time overhead for sorting $\pi_i^*(u)$ is $O(r \log_2 r)$. Thus, the total time complexity for computing $\bar{s}$ is $O(r \log_2 r + \sum_{u \in R} d_u(\tilde{G}))$. Since both r and $|R|$ are not very large, the upper bound $|R| + \bar{s}$ can be efficiently derived.

TABLE I
DATASETS

Dataset	$n = \|V\|$	$m = \|E\|$	$d_{\max}$	δ
AskUbuntu	157,522	455,691	5,401	48
SuperUser	192,409	714,570	14,294	61
CaHepTh	22,908	2,444,798	8,718	561
WikiTalk	1,094,018	2,787,967	141,951	124
DBLP	1,843,614	8,350,259	2,213	279

Implementation details. Note that we can easily integrate the above upper bounds into Algorithm 4 to find a maximum (k,τ)-clique. Specifically, we first maintain the size of the largest maximal (k,τ)-clique σ found so far in line 9. Then, in line 12 of Algorithm 4, we first compute the basic color-based upper bound, i.e., $|R| + \mathsf{col}(C)$. If $|R| + \mathsf{col}(C) \leq \sigma$, the algorithm can safely prune the current search subspace. Otherwise, the algorithm computes the above two advanced color-based upper bounds to determine whether the current search subspace can be pruned or not. In our experiments, we will show that the algorithm equipped with all the above color-based upper bounds is two orders of magnitude faster than the state-of-the-art algorithm [21].

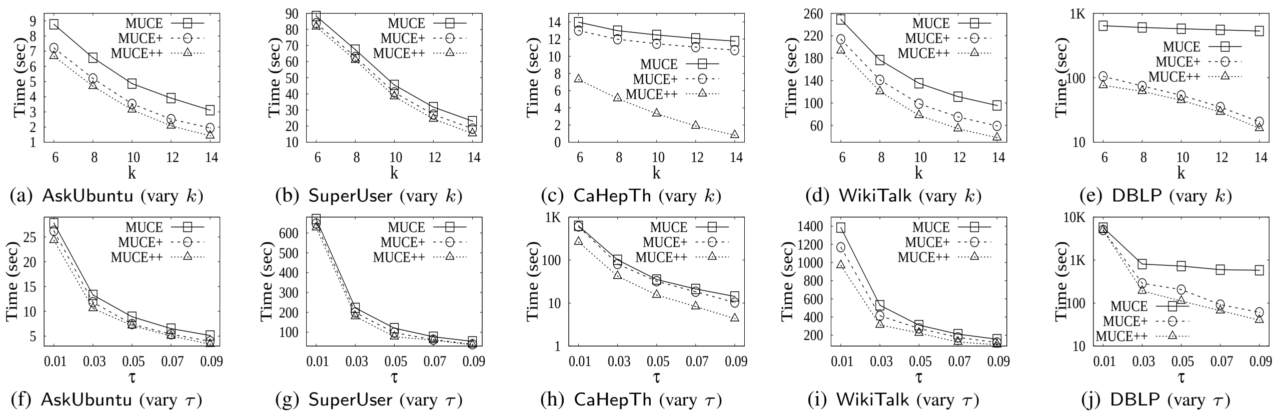
VI. EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the efficiency and effectiveness of the proposed algorithms. Below, we first describe the experimental setup and then report our results.

A. Experimental setup

We implement two algorithms DPCore and DPCore+ to compute the (k,τ)-core. DPCore is the state-of-the-art DP algorithm proposed in [16], and DPCore+ is our DP algorithm (Algorithm 2). To enumerate all maximal (k,τ)-cliques, we implement three algorithms: MUCE, MUCE+, and MUCE++. MUCE is the state-of-the-art algorithm proposed in [18], [19]. Essentially, MUCE is Algorithm 4 that does not use any core-based pruning rule developed in this paper, but it is integrated all pruning rules proposed in [19]. MUCE+ is Algorithm 4 with the (k,τ)-core pruning rule. MUCE++ is Algorithm 4 with the (Top_k,τ)-core pruning rule. Note that both MUCE+ and MUCE++ are integrated with the cut-based optimization technique (see line 3 in Algorithm 4). We also implement three algorithms for finding one of maximum (k,τ)-cliques: MaxUC, MaxRDS, and MaxUC+. MaxUC is a variant of Algorithm 4 that only uses the candidate set size based upper bound for pruning. MaxRDS is the state-of-the-art maximum (k,τ)-clique search algorithm proposed in [21], and MaxUC+ is our algorithm that is integrated with three color-based upper bounds proposed in Section V. All algorithms are implemented in C++. We conduct all experiments on a PC with a 2.4GHz Xeon CPU and 16GB memory running Red Hat Linux 6.4.

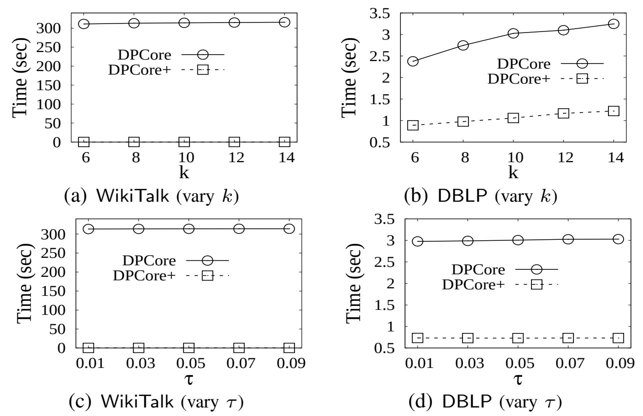
Datasets. We make use of five real-world graphs to evaluate the efficiency of various algorithms. Table I provides the statistics, where the last two columns denote the maximum



(a) AskUbuntu (vary k) (b) SuperUser (vary k) (c) CaHepTh (vary k) (d) WikiTalk (vary k) (e) DBLP (vary k)

(f) AskUbuntu (vary τ) (g) SuperUser (vary τ) (h) CaHepTh (vary τ) (i) WikiTalk (vary τ) (j) DBLP (vary τ)

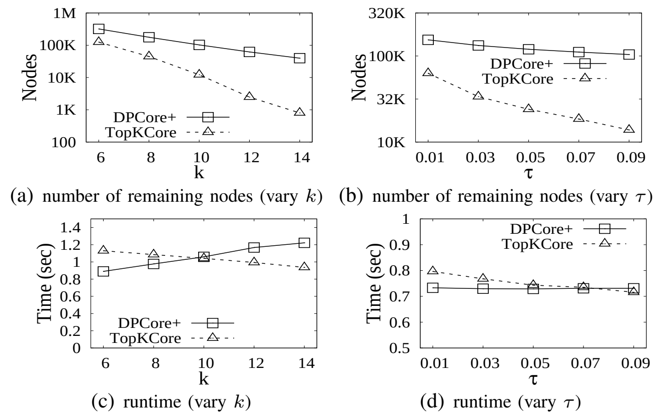
Fig. 3. Runtime of different algorithms for enumerating all maximal (k, τ)-cliques.



(a) WikiTalk (vary k) (b) DBLP (vary k)

(c) WikiTalk (vary τ) (d) DBLP (vary τ)

Fig. 2. Runtime of DPCore and DPCore+.



(a) number of remaining nodes (vary k) (b) number of remaining nodes (vary τ)

(c) runtime (vary k) (d) runtime (vary τ)

Fig. 4. Comparison between different core-based pruning rules (DBLP).

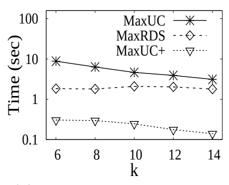
degree and degeneracy of the graph respectively. AskUbuntu, SuperUser, and WikiTalk are weighted *online communication* networks where the weight on an edge denotes the number of interactions between two users. Both CaHepTh and DBLP are weighted scientific collaboration networks where the weight on an edge denotes the number of papers co-authored by two researchers. AskUbuntu, SuperUser, and WikiTalk are downloaded from the Stanford network dataset collection (snap.stanford.edu), CaHepTh is downloaded from (konect.uni-koblenz.de/), and DBLP is extracted from (dblp.uni-trier.de/xml). We adopt a standard method in the uncertain graph mining literature [23], [24] to generate an uncertain graphs for each dataset. In particular, for each edge (u, v), we make use of an exponential cumulative distribution with mean $\lambda = 2$ to the weight of (u, v) to generate a probability (i.e., $p_{uv} = 1 - \exp(-w_{uv}/\lambda)$ [23], [24]).

Parameters. There are two parameters in our algorithms: k and τ. The parameter k is chosen from the interval $[6, 14]$ with a default value of $k = 10$; τ is selected from the interval $[0.01, 0.1]$ with a default value of $\tau = 0.1$. Unless otherwise specified, the value of the other parameter is set to its default value when varying a parameter.

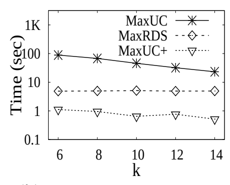
B. Efficiency results

Exp-1: Runtime of DPCore **and** DPCore+**.** Fig. 2 shows the runtime of DPCore and DPCore+ on WikiTalk and DBLP datasets with varying parameters. Similar results can also be observed on the other datasets. We can see that the runtime of both DPCore and DPCore+ are robust with varying k and τ. As desired, DPCore+ is much faster than DPCore on both WikiTalk and DBLP. Moreover, DPCore+ is three orders of magnitude faster than DPCore on WikiTalk. For instance, when $k = 10$ and $\tau = 0.1$, DPCore+ only takes 0.28 seconds while DPCore consumes 314 seconds on WikiTalk. This is because on the WikiTalk dataset, the maximum degree is much larger than the degeneracy (see Table I), thus our DPCore+ algorithm is much faster than DPCore. This result confirms our theoretical analysis in Section III-A.

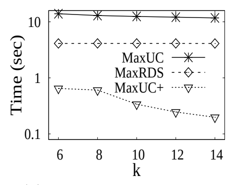
Exp-2: Runtime of MUCE**,** MUCE+**, and** MUCE++**.** In this experiment, we evaluate the runtime of MUCE, MUCE+, and MUCE++ for enumerating all maximal (k, τ)-cliques. Fig. 3 reports the runtime of these algorithms on all datasets with varying values for k and τ. As can be seen, MUCE+ is consistently faster than MUCE, and MUCE++ is significantly faster than MUCE+ under all parameter settings. Moreover, we can see that on large datasets, both MUCE++ and MUCE+ are at least one order of magnitude faster than MUCE. For example, in Fig. 3(e), when $k = 10$ and $\tau = 0.1$, MUCE++, MUCE+, and MUCE takes 44.7 seconds, 53.4 seconds, and 576.2 seconds to enumerate all maximal (k, τ)-cliques on DBLP, respectively. These results confirm that our pruning techniques developed in Section III are indeed very powerful in enumerating all maximal (k, τ)-cliques on large uncertain graphs. In general, the runtime of all the three algorithms decrease as k (or τ) increases. This is because when k (or



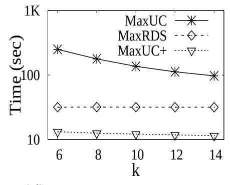
(a) AskUbuntu (vary k)



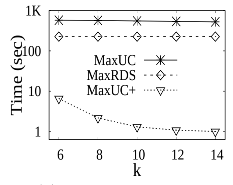
(b) SuperUser (vary k)



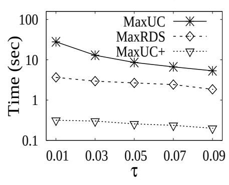
(c) CaHepTh (vary k)



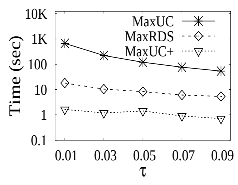
(d) WikiTalk (vary k)



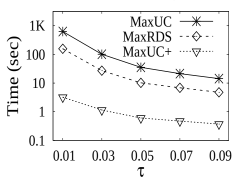
(e) DBLP (vary k)



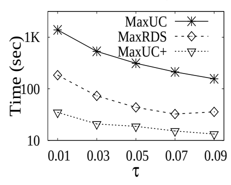
(f) AskUbuntu (vary τ)



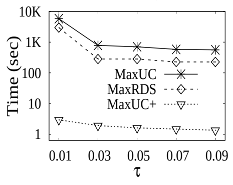
(g) SuperUser (vary τ)



(h) CaHepTh (vary τ)

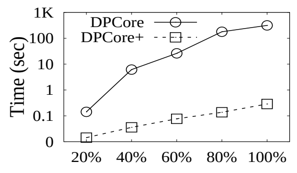


(i) WikiTalk (vary τ)

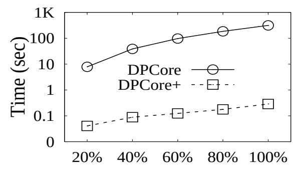


(j) DBLP (vary τ)

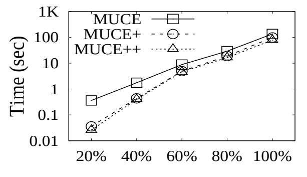
Fig. 5. Runtime of different algorithms for finding a maximum (k,τ)-clique.



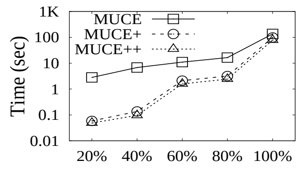
(a) vary $|V|$



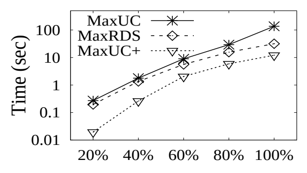
(b) vary $|E|$



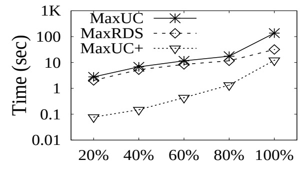
(c) vary $|V|$



(d) vary $|E|$



(e) vary $|V|$



(f) vary $|E|$

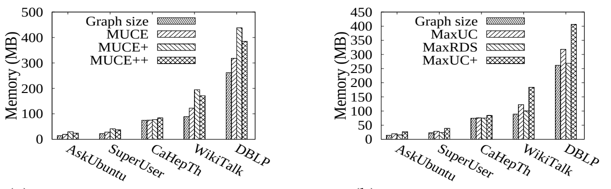
Fig. 6. Scalability of various algorithms (WikiTalk).

τ) increases, the number of maximal (k,τ)-cliques decreases and the algorithm can prune search subspaces early in the enumeration, thus the time overhead of the algorithm decreases.

Exp-3: Comparison of the core-based pruning rules. In this experiment, we evaluate the pruning performance of the (k,τ)-core and (Top_k,τ)-core pruning techniques. Fig. 4 shows the results on DBLP using the default parameter setting. The results on the other datasets are consistent. From Figs. 4(a-b), we can see that the number of remaining nodes obtained by the (Top_k,τ)-core pruning is much smaller than that obtained by the (k,τ)-core pruning rule. For example, when $k=14$ and $\tau=0.1$, the number of remaining nodes on DBLP is only 802 after performing the (Top_k,τ)-core pruning, while using the (k,τ)-core pruning rule, the number of remaining nodes is 39,823. These results indicate that the (Top_k,τ)-core pruning is more effective than the (k,τ)-core pruning in enumerating all maximal (k,τ)-cliques, which also confirms our theoretical analysis in Section III. In addition, from Figs. 4(c-d), we can observe that the time overhead for computing the (k,τ)-core and (Top_k,τ)-core are comparable with all parameter settings. This is because both (k,τ)-core and (Top_k,τ)-core can be computed in near linear time using the proposed algorithms.

Exp-4: Runtime of MaxUC, MaxRDS, and MaxUC+. Here we evaluate the performance of various algorithms for finding one of maximum (k,τ)-cliques. Fig. 5 shows the results on all datasets with varying values for parameters k and τ. As can be seen, MaxUC+ significantly outperforms other competitors. For example, when $k=10$ and $\tau=0.1$, MaxUC+ only takes 1.3 seconds to find a maximum (k,τ)-clique on DBLP, while MaxRDS consumes 225.8 seconds and MaxUC uses 553.7 seconds. In this case, our MaxUC+ algorithm is two orders of magnitude faster than MaxRDS and MaxUC. These results indicate that our color-based upper-bounding techniques are very effective to prune the search paths in finding a maximum (k,τ)-clique. Moreover, we can see that MaxUC+ takes less than 10 seconds on all datasets under most parameter settings. These results further demonstrate the high efficiency of our MaxUC+ algorithm. Generally, the runtime of all three algorithms decrease with increasing k or τ. The reason is because with a larger value of k (or τ), the algorithm can prune search subspaces aggressively early in the search procedure.

Exp-5: Scalability testings. We use the WikiTalk dataset to test the scalability of all the proposed algorithms. Specifically, we generate four subgraphs by randomly sampling 20-80% of the nodes (edges) from WikiTalk and evaluate the time costs of our algorithms on these subgraphs. Figs. 6(a-b) show the scalability results of DPCore and DPCore+ under the default parameter setting. As can be seen, the runtime of our DPCore+ algorithm increases smoothly with a varying $|V|$ or $|E|$, while the runtime of DPCore increases sharply. These results demonstrate the high scalability of our DPCore+ algorithm. Figs. 6(c-d) depict the scalability results of MUCE, MUCE+, and MUCE++. We can see that the runtime of all three algorithms increase not very sharply with increasing $|V|$ or $|E|$. Moreover, our algorithms (MUCE+ and MUCE++) consistently outperform MUCE. These results indicate that our algorithms are scalable when handling large uncertain graphs. Similarly, in Figs. 6(e-f), the results suggest that our MaxUC+



(a) maximal clique enumeration (b) maximum clique search

Fig. 7. Memory overhead of various algorithms.

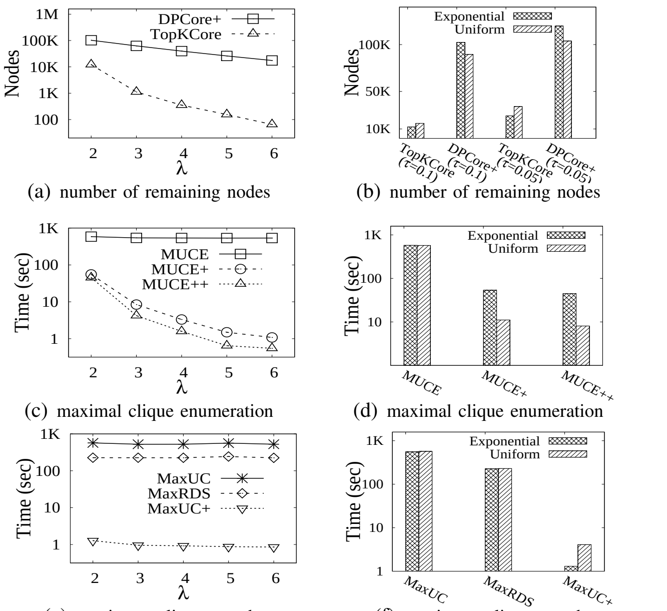
algorithm exhibits very good scalability in computing one of maximum (k, τ)-cliques.

Exp-6: Memory overhead. Fig. 7(a) reports the memory overhead of three maximal (k, τ)-clique enumeration algorithms (MUCE, MUCE+, and MUCE++) for all datasets. The results demonstrate that the memory usages of all three algorithms are slightly higher than the graph size but clearly lower than twice the size of the graph. These results confirm the linear space complexity of Algorithm 4. Fig. 7(b) shows the memory consumption of three maximum (k, τ)-clique search algorithms (MaxUC, MaxRDS, and MaxUC+). As can be seen, all three algorithms take linear space cost w.r.t. the uncertain graph size. This is because all three algorithms follow a depth-first search manner, thus the space overhead is linear.

Exp-7: Effect of different probability distributions. Here we study the performance of our algorithms with different probability distributions. Recall that in all previous experiments, the probability on each edge is generated by an exponential distribution with a parameter λ. In this experiment, we first investigate the impact of parameter λ for various algorithms (varying λ from 2 to 6). Second, we make use of a uniform $[0, 1]$ distribution to generate edge probabilities for each dataset, and then evaluate the performance of different algorithms on these datasets. Fig. 8 shows the results of different algorithms on DBLP using default parameter values ($k = 10, \tau = 0.1, \lambda = 2$). Similar results can also be observed on other datasets or using other parameter values. For convenience, we refer to DBLP with uniform distribution as DBLP-U, and DBLP with exponential distribution as DBLP-E.

In Fig. 8(a), we can see that the number of remaining nodes obtained by both TopKCore and DPCore+ decreases with an increasing λ. This is because the probability of an edge decreases as λ increases, thus the size of both (k, τ)-core and (Top_k, τ)-core reduces when λ increases. As shown in Fig. 8(b), the pruning performance of TopKCore is slightly better on DBLP-E. For DPCore+, however, the pruning performance is better on DBLP-U. The reason could be that for a high-degree node u, the average edge probabilities (for all u's outgoing edges) generated by uniform distributions are typically smaller than those obtained by exponential distributions, but the top-k probabilities obtained by uniform distributions may be larger than those computed by exponential distributions. Additionally, Figs. 8(a-b) also show that TopKCore significantly outperforms DPCore+ for pruning, which is consistent with our previous results.

From Fig. 8(c), we can observe that the runtime of MUCE++ (or MUCE+) decreases with an increasing λ, due to the edge



(a) number of remaining nodes (b) number of remaining nodes

(c) maximal clique enumeration (d) maximal clique enumeration

(e) maximum clique search (f) maximum clique search

Fig. 8. Effect of different probability distributions (DBLP).

probabilities reducing. As shown in Fig. 8(d), the runtime of MUCE++ (or MUCE+) on DBLP-U is significantly lower than that on DBLP-E. This is because the number of maximal (k, τ)-cliques on DBLP-U is smaller than that on DBLP-E. Interestingly, from Figs. 8(e-f), we can see that the maximum clique search algorithms are robust with respect to different probability distributions in most cases. As an exception shown in Fig. 8(f), MaxUC+ performs significantly better on DBLP-E than that on DBLP-U. This is because the clique size on DBLP-U is typically smaller than that on DBLP-E, reducing the effectiveness of the color-based pruning techniques. In addition, we can clearly see that MaxUC+ significantly outperforms two baselines. These results further confirm the high efficiency of our MaxUC+ algorithm.

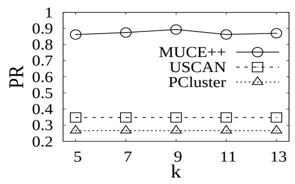
C. Effectiveness results

Here we conduct a case study on a protein-protein interaction (PPI) network to evaluate the effectiveness of the proposed algorithms. For this experiment, we use a real-world PPI network CORE which is an uncertain graph provided by Krogan et al. [31]. In particular, CORE contains 2,708 nodes and 7,123 edges, where a node is a protein and an edge denotes an interaction between two proteins. Each edge in CORE is associated with a probability, denoting the confidence of the interaction between two proteins. In addition, we can obtain the ground-truth protein complexes for CORE based on the MIPS protein database [32], [33], where a protein complex is a cohesive subgraph of CORE. Below, we show the effectiveness of our algorithm to detect protein complexes in CORE.

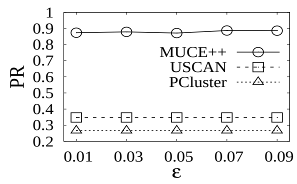
Note that based on the ground truth, we are able to calculate the number of true positives (TP), the number of false positives (FP), and the precision (PR=TP/(TP+FP)) obtained by various protein complex detection algorithms. More specifically, TP denotes the number of correctly matched interactions in predicted complexes with that in MIPS, and FP is the total number

TABLE II
PRECISION OF VARIOUS ALGORITHMS

Algorithm	#Results	TP	FP	PR
USCAN	456	1086	2037	0.348
PCluster	475	1027	3021	0.266
MUCE++	694	7020	940	**0.882**



(a) vary k



(b) vary τ

Fig. 9. Precision of different algorithms with varying parameters (CORE).

of interactions in predicted complexes minus TP. We adopt the same method to compute TP, FP, and PR as used in [32], [33]. We compare the proposed algorithm (MUCE++) with two state-of-the-art protein complex detection algorithms USCAN [33] and PCluster [32] based on the TP, FP, and PR metrics. For both USCAN and PCluster, we adopt the default parameter values as used in their original experiments [33], [32]. Table II shows the results of different algorithms. Note that the parameters of MUCE++ are also set to default values (i.e., $k = 10$ and $\tau = 0.1$). As can be seen, MUCE++ substantially outperforms the two baseline algorithms in terms of TP, FP, and PR. For example, the precision of MUCE++ is 0.882, while the precision of USCAN and PCluster is only 0.348 and 0.266 respectively. The reason is that each protein complex may be corresponding to a small cohesive subgraph, which can be well characterized by a maximal (k, τ)-clique. However, both USCAN and PCluster are clustering-based algorithms which may generate large-size clusters, thus decreasing precision. In Fig. 9, we also study the effect of parameters k and τ in our algorithm. The results show that the precision of MUCE++ is robust with respect to both k and τ. These results indicate that our algorithm is very effective to detect protein complexes in PPI networks.

VII. RELATED WORK

Uncertain graph mining. Mining uncertain graphs has attracted much attention in the database and data mining communities [34], [35], [16], [36], [14], [37], [38], [33]. Jin et al. [34] studied a problem of mining highly reliable connected subgraphs in an uncertain graphs. Liu et al. [35] proposed a clustering algorithm to identify reliable clusters in an uncertain graph. Bonchi et al. [16] investigated the k-core decomposition problem on uncertain graphs. Parchas et al. [36] proposed a framework to analyze uncertain graphs based on representative instances. Mehmood et al. [14] studied an influence cascade problem where the influence network is modeled as an uncertain graph. Huang et al. [37] proposed a k-truss search problem on uncertain graphs. Gao et al. [38] proposed an approach to find RkNN on uncertain graphs. More recently, Qiu et al. [33] studied the graph structural clustering problem on uncertain graphs. For the maximal clique mining problem, Zou et al. [39] proposed an algorithm to find the top-k maximal cliques in an uncertain graph. Mukherjee et al. [18], [19] simplified the maximal clique model proposed in [39], and proposed a backtracking enumeration algorithm to find all maximal cliques. However, the enumeration algorithm proposed in [18], [19] is very costly for large uncertain graphs. In this work, we focus mainly on developing fast solutions to enumerate maximal cliques in a large uncertain graph. We also propose an algorithm to find one of maximum cliques in an uncertain graph based on several novel pruning techniques.

Maximal clique mining. Maximal clique mining is a fundamental graph mining task which has been widely used in many real-world applications [40], [7], [41], [8], [9]. The practical algorithms for enumerating all maximal cliques are the classic Bron-Kerbosch algorithm [40] and its variants [7], [9]. Tomita et al. [7] proved that the Bron-Kerbosch algorithm with a greedy pivoting technique can achieve optimal time complexity for listing all maximal cliques. Eppstein et al. [9] further improved the Bron-Kerbosch algorithm based on a degeneracy ordering heuristics. Recently, Cheng et al. proposed an I/O-efficient maximal clique enumeration algorithm for disk-resident graphs [41] and a parallel maximal clique enumeration algorithm using limited memory [8]. The maximal clique enumeration problem has also been studied for special graph data. For instance, Viard et al. [42] investigated a problem of enumerating maximal cliques in temporal graphs, in which each edge is associated with a timestamp. Li et al. [43] studied a problem of enumerating maximal signed cliques in a signed graph, where the edges in the graph can be positive or negative.

Another related problem is to find a maximum clique in a graph where the goal is to identify a clique with the largest size. Most practical algorithms for this problem are based on the branch and bound search technique [44], [45], [46], [5], [47]. Ostergard [44] proposed a branch and bound algorithm with a Russian Doll Search framework to compute a maximum clique. Li [45] proposed a branch and bound algorithm based on a so-called MaxSAT bound which is shown to be better than the classic coloring based bound. Rossi et al. [46] proposed a parallel algorithm to find a maximum clique using the degeneracy and coloring based bounds. Lu et al. [5] presented a randomized algorithm to find near maximum clique in a large sparse graph. Tomita [47] proposed several efficient branch and bound algorithms to compute a maximum clique based on the coloring order heuristics. All these techniques are tailored to deterministic graphs and cannot be directly used for uncertain graphs. More recently, Miao et al. [21] proposed an algorithm for finding a maximum clique in an uncertain graph based on the classic Russian Doll Search framework. Their algorithm, however, is not very efficient for large uncertain graphs, as its time complexity is proportional to 2^n where n is the number of nodes in the uncertain graph. In this work, we develop a faster algorithm to compute a maximum clique in an uncertain graph based on several powerful pruning techniques.

VIII. CONCLUSION

In this paper, we develop several new solutions for maximal (k, τ)-clique search in uncertain graphs. Specifically, we first propose an efficient (k, τ)-core based pruning approach to prune the nodes in an uncertain graph which are not contained in any maximal (k, τ)-clique. We also develop a novel (Top_k, τ)-core based pruning strategy, as well as a cut-based optimization technique to further improve the pruning performance. On the pruned uncertain graph, we propose a new algorithm to enumerate all maximal (k, τ)-cliques, and a new algorithm to compute one of maximum (k, τ)-cliques based on several carefully-designed upper-bounding techniques. Comprehensive experiments on six real-world datasets demonstrate the efficiency and effectiveness of our algorithms.

References

[1] R.-H. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2453–2465, 2014.

[2] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," *SIGMOD*, 2014.

[3] L. Qin, R. Li, L. Chang, and C. Zhang, "Locally densest subgraph discovery," in *KDD*, 2015.

[4] R.-H. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *PVLDB*, vol. 8, no. 5, pp. 509–520, 2015.

[5] C. Lu, J. X. Yu, H. Wei, and Y. Zhang, "Finding the maximum clique in massive graphs," *PVLDB*, vol. 10, no. 11, pp. 1538–1549, 2017.

[6] R. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng, "Skyline community search in multi-valued networks," in *SIGMOD*, 2018.

[7] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theor. Comput. Sci.*, vol. 363, no. 1, pp. 28–42, 2006.

[8] J. Cheng, L. Zhu, Y. Ke, and S. Chu, "Fast algorithms for maximal clique enumeration with limited memory," in *KDD*, 2012.

[9] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *ACM Journal of Experimental Algorithmics*, vol. 18, 2013.

[10] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, pp. 814–818, 2005.

[11] G. Creamer, R. Rowe, S. Hershkop, and S. J. Stolfo, "Segmentation and automated social hierarchy detection through email network analysis," in *1st International Workshop on Social Networks Analysis, SNA-KDD*, 2007.

[12] 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*, vol. 24, no. 7, pp. 976–986, 2008.

[13] J. S. Bader, A. Chaudhuri, J. M. Rothberg, and J. Chant, "Gaining confidence in high-throughput protein interaction networks," *Nature Biotechnology*, vol. 22, no. 1, pp. 78–85, 2004.

[14] Y. Mehmood, F. Bonchi, and D. García-Soriano, "Spheres of influence for more effective viral marketing," in *SIGMOD*, 2016.

[15] H. Kawahigashi, Y. Terashima, N. Miyauchi, and T. Nakakawaji, "Modeling ad hoc sensor networks using random graph theory," in *CCNC*, 2005.

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

[17] X. Huang, W. Lu, and L. V. S. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *SIGMOD*, 2016.

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

[19] ——, "Enumeration of maximal cliques from an uncertain graph," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 3, pp. 543–555, 2017.

[20] R. Rymon, "Search through systematic set enumeration," in *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, 1992.

[21] Z. Miao, B. Balasundaram, and E. L. Pasiliao, "An exact algorithm for the maximum probabilistic clique problem," *J. Comb. Optim.*, vol. 28, no. 1, pp. 105–120, 2014.

[22] S. B. Seidman, "Network structure and minimum degree," *Social Networks*, vol. 5, no. 3, pp. 269–287, 1983.

[23] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios, "k-nearest neighbors in uncertain graphs," *PVLDB*, vol. 3, no. 1, pp. 997–1008, 2010.

[24] R. Jin, L. Liu, B. Ding, and H. Wang, "Distance-constraint reachability computation in uncertain graphs," *PVLDB*, vol. 4, no. 9, pp. 551–562, 2011.

[25] R. Li, J. X. Yu, R. Mao, and T. Jin, "Efficient and accurate query evaluation on uncertain graphs via recursive stratified sampling," in *ICDE*, 2014.

[26] ——, "Recursive stratified sampling: A new framework for query evaluation on uncertain graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 2, pp. 468–482, 2016.

[27] V. Batagelj and M. Zaversnik, "An O(m) algorithm for cores decomposition of networks," *CoRR*, vol. cs.DS/0310049, 2003.

[28] ——, "Fast algorithms for determining (generalized) core groups in social networks," *Adv. Data Analysis and Classification*, vol. 5, no. 2, pp. 129–145, 2011.

[29] M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM*, vol. 44, no. 4, pp. 585–591, 1997.

[30] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Ordering heuristics for parallel graph coloring," in *SPAA*, 2014.

[31] N. J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, S. Pu, N. Datta, A. P. Tikuisis, and others, "Global landscape of protein complexes in the yeast saccharomyces cerevisiae," *Nature*, vol. 440, no. 7084, pp. 637–643, 2006.

[32] G. Kollios, M. Potamias, and E. Terzi, "Clustering large probabilistic graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 2, pp. 325–336, 2013.

[33] Y.-X. Qiu, R.-H. Li, J. Li, S. Qiao, G. Wang, J. X. Yu, and R. Mao, "Efficient structural clustering on probabilistic graphs," *IEEE Transactions on Knowledge and Data Engineering*, to appear. [Online]. Available: https://doi.org/10.1109/TKDE.2018.2872553

[34] R. Jin, L. Liu, and C. C. Aggarwal, "Discovering highly reliable subgraphs in uncertain graphs," in *KDD*, 2011.

[35] L. Liu, R. Jin, C. C. Aggarwal, and Y. Shen, "Reliable clustering on uncertain graphs," in *ICDM*, 2012.

[36] P. Parchas, F. Gullo, D. Papadias, and F. Bonchi, "Uncertain graph processing through representative instances," *ACM Trans. Database Syst.*, vol. 40, no. 3, pp. 20:1–20:39, 2015.

[37] X. Huang, W. Lu, and L. V. S. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *SIGMOD*, 2016.

[38] Y. Gao, X. Miao, G. Chen, B. Zheng, D. Cai, and H. Cui, "On efficiently finding reverse k-nearest neighbors over uncertain graphs," *The VLDB Journal*, pp. 1–26, 2017.

[39] Z. Zou, J. Li, H. Gao, and S. Zhang, "Finding top-k maximal cliques in an uncertain graph," in *ICDE*, 2010.

[40] C. Bron and J. Kerbosch, "Finding all cliques of an undirected graph (algorithm 457)," *Commun. ACM*, vol. 16, no. 9, pp. 575–576, 1973.

[41] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Trans. Database Syst.*, vol. 36, no. 4, pp. 21:1–21:34, 2011.

[42] J. Viard, M. Latapy, and C. Magnien, "Computing maximal cliques in link streams," *Theor. Comput. Sci.*, vol. 609, pp. 245–252, 2016.

[43] R.-H. Li, Q. Dai, L. Qin, G. Wang, X. Xiao, J. X. Yu, and S. Qiao, "Efficient signed clique search in signed networks," in *ICDE*, 2018.

[44] P. R. J. Östergård, "A fast algorithm for the maximum clique problem," *Discrete Applied Mathematics*, vol. 120, no. 1-3, pp. 197–207, 2002.

[45] C. M. Li and Z. Quan, "An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem," in *AAAI*, 2010.

[46] R. A. Rossi, D. F. Gleich, and A. H. Gebremedhin, "Parallel maximum clique algorithms with applications to network analysis," *SIAM J. Scientific Computing*, vol. 37, no. 5, 2015.

[47] E. Tomita, "Efficient algorithms for finding maximum and maximal cliques and their applications," in *WALCOM*, 2017, pp. 3–15.