Core Decomposition on Uncertain Graphs Revisited

Qiangqiang Dai, Rong-Hua Li, Guoren Wang, Rui Mao, Zhiwei Zhang, and Ye Yuan

Abstract—Core decomposition on uncertain graphs is a fundamental problem in graph analysis. Given an uncertain graph $\mathcal{G}$, the core decomposition problem is to determine all (k,η)-cores in $\mathcal{G}$, where a (k,η)-core is a maximal subgraph of $\mathcal{G}$ such that each node has an η-degree no less than k within the subgraph. The η-degree of a node v is defined as the maximum integer r such that the probability that v has a degree no less than r is larger than or equal to the threshold $\eta \in [0,1]$. The state-of-the-art algorithm for solving this problem is based on a peeling technique which iteratively removes the nodes with the smallest η-degrees and also dynamically updates their neighbors' η-degrees. Unfortunately, we find that such a peeling algorithm with the dynamical η-degree updating technique is incorrect due to the inaccuracy of the recursive floating-point number division operations involved in the dynamical updating procedure. To correctly compute the (k,η)-cores, we first propose a bottom-up algorithm based on an on-demand η-degree computational strategy. To further improve the efficiency, we also develop a more efficient top-down algorithm with several nontrivial optimization techniques. Both of our algorithms do not involve any floating-point number division operations, thus the correctness can be guaranteed. In addition, we also develop the parallel variants of all the proposed algorithms. Finally, we conduct extensive experiments to evaluate the proposed algorithms using five large real-life datasets. The results show that our algorithms are at least three orders of magnitude faster than the existing exact algorithms on large uncertain graphs. The results also demonstrate the high scalability and parallel performance of the proposed algorithms.

Index Terms—Uncertain graphs, cohesive subgraph mining, uncertain core decomposition

1 Introduction

Real-world graphs, such as social networks, web graphs, and biological networks often contain cohesive subgraph structures. Mining cohesive subgraphs from a graph is a fundamental problem in graph analysis which has attracted much attention in the database and data mining communities [1], [2], [3], [4]. However, many real-world graphs, such as protein-protein interaction (PPI) networks [5], sensor networks [6], and influence networks [7], are typically uncertain in nature, where each edge is associated with a probability denoting the likelihood of the existence of the edge.

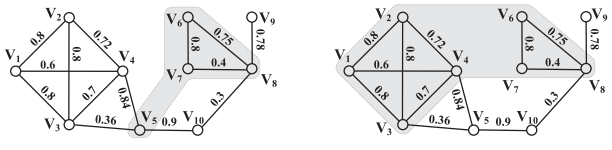
Recently, many cohesive subgraph mining models on uncertain graphs have been proposed. Notable examples including the (k,η)-core model [8], [9], [10], the (k,η)-truss model [11], [12], and the uncertain maximal clique model [13], [14], [15]. In this paper, we focus mainly on the (k,η)-core model, since such a model is simple and easy to compute than the other models.

- *Qiangqiang Dai, Rong-Hua Li, Guoren Wang, Zhiwei Zhang, and Ye Yuan are with the Beijing Institute of Technology, Beijing 100081, China. E-mail: {qiangd66, lironghuascut}@gmail.com, wanggrbit@126.com, cszwzhang@outlook.com, yuanye@mail.neu.edu.cn.*
- *Rui Mao is with Shenzhen University, Shenzhen 518060, China. E-mail: mao@szu.edu.cn.*

Manuscript received 15 December 2020; revised 6 May 2021; accepted 27 May 2021. Date of publication 11 June 2021; date of current version 7 December 2022.
This work was supported in part by the National Key Research and Development Program of China under Grant 2020AAA0108503, in part by NSFC under Grants 62072034 and 61772346, and in part by CCF-Baidu Open Fund.
(Corresponding author: Rong-Hua Li.)
Recommended for acceptance by B. Glavic.
Digital Object Identifier no. 10.1109/TKDE.2021.3088504

Given an uncertain graph $\mathcal{G}$ and a probabilistic threshold $\eta \in [0,1]$, a (k,η)-core is a maximal subgraph $\mathcal{C}$ of $\mathcal{G}$ satisfying that each node in $\mathcal{C}$ has an η-degree no less than k. Here the η-degree of a node v is defined as the maximum integer r such that the probability that v has a degree no less than r is greater than or equal to $\eta \in [0,1]$. To compute all the (k,η)-cores of $\mathcal{G}$, Bonchi *et al.* [8] proposed a peeling algorithm with an efficient dynamical η-degree updating technique. The time complexity of such a peeling algorithm is $O((m+n)d_{\max})$, where m, n, and $d_{\max}$ denote the number of edges, the number of nodes, and the maximum degree of the deterministic graph of $\mathcal{G}$ respectively. Recently, Yang *et al.* [9] proposed an index-based algorithm to query the (k,η)-core of an uncertain graph $\mathcal{G}$ in optimal time for any k and η, where the index is constructed based on the peeling algorithm proposed in [8]. Li *et al.* [15] proposed an improved peeling algorithm with time complexity $O((m+n)\delta)$ to compute all the (k,η)-cores of $\mathcal{G}$, where δ ($\delta \leq d_{\max}$) is the maximum core number of the deterministic graph of $\mathcal{G}$.

Unfortunately, we discover that all the above mentioned peeling algorithms [8], [9], [15] that are based on the dynamical η-degree updating technique are incorrect. The reason is that the dynamical η-degree updating procedure involves a recursive floating-point number division operation which will rapidly increase the floating-point number errors when updating the η-degrees. More specifically, when updating the η-degree of a node u using the algorithms proposed in [8], [9], [15], the floating-point number error can be up to $O(\frac{1}{1-p_e})^i$ where i is η-degree of u and p_e is the probability of an edge incident to u, and thus the η-degrees computed by the dynamical updating procedure is extremely imprecise. As a consequence, the (k,η)-cores computed by the state-of-the-art



(a) Accurate results (b) Inaccurate results

Fig. 1. Running example of core decomposition ($\eta = 0.3$, the gray area contains the nodes with η-core numbers equaling 2).

peeling algorithms [8], [9], [15] are incorrect. As an illustrative example shown in Fig. 1, the correct η-core numbers of the nodes $\{v_5, \ldots, v_8\}$ are 2, while the $(2, \eta)$-core obtained by the state-of-the-art algorithms is $\{v_1, \ldots, v_4, v_6, \ldots, v_8\}$ which is incorrect.

To precisely compute the (k, η)-cores, a basic solution is to re-compute the η-degree of a node when it needs to update. However, such a basic solution is inefficient due to a large number of from-scratch re-computations of the η-degrees. To overcome this critical issue, we first develop a bottom-up algorithm with an on-demand η-degree re-computation technique which can efficiently and precisely compute all (k, η)-cores on $\mathcal{G}$. Specifically, the bottom-up algorithm applies a lower bound of the η-core number to prune unnecessary η-degree re-computations, and also adopts a lazy update technique to reduce redundant computations. To further improve the efficiency, we also propose a top-down algorithm with several non-trivial optimization techniques. The top-down algorithm computes the (k, η)-cores following a decreasing order of k (i.e., it first computes the (k, η)-core with largest k). The striking feature of this algorithm is that in many cases the η-degrees can be computed by an incremental updating technique without sacrificing accuracy. In addition, we also develop parallel variants of all the proposed algorithms to further improve the scalability of our algorithms. Finally, we conduct extensive experiments using five large real-world datasets to evaluate the proposed algorithms. The results show that the top-down algorithm can achieve around $40\times$ speedup over the bottom-up algorithm and at least $1000\times$ speedup over the basic algorithm on a large uncertain graph with 2,180,759 nodes and 228,985,632 edges. The results also reveal that the speedup ratio of our parallel bottom-up and top-down algorithms can be up to 10 when using 16 threads. Additionally, we also evaluate the accuracy of the state-of-the-art peeling algorithms [8], [9], [15]. The results show that the η-core numbers of the nodes obtained by these algorithms are nearly 100% incorrect for a large k, indicating that our work is critical.

In summary, the main contributions of this work are as follows.

- We point out a critical error in the state-of-the-art algorithms for computing the (k, η)-cores on uncertain graphs.
- We propose two new algorithms which can efficiently and correctly compute the (k, η)-cores on uncertain graphs. We also develop parallel variants for all our algorithms.
- We conduct extensive experiments using five large real-world datasets to show the efficiency, scalability, and accuracy of the proposed algorithms. The source code of this paper is publicly accessible at (https://github.com/qq-dai/UncertainCore).

Organizations. Section 2 introduces the notations, discuss the existing solutions and analyze their defects. The proposed bottom-up and top-down algorithms are presented in Sections 3 and 4 respectively. The parallel versions of our algorithms are studied in Section 5. Section 6 reports the experimental results. We review related works in Section 7 and conclude this paper in Section 8.

2 Preliminaries

Core Decomposition on Deterministic Graphs. Let $G = (V, E)$ be a deterministic undirected graph, where V and E are the set of nodes and edges respectively. Denote by n and m the number of nodes and edges of G respectively. The neighbor set of $v \in V$ is denoted by $N_v(G) = \{u | (v, u) \in E\}$. The degree of $v \in V$, denoted by $d_v(G)$, is the cardinality of $N_v(G)$, i.e., $d_v(G) = |N_v(G)|$. Let $G(C) = (V_C, E_C)$ be an induced subgraph of G if $V_C \in V$ and $E_C = \{(v, u) \in E | v, u \in V_C\}$. Given a deterministic undirected graph $G = (V, E)$ and a positive integer k, a k-core is a maximal induced subgraph $G(C)$ in which every node $v \in V_C$ has a degree no less than k, i.e., $\forall v \in V_C, d_v(G(C)) \geq k$ [16]. The core number of a node v in G, denoted by $\mathsf{core}(v)$, is the largest integer k such that there exists a k-core containing u. The core decomposition of a deterministic graph G is a problem of computing the core numbers for all nodes in G. As shown in [17], such a core decomposition problem can be solved in $O(n + m)$ time.

Core Decomposition on Uncertain Graphs. Let $\mathcal{G} = (V, E, p)$ be an uncertain graph, where p is a function that maps the existence probability of each edge to a real value in [0,1]. We denote by $N_v(\mathcal{G}) = \{u | (u, v) \in E\}$ the neighborhood of $v \in V$ in $\mathcal{G}$. Similarly, the degree of $v \in V$ is denoted by $d_v(\mathcal{G}) = |N_v(\mathcal{G})|$. Following the standard uncertain graph model [18], [19], [20], [21], we assume that the existence probability of each edge in $\mathcal{G}$ is independent. Based on this, the well-known possible world semantics [18] can be applied to analyze uncertain graphs. More specifically, let $G = (V, E_G)$ be a possible world of $\mathcal{G}$. Then, the probability of an observing possible world G, called $Pr(G)$, is defined as

$$Pr(G) = \prod_{e \in E_G} p_e \prod_{e \in E/E_G} (1 - p_e). \tag{1}$$

Clearly, there are $2^{|E|}$ possible worlds for an uncertain graph $\mathcal{G}$. Based on the possible word semantic, Bonchi *et al.* [8] introduced a k-core model for uncertain graphs, called (k, η)-core, which is defined as follows.

Definition 1 ((k, η)-core). *Given an uncertain graph $\mathcal{G}$ and a probability threshold $\eta \in [0, 1]$, the (k, η)-core is a maximal induced subgraph $\mathcal{G}' = (V', E', p)$ in which the probability that each node has a degree no less than k is greater than or equal to η, i.e., $\forall v \in V', Pr[d_v(\mathcal{G}') \geq k] \geq \eta$.*

Based on the concept of (k, η)-core, the core number of a node v in $\mathcal{G}$, denoted by η-$\mathsf{core}(v)$, is the largest k such that there exist a (k, η)-core containing v. Given an uncertain graph $\mathcal{G}$, and a probability threshold $\eta \in [0, 1]$, the core decomposition on an uncertain graph is a problem of determining the η-core number for each node in $\mathcal{G}$.

2.1 Existing Solutions

Let $\mathcal{G}_v^{\geq k}$ be the set of all possible world subgraphs of $\mathcal{G}$ such that each node in a possible world subgraph has a degree no less than k. Therefore, the probability $Pr[d_v(\mathcal{G}) \geq k]$ of a node v can be computed by

$$Pr[d_v(\mathcal{G}) \geq k] = \sum_{G \in \mathcal{G}_v^{\geq k}} Pr(G). \tag{2}$$

By Eq. (2), the definition of the η-degree of a node v in $\mathcal{G}$, denoted by $\eta\text{-deg}_v(\mathcal{G})$, is given as follows.

Definition 2 (η-degree). *Given an uncertain graph $\mathcal{G}$ and a probability threshold $\eta \in [0, 1]$, the η-degree of $v \in V$ in $\mathcal{G}$ is the largest integer k that satisfies $Pr[d_v(\mathcal{G}) \geq k] \geq \eta$.*

Based on Definition 2, Bonchi *et al.* [8] developed a peeling algorithm to compute the η-core number for each node in the uncertain graph. Specifically, the peeling algorithm iteratively removes the node that has the minimum η-degree among all remaining nodes in $\mathcal{G}$. The key step of this peeling algorithm is to compute and update the η-degrees of nodes. Below, we describe the dynamic programming (DP) algorithm proposed in [8] to compute the η-degrees.

The DP Algorithm. First, we can easily derive the following equation to compute the η-degree of a node

$$Pr[d_v(\mathcal{G}) \geq k] = 1 - \sum_{i=0}^{k-1} Pr[d_v(\mathcal{G}) = i]. \tag{3}$$

By Eq. (3), the η-degree of $v \in V$ can be obtained by computing $Pr[d_v(\mathcal{G}) = i]$. Let $E_v(\mathcal{G}) = \{e_1, e_2, \ldots, e_{d_v(\mathcal{G})}\}$ be the set of edges that incident to v in $\mathcal{G}$. Denote by $E_v^h(\mathcal{G}) = \{e_1, e_2, \ldots, e_h\}$ a subset that contains the first h edges of $E_v(\mathcal{G})$, where $h \leq d_v(\mathcal{G})$. Denote by $d_v(\mathcal{G}_v^h)$ the degree of v in $\mathcal{G}_v^h \subseteq \mathcal{G}$, where $\mathcal{G}_v^h = (V, E \setminus (E_v(\mathcal{G}) \setminus E_v^h(\mathcal{G})), p)$ is a subgraph of $\mathcal{G}$ that excludes the edges in $E_v(\mathcal{G}) \setminus E_v^h(\mathcal{G})$. The DP algorithm is based on the following observation. If a node v has a degree i, then there are two cases to be considered: (i) either the edge $e_{d_v(\mathcal{G})}$ exists and $d_v(\mathcal{G}_v^{d_v(\mathcal{G})-1}) = i - 1$; (ii) or the edge $e_{d_v(\mathcal{G})}$ does not exist and $deg_v(\mathcal{G}_v^{d_v(\mathcal{G})-1}) = i$. Denote by $X_v(h, i)$ the probability of a node v that has a degree i in $\mathcal{G}_v^h$. Then, the DP equation is as follows:

$$X_v(h, i) = p_{e_i} X_v(h-1, i-1) + (1 - p_{e_i}) X_v(h-1, i). \tag{4}$$

The initial states of the DP equation are $X_v(0, 0) = 1$, $X_v(i, -1) = 0$ for all $i \in [0, d_v(\mathcal{G})]$, and $X_v(h, i) = 0$ for all $0 \leq h < i \leq d_v(\mathcal{G})$. Obviously, $X_v(d_v(\mathcal{G}), i)$ is equal to $Pr[d_v(\mathcal{G}) = i]$ for a node v, where $i \in [0, d_v(\mathcal{G})]$. The time complexity for computing the η-degree of a node v is $O(d_v(\mathcal{G})\eta\text{-deg}_v(\mathcal{G}))$.

Note that after removing a node v, the peeling algorithm needs to update the η-degrees of the neighbor nodes of v. A basic algorithm is to recompute the η-degrees for all neighbors of v using the DP algorithm. Clearly, such an algorithm is costly. To avoid from-scratch re-computations, Bonchi *et al.* [8] proposed a more-efficient updating technique which can dynamically update the η-degrees of the neighbor nodes. Let $e = (v, u) \in E_v(\mathcal{G})$ be an incident edge v. When removing a node v, the edge e is also deleted. Let $\mathcal{G}_{\neg e} = (V, E \setminus e, p)$ be the subgraph of $\mathcal{G}$ after deleting e. The η-degree of u can be updated by computing $Pr[d_v(\mathcal{G}_{\neg e}) = i]$, where $i \in [0, \eta\text{-deg}_u(\mathcal{G})]$. Specifically, $Pr[d_v(\mathcal{G}_{\neg e}) = i]$ can be computed by the following updating equation:

$$Pr[d_u(\mathcal{G}_{\neg e}) = i] = \frac{Pr[d_u(\mathcal{G}) = i] - p_e Pr[d_u(\mathcal{G}_{\neg e}) = i - 1]}{1 - p_e}. \tag{5}$$

Given that $Pr[d_u(\mathcal{G}_{\neg e}) = 0] = Pr[d_u(\mathcal{G}) = 0]/(1 - p_e)$, the probability $Pr[d_u(\mathcal{G}_{\neg e}) = i]$ for all $i \in [1, \eta\text{-deg}_u(\mathcal{G})]$ can be calculated by Eq. (5) in $O(\eta\text{-deg}_u(\mathcal{G}))$ time.

As shown in [8], the worst-case time complexity of the peeling algorithm equipped with the updating technique is $O((n + m)d_{\max})$, where $d_{\max}$ is the maximum degree among all nodes in $\mathcal{G}$. Recently, Li *et al.* [15] developed a more efficient algorithm based on an improved DP procedure and a similar updating technique which reduces the time complexity from $O((n + m)d_{\max})$ to $O((n + m)\delta)$, where δ ($\delta < d_{\max}$) is the maximum core number of the deterministic graph of $\mathcal{G}$.

2.2 Defect of Existing Solutions

Although the updating technique developed in [8], [9], [15] is elegant and is also able to significantly improve the efficiency of the peeling algorithm, it unfortunately cannot obtain correct η-degrees due to the inaccuracy of the recursive floating-point number division operation. As a consequence, the core decomposition obtained by the peeling algorithm with such an updating technique is incorrect. The reasons are analyzed as follows.

Since the probability of each edge is a floating-point number, it often cannot be precisely stored in modern computers. Let ϵ (e.g., $\epsilon \approx 10^{-16}$) be the error of the floating-point number representation of a computer. Then, in Eq. (5), the recursive division operation on floating-point numbers will rapidly increase the errors of the η-degrees. Suppose that the probabilities $Pr[d_u(\mathcal{G}) = i]$ for all $i \in [0, \eta\text{-deg}_u(\mathcal{G})]$ have an ϵ error. Then, by $Pr[d_u(\mathcal{G}_{\neg e}) = 0] = Pr[d_u(\mathcal{G}) = 0]/(1 - p_e)$, the error of $Pr[d_u(\mathcal{G}_{\neg e}) = 0]$ is increased to $\epsilon/(1 - p_e)$. By Eq. (5), it is easy to derive that the error of $Pr[d_u(\mathcal{G}_{\neg e}) = 1]$ is further enlarged to $\epsilon/(1 - p_e)^2$. As a result, the error of $Pr[d_u(\mathcal{G}_{\neg e}) = i]$ will be increased to $\epsilon/(1 - p_e)^{i+1}$ which is rather imprecise. Recall that to update the η-degree of u, we need to compute $Pr[d_u(\mathcal{G}_{\neg e}) = i]$ for all $i \in [0, \eta\text{-deg}_u(\mathcal{G})]$. Therefore, the error of the η-degree of a node u obtained by the updating technique can be up to $O(\epsilon/(1 - p_e)^k)$ where $k = \eta\text{-deg}_u(\mathcal{G})$, resulting in that the core decomposition is largely incorrect. The following illustrative example shows the incorrect core decomposition obtained by the peeling algorithm with the updating technique.

Example 1. Consider an uncertain graph $\mathcal{G}$ in Fig. 1. Suppose that the parameter $\eta = 0.3$ and the floating-point number precision of the computer is 10^{-3} (i.e., the computer only retains three significant digits for each probability). Fig. 1a shows the nodes with an accurate η-core number 2, while Fig. 1b shows the nodes with an η-core number 2 computed by the peeling algorithm with the updating technique. Specifically, the peeling algorithm first deletes the nodes v_9 and v_{10}, since their η-degrees are equal to 1. After removing v_{10}, the algorithm needs to update the η-degrees of its neighbor nodes v_5 and v_8. Let us consider the neighbor node v_5. Note that before removing v_{10}, the probabilities of each possible degree of v_5 are $Pr[d_{v_5} = 0] = 0.0102$, $Pr[d_{v_5} = 1] = 0.152$, $Pr[d_{v_5} = 2] = 0.566$ and $Pr[d_{v_5} = 3] = 0.272$, respectively.

After deleting the edge (v_{10}, v_5), the resulting probabilities of v_5 are $Pr[d_{v_5} = 0] = 0.102$, $Pr[d_{v_5} = 1] = (0.152 - 0.9 \times 0.102)/0.1 = 0.602$ and $Pr[d_{v_5} = 2] = (0.566 - 0.9 \times 0.602)/0.1 = 0.242$ respectively based on Eq. (5). Clearly, the η-degree of v_5 is equal to 1, thus v_5 also needs to be removed from $\mathcal{G}$ after deleting v_{10}. As a result, the η-core number of v_5 obtained by the peeling algorithm is 1 which is incorrect. In addition, we observe that the sum of $Pr[deg_{v_5} = 0]$, $Pr[deg_{v_5} = 1]$ and $Pr[deg_{v_5} = 2]$ is less than 1 which further implies that Eq. (5) leads to a significant error for updating the probabilities. Similarly, we can derive that the η-core numbers of the nodes $\{v_1, \ldots, v_4\}$ computed by the peeling algorithm are 2. However, the probabilities of each possible degree of v_5 in the subgraph induced by $\{v_1, \ldots, v_5\}$ are $Pr[deg_{v_5} = 0] = 0.102$, $Pr[deg_{v_5} = 1] = 0.595$, $Pr[deg_{v_5} = 2] = 0.302$ respectively based on the Eq. (4), which indicates that the correct η-core number of v_5 is 2. Furthermore, we can also compute the probabilities that each node has a degree of 3 in the subgraph induced by $\{v_1, \ldots, v_4\}$ which are $Pr[deg_{v_1} = 3] = 0.384$, $Pr[deg_{v_2} = 3] = 0.461$, $Pr[deg_{v_3} = 3] = 0.448$, $Pr[deg_{v_4} = 3] = 0.302$ respectively. As a result, the correct η-core numbers of $\{v_1, \ldots, v_4\}$ are equal to 3 in this example. These results indicate that the state-of-the-art peeling algorithms cannot obtain correct core decomposition on uncertain graphs.

Based on the above analysis, we know that all the previous peeling algorithms with the updating technique developed in [8], [9], [15] are doomed to obtain incorrect core decomposition on uncertain graphs. A natural question is that is there any other solutions that can correctly compute all (k, η)-cores of $\mathcal{G}$? We answer this question affirmatively. In fact, the basic peeling algorithm without using the updating technique can obtain correct core decomposition. When deleting a node v, the basic peeling algorithm recomputes the η-degrees of v's neighbors using the DP algorithm (Eq. (4)), instead of using the updating technique (Eq. (5)). Unlike Eq. (5), the DP equation (Eq. (4)) does not include a division operation, thus the error of the floating-point operation will not increase. Specifically, let $a = p_a \pm \epsilon$ and $b = p_b \pm \epsilon$ be the values with an ϵ error in a computer. Based on the analysis in [22], the errors for floating-point operations excluding division are as follows: (i) $a \times b = p_a \times p_b + O((p_a + p_b)\epsilon + \epsilon^2)$; (ii) $a + b = p_a + p_b + O(\epsilon)$; (iii) $a - b = p_a - p_b + O(\epsilon)$. Note that all floating-point values in uncertain graphs are no larger than 1, thus $O((p_a + p_b)\epsilon + \epsilon^2)$ is bounded by $O(\epsilon)$, while $O(\epsilon/(1 - p_e))$ is hard to be bounded. It can be seen that the error of each floating-point operation excluding division is bounded by $O(\epsilon)$. For a modern computer, the error $\epsilon \approx 10^{-16}$ is extremely small, thus the floating-point computations for a computer that do not involve floating-point divisions are often very accurate. As a result, the η-degrees computed by the DP algorithm are very accurate, and thus the basic peeling algorithm can obtain correct core decomposition. However, such a peeling algorithm is inefficient, as it needs to frequently recompute the η-degrees. In the following sections, we will develop two novel algorithms which can significantly reduce the from-scratch re-computations without sacrificing accuracy.

3 The Bottom-Up Algorithm

In this section, we propose a bottom-up algorithm to correctly compute the core decomposition on uncertain graphs based on an on-demand η-degree computation technique. Our algorithm follows a bottom-up computational manner which first computes the η-core numbers for the low-core nodes and then calculates the η-core numbers for the high-core nodes. To reduce the re-computations of η-degrees, our algorithm adopts an on-demand computation technique. Specifically, the algorithm maintains a lower bound of the η-core number for each node. After peeling a node v, the neighbors with lower bounds no less than v's η-degree are definitely not removed from $\mathcal{G}$ in this iteration, and thus we do not need to recompute the η-degrees for all these neighbors. Below, we first introduce two existing lower bounds of the η-core numbers, and then present our bottom-up algorithm and a lazy update optimization.

3.1 Two Existing Lower Bounds

The (η, k)-topcore Based Lower Bound. In [15], Li *et al.* introduced a different concept of k-core on uncertain graphs, called (η, k)-topcore, in which the η-topcore number was shown to be a lower bound of the η-core number for any node.

Let $N_v^k(\mathcal{G})$ be the set of k edges with the highest probabilities in $N_v(\mathcal{G})$. The η-topdegree of a node is defined as follows.

Definition 3 (η-topdegree). *Given an uncertain graph $\mathcal{G}$ and a probabilistic threshold $\eta \in [0, 1]$, the η-topdegree of $v \in V$, denoted by η-topdeg$_v(G)$, is a maximal k such that $\pi_v^k(\mathcal{G}) \geq \eta$, where $\pi_v^k(\mathcal{G}) = \prod_{e \in N_v^k(\mathcal{G})} p_e$, i.e., η-topdeg$_v(\mathcal{G}) = \max\{k | \pi_v^k(\mathcal{G}) \geq \eta\}$.*

By Definition 3, the definition of (η, k)-topcore is given below.

Definition 4 ((η, k)-topcore). *Given an uncertain graph $\mathcal{G}$, a positive integer k, and a probabilistic threshold $\eta \in [0, 1]$, an (η, k)-topcore is a maximal subgraph $\mathcal{G}' = (V', E', p)$ of $\mathcal{G}$ such that each node $v \in V'$ has a η-topdegree no less than k.*

By Definition 4, the η-topcore number of v, called η-topcore(v), is a largest integer k such that there exists a subgraph (η, k)-topcore containing v. Li *et al.* shown that η-topcore(v) is a lower bound of η-core(v) for each node v in $\mathcal{G}$ with any given $\eta \in [0, 1]$ [15]. Moreover, as shown in [15], all the (η, k)-topcores can be computed by a similar peeling algorithm using $O(m \log(d_{\max}))$ time.

The Beta-Function Based Lower Bound. Another existing lower bound of the η-core is based on a regularized beta function which was proposed in [8]. In particular, Bonchi *et al.* [8] proved the following inequality:

$$Pr[d_v(\mathcal{G}) \geq k] \geq I_{p_{\min}(v)}(k, d_v(\mathcal{G}) - k + 1), \tag{6}$$

where $p_{\min}(v)$ is the minimum probability in $E_v(\mathcal{G})$ and $I_z(a, b) = \sum_{i=a}^{a+b-1} a + b - 1 i z^i (1 - z)^{a+b-1-i}$ is the regularized beta function [23]. This regularized beta function is a cumulative distribution of an event with an existence probability z that occurs no less than a times of $a + b - 1$ repeated experiments. By Eq. (6), it is easy to derive that η-LB$(v) = \max\{k \in [0, \ldots, d_v(\mathcal{G})] | I_{p_{\min}(v)}(k, d_v(\mathcal{G}) - k + 1) \geq \eta\}$ is a lower bound of the η-degree of v. Based on η-LB, we can use a peeling

algorithm which iteratively removes the node with the smallest η-LB to obtain a lower bound of the η-core number for each node. Bonchi *et al.* [8] shown that such a beta-function based lower bound can be computed in $O(m\log(d_{\max}))$ time.

Remark. It is worth remarking that the above two lower bounds can be computed accurately. This is because both of these two lower bounds can be efficiently updated after deleting an edge without involving recursive floating-point number division operations [8], [15], and thus the algorithm can obtain accurate lower bounds.

Algorithm 1. BottomUpUCD$(\mathcal{G}, \eta)$

Input: an uncertain graph $\mathcal{G} = (V, E, p)$ and a parameter $\eta \in [0, 1]$
Output: η-core(v) for all $v \in V$
1 ub $\leftarrow$ core numbers of all nodes in the deterministic graph of $\mathcal{G}$
2 lb $\leftarrow$ the final lower bound of all nodes
3 $deg(v) \leftarrow 0$ for all $v \in V$
4 $k \leftarrow 0$
5 **while** $V \neq \emptyset$ **do**
6 $\quad C \leftarrow \{v \in V | \text{lb}(v) = k\}$
7 $\quad$ **foreach** $v \in C$ **do**
8 $\quad\quad deg(v) \leftarrow \text{DP}(\mathcal{G}, v, \text{ub}(v))$
9 $\quad D \leftarrow \{v \in V | deg(v) \leq k, \text{lb}(v) \leq k\}$
10 $\quad$ **foreach** $v \in D$ **do**
11 $\quad\quad \eta\text{-core}(v) \leftarrow k$
12 $\quad\quad$ Remove v from D and V
13 $\quad\quad$ **foreach** $u \in N_v(\mathcal{G})$ s.t. $deg(u) > k$ **do**
14 $\quad\quad\quad deg(u) \leftarrow \text{DP}(\mathcal{G}, u, \text{ub}(u))$
15 $\quad\quad\quad$ **if** $deg(u) \leq k$ $D \leftarrow D \cup \{u\}$
16 $\quad k \leftarrow k + 1$
17 **Procedure** DP$(\mathcal{G}, v, \text{ub}(v))$
18 Let $N_v(\mathcal{G}) = \{e_0, e_1, \ldots, e_{d_v(\mathcal{G})-1}\}$ be the set of neighbors of v
19 $F(i, 0) \leftarrow 1$ for all $i \in [0, d_v(\mathcal{G})]$
20 $F(i-1, i) \leftarrow 0$ for all $0 < i \leq \text{ub}(v)$
21 **for** $i = 1$ to ub(v) **do**
22 $\quad$ **for** $j = i$ to $d_v(\mathcal{G})$ **do**
23 $\quad\quad F(j, i) = p_{e_j} F(j-1, i-1) + (1 - p_{e_j}) F(j-1, i)$
24 $\quad$ **if** $F(d_v(\mathcal{G}), i) < \eta$ return $i - 1$
25 return ub(v)

3.2 The Basic Bottom-Up Algorithm

Equipped with the above two lower bounds, we present a bottom-up algorithm to compute the η-cores for all nodes. Initially, the algorithm calculates two lower bounds for each node, instead of computing the η-degrees. For each node, the final lower bound is obtained by taking the maximum value between these two lower bounds. The algorithm iteratively processes the nodes following a non-decreasing order of the final lower bounds. In the kth iteration, the algorithm only computes the η-degrees for the nodes with lower bound equaling k. Then, the algorithm removes these nodes and recomputes the η-degrees of the neighbor nodes if 1) their previous η-degrees are larger than k and 2) their lower bounds are less than or equal to k. The algorithm terminates if all nodes are deleted. The detailed implementation of this algorithm is shown in Algorithm 1.

Specifically, Algorithm 1 first computes the core numbers of all nodes in the deterministic graph of $\mathcal{G}$ (line 1), which will be applied to speed up the DP procedure (lines 17-25) to calculate the η-degree for each node as used in [15]. Then, Algorithm 1 computes the final lower bound for each node (line 2) and sets the initial η-degrees of all nodes to 0 (line 3). After that, the algorithm iteratively calculates the (k, η)-cores from $k = 0$ to $k = k_{\max}$ (lines 5-16). In the kth iteration, the algorithm computes the η-degrees of the nodes with lower bounds equaling k (lines 6-8). The algorithm makes use of a set D to maintain all the nodes whose η-degrees and lower bounds are no larger than k (line 9). Then, for each node $v \in D$, the algorithm removes v from $\mathcal{G}$ and sets the η-core number of v to k (lines 10-12). Subsequently, the algorithm invokes the DP procedure to recompute the η-degrees of v's neighbors if their current η-degrees are larger than k (lines 13-14). Note that the η-degrees of the v's neighbors with lower bounds larger than k must be equal to the initial value 0. Therefore, the algorithm does not need to recompute the η-degrees of those neighbors. For a neighbor node u, if its updated η-degree is smaller than or equal to k, the algorithm adds it to the set D (line 15). By this algorithm, all the nodes with η-core numbers equaling k can be identified in the kth iteration.

Since Algorithm 1 recomputes the η-degrees using the DP algorithm, the correctness of the algorithm can be guaranteed. The following example illustrates how the algorithm works.

Example 2. Consider the uncertain graph $\mathcal{G}$ in Fig. 1. Suppose that $\eta = 0.3$. Then, we can easily derive that the η-topcore numbers of the nodes $\{v_9, v_{10}\}$ are 1, the η-topcore numbers of the nodes $\{v_5, \ldots, v_8\}$ are 2, and the η-topcore numbers of the remaining nodes $\{v_1, \ldots, v_4\}$ are 3. Since the topcore based lower bound is tight in this example, we do not consider the beta-function based lower bound. In the peeling stage, Algorithm 1 calculates the η-degrees of nodes $\{v_9, v_{10}\}$ which are equal to 1, and thus obtain $D = \{v_9, v_{10}\}$. Then, Algorithm 1 sets the η-core numbers for $\{v_9, v_{10}\}$ to 1. When removing nodes in D, none of their neighbors need to recompute the η-degrees, because the lower bounds of all the neighbors are larger than 1. In the second iteration, the algorithm calculates the η-degrees for the nodes $\{v_5, \ldots, v_8\}$, and then set $D = \{v_5, \ldots, v_8\}$. Then, for each node in D, the algorithm removes it from $\mathcal{G}$ and sets the η-core number to 2. Similarly, there is no neighbor whose η-degree needs to be recomputed. In the third iteration, we can easily obtain that the nodes $\{v_1, \ldots, v_4\}$ will be removed from $\mathcal{G}$ whose η-core numbers are set to 3.

The above example indicates that Algorithm 1 with a good lower bound can significantly avoid from-scratch recomputations of η-degrees. In our experiments, we will show that Algorithm 1 is indeed much more efficient than the existing peeling algorithm. Below, we analyze the time and space complexity of Algorithm 1.

Theorem 1. *The worst-case time and space complexity of Algorithm 1 is $O((m+n)d_{\max}\delta)$ and $O(m+n)$ respectively, where δ is the maximum core number of the deterministic graph of $\mathcal{G}$.*

Proof. For the time complexity, the algorithm first takes $O((m+n)\log(d_{\max}))$ to compute the lower bounds. When peeling a node v, the algorithm needs to 1) compute the η-degree of v which consumes $O(d_v c_v)$ time complexity,

and 2) recompute the η-degrees of v's neighbors which takes at most $O(\sum_{u \in N_v(\mathcal{G})} c_u \times d_u)$. As a result, the total time overhead consumed in the peeling stage is $O(\sum_{v \in V} \sum_{u \in N_v(\mathcal{G})} (c_u \times d_u)) < O((m+n)d_{\max}\delta)$. For the space complexity, the algorithm only uses several linear-size arrays to maintain the η-degrees, the η-core numbers and the set D. In the DP procedure, we can use two arrays to maintain the values of F (line 23) which consumes at most $O(n)$ space. Therefore, the total space complexity of Algorithm 1 is $O(m+n)$. □

Algorithm 2. ImpBottomUpUCD$(\mathcal{G}, \eta)$

Input: an uncertain graph $\mathcal{G} = (V, E, p)$ and a parameter $\eta \in [0, 1]$
Output: η-core(v) for all $v \in V$

1 Lines 1-9 of Algorithm 1
2 **while** $D \neq \emptyset$ **do**
3 $\quad Q \leftarrow \bigcup_{v \in D} \{u \in N_v(\mathcal{G}) | deg(u) > k\}$ η-core$(v) \leftarrow k$ for all $v \in D$
4 $\quad$ Remove all nodes of D from $\mathcal{G}$ and set $D \leftarrow \emptyset$
5 $\quad$ **foreach** $u \in Q$ **do**
7 $\quad\quad deg(u) \leftarrow$ DP$(\mathcal{G}, u, \text{ub}(u))$
8 $\quad\quad$ **if** $deg(u) \leq k$ $D \leftarrow D \cup \{u\}$
9 Lines 16-25 of Algorithm 1

3.3 The Lazy Update Optimization

Although Algorithm 1 can largely avoid from-scratch re-computations of η-degrees with a tight lower bound, there is still much room for optimization. In particular, we observe that if two nodes u and v are removed in the same iteration, the η-degrees of their common neighbors may be recomputed twice by u and v respectively. Based on this observation, we develop a lazy update optimization to further reduce the from-scratch re-computations of η-degrees. Our optimized algorithm is shown in Algorithm 2.

Similar to Algorithm 1, Algorithm 2 first computes the upper and lower bounds of all nodes in the initial stage (line 1). Then, in the peeling stage, Algorithm 2 also obtains a node set D which contains all nodes to be removed in the current iteration. Unlike Algorithm 1, when removing a node $v \in D$ from $\mathcal{G}$, Algorithm 2 does not immediately recompute the η-degree of v's neighbors that have an η-degree larger than k. Instead, it pushes those nodes into a set Q in which each node only appears once (line 3). After that, Algorithm 2 recomputes the η-degrees for each node in Q (lines 6-8). Clearly, Algorithm 2 only recomputes the η-degree of a neighbor node at most once when removing the nodes in D. Thus, Algorithm 2 can significantly improve the efficiency of the basic bottom-up algorithm, which is also confirmed in our experiments.

4 The Top-Down Algorithm

In the previous section, we have proposed a bottom-up algorithm to reduce from-scratch re-computations of η-degrees. However, its performance is dependent on the quality of the lower bound. Since two existing lower bounds used in Algorithm 1 are often not very tight in real-world graphs, the algorithm may still be very costly when handling large real-world graphs. To further improve the efficiency, in this section, we develop a top-down algorithm which does not rely on the lower bounds.

4.1 The Basic Top-Down Algorithm

Unlike the bottom-up algorithm, the top-down algorithm computes the (k, η)-cores from $k = k_{\max}$ to $k = 0$. The challenges in developing an efficient top-down algorithm are twofold: 1) how to efficiently identify the $(k_{\max}, \eta)$-core from the uncertain graph $\mathcal{G}$; 2) how to reduce the unnecessary computations when computing (k', η)-core given that all the (k, η)-cores with $k > k'$ have already been obtained. Below, we describe our solution to tackle these challenges.

For each node u, let ub$(u) =$ core(u) be the core number of u in the deterministic graph G of $\mathcal{G}$ which is an upper bound of the η-core number of u. Let $\mathcal{C}^{\geq k} = (V^{\geq k}, E^{\geq k}, p)$ be a subgraph of $\mathcal{G}$ induced by the nodes in $V^{\geq k}$, where each node in $V^{\geq k}$ has a core number no less than k (core$(v) \geq k$ for each $v \in V^{\geq k}$). Clearly, the (k, η)-core of $\mathcal{G}$ must be contained in the subgraph $\mathcal{C}^{\geq k}$ for any parameter $\eta \in [0, 1]$. That is to say, we can exactly compute the (k, η)-core in the subgraph $\mathcal{C}^{\geq k}$, instead of in the original uncertain graph $\mathcal{G}$. Based on this, we can devise an efficient binary-search algorithm to identify the $(k_{\max}, \eta)$-core.

In particular, we assume that maxc is the maximum core number of the deterministic graph G of $\mathcal{G}$. The binary-search algorithm first computes the $(\frac{\text{maxc}}{2}, \eta)$-core in $\mathcal{C}^{\geq \frac{\text{maxc}}{2}}$. Note that for a given $k = \frac{\text{maxc}}{2}$, we can make use of a peeling algorithm to compute the (k, η)-core by iteratively removing the nodes with η-degrees smaller than k. If there is no such a $(\frac{\text{maxc}}{2}, \eta)$-core, the algorithm continues to detect the $(\frac{\text{maxc}}{4}, \eta)$-core in $\mathcal{C}^{\geq \frac{\text{maxc}}{4}}$. Otherwise, it tries to find the $(\frac{3\text{maxc}}{4}, \eta)$-core in $\mathcal{C}^{\geq \frac{3\text{maxc}}{4}}$. The binary-search algorithm terminates until the $(k_{\max}, \eta)$-core is identified. The number of iterations of the binary search procedure is bounded by $O(\log(\text{maxc}))$.

After obtaining the $(k_{\max}, \eta)$-core, the top-down algorithm then sequentially computes the other (k, η)-cores from $k = k_{\max} - 1$ to $k = 0$. Below, we develop several non-trivial optimization techniques to reduce the unnecessary computations by fully using the already computed information.

(i) Optimizing the binary search procedure. Recall that the binary search algorithm tries to compute (k', η)-core in $\mathcal{C}^{\geq k'}$ if the (k, η)-core exists in $\mathcal{C}^{\geq k}$, where k' is a mean value between k and the upper bound of the maximum η-core. We observe that it is not necessary to detect the (k', η)-core in $\mathcal{C}^{\geq k'}$, since the (k', η)-core is included in the (k, η)-core with $k' > k$. Therefore, it is sufficient to compute the (k', η)-core in the subgraph induced by the nodes in the (k, η)-core with η-degrees no less than k'. Clearly, such an induced subgraph is no larger than $\mathcal{C}^{\geq k'}$, thus the binary search procedure can be accelerated by using this optimization.

(ii) Optimizing the (k, η)-core computation for large k values. Let maxlb be the minimal k such that the (k, η)-core is successfully detected by the binary search algorithm (i.e., the first k that the binary search algorithm detects a (k, η)-core). Clearly, the η-degrees of the nodes in the (maxlb, η)-core have already been obtained after detecting the (maxlb, η)-core. When computing the (k, η)-core for each $k \in [\text{maxlb} + 1, k_{\max} - 1]$, we do not need to compute the η-degrees for the nodes in $\mathcal{C}^{\geq k}$. Instead, we can directly use

the η-degrees of the nodes that have been already computed in the (maxlb, η)-core to compute the (k, η)-core for all $k \in [\mathsf{maxlb}+1, k_{\max}-1]$. This is because the (k, η)-core for each $k \in [\mathsf{maxlb}+1, k_{\max}-1]$ is contained in the (maxlb, η)-core, thus it is sufficient to compute such (k, η)-cores in the (maxlb, η)-core.

(iii) Incremental update of η-degrees. When the algorithm calculates the (k, η)-core for a particular k, it needs to initially compute the η-degree for each node in $\mathcal{C}^{\geq k}$. Interestedly, we find that by the top-down computational manner, the η-degrees of the nodes in the initial stage of computing the (k, η)-core can be incrementally updated, instead of recomputing them from scratch. Specifically, suppose that we have already obtained the (k, η)-cores for $k \geq \bar{k}$. Then, when we compute the (k, η)-cores for all $k < \bar{k}$, the η-degree of a node v in $\mathcal{C}^{\geq k}$ can be updated based on the η-degree of v in $\mathcal{C}^{\geq \bar{k}}$ by using Eq. (4). This is because $\mathcal{C}^{\geq \bar{k}}$ is a subgraph of $\mathcal{C}^{\geq k}$, thus to compute the η-degree of v in $\mathcal{C}^{\geq k}$, we only need to add the edges from $\mathcal{C}^{\geq k} \backslash \mathcal{C}^{\geq \bar{k}}$ to update the η-degree of v in $\mathcal{C}^{\geq \bar{k}}$ which can be done by using Eq. (4). It is important to note that Eq. (4) does not involve any floating-point number division operations, thus the incremental computations of η-degrees can be accurate.

(iv) Optimizing the computation of η-degrees. As shown in the DP procedure (see lines 17-25 of Algorithm 1), the time cost for computing the η-degree of a node v relies on the upper bound of v. We note that the upper bound of the η-core number for each node can be incrementally refined in our top-down algorithm. Specifically, if a (k, η)-core H_k is computed, the upper bounds of the nodes in $V \backslash H_k$ must be smaller than k, thus we can use $k-1$ to refine the upper bounds for all nodes in $V \backslash H_k$. Based on the refined upper bounds, we can further reduce the costs of computing the η-degrees. In addition, if the refined upper bound of a node v in $\mathcal{C}^{\geq k}$ is less than k and the current η-degree of v is larger than or equal to k, then we do not need to update the η-degree for v. This is because if the η-degree of a node in a subgraph is no less than the upper bound of its η-core number, then the η-degree truncated by its upper bound is still equal to the upper bound, and thereby it is sufficient to correctly compute the (k, η)-cores based on such truncated η-degrees as shown in [15].

Implementation Details. The detailed implementation of our algorithm is shown in Algorithm 3. To simplify the description, Algorithm 3 only integrates the optimizations (i), (ii) and (iii). Specifically, Algorithm 3 first invokes a binary search procedure to detect the $(k_{\max}, \eta)$-core (lines 5-14), in which optimization (i) (line 10) is employed after obtaining the (maxlb, η)-core. Subsequently, Algorithm 3 starts to compute all (k, η)-cores from $k_{\max}-1$ to 0 with optimization (ii) (lines 16-24). In particular, if $k \geq \mathsf{maxlb}$ (lines 17-18), the algorithm directly computes the (k, η)-core by iteratively removing nodes that have an η-degree less than k in the (maxlb, η)-core. Otherwise, the algorithm has to compute or incrementally update the η-degrees of nodes in $V^{\geq k}$ whose η-core is less than k (lines 20-22), and then it invokes the peeling algorithm (lines 25-30) to compute the (k, η)-core. Note that the optimization (iii) is implemented in line 9 and line 22, which can significantly reduce the redundant computations of η-degrees.

Example 3. Consider the uncertain graph $\mathcal{G}$ in Fig. 1. Suppose that $\eta = 0.3$. Algorithm 3 first computes the upper bound of each node's η-core. Specifically, the upper bound of v_9 is 1, the upper bounds of $\{v_5, \ldots, v_8, v_{10}\}$ are 2 and the upper bounds of the other nodes are 3. Then, in the binary search stage, the algorithm computes the $(\mathsf{maxlb}, 0.3)$-core and $(k_{\max}, 0.3)$-core which are $\{v_1, \ldots, v_8\}$ and $\{v_1, \ldots, v_4\}$ respectively, where $\mathsf{maxlb} = 2$ and $k_{\max} = 3$. Next, the algorithm starts to compute other $(k, 0.3)$-cores with $k \leq 2$ following a top-down manner. For $k = 2$, the algorithm can directly ignore it since (2,0.3)-core has been computed previously. For $k = 1$, only two nodes $\{v_9, v_{10}\}$ need to be computed. The algorithm first computes the η-degree of v_9 and updates the η-degree of v_{10}, and then it invokes the peeling procedure to return the whole uncertain graph $\mathcal{G}$ as the (1,0.3)-core.

Algorithm 3. $\mathsf{TopDownUCD}(\mathcal{G}, \eta)$

Input: an uncertain graph $\mathcal{G} = (V, E, p)$ and a parameter $\eta \in [0, 1]$
Output: η-$\mathsf{core}(v)$ for all $v \in V$
1 $\mathsf{ub} \leftarrow$ core numbers of all nodes in the deterministic graph of $\mathcal{G}$
2 $V^{\geq k} \leftarrow \{v \in V | \mathsf{ub}(v) \geq k\}$; η-$\mathsf{core}(v) \leftarrow 0$ for all $v \in V$
3 Denote by $\mathcal{C}^{\geq k}$ the subgraph of $\mathcal{G}$ induced by $V^{\geq k}$
4 $\mathsf{minc} \leftarrow 0$; $\mathsf{maxlb} \leftarrow 0$; $\mathsf{maxc} \leftarrow \max_{v \in V}\{\mathsf{ub}(v)\}$
5 **while** $\mathsf{minc} \leq \mathsf{maxc}$ **do**
6 $\quad k \leftarrow (\mathsf{minc} + \mathsf{maxc})/2$
8 $\quad$ **for** $v \in V^{\geq k}$ s.t $\mathsf{maxlb} = 0$ **do**
9 $\quad\quad$ **if** $\mathsf{ub}(v) \leq \mathsf{maxc}$ **then** $deg(v) \leftarrow \mathsf{DP}(\mathcal{C}^{\geq k}, v, \mathsf{ub}(v))$;
10 $\quad\quad$ **else** Incrementally update $deg(v)$;
11 $\quad$ **if** $\mathsf{maxlb} \neq 0$ $\mathcal{C}^{\geq k} \leftarrow (k', \eta)$-core; **then**
12 $\quad$ **if** $\mathsf{Core}(\mathcal{C}^{\geq k}, deg, k, \eta)$ **then** $\mathsf{maxlb} = 0$ $\mathsf{maxlb} \leftarrow k$;
13 $\quad\quad$ $\mathsf{minc} \leftarrow k+1$; $k_{\max} \leftarrow k$; $k' \leftarrow k$
14 $\quad$ **else** $\mathsf{maxc} \leftarrow k-1$; $k_{\max} \leftarrow \mathsf{maxc}$;
15 $k \leftarrow k_{\max} - 1$
16 **while** $k > 0$ **do**
17 $\quad$ **if** $k \geq \mathsf{maxlb}$ **then**
18 $\quad\quad$ $\mathcal{H} \leftarrow (\mathsf{maxlb}, \eta)$-core; $\mathsf{Core}(\mathcal{H}, deg(\mathcal{H}), k, \eta)$
19 $\quad$ **elase**
20 $\quad\quad$ **for** $v \in V^{\geq k}$ s.t η-$\mathsf{core}(v) < k$ **do**
21 $\quad\quad\quad$ **if** $\mathsf{ub}(v) = k$ **then** $deg(v) \leftarrow \mathsf{DP}(\mathcal{C}^{\geq k}, v, \mathsf{ub}(v))$;
22 $\quad\quad\quad$ **else** Incrementally update $deg(v)$
23 $\quad\quad$ $\mathsf{Core}(\mathcal{C}^{\geq k}, deg, k, \eta)$
24 $\quad$ $k \leftarrow k - 1$
25 **Procedure** $\mathsf{Core}(\mathcal{G}, deg, k, \eta)$
26 $curdeg \leftarrow deg$
27 Iteratively remove the node v with $curdeg(v) < k$ from $\mathcal{G}$, and recompute the η-degree of its neighbors whose η-core is less than k **foreach** remaining v in $\mathcal{G}$ s.t. η-$\mathsf{core}(v) < k$ **do**η-$\mathsf{core}(v) \leftarrow k$;
29 **if** $\mathcal{G} \neq \emptyset$ **then return true**
30 **else return false**

Complexity Analysis. We analyze the time and space complexity of Algorithm 3 in the following theorem.

Theorem 2. *The worst-case time and space complexity of Algorithm 3 is $O((m+n)d_{\max}\delta)$ and $O(m+n)$ respectively, where δ is the maximum core number of the deterministic graph of $\mathcal{G}$.*

Proof. We first analyze the time complexity. First, Algorithm 3 takes $O((m+n)d_{\max})$ time to compute a (k, η)-core for a particular k based on a peeling algorithm (lines 25-

30). Second, Algorithm 3 invokes a binary search procedure to compute the $(k_{\max}, \eta)$-core which consumes at most $O((m+n)d_{\max}\log(\delta))$ time (lines 5-14). Third, to compute the other (k, η)-cores one by one which takes at most $O((m+n)d_{\max}\delta)$. As a result, the total time complexity of Algorithm 3 is $O((m+n)d_{\max}\delta)$. For the space complexity, the algorithm only uses several additional linear-size arrays to maintain the upper bounds and η-degrees. Therefore, the space overhead of our algorithm is $O(m+n)$. □

Note that the worst-case time complexity of Algorithm 3 is the same as that of Algorithm 2. However, as shown in our experiments, the practical performance of Algorithm 3 is often much better than that of Algorithm 2 on large uncertain graphs due to the high pruning performance of the proposed optimization techniques.

4.2 The Landmark-Core Based Algorithm

We notice that in Algorithm 3, there exist some nodes that have an η-degree no less than k in $\mathcal{C}^{\geq k}$ but their η-core numbers are much less than k. As a consequence, the η-degrees of such nodes may be frequently recomputed in the peeling procedure for computing the (k, η)-cores. To reduce such unnecessary re-computations, we develop two landmark-core based top-down algorithms. The key idea of our algorithms is as follows. First, similar to Algorithm 3, the algorithms invoke a binary search procedure to compute the (maxlb, η)-core and the $(k_{\max}, \eta)$-core. Second, to compute the (k, η)-cores for $k \in [0, \mathsf{maxlb}-1]$, the algorithms first partition the interval $[0, \mathsf{maxlb}-1]$ by selecting some landmark k values in $[0, \mathsf{maxlb}-1]$, and then compute the (k, η)-cores based on the selected landmarks using the peeling procedure. After that, the algorithms compute the (k, η)-cores for all k values in each subinterval on the subgraph induced by the corresponding landmark (k, η)-core (similar to optimization (ii)).

More specifically, our first landmark selection strategy is inspired by a binary search strategy. First, we partition the interval $[0, \mathsf{maxlb}-1]$ into $\log(\mathsf{maxlb})$ subintervals. Let $\mathcal{C}^{\geq \frac{\mathsf{maxlb}}{2^i}}$ be a subgraph induced by the node set $V^{\geq \frac{\mathsf{maxlb}}{2^i}}$, where i is a positive integer in the interval $[1, \lfloor \log(\mathsf{maxlb}) \rfloor + 1]$ (we set $2^i = \mathsf{maxlb}$ if $i > \log(\mathsf{maxlb})$). Then, we compute the $(\frac{\mathsf{maxlb}}{2^i}, \eta)$-core in $\mathcal{C}^{\geq \frac{\mathsf{maxlb}}{2^i}}$ according to the non-decreasing order of i. After that, we compute the (k, η)-core for each $k \in [\frac{\mathsf{maxlb}}{2^i} + 1, \frac{\mathsf{maxlb}}{2^{i-1}}]$ in the subgraph induced by the nodes that are 1) contained in the $(\frac{\mathsf{maxlb}}{2^i}, \eta)$-core, and 2) with η-degrees no less than k. An alternative landmark selection strategy is based on an isometric partition method. Specifically, the interval $[0, \mathsf{maxlb}-1]$ is first partitioned into equal-size subintervals, and then a similar landmark core based method is applied to compute the (k, η)-cores. In our experiments, we will show that such a simple isometric partition strategy is very efficient in practice.

5 The Parallel Algorithms

In this section, we develop parallel variants of all the proposed algorithms. Below, we first propose a parallel algorithm for computing the lower bounds used in the bottom-up algorithms. Then, we will show how to parallelize the (k, η)-cores computation algorithms.

Algorithm 4. ParallelLowerBound$(\mathcal{G}, \eta)$

```
    Input: an uncertain graph G = (V, E, p) and a parameter
           η ∈ [0, 1]
    Output lb(v) for all v ∈ V
 1  Compute η-topdeg_v(G) for each v ∈ V in parallel;
 2  B[i] ← ∅ parallel for each i ∈ [1, d_max]; k ← 0
 3  cnt(v) ← 1 for each v ∈ V in parallel;
 4  while G is not empty do
 5      C ← {v ∈ V | η-topdeg_v(G) = k}
 6      while C ≠ ∅ do
 7          parallel for v ∈ C do
 8              lb(v) ← k
 9              Remove v from C and G
10              foreach u ∈ N_v(G) s.t. η-topdeg_u(G) > k do
11                  i ← cnt(u)++ (atomic operation)
12                  B[i] ← B[i] ∪ {u} (atomic operation)
13          for i = 1 to i = len(B) do
14              parallel for u ∈ B[i] do
15                  Update η-topdeg_u(G)
16                  if η-topdeg_u(G) ≤ k s.t. u ∉ C then
17                      C ← C ∪ {u} (atomic operation)
18                  cnt(u) ← 1
19              B[i] ← ∅
20      k ← k + 1
```

5.1 Parallel Lower Bound Computation

Recall that the (η, k)-topcore based lower bound and the beta-function based lower bound are derived by an iterative peeling procedure. Here we develop an efficient parallel algorithm to compute these lower bounds. Below, we focus mainly on computing the (η, k)-topcore based lower bound, and the same technique can be easily adapted to calculate the beta-function based lower bound.

The peeling algorithm computes the (η, k)-topcore by iteratively removing the nodes with the smallest η-topdegree. When deleting a node v, the algorithm also needs to update the η-topdegrees of v's neighbors. Clearly, the procedure of updating neighbors' η-topdegrees can be done in parallel. However, such a straightforward parallel strategy is not very efficient. This is because many nodes often have a few neighbors that are needed to update their η-topdegrees, thus the degree of parallelism of this algorithm can be very low. Below, we develop a new edge-parallel strategy to improve the parallel performance of the peeling algorithm.

The key idea of our algorithm is described as follows. When removing a node v, we do not immediately update the η-topdegrees of v's neighbors. Instead, for each deleted edge (v, u), we record the neighbor node u using a set B. Clearly, for different neighbors recorded in B, we can update the η-topdegrees in parallel. However, a node u may be a neighbor of several nodes that are deleted in the kth iteration, thus for the same neighbor node u, we cannot update u's η-topdegree in parallel. To overcome this issue, we can make use of an array of sets $B[i]$ (for $i \leq d_{\max}$) to record the neighbors, satisfying that the neighbor nodes in $B[i]$ for a particular i must be different. Then, we can process each $B[i]$ in parallel. The detailed description of our algorithm is shown in Algorithm 4.

Algorithm 4 first computes the topcore for all nodes in $\mathcal{G}$ in parallel (line 1). Then, the algorithm initializes an array of sets $B[i]$ (for $i \leq d_{\max}$) and a counting array cnt which is used to

record the number of times that a node acts as a neighbor of the deleted nodes (lines 2-3). After that, the algorithm iteratively removes nodes with η-topdegrees equaling k (from $k = 0$ to $k = k_{\max}$) to compute the (η, k)-topcores (lines 4-20). In the kth iteration, the algorithm first identifies all nodes C with η-topdegrees equaling k (line 5). Then, the algorithm processes the nodes in C in parallel (lines 7-12). For each $v \in C$, the algorithm records the neighbors of v in a set $B[i]$ (lines 10-12). Subsequently, the algorithm updates the η-topdegrees for all nodes in $B[i]$ in parallel (lines 13-19). Note that the time complexity of constructing the array of sets $B[i]$ (for all i) is $O(r)$, where r is the number of removed edges in the kth iteration (for any k). Thus, the proposed edge-parallel strategy does not increase the total time complexity of the algorithm. Moreover, the degree of parallelism of the proposed edge-parallel strategy is higher than that of the straightforward parallel strategy. For example, suppose that there are $C = \{v_1, \ldots, v_t\}$ nodes to be deleted in the k-iteration. Let d_i be the degree of $v_i \in C$. Then, the maximum degree of parallelism of the straightforward strategy is $\hat{d} = \max\{d_1, \ldots, d_t\}$. However, for the edge-parallel strategy, the maximum degree of parallelism can be up to $\tilde{d} = |\bigcup_{v \in C}\{N_v(\mathcal{G})\}| = B[1]$ which is no less than $\hat{d}$.

5.2 Parallel Uncertain Core Decomposition

We find that Algorithms 2 and 3 can be easily parallelized, since the re-computations of nodes' η-degrees are completely independent in these algorithms. The critical thing that should be noted for devising the parallel variants of these algorithms is as follows. In the kth iteration, all the neighbors of the deleted nodes that have an η-degree larger than k must be sequentially pushed into a set Q, and then we can recompute their η-degrees in parallel. Due to the space limit, we only introduce the details of the parallel version of Algorithm 2, since the parallel peeling procedure of Algorithm 3 can be implemented in a similar manner. The detailed description of our parallel algorithm is shown in Algorithm 5.

Algorithm 5. PUCD$(\mathcal{G}, \eta)$

Input: an uncertain graph $\mathcal{G} = (V, E, p)$ and a parameter $\eta \in [0, 1]$
Output: η-core(v) for all $v \in V$
1 ub $\leftarrow$ parallel core decomposition in the deterministic graph of $\mathcal{G}$
2 lb $\leftarrow$ ParallelLowerBound$(\mathcal{G}, \eta)$;
3 $deg(v) \leftarrow 0$ for all $v \in V$ in parallel;
4 $k \leftarrow 0$
5 **while** $\mathcal{G}$ is not empty **do**
6 $\quad C \leftarrow \{v \in V | \text{lb}(v) = k\}$
7 $\quad deg(v) \leftarrow$ DP$(\mathcal{G}, v, \text{ub}(v))$ for all $v \in C$ in parallel;
8 $\quad D \leftarrow \{v \in V | deg(v) \leq k, \text{lb}(v) \leq k\}; Q \leftarrow \emptyset$
9 $\quad$ **while** $D \neq \emptyset$ **do**
10 $\quad\quad$ **parallel for** $v \in D$ **do**
11 $\quad\quad\quad \eta$-core$(v) \leftarrow k$
12 $\quad\quad\quad$ Remove v from D and $\mathcal{G}$
13 $\quad\quad\quad$ **foreach** $u \in N_v(\mathcal{G})$ s.t. $deg(u) > k$ **do**
14 $\quad\quad\quad\quad$ **if** $u \notin Q$ $Q \leftarrow Q \cup \{u\}$ (atomic operation);
15 $\quad\quad$ **parallel for** $u \in Q$ **do**
16 $\quad\quad\quad deg(u) \leftarrow$ DP$(\mathcal{G}, u, \text{ub}(u))$
17 $\quad\quad\quad$ **if** $deg(u) \leq k$ $D \leftarrow D \cup \{u\}$ (atomic operation);
18 $\quad k \leftarrow k + 1$

In the initial stage, Algorithm 5 first invokes the parallel lower and upper bound algorithms to obtain the bounds of the η-core number for each node (lines 1-2). Note that we can use an existing parallel core decomposition algorithm for deterministic graphs [24] to compute the upper bound. Then, in the peeling stage (lines 5-18), the algorithm first computes the η-degrees of nodes whose lower bounds are equal to k in parallel (lines 6-7), and adds all nodes to be deleted into a set D (line 8). Subsequently, the algorithm processes the nodes in D in parallel (lines 10-14). Note that for each neighbor of a node in D, if it needs to update its η-degree, the algorithm adds it into a set Q with an *atomic add* operation, because Q is a shared variable by different threads. After that, the algorithm recomputes the η-degrees of the nodes in Q in parallel (lines 15-17), since the re-computation of the η-degree of each node in Q is independent. Similarly, the algorithm also invokes an *atomic add* operation to maintain the set D (line 17). The algorithm terminates if all nodes are removed from $\mathcal{G}$. Clearly, the total CPU time of Algorithm 5 is equal to that of Algorithm 2.

6 Experiments

In this section, we conduct extensive experiments to evaluate the efficiency, scalability, and accuracy of different algorithms. Below, we first introduce the experimental setup and then report our results.

6.1 Experimental Setup

We implement five sequential algorithms Basic, BU, BU+, TD, and TD+ to compute the accurate (k, η)-cores. Basic is the state-of-the-art exact peeling (k, η)-core algorithm proposed by Bonchi *et al.* [8], which invokes the DP algorithm to recompute η-degrees in each iteration. BU denotes our basic bottom-up algorithm described in Algorithm 1, and BU+ is the bottom-up algorithm with the lazy update optimization, i.e., Algorithm 2. TD denotes the basic top-down algorithm presented in Algorithm 3, and TD+ is the landmark-core based top-down algorithm proposed in Section 4.2. Note that for TD+, we make use of the isometric-partition based landmark selection strategy, as it is more efficient than the binary-search based partition strategy. We will evaluate the effect of different landmark selection strategies in Exp-2. In addition, we also implement two parallel versions of BU+ and TD+ using OpenMP which are denoted by PBU+ and PTD+ respectively. All algorithms are implemented in C++. All experiments are tested on a PC with two 2.1 GHz Xeon CPUs (16 cores in total) and 128GB memory running CentOS 7.6.

Datasets. We make use of five large real-world graphs to evaluate the efficiency and scalability of our algorithms. Table 1 shows the detailed statistics of each dataset, where the last two columns represent the maximum degree and maximum core number of the graph, respectively. DBLP is a scientific collaboration network which is extracted from the DBLP computer science bibliography (http://dblp.uni-trier.de/xml/). Both Wiki and Stackof are the communication networks, while both SocLJ and Hollywood are social networks. DBLP, Wiki, and Stackof are weighted graphs where each edge (u, v) denotes the interaction frequency between u and v, while SocLJ and Hollywood are unweighted graphs. We download the Wiki and SocLJ datasets

TABLE 1
Datasets

Dataset	$\|V\|$	$\|E\|$	$d_{\max}$	δ
DBLP	2,536,460	22,056,096	3,087	263
Wiki	2,987,535	24,981,163	146,311	210
Stackof	2,601,977	63,497,050	44,065	198
SocLJ	4,847,571	68,475,391	22,887	372
Hollywood	2,180,759	228,985,632	13,107	1,296

from (http://konect.cc/networks/). Stackof and Hollywood are downloaded from (http://snap.stanford.edu/data/) and (http://law.di.unimi.it/datasets.php), respectively.

For each dataset, we generate an uncertain graph using two different methods. For the first method, we independently generate a probability for each edge of a graph based on a [0,1] uniform distribution. For the second method, we adopt an exponential distribution to generate the probability for each edge which has been widely used in the uncertain graph mining literature [9], [18], [21]. More specifically, for the weighted graphs, we make use of an exponential cumulative distribution with expectation 2 to the weight of an edge to generate a probability. For the un-weighted graphs, we first randomly assign a weight to each edge, and then use the same exponential distribution method to generate the probabilities. In our experiments, the dataset named with a suffix "uni" ("exp") denotes the uncertain graph generated by the first (second) method.

Parameters. There is only one parameter in our algorithms: $\eta \in [0, 1]$. In our experiments, the default value of η is 0.4 unless otherwise specified. Note that if $\eta = 0$, the (k, η)-core of $\mathcal{G}$ is the same as the k-core of the corresponding deterministic graph.

6.2 Efficiency Testings

Exp-1: Efficiency of Sequential Algorithms. Fig. 2 shows the runtime of Basic, BU, BU+, TD, and TD+ on each dataset with a varying η. The symbol "INF" means that the algorithm cannot terminate within 10 days. From Fig. 2, we can see that all the proposed algorithms are much faster than the state-of-the-art exact algorithm Basic. As expected, the Basic algorithm is very costly for computing the (k, η)-core decomposition on all datasets due to the extensive from-scratch recomputations of η-degrees. It even takes several days to complete the computation on a medium-sized graph Wiki. Our best algorithm TD+, however, takes only a few seconds to compute the (k, η)-core decomposition on the same graph. Generally, the proposed top-down algorithms are more efficient than the bottom-up algorithms. The best top-down algorithm TD+ significantly outperforms all the other competitors on all datasets. In particular, TD+ can achieve up to one order of magnitude faster than BU+, two orders of magnitude faster than BU, and three orders of magnitude faster than Basic respectively on the largest dataset Hollywood. For example, in Fig. 2j, when $\eta = 0.4$, TD+ only takes 2,444 seconds while TD, BU+, and BU consume 12,847 seconds, 91,073 seconds, and 293,668 seconds respectively to compute all (k, η)-cores on Hollywood-uni. Generally, BU+ is $3\times$ to $9\times$ faster than BU, and TD+ is $1\times$ to $5\times$ faster than TD on most datasets due to the effect of the proposed optimization technique. Moreover, we can see that all the proposed algorithms are robust w.r.t. the parameter η on most datasets. These results demonstrate the high efficiency of the proposed algorithms.

Additionally, we observe that the time overheads of BU+ and BU on the uncertain graphs with probabilities generated by the exponential cumulative distribution are generally less than those of the same algorithms on the uncertain graphs with uniform edge probabilities. This is because the edge probabilities generated by the exponential cumulative distribution are often larger than the uniform edge probabilities. Thus, the lower bound (used in the bottom-up algorithms) on the uncertain graphs with probabilities generated by the exponential cumulative distribution is often much tighter, compared to the uncertain graphs with uniform edge probabilities. As expected, the runtime of TD+ and TD is not very sensitive with respect to (w.r.t.) the two different types of uncertain graphs. This is because the top-down algorithms do not rely on the lower bound.

Exp-2: The Effect of Different Landmark Techniques. In this experiment, we evaluate the effect of different landmark selection strategies used in the top-down algorithm. Let TD-B (TD-I) be the binary-search partition (isometric-partition) based landmark selection strategy. Fig. 3 shows the runtime of different algorithms on SocLJ and Hollywood. The results on the other datasets are consistent. From the Fig. 3, we observe that TD-I significantly outperforms TD-B on all datasets with all parameter settings. For example, when $\eta =$

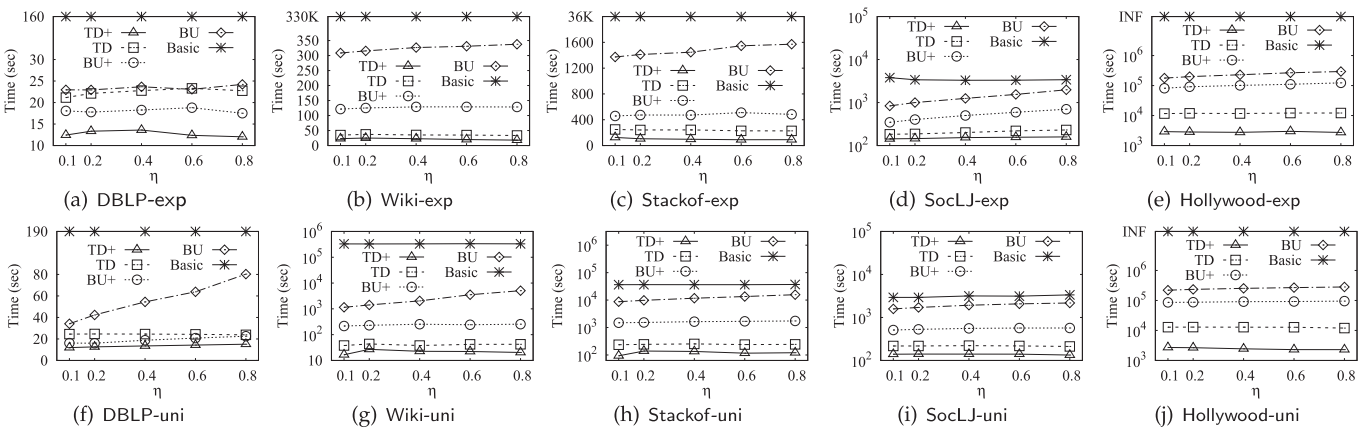


Fig. 2. Runtime of different sequential algorithms.

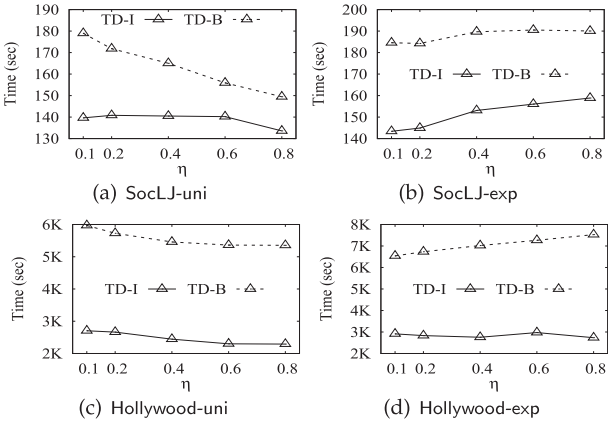


Fig. 3. Runtime of the top-down algorithms with different landmark selection strategies.

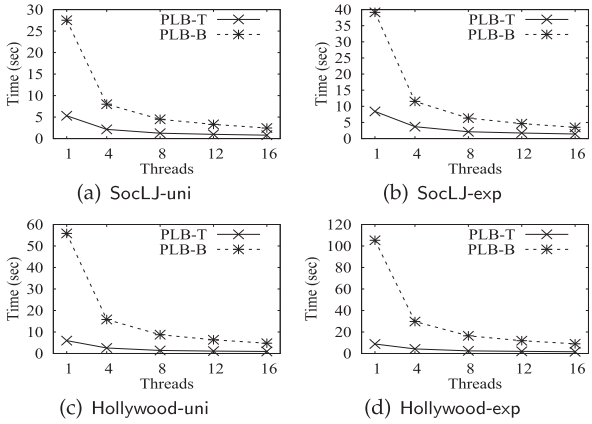


Fig. 5. Runtime of parallel lower bound algorithms ($\eta = 0.4$).

0.4, TD-I takes 2,755 seconds to compute all (k, η)-cores on Hollywood-exp, while TD-B consumes 7,021 seconds to complete the computation. These results indicate that the isometric-partition strategy is more effective than the binary-search based partition strategy to reduce the from-scratch re-computations of η-degrees in our top-down algorithm.

Exp-3: Efficiency of Parallel Algorithms. In this experiment, we evaluate the performance of the two parallel algorithms: PBU+ and PTD+. Fig. 4 shows the runtime of each algorithm on five uncertain graphs with probabilities generated by exponential cumulative distribution. The results on the uncertain graphs with uniform edge probabilities are consistent. From Fig. 4, we can see that the time consumptions of all algorithms rapidly decrease as the number of threads increases. Moreover, on most datasets, the speedup ratios of both PTD+ and PBU+ can reach up to 10 using 16 threads. For example, on Hollywood-exp, PTD+ and PBU+ take 2,755 and 100,979 seconds to compute all (k, η)-cores with one thread respectively. However, when using 16 threads, they only consume 258 and 8,016 seconds to complete the (k, η)-cores computation respectively. These results indicate that both PTD+ and PBU+ exhibit a very good parallel efficiency.

Exp-4: Efficiency of the Parallel Lower Bound Algorithms. Here we evaluate the parallel performance of two lower-bound computation algorithms with the edge-parallel strategy proposed in Section 5.1. Let PLB-T and PLB-B be the algorithms for computing topcore and the beta function based lower bounds, respectively. Fig. 5 shows the runtime of each lower-bound computation algorithm using different number of threads on SocLJ and Hollywood. Similar results can also be obtained on the other datasets. As can be seen in Fig. 5, the runtime of each parallel algorithm quickly decreases as the number of threads increases. The speedup ratio of PLB-B is around 10 when using 16 threads, while the speedup ratio of PLB-T is around 6.7. Moreover, we can see that PLB-T is significantly faster than PLB-B, because the update of the beta function is more expensive than the update of the η-topdegree. These results confirm the high efficiency of the parallel lower bound computation algorithms with the edge-parallel strategy.

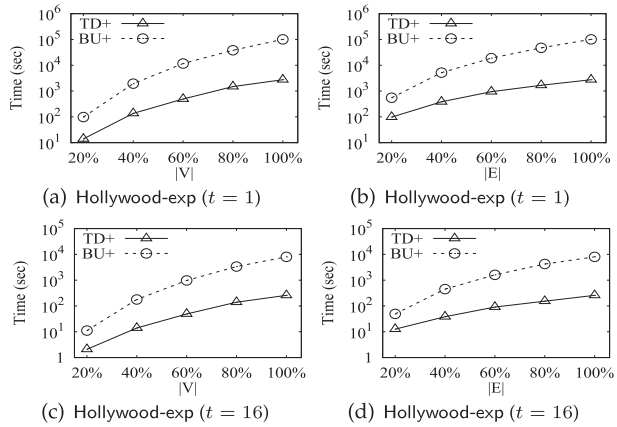


Fig. 6. Scalability of different algorithms.

Exp-5: Scalability Testing. Here we investigate the scalability of the best bottom-up and top-down algorithms using the largest dataset Hollywood. Specifically, we generate four subgraphs by randomly sampling 20–80% of nodes (edges) from Hollywood, and evaluate the time overheads of our algorithms on the four subgraphs. Figs. 6a and 6b show the scalability results of BU+ and TD+ on Hollywood-exp using a single thread. As can be seen, the runtime of TD+ increases smoothly as $|V|$ or $|E|$ increases, while the runtime of BU+ increases sharply. In addition, we also evaluate the scalability of the parallel versions of BU+ and TD+. Figs. 6c and 6d show the scalability results of parallel BU+ and TD+ using 16 threads. We can

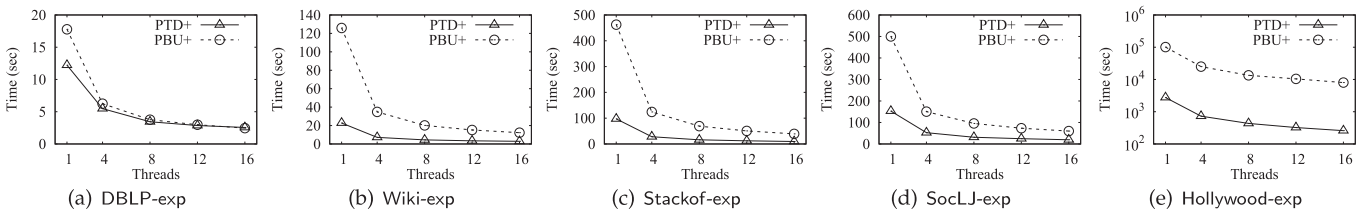


Fig. 4. Runtime of different parallel algorithms ($\eta = 0.4$).

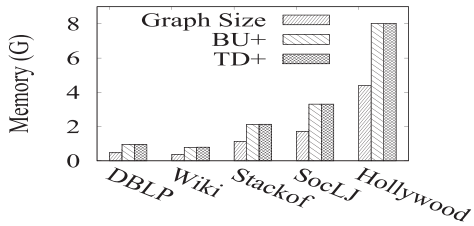


Fig. 7. Memory overheads of different algorithms.

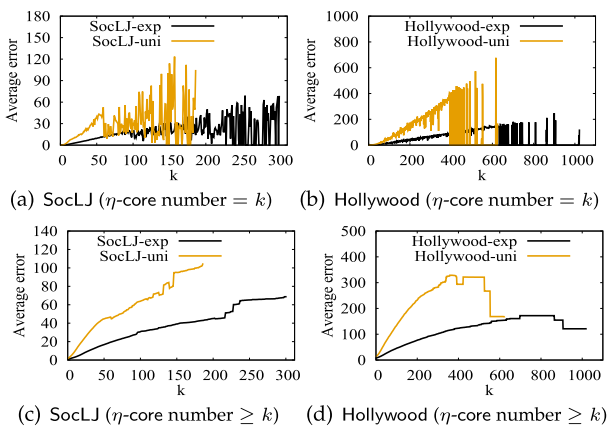
observe that the scalability of parallel algorithms is similar to the corresponding sequential algorithms with varying $|V|$ and $|E|$. These results indicate that both the sequential and parallel versions of TD+ exhibit a very good scalability performance in computing all (k, η)-cores.

Exp-6: Memory Overhead. In this experiment, we evaluate the space consumptions of BU+ and TD+. Fig. 7 shows the memory overheads of BU+ and TD+ on all datasets. As can be seen, the space usages of BU+ and TD+ is around twice the graph size on all datasets, because both of these algorithms have a linear space complexity. These results indicate that both BU+ and TD+ are highly space-efficient.

6.3 Accuracy Testings

Exp-7: Accuracy of the Existing Algorithms. Recall that existing algorithms [8], [9], [15] that use Eq. (5) to update the η-degrees will obtain incorrect core decomposition due to the inaccuracy of the recursive floating-point number division operations. In this experiment, we study the accuracy of such an inexact algorithm proposed in [15], denoted by InExactUCD, which is shown to be faster than the inexact algorithms proposed in [8], [9]. Note that we use 64 bits (the double data type in C++) to represent a floating-point number in the implementation of the InExactUCD algorithm.

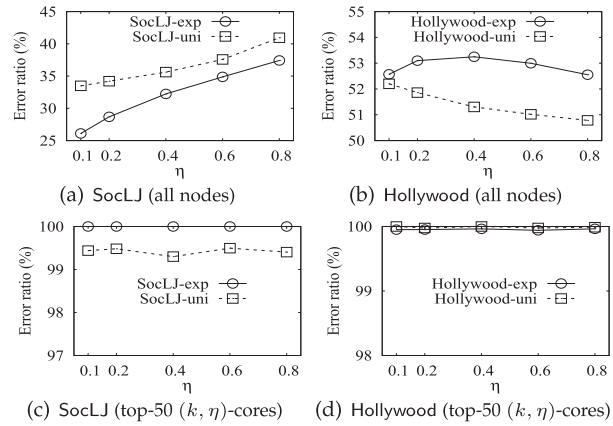
Let η-$\overline{\text{core}}(v)$ be the η-core number of $v \in V$ generated by InExactUCD, and the η-core(v) is the correct η-core number of v computed by our algorithms. We first evaluate the average errors of the η-cores computed by InExactUCD using the following two metrics: (i) $(\sum_{v \in C^k}(|\eta\text{-core}(v) - \eta\text{-}\overline{\text{core}}(v)|))/|C^k|$, and (ii) $(\sum_{v \in C^{\geq k}}(\eta\text{-core}(v) - \eta\text{-}\overline{\text{core}}(v)|))/|C^{\geq k}|$, where C^k is a set of nodes with η-core numbers equaling k, and $C^{\geq k}$ is a set of node with η-core number no less than k. Fig. 8 shows the result on SocLJ and Hollywood, and the results on the other datasets



(a) SocLJ (η-core number $= k$) (b) Hollywood (η-core number $= k$)

(c) SocLJ (η-core number $\geq k$) (d) Hollywood (η-core number $\geq k$)

Fig. 8. Average errors of the η-core numbers computed by the InExactUCD algorithm proposed in [15].



(a) SocLJ (all nodes) (b) Hollywood (all nodes)

(c) SocLJ (top-50 (k, η)-cores) (d) Hollywood (top-50 (k, η)-cores)

Fig. 9. The ratio of nodes with incorrect η-core numbers computed by InExactUCD.

are consistent. In Figs. 8a and 8b, we can observe that for most k values, the average errors of InExactUCD based on the first metric are very high on both SocLJ and Hollywood. For a particular k, the maximum average error of InExactUCD can be higher than 677 and 248 on Hollywood-uni and Hollywood-exp respectively. From Figs. 8c and 8d, we can see that the average errors of InExactUCD based on the second metric are also very high. In general, the average errors of InExactUCD increase with k increases, indicating that the high-order (k, η)-cores computed by InExactUCD are often much less precise than the lower-order (k, η)-cores obtained by InExactUCD. These results indicate that the existing algorithms that apply Eq. (5) to update the η-degrees are extremely imprecise for computing the (k, η)-cores on uncertain graphs.

To further evaluate the accuracy of the InExactUCD algorithm, we also investigate the ratio of nodes that are with incorrect η-core numbers computed by InExactUCD. Figs. 9a and 9b show the error ratios among all nodes, while Figs. 9c and 9d show the error ratios by only considering the nodes in the top-50 (k, η)-cores (i.e., the nodes with the top-50 highest η-core numbers). From Figs. 9a and 9b, we can clearly see that there are at least 25% and 50% nodes that have incorrect η-core numbers on SocLJ and Hollywood respectively. Moreover, the case is even worse when only considering the nodes in the top-50 (k, η)-cores. As can be seen from Figs. 9c and 9d, the error ratios are near to 100% on both SocLJ and Hollywood. These results further confirm that the InExactUCD algorithm is incorrect for (k, η)-cores computation.

Exp-8: The Effect of Floating-Point Number Precision. As shown in Exp-7, the InExactUCD algorithm is extremely imprecise when using 64-bits to represent a floating-point number. A natural question is that can we improve the accuracy of InExactUCD by using more bits to represent a floating-point number. To answer this question, we implement InExactUCD using a high-precision floating-point number operation library, namely GMP library (https://gmplib.org), to achieve high-precision representations of floating-point numbers. Let p ($p > 64$) denotes the number of bits used to represent a floating-point number. For convenience, we set $p = sys$ to denote that a floating-point number is represented by using 64 bits (i.e., the double data type in C++). Fig. 10 shows the results of error ratios of nodes in the top-50 η-cores with different floating-point number precisions. The results of error ratios among all nodes are consistent.

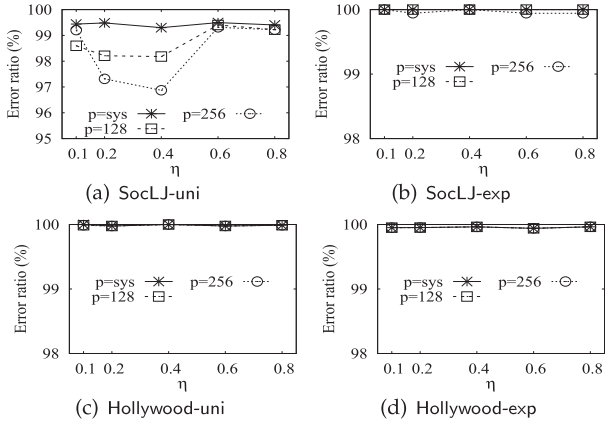


Fig. 10. The ratios of nodes with incorrect η-core numbers in the top-50 (k, η)-cores with different floating-point number precisions.

From Fig. 10, we can see that the error ratios do not significantly decrease as p increases. Moreover, in most cases, the error ratios keep unchanged with an increasing p. For example, when $\eta = 0.2$, the error ratios of nodes are 100%, 100%, and 99.94% on SocLJ-exp with $p = sys$, $p = 128$, and $p = 256$, respectively. These results indicate that we cannot overcome the defect of existing algorithms by using high-precision floating-point number operations.

Moreover, using high-precision floating-point number operations with the GMP library also incurs high computational costs. Fig. 11 shows the runtime of the InExactUCD algorithm with different floating-point number precisions on SocLJ and Hollywood. We can see that the runtime of InExactUCD increases as p increases. As expected, there is a large gap in runtime between the algorithms implemented by using or not using the GMP library. In particular, the InExactUCD algorithm implemented with the 64-bits floating-point number precision can be one order of magnitude faster than the algorithm implemented by using the GMP library on most datasets. Furthermore, by combining the results shown in Fig. 2, we can see that our top-down algorithm TD+ also can achieve $5\times$ faster than InExactUCD implemented by using the GMP library with $p = 128$. These results further confirm that TD+ is very efficient, and it can achieve comparable runtime of the InExactUCD algorithm even when it is implemented with the 64-bits floating-point number precision.

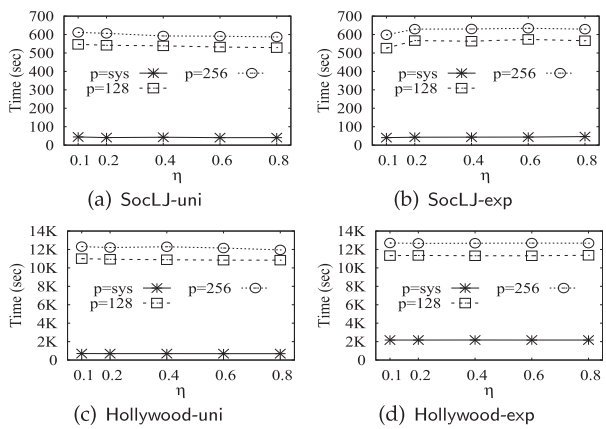


Fig. 11. Runtime of InExactUCD with different floating-point number precisions.

7 Related Works

Uncertain Graph Mining. Uncertain graph mining has attracted much attention in the database community. Notable examples include mining maximal cliques in uncertain graphs [13], [14], [25], identifying reliable connected subgraphs in uncertain graphs [26], finding reliable clustering in uncertain graphs [27], [28], performing influence analysis on uncertain graphs [7], and clustering uncertain graphs [29]. Recently, several cohesive subgraph models, including k-core, k-truss, and maximal clique, have also been extended to uncertain graphs. Specifically, Bonchi *et al.* [8] extended the k-core model to uncertain graphs and proposed a peeling algorithm to compute the core decomposition. Huang *et al.* [11] and Zou *et al.* [12] independently extended the k-truss model to uncertain graphs and also developed peeling-style algorithms to compute the truss decomposition. Mukherjee [14], [25] generalized the traditional maximal clique model to the uncertain graph setting and developed a branch-and-bound algorithm to enumerate all maximal cliques on uncertain graphs. Based on the same clique model, Li *et al.* [15] proposed an improved branch-and-bound algorithm to enumerate maximal cliques on uncertain graphs.

Core Decomposition on Graphs. Core decomposition is a basic graph mining operator which has been widely applied in many graph analysis applications [3], [30], [31], [32]. The concept of k-core was first proposed by Seidman [16]. Batagelj *et al.* [17] shown that the core decomposition of a graph can be computed in linear time based on an elegant peeling algorithm. Montresor *et al.* [33] developed an h-index iteration algorithm to compute the core decomposition of a graph in a distributive setting. Based on the same idea, Wen *et al.* [34] proposed an I/O-efficient algorithm to calculate the core decomposition on disk-resident graphs. Recently, the problems of core decomposition have also been studied on different types of graphs, including direct graphs [35], streaming graphs [36], [37], [38], temporal graphs [39], bipartite graphs [4], [40] and uncertain graphs [8], [9]. As we discussed previously, the state-of-the-art algorithms for core decomposition on uncertain graphs cannot obtain the correct results. Our work focuses mainly on developing correct and efficient core decomposition algorithms for uncertain graphs.

8 Conclusion

In this paper, we study the problem of computing the (k, η)-core decomposition on uncertain graphs. We first discover that the state-of-the-art algorithms for solving this problem are incorrect due to the imprecision of the recursive floating-point number division operations. To solve this issue, we propose an efficient bottom-up algorithm based on an on-demand η-degree computation strategy which does not involve any floating-point number division operation. To further improve the efficiency, we also present a top-down framework with several carefully-designed optimization techniques to compute all (k, η)-cores. In addition, we also develop parallel variants for all our proposed algorithms. The results of extensive experiments on five large uncertain graphs demonstrate the efficiency and scalability of our algorithms, as well as the inaccuracy of the existing algorithms.

REFERENCES

[1] A. Conte, D. Firmani, C. Mordente, M. Patrignani, and R. Torlone, "Fast enumeration of large k-plexes," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 115–124.

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

[3] R. Li *et al.*, "Skyline community search in multi-valued networks," in *Proc. Int. Conf. Manage. Data*, 2018, pp. 457–472.

[4] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient (,)-core computation: An index-based approach," in *Proc. Int. Conf. World Wide Web*, 2019, pp. 1130–1141.

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

[6] H. Kawahigashi, Y. Terashima, N. Miyauchi, and T. Nakakawaji, "Modeling ad hoc sensor networks using random graph theory," in *Proc. 2nd IEEE Consum. Commun. Netw. Conf.*, 2005, pp. 104–109.

[7] Y. Mehmood, F. Bonchi, and D. Garc ía-Soriano, "Spheres of influence for more effective viral marketing," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 711–726.

[8] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich, "Core decomposition of uncertain graphs," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 1316–1325.

[9] B. Yang, D. Wen, L. Qin, Y. Zhang, L. Chang, and R.-H. Li, "Index-based optimal algorithm for computing K-cores in large uncertain graphs," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 64–75.

[10] F. Esfahani, V. Srinivasan, A. Thomo, and K. Wu, "Efficient computation of probabilistic core decomposition at web-scale," in *Proc. Int. Conf. Extending Database Technol.*, 2019, pp. 325–336.

[11] X. Huang, W. Lu, and L. V. S. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 77–90.

[12] Z. Zou and R. Zhu, "Truss decomposition of uncertain graphs," *Knowl. Inf. Syst.*, vol. 50, no. 1, pp. 197–230, 2017.

[13] Z. Zou, J. Li, H. Gao, and S. Zhang, "Finding top-k maximal cliques in an uncertain graph," in *Proc. IEEE 26th Int. Conf. Data Eng.*, 2010, pp. 649–652.

[14] A. P. Mukherjee, P. Xu, and S. Tirthapura, "Mining maximal cliques from an uncertain graph," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 243–254.

[15] R.-H. Li, Q. Dai, G. Wang, Z. Ming, L. Qin, and J. X. Yu, "Improved algorithms for maximal clique search in uncertain networks," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 1178–1189.

[16] S. B. Seidman, "Network structure and minimum degree," *Soc. Netw.*, vol. 5, no. 3, pp. 269–287, 1983.

[17] V. Batagelj and M. Zaversnik, "An O(m) algorithm for cores decomposition of networks," 2003, *arXiv:cs/0310049*.

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

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

[20] R.-H. Li, J. X. Yu, R. Mao, and T. Jin, "Efficient and accurate query evaluation on uncertain graphs via recursive stratified sampling," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 892–903.

[21] R.-H. Li, J. X. Yu, R. Mao, and T. Jin, "Recursive stratified sampling: A new framework for query evaluation on uncertain graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 2, pp. 468–482, Feb. 2016.

[22] R. Butt, "Introduction to numerical analysis using MATLAB," Jones & Bartlett Learning, 1st ed., 2009.

[23] E. W. Weisstein, "Binomial distribution," 2002. [Online]. Available: https://mathworld.wolfram.com/BinomialDistribution.html

[24] H. Kabir and K. Madduri, "Parallel k-core decomposition on multicore platforms," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2017, pp. 1482–1491.

[25] A. P. Mukherjee, P. Xu, and S. Tirthapura, "Enumeration of maximal cliques from an uncertain graph," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 3, pp. 543–555, Mar. 2017.

[26] R. Jin, L. Liu, and C. C. Aggarwal, "Discovering highly reliable subgraphs in uncertain graphs," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2011, pp. 992–1000.

[27] L. Liu, R. Jin, C. Aggarwal, and Y. Shen, "Reliable clustering on uncertain graphs," in *Proc. IEEE 12th Int. Conf. Data Mining*, 2012, pp. 459–468.

[28] Y.-X. Qiu *et al.*, "Efficient structural clustering on probabilistic graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 10, pp. 1954–1968, Oct. 2019.

[29] K. Han *et al.*, "Efficient and effective algorithms for clustering uncertain graphs," *Proc. VLDB Endowment*, vol. 12, no. 6, pp. 667–680, 2019.

[30] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, "K-core decomposition of internet graphs: Hierarchies, self-similarity and measurement biases," 2005, *arXiv:cs/0511007v4*.

[31] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, "Large scale networks fingerprinting and visualization using the k-core decomposition," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2005, pp. 41–50.

[32] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis, "Evaluating cooperation in communities with the k-core structure," in *Proc. Int. Conf. Advances Soc. Netw. Anal. Mining*, 2011, pp. 87–93.

[33] A. Montresor, F. De Pellegrini , and D. Miorandi, "Distributed k-core decomposition," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 2, pp. 288–300, Feb. 2013.

[34] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. X. Yu, "I/O efficient core graph decomposition at web scale," in *Proc. IEEE 32nd Int. Conf. Data Eng.*, 2016, pp. 133–144.

[35] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis, "D-cores: Measuring collaboration of directed graphs based on degeneracy," *Knowl. Inf. Syst.*, vol. 35, no. 2, pp. 311–343, 2013.

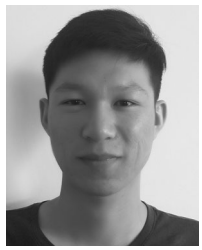
[36] A. E. Sariyüce, B. Gedik, G. Jacques-Silva , K.-L. Wu, and Ü. V. Çatalyürek, "Streaming algorithms for k-core decomposition," *Proc. VLDB Endowment*, vol. 6, no. 6, pp. 433–444, 2013.

[37] 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, Oct. 2014.

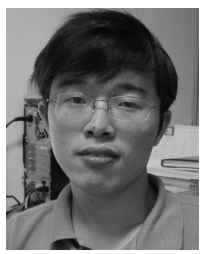
[38] Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin, "A fast order-based approach for core maintenance," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 337–348.

[39] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, "Persistent community search in temporal networks," in *Proc. IEEE 34th Int. Conf. Data Eng.*, 2018, pp. 797–808.

[40] D. Ding, H. Li, Z. Huang, and N. Mamoulis, "Efficient fault-tolerant group recommendation using alpha-beta-core," in *Proc. ACM Conf. Inf. Knowl. Manage.*, 2017, pp. 2047–2050.



Qiangqiang Dai is currently working toward the PhD degree at the Beijing Institute of Technology (BIT), Beijing, China. His research interests include graph data management and mining, social network analysis, and graph computation systems.



Rong-Hua Li received the PhD degree from the Chinese University of Hong Kong, Hong Kong, in 2013. He is currently a professor with the Beijing Institute of Technology (BIT), Beijing, China. Before joining BIT in 2018, he was an assistant professor with Shenzhen University. His research interests include graph data management and mining, social network analysis, graph computation systems, and graph-based machine learning.

Guoren Wang received the BS, MS, and PhD degrees from the Department of Computer Science, Northeastern University, Shenyang, China, in 1988, 1991, and 1996, respectively. Currently, he is a professor with the Beijing Institute of Technology (BIT), Beijing, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management.

Rui Mao received the PhD degree in computer science from the University of Texas at Austin, Austin, Texas, in 2007. He is currently a professor with Shenzhen University. His research interests include big data analysis and management, content-based similarity query of multimedia and biological data, data mining, and machine learning.

Zhiwei Zhang received the BS degree from the Renmin University of China, Beijing, China, in 2010, and the PhD degree from the Chinese University of Hong Kong, Hong Kong, in 2014. He is currently a professor with the Beijing Institute of Technology (BIT), Beijing, China. His research interests include federal learning, data pricing and transaction, distributed system, blockchain, and algorithm analysis.

Ye Yuan received the BS, MS, and PhD degrees in computer science from Northeastern University, Boston, Massachusetts, in 2004, 2007, and 2011, respectively. He is now a professor with the Beijing Institute of Technology (BIT), Beijing, China. His research interests include graph databases, probabilistic databases, and social network analysis.