2019 IEEE 35th International Conference on Data Engineering (ICDE)

Index-based Optimal Algorithm for Computing K-Cores in Large Uncertain Graphs

Bohua Yang[♮], Dong Wen[♮], Lu Qin[♮], Ying Zhang[♮], Lijun Chang[§] and Rong-Hua Li[‡]

[♮]*Centre for Artificial Intelligence, University of Technology Sydney, Australia*
[§]*The University of Sydney, Australia* [‡]*Beijing Institute of Technology, China*
[♮]bohua.yang@student.uts.edu.au; {dong.wen, lu.qin, ying.zhang}@uts.edu.au;
[§]lijun.chang@sydney.edu.au; [‡]lironghuascut@gmail.com

***Abstract*—Uncertainty in graph data occurs for a variety of reasons, such as noise and measurement errors. Recently, uncertain graph management and analysis have attracted many research attentions. Among them, computing k-cores in uncertain graphs (aka, (k, η)-cores) is an important problem and has emerged in many applications, for example, community detection, protein-protein interaction network analysis and influence maximization. Given an uncertain graph, the (k, η)-cores can be derived by iteratively removing the vertex with an η-degree of less than k and updating the η-degrees of its neighbors. However, the results heavily depend on the two input parameters k and η, and the settings for these parameters are unique to the specific graph structure and the user's subjective requirements. Additionally, computing and updating the η-degree for each vertex is the most costly component of the algorithm, and that cost is high. To overcome these drawbacks, we have developed an index-based solution for computing (k, η)-cores in this paper. The size of the index is well bounded by $O(m)$, where m is the number of edges in the graph. Based on this index, queries for any k and η can be answered in optimal time. Further, the method is accompanied by several different optimizations to speed up construction of the index. We conduct extensive experiments on eight real-world datasets to practically evaluate the performance of all the proposed algorithms. The results demonstrate that this index-based approach is several orders of magnitude faster at processing queries than the traditional online approaches.**

I. Introduction

Graphs have been widely used to model sophisticated relationships between different entities due to their strong representative properties. Social networks, traffic networks and biological networks are among the applications that benefit from being expressed as graphs. However, many real-world applications contain uncertainty in the form of noise [1], measurement errors [2], the accuracy of predictions [3], privacy concerns [4], and so on. These uncertain relationships are often modeled as an uncertain graph, where the actual existence of each edge is assigned an "existence probability".

A large number of studies on uncertain graph analysis and management have involved combining fundamental graph problems with uncertain graph models. These studies span a range of tasks, such as reliability searches [5], frequent pattern mining [6] and dense subgraph detection [7]. Among the solutions, k-core is a popular and well-studied cohesive subgraph metric [8], and the k-core conception in the uncertain graph model is originally formalized in [9].

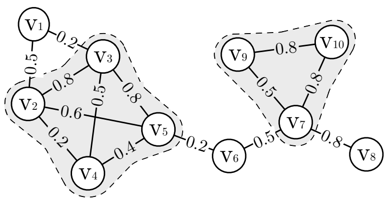

Fig. 1. The (k, η)-cores of $\mathcal{G}$ for $k = 2$ and $\eta = 0.3$

k-Cores in Deterministic Graphs. Given a deterministic graph, a k-core is a maximal connected subgraph where each vertex has a degree of at least k [8]. k-cores are computed by iteratively removing the vertex with the minimum degree and incident edges. This is done in linear time. Computing k-cores has a large number of real-world applications: community detection [10], [11], network visualization [12], network topology analysis [8], system structure analysis [13], protein-protein interaction network analysis [14], and so on. It also serves to find an approximation result for densest subgraph [15], betweenness score [16].

$(\mathbf{k}, \eta)$-Cores in Uncertain Graphs. In the context of uncertain graph models, the degree of each vertex is uncertain. A (k, η)-core model in uncertain graphs is formalized in [9]. A (k, η)-core is a maximal subgraph where each vertex has at least a probability of η that the degree of this vertex is at least k. Note that, in this paper, we have imposed a connectivity constraint to ensure the cohesiveness of the resulting subgraph, i.e., a (k, η)-core is connected. Figure 1 illustrates an example of the (k, η)-cores. Here, given an integer $k = 2$ and a probability threshold $\eta = 0.3$, the uncertain graph contains two $(2, 0.3)$-cores — $\mathcal{G}[\{v_2, v_3, v_4, v_5\}]$ and $\mathcal{G}[\{v_7, v_9, v_{10}\}]$. Computing the (k, η)-cores can be naturally applied in the aforementioned areas. For example, in DBLP collaboration network, each vertex represents an author, and edges represent co-authorships. The edge probability is an exponential function based on the number of collaborations [17]. A (k, η)-core in this case may be a research group.

Additionally, [9] introduced some specific applications for (k, η)-cores associated with uncertain graph models, such as influence maximization and task-driven team formation.

Given an uncertain graph $\mathcal{G}$, an integer k and a probability threshold η, this paper explores the problem of efficiently computing all the (k, η)-cores in $\mathcal{G}$. In other words, our aim is to compute a set of vertex sets, and the induced subgraph of each vertex set is a (k, η)-core.

The Online Approach. In the original paper [9], (k, η)-cores are derived using an η-core decomposition algorithm, given an uncertain graph $\mathcal{G}$ and a probability threshold η. The algorithm computes an η-core number for each vertex u in $\mathcal{G}$, where the η-core number for u is the largest integer k such that a (k, η)-core containing u exists. Let the η-degree of a vertex u be the largest possible degree such that the probability of u to have that degree is no less than η. The algorithm iteratively removes the vertex with the minimum η-degree and updates the η-degrees of the neighbors. With a small modification, this algorithm could compute all the (k, η)-cores in our problem. Specifically, iteratively removing all the vertices with η-degrees of less than k would result in a vertex set that matches our specifications. The final result can then be generated by performing a connected component detection procedure on that vertex set.

Motivation. Even though the online approach can successfully compute the (k, η)-cores, several challenges remain:

- *Parameters Tuning.* The results heavily depend on two sensitive input parameters, k and η, and these parameter settings usually depend on the topological structure of the input graph along with user's subjective requirements. To arrive at a satisfying result, users may need to run the algorithm several times to properly tune the parameters.
- *Query Efficiency.* Computing and updating the η-degree for each vertex is costly and accounts for the majority of the running time in the algorithm. Even though [9] proposes a dynamic programming approach to partially offset this problem, the algorithm is still time-intensive and is not scalable to large uncertain graphs.

An Index-based Approach. Motivated by these challenges, we have developed an order-based index structure, called $UCO\text{-}Index$. The general idea is to retain the resulting vertices for every possible k and η. Specifically, a probability order for each vertex is maintained. Given an integer k and a probability threshold η, a vertex in the result set is identified by comparing the k-th value in the order for this vertex with η. The final result is then produced by performing a connected component detection on the vertex set. We have imposed a bound on the length of the order for each vertex according to the core number, i.e., the largest integer k such that a k-core exists containing this vertex. Therefore, the space for the $UCO\text{-}Index$ is well-bounded by $O(m)$, where m is the number of edges in the graph. The time complexity for query processing is $O(n + \sum_{u \in C} Deg(u))$ for every possible parameter setting of k and η, where n is the number of vertices and $\sum_{u \in C} Deg(u)$ is the sum of degrees of all the vertices in the result set C.

However, there is still room to reduce the amount of time it takes for query processing based on the $UCO\text{-}Index$. Hence, we also propose an alternative method for computing the (k, η)-cores based on a forest index structure, called $UCF\text{-}Index$. In this method, rather than maintaining the order of each vertex, $UCF\text{-}Index$ maintains a tree structure for each integer k. Each tree node contains a set of vertices, and a probability value is assigned to the tree node, which means a corresponding (k, η)-core that contains these vertices exists. The size of $UCF\text{-}Index$ is also bounded by $O(m)$. Using the $UCF\text{-}Index$, we make the time complexity of query processing optimal. In other words, let $|C|$ be the number of vertices in the result set. The running time of the query algorithm is bounded by $O(|C|)$.

Further, we have explored two optimizations to speed up construction of the index. The first one is called *core-based reduction*. By computing the core number of each vertex, some unnecessary neighbors of each vertex are pruned to reduce the running time required to compute and update the probabilities for each vertex. This approach is especially effective in the last few iterations of the index construction algorithm.

The second optimization is called *core-based ordering*. This approach avoids the need for repeated computations of each vertex in the dynamic programming schema as each iteration of index construction algorithm proceeds without breaking the correctness. Our experiments demonstrate a significant increase in speed as a result of these two optimizations.

Contributions. The main contributions of this paper are summarized as follows:

- *The first index-based solution for computing (k, η)-cores in uncertain graphs.* This study presents an effective index structure, called $UCF\text{-}Index$, for computing all the (k, η)-cores. The size of $UCF\text{-}Index$ is well-bounded by $O(m)$. To the best of our knowledge, this is the first index-based solution to this problem.
- *Optimal query processing.* We also present an efficient query algorithm based on $UCF\text{-}Index$ for any possible k and η. The time complexity is optimal and linear to the number of vertices in the result set.
- *Optimizations for index construction.* Two optimizations for the index construction are included: *core-based reduction* and *core-based ordering*. Using these two optimizations, the query processing time is significantly reduced.
- *Extensive performance studies on both real-world and synthetic datasets.* Extensive experiments were conducted with all the proposed algorithms on eight real-world datasets. The results demonstrate that this index-based approach is several orders of magnitude faster than the online approach.

Outline. The rest of this paper is summarized as follows. Section II provides some preliminary concepts and formally defines the problem. In Section III, we review an existing solution and explain the online approach in detail. Section IV describes the basic structure of the index. Section V presents the optimized forest-based index structure. Section VI practically evaluates the proposed algorithms in practical terms. Section VII summarizes related works, and Section VIII concludes the paper. Due to the space limitations, we omit the proofs for some lemmas and theorems.

II. Preliminaries

Before stating the problem, we first introduce the concept of k-cores in deterministic graphs, followed by the definition of (k, η)-core in uncertain graphs.

K-Cores in Deterministic Graphs. Given a deterministic undirected graph $G(V, E)$, V is the set of vertices and E is the set of edges. Given a vertex u in G, the neighbor set of u is denoted as $N(u, G)$, i.e., $N(u, G) = \{v \in V | (u, v) \in E\}$. The degree of u is denoted as $deg(u, G)$, i.e., $deg(u, G) = |N(u, G)|$. We use the terms $N(u)$ and $deg(u)$ for simplicity when the context is clear. Given a set of vertices V', the induced subgraph of V' is denoted as $G[V']$, i.e., $G[V'] = (V', \{(u, v) \in E | u, v \in V'\})$. The definitions for k-core and core number are presented as follows.

DEFINITION 1. (k-CORE) *Given a graph $G(V, E)$ and an integer k, the k-core is a maximal connected induced subgraph $G'[V']$ in which every vertex has a degree of at least k, i.e., $\forall u \in V', deg(u, G') \geq k$. [8]*

DEFINITION 2. (CORE NUMBER) *Given a graph $G(V, E)$, the core number for a vertex u, denoted as $core(u)$, is the largest integer of k such that a k-core containing u exists.*

Given a deterministic graph G, the core numbers for all vertices can be computed by iteratively removing the vertex with the minimum degree. The pseudocode is given in Algorithm 1.

Algorithm 1: CORE DECOMPOSITION

```
Input: A Graph $G(V, E)$
Output: The core numbers for all vertices in $G$
1 while $V \neq \emptyset$ do
2     $k \leftarrow \min_{u \in V} deg(u, G)$;
3     while $\exists u \in V$ s.t. $deg(u, G) \leq k$ do
4         $core(u) \leftarrow k$;
5         foreach $v \in N(u, G)$ do
6             remove edge $(u, v)$ from $G$;
7             $deg(v, G) \leftarrow deg(v, G) - 1$;
8         $V \leftarrow V \setminus \{u\}$;
9 return $core(u)$ for all vertices $u$;
```

K-Core in Uncertain Graphs. Given an uncertain graph $\mathcal{G}(V, E, p)$, p is a function that maps each edge to a probability value in $[0, 1]$ in addition to the vertex set V and the edge set E. The probability of an edge $e \in E$ is denoted by p_e. We denote the neighbor set and the degree of a vertex u in an uncertain graph $\mathcal{G}$ as $\mathcal{N}(u, \mathcal{G})$ and $Deg(u, \mathcal{G})$ respectively.

In line with existing works, we assume that the probability of each edge actually existing is independent, and adopt the well-known possible-world semantics for uncertain graph analysis. There exist $2^{|E|}$ possible graph instances under this assumption. The probability of observing a graph instance $G(V, E')$, denoted by $Pr(G)$, is:

$$Pr(G) = \prod_{e \in E'} p_e \prod_{e \in E \setminus E'} (1 - p_e). \quad (1)$$

The concept of (k, η)-cores, originally defined in [9], is based on possible-world semantics.

DEFINITION 3. ((k, η)-CORE) *Given an uncertain graph $\mathcal{G}(V, E, p)$, an integer k and a probabilistic threshold $\eta \in [0, 1]$, the (k, η)-core of $\mathcal{G}$ is a maximal connected induced subgraph $\mathcal{G}'[V']$ such that the probability that each vertex $u \in V'$ has a degree of at least k in $\mathcal{G}'$ is not less than η, i.e., $\forall u \in V', Pr[deg(u, \mathcal{G}') \geq k] \geq \eta$.*

Note that we have slightly revised this definition by adding a connectivity constraint to the (k, η)-cores. An example of (k, η)-cores is given as follows.

EXAMPLE 1. *Consider the uncertain graph $\mathcal{G}$ in Figure 1. Given an integer $k = 2$ and a probability threshold $\eta = 0.3$, we identify two $(2, 0.3)$-cores, as marked in the figure. One is $\mathcal{G}[\{v_2, v_3, v_4, v_5\}]$, and the other one is $\mathcal{G}[\{v_7, v_9, v_{10}\}]$. We denote $\mathcal{G}[\{v_2, v_3, v_4, v_5\}]$ as $\mathcal{G}_1$ for simplicity. Consider the vertex v_2 in $\mathcal{G}_1$. There are three edges connected to v_2 in $\mathcal{G}_1$, and we have $Pr[deg(v_2, \mathcal{G}_1) \geq 2] = 0.568$. Similarly, we have $Pr[deg(v_3, \mathcal{G}_1) \geq 2] = 0.8$, $Pr[deg(v_4, \mathcal{G}_1) \geq 2] = 0.3$ and $Pr[deg(v_5, \mathcal{G}_1) \geq 2] = 0.656$. $\mathcal{G}_1$ is maximal. Assume that we add v_1 in to $\mathcal{G}_1$. We have $Pr[deg(v_1, \mathcal{G}_1) \geq 2] = 0.1 < 0.3$. Therefore, v_1 cannot be in the $(2, 0.3)$-core.*

Based on Definition 3, the problem is defined as follows.

Problem Definition. Given an uncertain graph $\mathcal{G}(V, E, p)$, an integer k and a probabilistic threshold $\eta \in [0, 1]$, we examine the problem of efficiently computing all the (k, η)-cores of $\mathcal{G}$.

Specifically, let C be the vertex set such that the induced subgraph $\mathcal{G}[C]$ of C is a (k, η)-core. The aim is to compute a set $\mathcal{R}$ containing all such vertex sets C without any duplication. In the case of $k = 2, \eta = 0.3$ in Figure 1, we return $\{\{v_2, v_3, v_4, v_5\}, \{v_7, v_9, v_{10}\}\}$.

III. ONLINE (k, η)-CORES COMPUTATION

In this section, we first review an existing solution [9] for the problem of η-core decomposition, as several key concepts and ideas intuitively fit our problem. Then, we provide our online solution for computing (k, η)-cores.

A. An Existing Solution for η-Core Decomposition

Given an uncertain graph $\mathcal{G}$, let $\mathcal{G}_u^{\geq k}$ be the set of all possible graph instances where u has a degree of at least k, i.e., $\mathcal{G}_u^{\geq k} = \{G \sqsubseteq \mathcal{G} | deg(u, G) \geq k\}$. We have the following equation [9]:

$$Pr[deg(u, \mathcal{G}) \geq k] = \sum_{G \in \mathcal{G}_u^{\geq k}} Pr(G). \quad (2)$$

Based on Equation 2, the definition for the η-degree for each vertex follows.

DEFINITION 4. (η-DEGREE) *Given an uncertain graph $\mathcal{G}(V, E, p)$ and a probabilistic threshold $\eta \in [0, 1]$, the η-degree of a vertex $u \in V$, denoted by η-$deg(u, \mathcal{G})$, is the largest integer of k that satisfies $Pr[deg(u, \mathcal{G}) \geq k] \geq \eta$. [9]*

Given a vertex u, $Pr[deg(u) \geq k]$ monotonously decreases when k increases. Next, we define the η-core number.

DEFINITION 5. (η-CORE NUMBER) *Given an uncertain graph $\mathcal{G}(V, E, p)$ and a probabilistic threshold $\eta \in [0, 1]$, the η-core number for a vertex u, denoted as η-$core(u)$, is the largest integer of k such that a (k, η)-core containing u exists.*

Based on Definition 3 and Definition 5, we have the following lemma.

LEMMA 1. *Given an uncertain graph $\mathcal{G}$ and a probability threshold $\eta \in [0, 1]$, a vertex u is in a (k, η)-core if and only if η-$core(u) \geq k$.*

Algorithm 2: η-CORE DECOMPOSITION

```
Input: An uncertain graph $\mathcal{G}(V, E, p)$ and a probability threshold $\eta$
Output: $\eta$-core numbers for all vertices in $\mathcal{G}$
1  compute $\eta$-$deg(u, \mathcal{G})$ for all $u \in V$;
2  while $\mathcal{G}$ is not empty do
3      $k \leftarrow \min_{u \in V} \eta$-$deg(u, \mathcal{G})$;
4      while $\exists u \in V$ s.t. $\eta$-$deg(u, \mathcal{G}) \leq k$ do
5          $\eta$-$core(u) \leftarrow k$;
6          foreach $v \in \mathcal{N}(u, \mathcal{G})$ do
7              remove edge $(u, v)$ from $\mathcal{G}$;
8              update $\eta$-$deg(v, \mathcal{G})$;
9          $V \leftarrow V \setminus \{u\}$;
10 return $\eta$-$core(u)$ for all vertices $u$;
```

The problem of computing the η-core numbers for all vertices in the uncertain graph $\mathcal{G}$ is called η-core decomposition. The solution proposed in [9] is provided in Algorithm 2.

Algorithm 2 shares the similar idea with Algorithm 1, and the pseudocode is self-explanatory. The key steps in the algorithm are computing (line 1) and updating (line 8) the η-degrees of the vertices. We introduce their details below.

η-Degree Computation. To compute the η-degree, we first present the following equation:

$$Pr[deg(u) \geq k] = \sum_{i=k}^{Deg(u)} Pr[deg(u) = i] = 1 - \sum_{i=0}^{k-1} Pr[deg(u) = i]. \tag{3}$$

Based on Equation 3, we can start with $Pr[deg(u) \geq 0] = 1$. Iteratively, we increase i by one and compute $Pr[deg(u) = i]$ for a vertex u. We calculate $Pr[deg(u) \geq i+1]$ as $Pr[deg(u) \geq i] - Pr[deg(u) = i]$. We repeat this step and terminate once $Pr[deg(u) \geq i+1] < \eta$. Then we have η-$deg(u) = i$.

To compute $Pr[deg(u) = i]$ for a vertex u, we use the dynamic-programming method given in [9]. Assume that $E(u) = \{e_1, e_2, ..., e_{Deg(u)}\}$ is the set of all the edges connected to u in some order. The intuitive idea of dynamic programming is that, if a vertex u has a degree of i, one of the following two cases applies: either (i) $i-1$ edges exist in $\{e_1, e_2, ..., e_{Deg(u)-1}\}$ and $e_{Deg(u)}$ exists; or (ii) i edges exist in $\{e_1, e_2, ..., e_{Deg(u)-1}\}$ and $e_{Deg(u)}$ does not exist.

Given a subset $E'(u) \subseteq E(u)$, let $deg(u|E'(u))$ be the degree of u in the subgraph $\mathcal{G}'(V, E \backslash (E(u) \backslash E'(u)), p)$, and $X(h, j) = Pr[deg(u|\{e_1, e_2, ..., e_h\}) = j]$. We have the following dynamic-programming recursive function [9] for all $h \in [1, Deg(u)], j \in [0, h]$:

$$X(h, j) = p_{e_h} X(h-1, j-1) + (1 - p_{e_h}) X(h-1, j). \tag{4}$$

Several initialization cases are also given as follows:

$$\begin{cases} X(0,0) = 1, & \\ X(h,-1) = 0, & \text{for all } h \in [0, Deg(u)], \\ X(h,j) = 0, & \text{for all } h \in [0, Deg(u)], j \in [h+1, i]. \end{cases} \tag{5}$$

The time complexity to compute the η-degree of a vertex is given as follows.

LEMMA 2. *The time complexity to compute the η-degree of a vertex u is $O(\eta\text{-}deg(u) \cdot Deg(u))$. [9]*

η-Degree Update. Given the incident edge set $E(u)$ of a vertex u, assume that an edge e is removed from $E(u)$. To compute the updated probability $Pr[deg(u|E(u)\backslash\{e\}) = i]$, we introduce the following equation [9]:

$$Pr[deg(u|E(u)\backslash\{e\}) = i)] = \frac{Pr[deg(u) = i] - p_e Pr[deg(u|E(u)\backslash\{e\}) = i-1]}{1 - p_e}. \tag{6}$$

Based on Equation 6, we compute $Pr[deg(u|E(u)\backslash\{e\}) = i]$ for each $i \in [1, \eta\text{-}deg(u)]$ in constant time, given that $Pr[deg(u|E(u)\backslash\{e\}) = 0] = \frac{1}{1-p_e} Pr[deg(u) = 0]$. The time complexity to update the η-degree is given as follows.

LEMMA 3. *Given an uncertain graph $\mathcal{G}$ and a removed incident edge e to a vertex u, the time complexity to update the η-degree of u is $O(\eta\text{-}deg(u))$. [9]*

We also provide the time complexity of Algorithm 2 below.

LEMMA 4. *Given an uncertain graph $\mathcal{G}(V, E, p)$, the time complexity of Algorithm 2 is $O(\sum_{u \in V} \eta\text{-}deg(u) \cdot Deg(u))$. [9]*

B. Our Approach to Compute (k, η)-Cores

Based on several concepts introduced in the previous section, we turn to the online approach for computing all the (k, η)-cores. Our approach is similar to Algorithm 2, which iteratively removes the vertex that does not belong to the result set. Before presenting the algorithm, we make the following observation about optimization.

OBSERVATION 1. *Given an uncertain graph $\mathcal{G}$ and a (k, η)-core $\mathcal{G}[C]$ for any parameter settings for k and η, there exists a k-core $G[C']$ containing $\mathcal{G}[C]$, i.e., $C \subseteq C'$.*

Based on Observation 1, we can first recursively remove the vertices with degrees of less than k, since these vertices cannot be in the result set for any (k, η)-core. We provide the pseudocode for our approach in Algorithm 3.

Lines 1–5 compute the k-cores. Lines 6–11 recursively remove the vertices with η-degrees of less than k and generate a subgraph containing all result vertices. Lines 12–14 determine the connected components in the result. The time complexity of Algorithm 3 is $O(\sum_{u \in V} \eta\text{-}deg(u) \cdot Deg(u))$, which is the same as that of Algorithm 2.

Drawbacks of the Online Approach. Even though Algorithm 3 successfully computes all the (k, η)-cores, several drawbacks still exist. First, changing the input parameters may heavily influence the resulting (k, η)-cores, especially in large graphs. We consider the case in Figure 1. If we change the input parameter η from 0.3 to 0.4 and keep $k = 2$, vertex v_4 will be removed and the result will change to $\{\{v_2, v_3, v_5\}, \{v_7, v_9, v_{10}\}\}$. Additionally, we find that the major cost in Algorithm 3 is computing and updating the η-degrees of the vertices. This is extremely time-consuming and means the algorithm cannot be scaled to big graphs.

Motivated by the above challenges, we propose an index-based approach. Based on the proposed index, we can answer a query for any given k and η with a time complexity that is only proportional to the size of the results. To make our

```
Algorithm 3: $(k, \eta)$-CORES COMPUTATION
Input: An uncertain graph $\mathcal{G}(V, E, p)$, an integer $k$ and a
       probability threshold $\eta$
Output: All $(k, \eta)$-cores in $\mathcal{G}$
1  while $\exists u \in V$ s.t. $Deg(u) < k$ do
2      foreach $v \in \mathcal{N}(u, \mathcal{G})$ do
3          remove the edge $(u, v)$ from $\mathcal{G}$;
4          $Deg(v) \leftarrow Deg(v) - 1$;
5      $V \leftarrow V \setminus \{u\}$;
6  compute $\eta$-$deg(u)$ for all $u \in V$;
7  while $\exists u \in V$ s.t. $\eta$-$deg(u) < k$ do
8      foreach $v \in \mathcal{N}(u, \mathcal{G})$ do
9          remove the edge $(u, v)$ from $\mathcal{G}$;
10         update $\eta$-$deg(v)$;
11     $V \leftarrow V \setminus \{u\}$;
12 $\mathcal{R} \leftarrow \emptyset$;
13 foreach connected component $\mathcal{G}[C] \in \mathcal{G}$ do
14     $\mathcal{R} \leftarrow \mathcal{R} \cup \{C\}$;
15 return $\mathcal{R}$;
```

solution scalable to big graphs, the index size is well-bounded, with an acceptable time cost for constructing the index.

We propose a basic index approach in Section IV, and in Section V, we optimize both the index structure and the query processing procedure.

IV. An Index-based Approach

A. The Index Structure

In this section, we introduce an index structure, called the uncertain core η-orders index (UCO-$Index$). The general idea of this index is to maintain the result vertices for every possible k and η. In other words, given an integer k and a probability threshold η, we aim to efficiently compute all the result vertices based on the index structure. To complete this task, we start by computing all result vertices from any given probability threshold η under a specific fixed integer k, as there is only a limited number of possible k. Based on Observation 1, we provide the range of integer k as follows.

OBSERVATION 2. *Given an uncertain graph $\mathcal{G}$, we only need to consider the parameter $1 \le k \le k_{max}$, where $k_{max} = \max_{u \in V} core(u)$.*

If $k > k_{max}$, the probability that a (k, η)-core exists is 0. We also provide the largest possible integer for k of each vertex in the following observation.

OBSERVATION 3. *Given an uncertain graph $\mathcal{G}$ and an integer k, a vertex u cannot be in the (k, η)-core if $core(u) < k$.*

Based on Observation 3, we derive a candidate set of result vertices by only considering the parameter k. That is $\{u \in V | core(u) \ge k\}$.

Now, given the candidate set for each integer k, we consider computing the exact result set by the probability threshold η. Recall that a vertex u is in the (k, η)-core if the η-degree of u is at least k. We have the following lemma.

LEMMA 5. *Given an uncertain graph $\mathcal{G}$, a parameter k and two probability threshold $0 \le \eta \le \eta' \le 1$, a vertex u is in (k, η)-core if it is in (k, η')-core.*

According to the monotonicity in Lemma 5, we only need to save the largest probability value η for each vertex u that will be in the (k, η)-core. We call such value the η-*threshold*, which is formally defined as follows.

DEFINITION 6. (η-THRESHOLD) *Given an uncertain graph $\mathcal{G}(V, E, p)$ and an integer k, the η-threshold of a vertex u, denoted by η-$threshold_k(u)$, is the largest η such that a (k, η)-core containing u exists.*

Based on Observation 3 and Definition 6, we have η-$threshold_k(u) = 0$ for any vertex u if $core(u) < k$, and we give a necessary and sufficient condition that a vertex will be in the (k, η)-core as follows.

LEMMA 6. *Given an uncertain graph $\mathcal{G}$, an integer k and a probability threshold η, a vertex u is in the (k, η)-core if and only if η-$threshold_k(u) \ge \eta$.*

To efficiently compute all result vertices, we save all η-thresholds of each vertex u in an order, which is formally defined as follows.

DEFINITION 7. (η-ORDER) *Given an uncertain graph $\mathcal{G}$ and a vertex u, the η-order of u, denoted by η-$order(u)$, is a probability order such that (i) the i-th value in η-$order(u)$ is η-$threshold_i(u)$, and (ii) the length of η-$order(u)$ is $core(u)$.*

k	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
1	0.6	0.92	0.92	0.76	0.92	0.6	0.9	0.8	0.9	0.9
2	0.1	0.48	0.48	0.3	0.48	0.1	0.4		0.4	0.4
3		0.04	0.04	0.04	0.04					

Fig. 2. The UCO-$Index$ of $\mathcal{G}$

EXAMPLE 2. *The η-orders for all vertices in the uncertain graph $\mathcal{G}$ in Figure 1 are given in Figure 2. We consider the example of vertex v_4. Given $k = 2$, we have η-$threshold_2(v_4) = 0.3$. That means v_4 is in a $(2, 0.3)$-core, but not in any $(2, \eta)$-core if $\eta > 0.3$.*

Given the η-order of a vertex u and an integer k, we use a constant time complexity to compute the η-$threshold_k(u)$. We save the η-orders for all vertices as our UCO-$Index$. The size of the UCO-$Index$ is well-bounded.

THEOREM 1. *Given an uncertain graph $\mathcal{G}(V, E, p)$, the space complexity of the UCO-$Index$ is $O(\sum_{u \in V} core(u))$.*

Since $core(u) \le Deg(u)$ for each vertex u, the size of the UCO-$Index$ is also roughly bounded by $O(|E|)$.

B. Query Processing

Before presenting the query processing algorithm, we first give an alternative definition for the (k, η)-core based on Definition 6.

LEMMA 7. *Given a set of vertices C in an uncertain graph $\mathcal{G}$, the induced subgraph $\mathcal{G}[C]$ is a (k, η)-core if and only if (i) $\forall u \in C, \eta$-$threshold_k(u) \ge \eta$; (ii) $\mathcal{G}[C]$ is connected; and (iii) C is maximal.*

Based on the above lemma, we present the pseudocode for the query processing in Algorithm 4. It first identifies all vertices whose η-threshold is not less than η in line 1.

Algorithm 4: UCO-BASED QUERY

```
Input: An uncertain graph $\mathcal{G}(V, E, p)$, an integer $k$, a
       probability threshold $\eta$ and $UCO\text{-}Index$
Output: All $(k, \eta)$-cores in $\mathcal{G}$
1 $V' \leftarrow \{u \in V | \eta\text{-}threshold_k(u) \geq \eta\}$;
2 $\mathcal{R} \leftarrow \emptyset$;
3 foreach connected component $\mathcal{G}[C] \in \mathcal{G}[V']$ do
4     $\mathcal{R} \leftarrow \mathcal{R} \cup \{C\}$;
5 return $\mathcal{R}$;
```

Given the input integer k, the η-threshold of a vertex u can be computed by checking the k-th item in the η-order of u according to Definition 7. The algorithm then computes each (k, η)-core in lines 3–4. The correctness of Algorithm 4 can be guaranteed according to Lemma 7. The running time of Algorithm 4 is analyzed as follows.

THEOREM 2. *Given an uncertain graph $\mathcal{G}(V, E, p)$, an integer k and a probability threshold η, the time complexity of Algorithm 4 is $O(|V| + \sum_{u \in C} Deg(u))$, where C is the set of all result vertices, i.e., $C = \{u \in V | \eta\text{-}threshold_k(u) \geq \eta\}$.*

C. Index Construction

Before introducing the algorithm used to construct the $UCO\text{-}Index$, we first give the following definition for the ease of presentation.

DEFINITION 8. (k-PROBABILITY) *Given an uncertain graph $\mathcal{G}$ and an integer k, the k-probability of a vertex u, denoted by $k\text{-}prob(u, \mathcal{G})$, is the probability that $Pr[deg(u, \mathcal{G}) \geq k]$.*

Based on Definition 8, the general idea to construct the $UCO\text{-}Index$ is iteratively removing the vertex with the minimum k-probability. The detailed pseudocode is given in Algorithm 5.

In Algorithm 5, an empty order is initialized for each vertex in line 2. The η-thresholds for all vertices under a specific k are computed from line 4 to line 18. Specifically, a vertex u with the minimum k-probability is selected in line 10. In line 11, the variable $curThres$ is used to save the η-threshold for the selected vertex u. We know that the $(k, curThres)$-core exists if $curThres > 0$ in line 12. The variable $isEnd$ is used to identify whether the iterations should be terminated, and we assign $isEnd$ with $false$ if $curThre > 0$ in line 13. We push the η-threshold of u into the η-order of u in line 14. We remove the vertex u and update the k-probability for each neighbor v of u in lines 15–18. The η-orders are derived when the algorithm is terminated. The prove is as follows.

THEOREM 3. *Given an uncertain graph $\mathcal{G}$, Algorithm 5 correctly computes the η-orders for all vertices in $\mathcal{G}$.*

THEOREM 4. *Given an uncertain graph $\mathcal{G}(V, E, p)$, the time complexity of Algorithm 5 is $O(k_{max}|V| \log |V| + k_{max}^2 |E|)$, where $k_{max} = \max_{u \in V} core(u)$.*

PROOF. *Based on Lemma 2 and Lemma 3, given a vertex u, computing (line 8) and updating (line 18) the k-probability cost $O(k \cdot Deg(u))$ time and $O(k)$ time respectively.*

In each iteration of lines 3–19, the time complexity of line 8 is $O(\sum_{u \in V} k \cdot Deg(u))$. We use a minimum priority queue (implemented by a Fibonacci heap) to maintain all vertices where the keys are their k-probabilities. The Fibonacci heap takes a constant amount of time for insertion, and linear amount of time to build the priority queue, plus a constant amount of time for updating if the involved key is decreased, and a logarithmic amount of time for delete-min. Thus, in each iteration of lines 3–19, line 10 and line 15 totally take $O(|V| \log |V|)$ time to remove the vertex with the minimum k-probability, and lines 16–18 totally take $O(\sum_{u \in V} k \cdot Deg(u))$ time for each k, since the k-probability of each vertex u can be updated at most $Deg(u)$ times.

Therefore each iteration in lines 3–19 takes $O(|V| \log |V| + \sum_{u \in V} k \cdot Deg(u))$ time, which can be arranged as $O(|V| \log |V| + k|E|)$. The number of iterations is bounded by $O(k_{max})$, where $k_{max} = \max_{u \in V} core(u)$. The time complexity of Algorithm 5 is $O(k_{max}|V| \log |V| + k_{max}^2 |E|)$.

Algorithm 5: UCO-INDEX CONSTRUCTION

```
Input: An uncertain graph $\mathcal{G}(V, E, p)$
Output: $UCO\text{-}Index$ of $\mathcal{G}$
1  $k \leftarrow 0$;
2  foreach $u \in V$ do $\eta\text{-}order(u) \leftarrow \emptyset$;
3  repeat
4      $k \leftarrow k + 1$;
5      $isEnd \leftarrow true$;
6      $\mathcal{G}' \leftarrow \mathcal{G}$;
7      $curThres \leftarrow 0$;
8      compute $k\text{-}prob(u)$ for each $u \in V'$;
9      while $\mathcal{G}'$ is not empty do
10         $u \leftarrow \arg\min_{v \in V'} k\text{-}prob(v, \mathcal{G}')$;
11         $curThres \leftarrow \max(k\text{-}prob(u, \mathcal{G}'), curThres)$;
12         if $curThres > 0$ then
13             $isEnd \leftarrow false$;
14             $\eta\text{-}order(u).push(curThres)$;
15         $V' \leftarrow V' \setminus \{u\}$;
16         foreach $v \in \mathcal{N}(u, \mathcal{G}')$ do
17             remove the edge $(u, v)$ from $\mathcal{G}'$;
18             update $k\text{-}prob(v, \mathcal{G}')$;
19 until $isEnd = true$;
20 return $\eta\text{-}order(u)$ for all vertices $u$;
```

V. MAKING QUERY PROCESSING OPTIMAL

We proposed a $UCO\text{-}Index$ based approach in the previous section. Even though computing the η-degree is avoided and the used space can be well-bounded, the $UCO\text{-}Index$ still needs to detect all vertices to compute the results in the query processing, and this may be hard to tolerate in big graphs. To address this issue, we propose a forest-based index structure, namely uncertain core η-forest index ($UCF\text{-}Index$). Based on the $UCF\text{-}Index$, we compute the result set in optimal time.

The index structure is introduced in Section V-A. We provide the query processing algorithm in Section V-B. We propose the algorithm to construct the $UCF\text{-}Index$ with several optimization techniques in Section V-C.

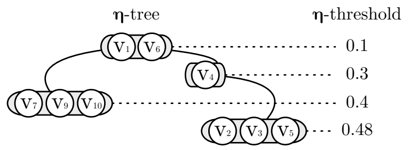

Fig. 3. The η-tree of $\mathcal{G}$ for $k = 2$

A. Forest-based Index Structure

According to Lemma 7, the key to computing all result (k, η)-cores is computing all vertices of u such that $\eta\text{-}threshold_k(u) \geq \eta$. This costs $O(|V|)$ time in Algorithm 4. A straightforward idea to improve the query's efficiency is to sort the vertices in a non-increasing order of their η-threshold for each integer k. Based on this structure, we can compute all result vertices in optimal time, and the total size of this structure can still be bounded by $O(\sum_{u \in V} core(u))$. However, given that there is no topological information between vertices, we still use $O(\sum_{u \in C} Deg(u))$ time to identify the connected components, where $C = \{u \in V | \eta\text{-}threshold_k(u) \geq \eta\}$.

Motivated by this, we propose a novel index, called the $UCF\text{-}Index$, which organizes the vertices and their η-thresholds into a tree structure, for each possible integer k. The tree is built based on Lemma 5, where vertices with smaller η-thresholds are on the upper side, and larger η-thresholds are on the lower side. We name the tree structure η-tree, which is denoted by $\eta\text{-}tree_k$. Specifically, let C_k be the set of vertices whose core numbers are not less than k, i.e., $C_k = \{u \in V | core(u) \geq k\}$. We divide all vertices in C_k into different tree nodes in $\eta\text{-}tree_k$. Considering a tree node $\mathbb{X}$ in the $\eta\text{-}tree_k$, the attributes of $\mathbb{X}$ are summarized as follows:

- $\mathbb{X}.vertices$: return a set of vertices.
- $\mathbb{X}.\eta\text{-}threshold$: return $\eta\text{-}threshold_k(u)$ for any vertex $u \in \mathbb{X}.vertices$.
- $\mathbb{X}.parent$: return the parent node of $\mathbb{X}$.
- $\mathbb{X}.children$: return the children nodes of $\mathbb{X}$.

The details to implement these attributes are presented below. Formally, the vertex set for each tree node is computed using the following rule.

LEMMA 8. *Given an uncertain graph $\mathcal{G}$ and an integer k, we group a vertex set S into a tree node $\mathbb{X}$, i.e., $\mathbb{X}.vertices = S$ if (i) $\forall u, v \in S, \eta\text{-}threshold_k(u) = \eta\text{-}threshold_k(v)$; (ii) let $\eta = \eta\text{-}threshold_k(u)$ for any $u \in S$, there is a (k, η)-core $\mathcal{G}[C]$, such that $S \subseteq C$; and (iii) S is maximal.*

Then we give the rules for the parent-children relationship of tree nodes. Let $\mathcal{G}[V_{\mathbb{X}}]$ be the $(k, \mathbb{X}.\eta\text{-}threshold)$-core containing $\mathbb{X}.vertices$, and $N(\mathcal{G}_{\mathbb{X}})$ be the set of tree nodes in which each tree node $\mathbb{Y}$ satisfies $\exists u \in V_{\mathbb{X}}, v \in \mathbb{Y}.vertices : (u, v) \in E \wedge v \notin V_{\mathbb{X}}$. Note that there does not exist a tree node $\mathbb{Y} \in N(\mathcal{G}_{\mathbb{X}})$ such that $\mathbb{Y}.\eta\text{-}threshold \geq \mathbb{X}.\eta\text{-}threshold$. Otherwise, the vertices in $\mathbb{Y}$ also belong to $V_{\mathbb{X}}$. The parent for each tree node is defined as follows.

LEMMA 9. *Given an uncertain graph $\mathcal{G}$ and an integer k, a tree node $\mathbb{Y}$ is the parent of the tree node $\mathbb{X}$ in $\eta\text{-}tree_k$, if $\mathbb{Y}$ is the tree node in $N(\mathcal{G}_{\mathbb{X}})$ with the largest η-threshold, i.e., $\mathbb{Y} = \arg\max_{\mathbb{Y} \in N(\mathcal{G}_{\mathbb{X}})} \mathbb{Y}.\eta\text{-}threshold$.*

Algorithm 6: UCF-BASED QUERY

```
Input: An uncertain graph $\mathcal{G}(V, E, p)$, an integer $k$, a
       probability threshold $\eta$ and $UCF\text{-}Index$ index
Output: All $(k, \eta)$-cores in $\mathcal{G}$
1  $\mathcal{T} \leftarrow$ the set of all tree nodes in $\eta\text{-}tree_k$;
2  $\mathcal{S} \leftarrow$ initialize an empty stack;
3  while $\mathcal{T}$ is not empty do
4      $\mathbb{X} \leftarrow \arg\max_{\mathbb{X} \in \mathcal{T}} \mathbb{X}.\eta\text{-}threshold$;
5      if $\mathbb{X}.\eta\text{-}threshold \geq \eta$ then $\mathcal{S}.push(\mathbb{X})$;
6      else break;
7      $\mathcal{T} \leftarrow \mathcal{T} \setminus \{\mathbb{X}\}$;
8  $\mathcal{R} \leftarrow \emptyset$;
9  while $\mathcal{S}$ is not empty do
10     $\mathbb{X} \leftarrow \mathcal{S}.pop()$;
11     if $\mathbb{X}$ is visited then continue;
12     $C \leftarrow \emptyset$;
13     $\mathcal{Q} \leftarrow$ initialize an empty queue;
14     $\mathcal{Q}.insert(\mathbb{X})$;
15     while $\mathcal{Q}$ is not empty do
16         $\mathbb{Y} \leftarrow \mathcal{Q}.pop()$;
17         mark $\mathbb{Y}$ as visited;
18         $C \leftarrow C \cup \mathbb{Y}.vertices$;
19         foreach $\mathbb{Z} \in \mathbb{Y}.children$ do $\mathcal{Q}.insert(\mathbb{Z})$;
20     $\mathcal{R} \leftarrow \mathcal{R} \cup \{C\}$;
21 return $\mathcal{R}$;
```

In the case of $N(\mathcal{G}_{\mathbb{X}}) = \emptyset$, the tree node $\mathbb{X}$ is the root node, and there may exist more than one trees for each integer k. We give an example of the tree node and the η-tree as follows.

EXAMPLE 3. *Still considering the uncertain graph $\mathcal{G}$ in Figure 1, we give the η-tree of $\mathcal{G}$ for $k = 2$ in Figure 3. The η-threshold for each tree node is listed on the right side. For the tree node $\{v_2, v_3, v_5\}$, the corresponding $(2, 0.48)$-core is the induced subgraph of the same vertex set. There are two neighbor tree nodes — $\{v_1, v_6\}$ and $\{v_4\}$. The η-threshold of $\{v_4\}$ is larger, and we set $\{v_4\}$ as the parent of $\{v_2, v_3, v_5\}$.*

THEOREM 5. *Given an uncertain graph $\mathcal{G}(V, E, p)$, the space complexity of the $UCF\text{-}Index$ is $O(\sum_{u \in V} core(u))$.*

B. Optimal Query Processing

We give an alternative definition for (k, η)-core based on the proposed $UCF\text{-}Index$.

LEMMA 10. *Given an uncertain graph $\mathcal{G}$, an integer k, and a probability threshold η, let $\mathbb{R}$ be a tree node in $\eta\text{-}tree_k$ such that (i) $\mathbb{R}.\eta\text{-}threshold \geq \eta$; and (ii) there does not exist a parent $\mathbb{R}'$ of $\mathbb{R}$ such that $\mathbb{R}'.\eta\text{-}threshold \geq \eta$. Let C be the set of all vertices in the subtree rooted by $\mathbb{R}$. The induced subgraph $\mathcal{G}[C]$ is a (k, η)-core.*

According to Lemma 10, the key to the query processing is collecting all tree nodes in the subtree rooted by the tree node $\mathbb{R}$ mentioned in the lemma. Following this general idea, we provide the pseudocode for query processing in Algorithm 6.

We first collect all result tree nodes in lines 1–7. We can use constant time to derive the tree node with the largest

η-threshold in line 4, if the tree nodes are sorted in a non-increasing order of their η-thresholds. The order can be precomputed in the index construction phase, which will be detailed introduced in the next subsection. Note that the number of tree nodes does not change in the order, and the total space complexity of the $UCF\text{-}Index$ is still $O(\sum_{u \in V} core(u))$.

We iteratively process each tree node in the stack in lines 9–20. Once we find an unvisited tree node in line 11, we find a root node satisfying the conditions in Lemma 10. We use a queue to compute all tree nodes rooted by $\mathbb{X}$, and collect all vertices in the tree nodes into C in lines 12–19. We add C into the result set in line 20.

EXAMPLE 4. *Given an example for computing the* $(k = 2, \eta = 0.3)$*-core in* $\mathcal{G}$ *of Figure 1 based on the* $UCF\text{-}Index$*. The* η*-tree for* $k = 2$ *is given in Figure 3. We first locate the tree nodes* $\mathbb{R}$ *in Lemma 10, which are* $\{v_4\}$ *and* $\{v_7, v_9, v_{10}\}$*. Then we get two result cores,* $\{v_4, v_2, v_3, v_5\}$ *and* $\{v_7, v_9, v_{10}\}$*.*

Lemma 10 guarantees the correctness of Algorithm 6. The time complexity of Algorithm 6 is summarized as follows.

THEOREM 6. *Given an uncertain graph* $\mathcal{G}(V, E, p)$*, an integer* k *and a probability threshold* η*, the time complexity of Algorithm 6 is* $O(|C|)$*, where* C *is the set of all result vertices, i.e.,* $C = \{u \in V | \eta\text{-}threshold_k(u) \geq \eta\}$*.*

Based on the above theorem, we claim that the time complexity of our query processing algorithm is optimal, since it is bounded by the result size.

C. Optimizations for the Index Construction Algorithm

The algorithm to construct the $UCF\text{-}Index$ is given in this section. We first introduce the general idea. For each integer, we compute the η-threshold for each vertex using the idea in Algorithm 5. Given the η-thresholds, we can build the η-tree using the disjoint-set data structure [18]. A similar idea can be found in [19], [20]. There are two main operations for the disjoint-set: Union$(\mathbb{X}, \mathbb{Y})$ merges the dynamic sets $\mathbb{X}$ and $\mathbb{Y}$. Find$(\mathbb{X})$ returns the set containing $\mathbb{X}$. With the two optimization techniques, *union by rank* and *path compression*, the amortized time per operation of the disjoint-set is only $O(\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function and is normally less than 5 [18].

The dominating cost in constructing the $UCF\text{-}Index$ is still computing the η-thresholds for the vertices. Specifically, we review the Algorithm 5. First, computing and updating the k-probability of the vertices in each iteration is time-consuming as the degree of each vertex may be very large. Second, the number of iterations reaches k_{max}, and the k-probability of the vertices are computed from scratch in each iteration.

To speed up computing the η-threshold, we propose two optimizations in response to the above two challenges.

Core-based Reduction. Given any vertex u in each iteration of lines 3–19 of Algorithm 5, the k-probability of u monotonically decreases due to the removal of its neighbors v with $\eta\text{-}threshold_k(v) \leq \eta\text{-}threshold_k(u)$, and $\eta\text{-}threshold_k(u) = k\text{-}prob(u, \mathcal{G}[\{v \in V | \eta\text{-}threshold_k(v) \geq \eta\text{-}threshold_k(u)\}])$. Without breaking the correctness, we do not need to consider the edge (u, v) when computing $k\text{-}prob(u)$, if $\eta\text{-}threshold_k(v) < \eta\text{-}threshold_k(u)$. According to Observation 1, the η-threshold of a vertex u for an integer k is 0 if the core number for u is less than k. Therefore, for each integer k, we can compute and update the k-probability of each vertex in the subgraph $\mathcal{G}[\{u \in V | core(u) \geq k\}]$.

Core-based Ordering. To conquer the second challenge, we propose a top-down strategy. For each vertex u, we arrange the neighbors of u in a non-increasing order of their core numbers. Specifically, we denote the set of neighbors v of u such that $core(v) \geq k$ by $N_k(u)$. Without losing generality, let $\{e_1, e_2, ..., e_i\}$ be the set of edges between u and $N_k(u)$, and $\{e_{i+1}, e_{i+2}, ..., e_j\}$ be the set of edges between u and $N_{k-1}(u) \setminus N_k(u)$.

According to the aforementioned core-based reduction, the k-probability of u, is initially computed based on the edge set $\{e_1, e_2, ..., e_i\}$, i.e., $k\text{-}prob(u) = Pr[deg(u|\{e_1, e_2, ..., e_i\}) \geq k]$. Let $X(l, r, k) = Pr[deg(u|\{e_l, e_{l+1}, ..., e_r\}) = k]$. Given the integer $k-1$, we aim to compute $(k-1)\text{-}prob(u)$ based on $\{e_1, e_2, ..., e_j\}$, which is equivalent to computing $X(1, j, h)$ for each $h \in [0, k-1]$. We give the following equation for computing $X(1, j, h)$:

$$X(1, j, h) = \sum_{l=0}^{h} X(1, i, l) \cdot X(i+1, j, h-l). \tag{7}$$

Note that the $X(1, i, l)$ of u for each $l \in [0, k]$ has been computed when computing the k-probability of u in $\{e_1, e_2, ..., e_i\}$. Given the integer $k-1$, we reduce the computation from $X(1, j, h)$ to $X(i+1, j, h)$ for each $h \in [0, k-1]$.

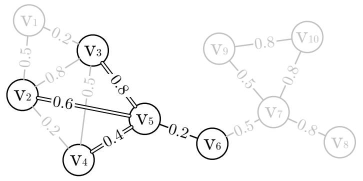

Fig. 4. The optimization of core-based ordering.

EXAMPLE 5. *An example of core-based ordering is given in Figure 4. Consider vertex* v_5*. Given an integer* $k = 2$*, we need to compute the k-probability of* v_5*, which is* $Pr[deg(v_5|\{(v_2, v_5), (v_3, v_5), (v_4, v_5), (v_6, v_5)\}) \geq 2]$*. Given that the* $Pr[deg(v_5|\{(v_2, v_5), (v_3, v_5), (v_4, v_5)\}) = i]$ *for* $i \in [0, 3]$ *has been computed in the previous iteration of* $k = 3$*, we only compute* $Pr[deg(v_5|\{(v_6, v_5)\}) = i]$ *for* $i \in [0, 2]$*.*

The Overall Algorithm. Based on the two proposed optimizations, we give the overall algorithm to construct the $UCF\text{-}Index$. The pseudocode is presented in Algorithm 7. For each vertex, we compute the core number and sort the neighbors by their core numbers in lines 1 and 2. We decrease k from k_{max} to 1 in each iteration of lines 4–19. We initialize the k-probability for each vertex based on Equation 7. Then similar to Algorithm 5, we iteratively remove the vertex with the minimum k-probability from the graph and push it into a stack $\mathcal{S}$ in line 14. With this strategy, the vertex with the smallest η-threshold is at the bottom of $\mathcal{S}$, while the vertex with the largest η-threshold is at the top of $\mathcal{S}$.

Based on the stack $\mathcal{S}$, we construct the η-tree in Algorithm 8. Given that the vertex is sorted by the η-threshold in

Algorithm 7: UCF-INDEX CONSTRUCTION

```
Input: An uncertain graph $\mathcal{G}(V, E, p)$
Output: $UCF\text{-}Index$ index of $\mathcal{G}$
1  invoke Algorithm 1 to compute $core(u)$ for all $u \in V$;
2  sort the neighbors of each $u \in V$ by their core numbers;
3  $k_{max} \leftarrow \max_{u \in V} core(u)$;
4  for $k \leftarrow k_{max}$ to $1$ do
5      $\mathcal{G}'(V', E', p) \leftarrow \mathcal{G}[\{u \in V | core(u) \geq k\}]$;
6      foreach $u \in V'$ do
7          compute $k\text{-}prob(u, \mathcal{G}')$ according to Equation 7;
8      $curThres \leftarrow 0$;
9      $\mathcal{S} \leftarrow$ initialize an empty stack;
10     while $\mathcal{G}'$ is not empty do
11         $u \leftarrow \arg\min_{v \in V'} k\text{-}prob(v, \mathcal{G}')$;
12         $curThres \leftarrow \max(k\text{-}prob(u, \mathcal{G}'), curThres)$;
13         $\eta\text{-}threshold_k(u) \leftarrow curThres$;
14         $\mathcal{S}.push(u)$;
15         foreach $v \in \mathcal{N}(u, \mathcal{G}')$ do
16             remove the edge $(u, v)$ from $\mathcal{G}'$;
17             update $k\text{-}prob(v, \mathcal{G}')$;
18         $V' \leftarrow V' \setminus \{u\}$;
19     construct $\eta\text{-}tree_k$ by invoking Algorithm 8;
20 return $\eta\text{-}tree_k$ for all $1 \leq k \leq k_{max}$;
```

Algorithm 8: η-TREE CONSTRUCTION

```
Input: An uncertain graph $\mathcal{G}$, an integer $k$ and a vertex
       stack $\mathcal{S}$
Output: The $\eta\text{-}tree_k$ of $\mathcal{G}$
1  while $\mathcal{S}$ is not empty do
2      $ct \leftarrow \eta\text{-}threshold_k(\mathcal{S}.top())$;
3      $H \leftarrow \emptyset$;
4      while $\mathcal{S} \neq \emptyset \wedge \eta\text{-}threshold_k(\mathcal{S}.top()) = ct$ do
5          $H \leftarrow H \cup \{\mathcal{S}.pop()\}$;
6      foreach connected component $\mathcal{G}[C] \in \mathcal{G}[H]$ do
7          $\mathbb{X} \leftarrow$ create a tree node;
8          $\mathbb{X}.vertices \leftarrow C$;
9          foreach $v \in \mathcal{N}(\mathcal{G}[C])$ do
10             if $\eta\text{-}threshold_k(v) < ct$ then continue;
11             $\mathbb{Y} \leftarrow$get the node containing $v$;
12             $\mathbb{Z} \leftarrow$get the root of $\mathbb{Y}$;
13             if $\mathbb{Z}.\eta\text{-}threshold > \mathbb{X}.\eta\text{-}threshold$ then
14                 $\mathbb{Z}.parent \leftarrow \mathbb{X}$;
15             else merge the tree nodes $\mathbb{X}$ and $\mathbb{Z}$;
16 return $\eta\text{-}trees_k$;
```

$\mathcal{S}$. We compute the set of vertices with the largest η-threshold in lines 2–5. According to Lemma 8, any two vertices u and v belong to the same η-tree node if $\eta\text{-}threshold_k(u) = \eta\text{-}threshold_k(v)$ and (u, v) exists. Therefore, we safely create a tree node $\mathbb{X}$ for the vertex set of each connected component (lines 7–8). We connect or merge $\mathbb{X}$ to existing tree nodes in lines 9–15. In line 9, we locate the neighbors of connected component $\mathcal{G}[C]$, i.e., $\mathcal{N}(\mathcal{G}[C]) = \{v \in V | u \in C, v \notin C, (u, v) \in E\}$. Note that $\eta\text{-}threshold_k(v) \neq ct$ for each neighbor v in line 9. Otherwise, v will be contained in the connected component $\mathcal{G}[C]$. If $\eta\text{-}threshold_k(v) < ct$, that means the tree node containing v has not been created, and that vertex is skipped in line 10. We locate the tree node $\mathbb{Y}$ containing v in line 11, and the root node $\mathbb{Z}$ of $\mathbb{Y}$ in line 12. We assign the parent of $\mathbb{Z}$ as $\mathbb{X}$ in line 14 if $\mathbb{Z}.\eta\text{-}threshold$ is larger. Otherwise, we have $\mathbb{X}.\eta\text{-}threshold = \mathbb{Z}.\eta\text{-}threshold$ and merge them into one tree node (line 15) even though $\mathbb{X}$ and $\mathbb{Z}$ are not directly connected. We analyze the running time of Algorithm 8 as follows.

LEMMA 11. *Given an uncertain graph $\mathcal{G}(V, E, p)$ and the vertex stack $\mathcal{S}$, the time complexity of Algorithm 8 is $O(|E_k|)$, where E_k is the set of edges in the subgraph induced by $\{u \in V | core(u) \geq k\}$.*

Based on Lemma 11 and Theorem 4, we summarize the time complexity of Algorithm 7 as follows.

THEOREM 7. *Given an uncertain graph $\mathcal{G}(V, E, p)$, the time complexity of Algorithm 7 is $O(k_{max}|V| \log |V| + k_{max}^2 |E|)$, where $k_{max} = \max_{u \in V} core(u)$.*

VI. EXPERIMENTS

We conducted extensive experiments to evaluate the performance of our proposed solutions. The algorithms evaluated in our experiments are summarized as follows:

- UC-Online: Algorithm 3.
- UCO-Query: Algorithm 4.
- UCF-Query: Algorithm 6.
- UCO-Construct: Algorithm 5.
- UCF-Construct: the naive algorithm to construct the $UCF\text{-}Index$, which invokes UCO-Construct to compute the η-threshold for each vertex.
- UCF-Construct-R: the algorithm to construct the $UCF\text{-}Index$, which applies the *core-based reduction* optimization from Section V.
- UCF-Construct* (Algorithm 7): the algorithm to construct the $UCF\text{-}Index$, which applies both *core-based order* and *core-based ordering* optimizations from Section V.

All algorithms were implemented in C++ and compiled using a g++ compiler at a -O3 optimization level. All the experiments were conducted on a Linux Server with an Intel Xeon 3.46GHz CPU and 96GB DDR3-RAM.

Datasets. We used eight publicly-available real-world graphs to evaluate the algorithms. The edge probabilities in the first four datasets came from real-world applications, while the probabilities in the last four datasets were randomly assigned.

Krogan is a protein-protein interaction (PPI) network [21]. The vertices represent proteins, and the edges represent the interactions between pairs of proteins. The edge probability represents the possibility of an interaction between the pair of proteins connected by this edge [22]. Flickr (https://www.flickr.com) is an online community for sharing photos. The edge probability is the Jaccard coefficient of interest groups two users share [9], [17]. DBLP (https://dblp.uni-trier.de) is

TABLE I
NETWORK STATISTICS

Datasets	$\|V\|$	$\|E\|$	deg_{max}	k_{max}
Krogan	2,559	7,031	141	15
Flickr	24,125	300,836	546	225
DBLP	684,911	2,284,991	611	114
BioMine	1,008,201	6,722,503	139,624	448
Web-Google	875,713	4,322,051	6,332	44
Cit-Patents	3,774,768	16,518,947	793	64
LiveJournal	3,997,962	34,681,189	14,815	360
Orkut	3,072,441	117,185,083	33,313	253

a computer science bibliography website. Each vertex corresponds to an author, and edges represent co-authorships. The edge probability is an exponential function based on the number of collaborations [9], [17]. BioMine (https://www.cs.helsinki.fi/group/biomine/) is a snapshot of the database of the BioMine project [23] containing biological interactions. The edge probability is based on the confidence that the interaction actually exists [9], [17].

Web-Google is a web network. Cit-Patents is a citation network. LiveJournal and Orkut are social networks. A detailed description of these four networks can be found in SNAP (http://snap.stanford.edu/index.html) with edge probabilities assigned uniformly and at random using the interval $[0, 1]$.

Detailed statistics of these datasets are summarized in Table I. The maximum degree (deg_{max}) and the maximum core number (k_{max}) are shown in the last two columns.

A. Performance of Query Processing

Our first set of experiments evaluate the performance of query processing by varying k and η. We adopt similar settings used in [9] for η, and choose 0.1, 0.3, 0.5, 0.7, and 0.9, with 0.5 as the default. We choose 5, 10, 15, 20, and 25 for k, with 15 as the default. Due to the space limitations, we only report the figures for DBLP, BioMine, LiveJournal and Orkut. The results on other datasets show the similar trends. The results for all datasets using the default parameter settings follow.

Evaluation-I: Varying k. The running time for UCF-Query, UCO-Query, and UC-Online when varying k is shown in Figure 5. UCF-Query is faster than UCO-Query for every k on all datasets, and the gap between them gradually increases as k grows. For example, in LiveJournal, UCF-Query takes about 14ms while UCO-Query takes around 435ms when $k = 5$. When k reaches 25, UCF-Query takes only 105μs (1μs = 10^{-6}s), while UCO-Query still takes approximately 36ms. Additionally, as k grows, UCF-Query shows a significant downward trend on all datasets, because the time UCF-Query takes to process is strictly dependent on the size of results, and the size of results becomes smaller as k increases. The time for UCO-Query is relatively steadier compared to UCF-Query. However, on some datasets, UC-Online shows slightly upward trends, since it takes more time to initialize and update η-degree for a large k. Overall, UCF-Query is significantly faster than UC-Online, particularly for a large k.

Evaluation-II: Varying η. Figure 6 shows the running time for UCF-Query, UCO-Query, and UC-Online when varying η. Compared to Figure 5, the changes as η increases for all algorithms are not as obvious. In fact, for some datasets,

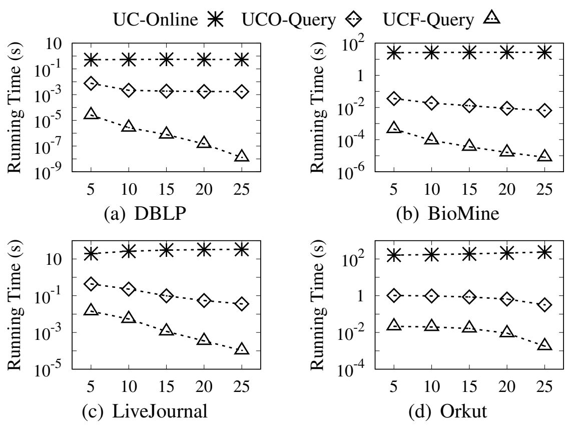

Fig. 5. Query time for different k ($\eta = 0.5$)

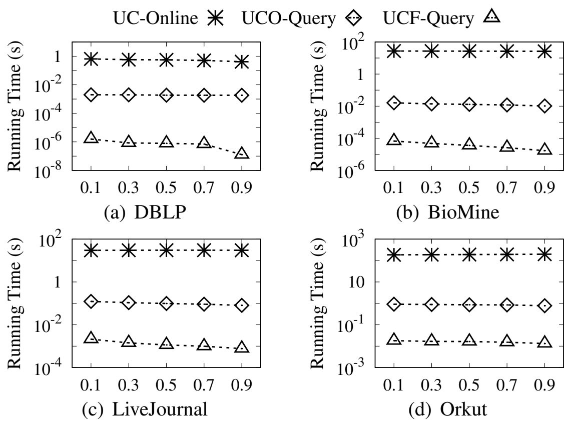

Fig. 6. Query time for different η ($k = 15$)

the trends are almost stable. For example, with BioMine, the running time for UCF-Query drops from 67μs with $\eta = 0.1$ to 17μs with $\eta = 0.9$. Meanwhile, UCO-Query drops from about 16ms to 10ms. Additionally, unlike Figure 5, the running time of UC-Online demonstrates a slight downward trend with some datasets. The dominating factor, in this case, is that the η-degree becomes smaller as η becomes larger. We can see that the running time for $UCO\text{-}Index$ and $UCF\text{-}Index$ is more sensitive to k than η. This is mainly because the size of k-core will be largely reduced when k increases, and the (k, η)-core is contained in a k-core. Overall, the UCF-Query significantly outperforms both UCO-Query and UC-Online.

Evaluation-III: Query Performance on Different Datasets. The running time for UCF-Query, UCO-Query, and UC-Online with the default parameters $k = 15$ and $\eta = 0.5$ on all datasets are shown in Figure 7. We can see that UCF-Query is not only more efficient than UCO-Query but is also several orders of magnitude faster than UC-Online on all datasets. The running time for UCF-Query on Krogan is about 0.012μs, which is the smallest value of all the results. Meanwhile, the cost for UCO-Query and UC-Online is about 8μs and 2ms respectively on the same dataset. On the Orkut dataset with over 100 million edges, UCF-Query only takes about 17ms, while UCO-Query and UC-Online takes approximately 857ms and 190s respectively.

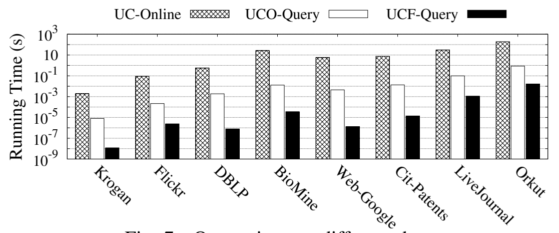

Fig. 7. Query time on different datasets

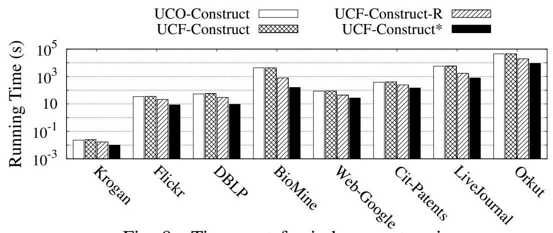

Fig. 9. Time cost for index construction

B. Performance of Index Construction

Evaluation-IV: Index Size with Different Datasets. The size of $UCF\text{-}Index$ for different datasets is reported in Figure 8 with $UCO\text{-}Index$ added as a comparison. The size of $UCF\text{-}Index$ gradually grows as the number of edges increases, and with most datasets, $UCF\text{-}Index$ needs more space due to the maintenance of the parent-children relationship in the η-tree. For example, in Cit-Patents, $UCO\text{-}Index$ takes up about 140MB, while $UCF\text{-}Index$ consumes about 160MB. However, we find that, with some datasets, the size of $UCF\text{-}Index$ is smaller than $UCO\text{-}Index$. For example, in Orkut, $UCO\text{-}Index$ takes up about 950MB, while $UCF\text{-}Index$ only needs 800MB. This is because several vertices were contracted into each tree node, and the η-threshold is only maintained for the tree node.

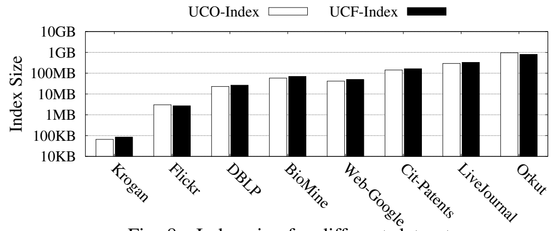

Fig. 8. Index size for different datasets

Evaluation-V: Construction Time on Different Datasets. This experiment is designed to evaluate our optimization techniques for index construction. UCF-Construct denotes the naive algorithm to construct the $UCF\text{-}Index$, with Algorithm 5 to compute the η-thresholds for all vertices and all integers of k. UCF-Construct-R denotes the *core-based reduction* technique described in Section V-C. UCF-Construct* denotes the combined *core-based reduction* and *core-based ordering* optimizations (Algorithm 7). We have also included the running time for UCO-Construct, the algorithm that constructs the $UCO\text{-}Index$, as a comparison. The results are reported in Figure 9. Compared to UCO-Construct, UCF-Construct originally requires additional time to construct the η-tree; however, our two optimizations significantly reduce this time. On the largest dataset Orkut, UCO-Construct, UCF-Construct, UCF-Construct-R, and UCF-Construct* takes approximately 12.5 hours, 12.6 hours, 5.5 hours, and 2.4 hours, respectively.

Evaluation-VI: Scalability Testing. This experiment tests the scalability of our proposed algorithms. Due to the space limitations, we have only included the results for two real-world graph datasets as representatives — BioMine and Orkut. The results using the other datasets show similar trends. For each dataset, we vary the graph size and graph density by randomly sampling nodes and edges from 20% to 100%. When sampling nodes, we derive the induced subgraph of the sampled nodes, and when sampling edges, we select the incident nodes of the edges as the vertex set. The time cost of UCF-Construct* at different percentages are reported in Figure 10, with UCO-Construct, UCF-Construct and UCF-Construct-R as comparisons.

As we can see as $|V|$ or $|E|$ increases, the processing time to construct all the indexes grows. However, the gap between UCF-Construct and UCO-Construct is not obvious, because the major cost in UCF-Construct is computing the η-thresholds. UCF-Construct* performs better than the other algorithms in all cases, and the gaps between UCF-Construct* and the other algorithms increase as the sampling ratio increases. Overall, these results imply that our optimization techniques are effective, especially with big graphs.

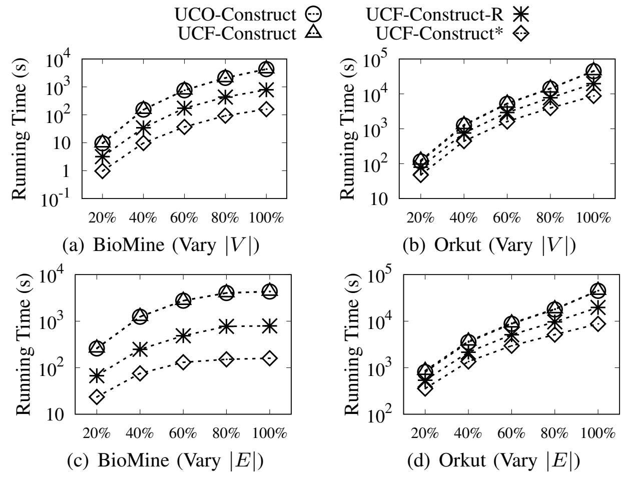

Fig. 10. Scalability of index construction

VII. Related Work

Uncertain Graphs. Many fundamental graph problems have been studied in uncertain graphs. Jin et al. [5] study the distance-constraint reachability problem on uncertain graphs. Potamias et al. [17] propose a framework to efficiently answer k-nearest neighbor queries over uncertain graphs. Gao et al. [24] study the problem of reverse k-nearest neighbor search on uncertain graphs. Zou et al. [6] investigate the problem of discovering and mining frequent subgraph patterns in uncertain graphs. Jin et al. [7] consider the problem of discovering highly reliable subgraphs of uncertain graphs. The truss decomposition of uncertain graphs is studied by [25].

K-Core Computations. k-core is first introduced by Seidman [8]. Batagelj and Zaversnik [26] propose a linear algorithm for core decomposition, which is presented in Section II. I/O efficient algorithms for core decomposition are studied in [27]–[29]. Montresor et al. [30] investigate a distributed algorithm for core decomposition. Core decomposition in random graphs is studied in [31]–[34]. Additionally, k-core is studied using weighted graphs in [35], directed graphs in [36], dynamic graphs in [37]–[39] and multi-dimensional graphs in [40]. [9] first explores the k-core model in uncertain graphs. The details of this approach are presented in Section III. A variation for the (k, η)-core, denoted by (k, θ)-core, is proposed in [41] to capture the k-core probability of each individual vertex in the uncertain graph.

VIII. Conclusion

This paper presents an index-based solution for computing all the (k, η)-cores in uncertain graphs. Our proposed index, called $UCF\text{-}Index$, maintains a tree structure for each integer k. The size of $UCF\text{-}Index$ is well-bounded by $O(m)$. Based on $UCF\text{-}Index$, queries for any input parameter k and η can be answered in optimal time. Two optimizations for $UCF\text{-}Index$ are also presented to speed up the index construction. The results of extensive performance studies demonstrate the effectiveness of this index-based approach and the efficiency of the query algorithm.

Acknowledgements. Lu Qin is supported by ARC DP160101513. Ying Zhang is supported by ARC FT170100128 and DP180103096. Lijun Chang is supported by ARC DE150100563 and DP160101513. Rong-Hua Li is supported by NSFC Grants 61772346 and Beijing Institute of Technology Research Fund Program for Young Scholars.

References

[1] C. C. Aggarwal, *Managing and Mining Uncertain Data*. Springer, 2009.

[2] E. Adar and C. Re, "Managing uncertainty in social networks." *IEEE Data Eng. Bull.*, vol. 30, no. 2, pp. 15–22, 2007.

[3] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *JASIST*, vol. 58, no. 7, pp. 1019–1031, 2007.

[4] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa, "Injecting uncertainty in graphs for identity obfuscation," *PVLDB*, vol. 5, no. 11, pp. 1376–1387, 2012.

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

[6] Z. Zou, H. Gao, and J. Li, "Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics," in *KDD*, 2010, pp. 633–642.

[7] R. Jin, L. Liu, and C. C. Aggarwal, "Discovering highly reliable subgraphs in uncertain graphs," in *KDD*, 2011, pp. 992–1000.

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

[9] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich, "Core decomposition of uncertain graphs," in *KDD*, 2014, pp. 1316–1325.

[10] W. Cui, Y. Xiao, H. Wang, and W. Wang, "Local search of communities in large graphs," in *SIGMOD*, 2014, pp. 991–1002.

[11] C. Giatsidis, F. D. Malliaros, D. M. Thilikos, and M. Vazirgiannis, "Corecluster: A degeneracy based graph clustering framework." in *AAAI*, vol. 14, 2014, pp. 44–50.

[12] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, "Large scale networks fingerprinting and visualization using the k-core decomposition," in *NIPS*, 2006, pp. 41–50.

[13] H. Zhang, H. Zhao, W. Cai, J. Liu, and W. Zhou, "Using the k-core decomposition to analyze the static structure of large-scale software systems," *The Journal of Supercomputing*, vol. 53, no. 2, pp. 352–369, 2010.

[14] G. D. Bader and C. W. Hogue, "An automated method for finding molecular complexes in large protein interaction networks," *BMC bioinformatics*, vol. 4, no. 1, p. 2, 2003.

[15] R. Andersen and K. Chellapilla, "Finding dense subgraphs with size bounds," in *International Workshop on Algorithms and Models for the Web-Graph*, 2009, pp. 25–37.

[16] J. Healy, J. Janssen, E. Milios, and W. Aiello, "Characterization of graphs using degree cores," in *International Workshop on Algorithms and Models for the Web-Graph*, 2006, pp. 137–148.

[17] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios, "K-nearest neighbors in uncertain graphs," *PVLDB*, vol. 3, no. 1-2, pp. 997–1008, 2010.

[18] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *JACM*, vol. 34, no. 3, pp. 596–615, 1987.

[19] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *PVLDB*, vol. 9, no. 12, pp. 1233–1244, 2016.

[20] A. E. Sariyüce and A. Pinar, "Fast hierarchy construction for dense subgraphs," *PVLDB*, vol. 10, no. 3, pp. 97–108, 2016.

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

[22] A. D. Fox, B. J. Hescott, A. C. Blumer, and D. K. Slonim, "Connectedness of ppi network neighborhoods identifies regulatory hub proteins," *Bioinformatics*, vol. 27, no. 8, pp. 1135–1142, 2011.

[23] L. Eronen and H. Toivonen, "Biomine: predicting links between biological entities using network models of heterogeneous databases," *BMC bioinformatics*, vol. 13, no. 1, p. 119, 2012.

[24] Y. Gao, X. Miao, G. Chen, B. Zheng, D. Cai, and H. Cui, "On efficiently finding reverse k-nearest neighbors over uncertain graphs," *The VLDB Journal*, vol. 26, no. 4, pp. 467–492, 2017.

[25] X. Huang, W. Lu, and L. V. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *SIGMOD*, 2016, pp. 77–90.

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

[27] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. X. Yu, "I/o efficient core graph decomposition at web scale," in *ICDE*, 2016, pp. 133–144.

[28] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo, "K-core decomposition of large networks on a single pc," *PVLDB*, vol. 9, no. 1, pp. 13–23, 2015.

[29] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *ICDE*, 2011, pp. 51–62.

[30] A. Montresor, F. De Pellegrini, and D. Miorandi, "Distributed k-core decomposition," *TPDS*, vol. 24, no. 2, pp. 288–300, 2013.

[31] S. Janson and M. J. Luczak, "A simple solution to the k-core problem," *Random Structures & Algorithms*, vol. 30, no. 1-2, pp. 50–62, 2007.

[32] M. Molloy, "Cores in random hypergraphs and boolean formulas," *Random Structures & Algorithms*, vol. 27, no. 1, pp. 124–135, 2005.

[33] T. Łuczak, "Size and connectivity of the k-core of a random graph," *Discrete Mathematics*, vol. 91, no. 1, pp. 61–68, 1991.

[34] B. Pittel, J. Spencer, and N. Wormald, "Sudden emergence of a giantk-core in a random graph," *Journal of Combinatorial Theory, Series B*, vol. 67, no. 1, pp. 111–151, 1996.

[35] A. Garas, F. Schweitzer, and S. Havlin, "A k-shell decomposition method for weighted networks," *New Journal of Physics*, vol. 14, no. 8, p. 083030, 2012.

[36] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis, "D-cores: Measuring collaboration of directed graphs based on degeneracy," in *ICDM*, 2011, pp. 201–210.

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

[38] R.-H. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *TKDE*, vol. 26, no. 10, pp. 2453–2465, 2014.

[39] Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin, "A fast order-based approach for core maintenance," in *ICDE*, 2017, pp. 337–348.

[40] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "When engagement meets similarity: efficient (k, r)-core computation on social networks," *PVLDB*, vol. 10, no. 10, pp. 998–1009, 2017.

[41] Y. Peng, Y. Zhang, W. Zhang, X. Lin, and L. Qin, "Efficient probabilistic k-core computation on uncertain graphs," in *ICDE*, 2018, pp. 1192–1203.