Computing K-Cores in Large Uncertain Graphs: An Index-based Optimal Approach

Journal:	*Transactions on Knowledge and Data Engineering*
Manuscript ID	TKDESI-2020-04-0417
Manuscript Type:	ICDE 2019
Keywords:	uncertain graph, k-core

Computing K-Cores in Large Uncertain Graphs: An Index-based Optimal Approach

Dong Wen, Bohua Yang, Lu Qin, Ying Zhang, Lijun Chang, and Rong-Hua Li

Abstract—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 such as 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. However, the results heavily depend on the two input parameters k and η. The settings for these parameters are unique to the specific graph structure and the user's subjective requirements. In addition, computing and updating the η-degree for each vertex is the most costly component in the algorithm, and the cost is high. To overcome these drawbacks, we propose an index-based solution for computing (k, η)-cores. The size of the index is well bounded by $O(m)$, where m is the number of edges in the graph. Based on the index, queries for any k and η can be answered in optimal time. We propose an algorithm for index construction with several different optimizations. We also propose a new algorithm for index construction in external memory. We conduct extensive experiments on eight real-world datasets to practically evaluate the performance of all proposed algorithms.

Index Terms—K-Core, Uncertain Graphs, Semi-External Algorithms.

1 Introduction

Graphs have been widely used to model sophisticated relationships between different entities due to their strong representative properties. 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].

k-Cores in Deterministic Graphs. Given a deterministic graph, a k-core is a maximal connected subgraph in which 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].

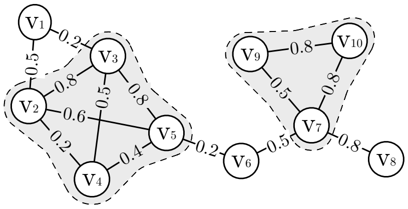


Fig. 1: The (k, η)-cores of $\mathcal{G}$ for $k = 2$ and $\eta = 0.3$

$(\mathbf{k}, \eta)$**-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 in which 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. In addition, [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

- *Dong Wen is with the University of Technology Sydney, Australia. Email: dong.wen@uts.edu.au.*
- *Bohua Yang is with the University of Technology Sydney, Australia. Email: bohua.yang@student.uts.edu.au.*
- *Lu Qin is with the University of Technology Sydney, Australia. Email: lu.qin@uts.edu.au.*
- *Ying Zhang is with the University of Technology Sydney, Australia. Email: ying.zhang@uts.edu.au.*
- *Lijun Chang is with the University of Sydney, Australia. Email: lijun.chang@sydney.edu.au.*
- *Rong-Hua Li is with the Beijing Institute of Technology, China. Email: lironghuascut@gmail.com.*

of each vertex set is a (k, η)-core.

The Online Approach. In [9], (k, η)-cores are derived using an η-core decomposition algorithm. 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, we can iteratively remove all the vertices with η-degrees of less than k and derive a set of resulting vertices. The final result can then be generated by performing a connected component detection procedure on the 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 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 the 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.

There is still room to reduce the amount of time it takes for query processing based on the $UCO\text{-}Index$. Hence, we further 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.

I/O Efficient Query Processing. We also study an index-based solution when a graph cannot be entirely loaded in memory. We adopt the semi-external setting [18], [19], which allows $O(n)$ memory usage. The structure of $UCF\text{-}Index$ can be naturally stored in external memory. We use the same strategy of in-memory query processing to derive results from the external index and achieve the optimal I/O complexity $O(|C|/B)$. To construct $UCEF\text{-}Index$ in external memory, straightforwardly using the strategy of in-memory index construction incurs significant I/O cost due to frequent random accesses of external memory. We propose a new framework for external index construction and derive the index by sequentially accessing the external graph in several iterations. Several optimizations are also given to further improve the efficiency.

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 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.* We give two optimizations, *core-based reduction* and *core-based ordering*, to improve the efficiency of index construction.
- *Index Construction in External Memory.* We propose a new framework for index construction in external memory. Several optimizations are given to reduce I/O cost and further improve the efficiency.
- *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 2 provides some preliminary concepts and formally defines the problem. In Section 3, we review an existing solution and explain the online approach in detail. Section 4 describes the basic structure of the index. Section 5 presents the optimized forest-based index structure. Section 6 proposes the algorithm for index construction in external memory. Section 7 practically evaluates the proposed algorithms. Section 8 summarizes related works, and Section 9 concludes the paper.

Extensions Beyond the Conference Version. The conference version of this paper can be found in [20]. The current version mainly adds a new Section 6, which proposes a new algorithm for index construction in external memory. Some corresponding experiments are added in Section 7. Due to the space limitation, we omit straightforward proofs for lemmas and theorems. We also remove several parts, which do not affect the understanding of the extension. 1) We remove Section 4.3, which proposes an algorithm for $UCO\text{-}Index$ construction. 2) We remove Section 5.3, which proposes an algorithm and optimizations for $UCF\text{-}Index$ construction. 3) We remove some experimental results, including the query time for different parameters, the index size, the running time of in-memory algorithms for index construction, and the scalability of the index construction.

2 Preliminaries

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 graph G, computing core numbers for all vertices is called core decomposition, which can be done by iteratively removing the vertex with the minimum degree. The time complexity is $O(m)$. [21], [22]

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 $[0,1]$. 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 $\mathcal{G}$ 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.

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.*

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

Let C be the vertex set such that the induced subgraph $\mathcal{G}[C]$ of C is a (k,η)-core. The problem aims 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 $\{\{v_2,v_3,v_4,v_5\},\{v_7,v_9,v_{10}\}\}$.

3 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.

3.1 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 and the η-core number 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]*

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.*

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 iff η-$core(u)\geq k$.*

The problem of computing the η-core numbers for all vertices is called η-core decomposition. The solution proposed in [9] is provided in Algorithm 1. Algorithm 1 shares

Algorithm 1: 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 similar idea with deterministic core decomposition, 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:

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

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

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

Algorithm 2: (k, η)-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 $\mathcal{R}$;
```

Based on the equation above, we compute $Pr[deg(u|E(u)\backslash\{e\}) = i]$ for each $i \in [1, \eta$-$deg(u)]$ in constant time, given that $Pr[deg(u|E(u)\backslash\{e\}) = 0] = \frac{1}{1-p_e} Pr[deg(u) = 0]$.

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 $O(\eta$-$deg(u))$. [9]*

LEMMA 4. *Given an uncertain graph $\mathcal{G}(V, E, p)$, the time complexity of Algorithm 1 is $O(\sum_{u \in V} \eta$-$deg(u) \cdot Deg(u))$. [9]*

3.2 An Online Approach to Compute (k, η)-Cores

Based on several concepts introduced in the previous section, we turn to the online approach for computing all (k, η)-cores. Our approach is similar to Algorithm 1, which iteratively removes the vertex that does not belong to the result set. Before presenting the algorithm, we make the following observation for 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 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 2.

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 2 is $O(\sum_{u \in V} \eta$-$deg(u) \cdot Deg(u))$, which is the same as that of Algorithm 1.

Drawbacks of the Online Approach. Even though Algorithm 2 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

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 $UCO\text{-}Index$ of $\mathcal{G}$

major cost in Algorithm 2 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.

4 An Index-based Approach

4.1 The Index Structure

In this section, we introduce an index structure, called the uncertain core η-orders index ($UCO\text{-}Index$). 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. We provide the range of integer k as follows.

Observation 2. *Given an uncertain graph $\mathcal{G}$, we only need to consider the parameter $1 \leq k \leq k_{max}$, 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 resulting vertices by only considering the parameter k, which is $\{u \in V | core(u) \geq k\}$. 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 $0 \leq \eta \leq \eta' \leq 1$, 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 $\eta\text{-}threshold_k(u)$, is the largest η such that a (k, η)-core containing u exists.*

Based on Observation 3 and Definition 6, we have $\eta\text{-}threshold_k(u) = 0$ 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.

Algorithm 3: 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 $\mathcal{R}$;
```

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 $\eta\text{-}threshold_k(u) \geq \eta$.*

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 $\eta\text{-}order(u)$, is a probability order such that (i) the i-th value in $\eta\text{-}order(u)$ is $\eta\text{-}threshold_i(u)$, and (ii) the length of $\eta\text{-}order(u)$ is $core(u)$.*

Example 2. *The η-orders for all vertices in the uncertain graph $\mathcal{G}$ in Figure 1 are given in Figure 2. Consider the vertex v_4. Given $k = 2$, we have $\eta\text{-}threshold_2(v_4) = 0.3$. 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 $\eta\text{-}threshold_k(u)$. We save the η-orders for all vertices as our $UCO\text{-}Index$. The size of the $UCO\text{-}Index$ is well-bounded.

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

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

4.2 Query Processing

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

Lemma 7. *Given a set of vertices C in $\mathcal{G}$, the induced subgraph $\mathcal{G}[C]$ is a (k, η)-core iff (i) $\forall u \in C, \eta\text{-}threshold_k(u) \geq \eta$; (ii) $\mathcal{G}[C]$ is connected; and (iii) C is maximal.*

Based on the above lemma, we present the pseudocode for query processing in Algorithm 3. It first identifies all vertices whose η-threshold is not less than η in line 1. 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 3 can be guaranteed by Lemma 7.

Theorem 2. *Given an uncertain graph $\mathcal{G}(V, E, p)$, an integer k and a probability threshold η, the time complexity of Algorithm 3 is $O(|V| + \sum_{u \in C} Deg(u))$, where C is the set of all result vertices, i.e., $C = \{u \in V | \eta\text{-}threshold_k(u) \geq \eta\}$.*

4.3 Index Construction

We give the following definition for 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 $Pr[deg(u, \mathcal{G}) \geq k]$.*

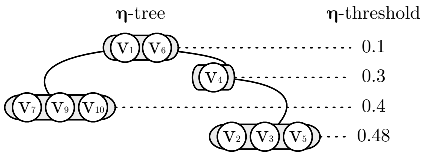


Fig. 3: The η-tree of $\mathcal{G}$ for $k = 2$

For each integer k, the η-thresholds for vertices in k-core can be derived by iteratively removing the vertex with the minimum k-probability. The detailed algorithm for $UCO\text{-}Index$ construction can be found in [20].

5 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 in query processing, and this may be hard to tolerate in big graphs. To address this issue, we propose a forest-based index structure, namely <u>u</u>ncertain <u>c</u>ore η-<u>f</u>orest <u>index</u> ($UCF\text{-}Index$). Based on the $UCF\text{-}Index$, we compute the result set in optimal time.

The index structure is introduced in Section 5.1. We provide the query processing algorithm in Section 5.2. We also propose an algorithm to construct the $UCF\text{-}Index$ and two optimizations to improve algorithmic efficiency. The details can be found in the conference version [20].

5.1 Forest-based Index Structure

According to Lemma 7, the key to query processing is computing all vertices of u such that $\eta\text{-}threshold_k(u) \geq \eta$. This costs $O(|V|)$ time in Algorithm 3. 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 the $UCF\text{-}Index$, which organizes the vertices and their η-thresholds into a tree structure, for each integer k. The tree is built based on Lemma 5. Vertices with smaller η-thresholds are on the upper side of the tree, and vertices with 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., $\mathbb{Y} = \arg\max_{\mathbb{Y}\in N(\mathcal{G}_{\mathbb{X}})} \mathbb{Y}.\eta\text{-}threshold$.*

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

```
Algorithm 4: 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 $\mathcal{G}$
  $\mathcal{T} \leftarrow$ the set of all tree nodes in $\eta\text{-}tree_k$;
  $\mathcal{S} \leftarrow$ initialize an empty stack;
  while $\mathcal{T}$ is not empty do
      $\mathbb{X} \leftarrow \arg\max_{\mathbb{X}\in\mathcal{T}} \mathbb{X}.\eta\text{-}threshold$;
      if $\mathbb{X}.\eta\text{-}threshold \geq \eta$ then $\mathcal{S}.push(\mathbb{X})$;
      else break;
      $\mathcal{T} \leftarrow \mathcal{T} \setminus \{\mathbb{X}\}$;
  $\mathcal{R} \leftarrow \emptyset$;
  while $\mathcal{S}$ is not empty do
      $\mathbb{X} \leftarrow \mathcal{S}.pop()$;
      if $\mathbb{X}$ is visited then continue;
      $C \leftarrow \emptyset$;
      $\mathcal{Q} \leftarrow$ initialize an empty queue;
      $\mathcal{Q}.insert(\mathbb{X})$;
      while $\mathcal{Q}$ is not empty do
          $\mathbb{Y} \leftarrow \mathcal{Q}.pop()$;
          mark $\mathbb{Y}$ as visited;
          $C \leftarrow C \cup \mathbb{Y}.vertices$;
          foreach $\mathbb{Z} \in \mathbb{Y}.children$ do $\mathcal{Q}.insert(\mathbb{Z})$;
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{C\}$;
  return $\mathcal{R}$;
```

5.2 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 η-$tree_k$ such that (i) $\mathbb{R}.\eta$-$threshold \geq \eta$; and (ii) there does not exist a parent $\mathbb{R}'$ of $\mathbb{R}$ such that $\mathbb{R}'.\eta$-$threshold \geq \eta$. The induced subgraph of all vertices in the subtree rooted by $\mathbb{R}$ is a (k, η)-core.*

According to Lemma 10, we process queries by collecting all tree nodes in the subtree rooted by the tree node $\mathbb{R}$. Following this idea, we provide the pseudocode for query processing in Algorithm 4.

We first collect all resulting tree nodes in lines 1–7. We use constant time to derive the tree node with the largest η-threshold in line 4, if tree nodes are sorted in a non-increasing order of their η-thresholds. The order can be precomputed in the index construction.

We iteratively process each tree node in the stack in lines 9–20. Once an unvisited tree node is found 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-$Index$. 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 $\{v_7, v_9, v_{10}\}$.*

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

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.

5.3 Optimizations for Index Construction

The algorithm to construct UCF-$Index$ is called UCF-Construct*. For each integer $1 \leq k \leq k_{max}$, UCF-Construct* contains two phases. The task of the first phase is to compute η-threshold for each vertex, which is the same as that of UCO-$Index$ construction. We further propose two optimizations in [20], called *core-based ordering* and *core-based reduction*, to speed up the first phase. The second phase constructs the η-tree. Given an integer k, we process vertices in non-increasing order of their η-thresholds, and the running time of η-tree construction can be bounded by $O(E_k)$ where E_k is the set of edges in the induced subgraph of k-core. The final pseudocode of UCF-Construct* can be found in [20].

6 Index Construction in External Memory

In this section, we discuss the (k, η)-core computation when graphs cannot be entirely stored in main memory. Assume that graph is stored in a CSR format in external memory. We adopt the *semi-external* setting, which allows $O(n)$ memory usage. This assumption is reasonable in practice, and it has been widely adopted in massive graph analysis [18], [19]. We design an index-based solution for I/O efficient (k, η)-core computation.

We first introduce the data structure to store UCF-$Index$ in external memory and give the corresponding query processing algorithm in Section 6.1. The following subsections propose the algorithm for index construction in external memory. Section 6.2 proposes a new strategy to locally compute η-thresholds, and Section 6.3 presents the corresponding algorithm for index construction. Section 6.4 proposes several optimizations to further reduce I/Os and improve efficiency.

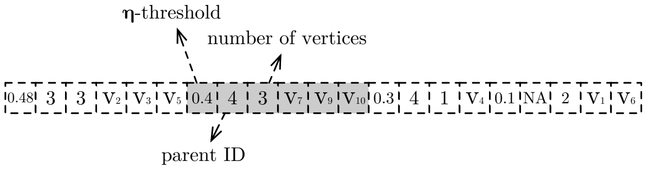


Fig. 4: The $UCEF$-$Index$ of $\mathcal{G}$ for $k = 2$

6.1 UCF-Index in External Memory

We can naturally extend the structure of UCF-$Index$ for the external memory setting, which is called $UCEF$-$Index$. Specifically, for each integer k, all tree nodes are arranged in a non-increasing order of their η-thresholds. For each tree node, we store the following three elements, (1) the node's η-threshold, (2) the node's parent ID, and (3) the containing vertices. Note that the nodes' IDs are assigned in the index construction by the order they arranged in the hard disk. Consequently, we can derive the current vertex ID in query processing accordingly and avoid the ID storage in the index. For the containing vertices of each node, we store an integer in the front to indicate the number of vertices. We give an example of the index as follows.

EXAMPLE 5. *We show the $UCEF$-$Index$ of $\mathcal{G}$ (Figure 1) for $k = 2$ in Figure 4. The corresponding tree structure is in Figure 3. We mark the fragment of the tree node containing $\{v_7, v_9, v_{10}\}$ by gray. The node is the second one in the sequence, and its implicit ID is 2. The parent ID is 4, which is the last tree node.*

I/O Efficient Query Processing. Based on $UCEF$-$Index$, we can use a similar idea as Algorithm 4 to answer (k, η)-core queries. We call the query processing algorithm for the external index UCEF-Query. Given an integer k and a probability threshold η, we sequentially scan the $UCEF$-$Index$ for k. Since tree nodes are arranged in a non-increasing order of η-thresholds, each scanned node naturally satisfies line 4 of Algorithm 4. After loading all nodes with η-threshold not less than η, we derive the tree structure based on the parent ID of each node. The tree size is bounded by $O(n)$, and the result can be computed using lines 8 – 21 of Algorithm 4. The I/O cost of UCEF-Query is still optimal.

THEOREM 5. *Given an integer k and a probability threshold η, the I/O complexity of* UCEF-Query *is $O(|C|/B)$, where $C = \{u \in V | \eta$-$threshold_k(u) \geq \eta\}$, and B is the block size.*

6.2 Local η-Threshold Computation

We can naively perform an in-memory algorithm, e.g., UCF-Construct* [20], to construct $UCEF$-$Index$. Specifically, for each integer k, we always first process the vertex with the smallest k-probability. Under the $O(n)$ memory limitation, we load neighbors of each vertex from external memory for computing k-probability and release the memory for loading neighbors of the next vertex. However, this strategy incurs significant I/O cost due to frequent random access of the external memory.

To improve the efficiency, we propose a new framework, called UCEF-Construct, tailored for the external memory setting. The framework releases the order limitation and computes the η-threshold of each vertex only using the neighbors' η-thresholds. Given an integer k and a probability value p, let $N_k^p(u)$ be the neighbors of u whose η-threshold is at least p in the (k,η)-core, i.e., $N_k^p(u) = \{v \in N(u) | \eta\text{-}threshold_k(v) \geq p\}$. The key theorem to support UCEF-Construct is given as follows.

THEOREM 6. *Given an integer k, a probability value p, and a vertex u with $core(u) \geq k$, we have $p = \eta\text{-}threshold_k(u)$ iff*

1) *the k-probability of u in $N_k^p(u)$ is not smaller than p, i.e., $k\text{-}prob(u, \mathcal{G}[\{u\} \cup N_k^p(u)]) \geq p$; and*
2) *there does not exists a probability value p' s.t. $p' > p$ and p' satisfies condition (1).*

EXAMPLE 6. *We give an example to explain Theorem 6. We consider the vertex v_4 in Figure 1. Given $k = 2$, the η-thresholds for neighbors of v_4 can be found in Figure 2. Assume that $p = 0.48$. We have $N_2^{0.48}(v_4) = \{v_2, v_3, v_5\}$. The probability that v_4 connects at least 2 neighbors in $N_2^{0.48}(v_4)$ is 0.3. Since $0.3 < 0.48$, p is not the η-threshold of v_4, which does not satisfy the condition 1 in Theorem 6. In this case, the correct η-threshold for v_4 should be 0.3.*

Based on Theorem 6, the procedure local_thres in Algorithm 5 shows the pseudocode for locally computing the η-threshold of u. Given a vertex u and a set of vertices N, we use N to represent the induced subgraph of u and N, i.e., $\mathcal{G}[\{u\} \cup N]$, for ease of presentation when context is clear. For example, in line 7 of local_thres, $k\text{-}prob(u, N_i)$ is short for $k\text{-}prob(u, \mathcal{G}[\{u\} \cup N_i])$.

LEMMA 11. *Given an integer k, a query vertex u and the η-thresholds for its neighbors N,* local_thres *correctly computes the η-threshold for u.*

PROOF. *According to the condition 1 of Theorem 6, all η-thresholds for the selected neighbors in computing k-probability are not smaller than p, and we have $\hat{t}(u) \leq p_1$. The computed k-probability is not smaller than p, and we have $\hat{t}(u) \leq p_2$. By setting $\hat{t}(u) = \min(p_1, p_2)$, we guarantee $\hat{t}(u)$ in each iteration always satisfies the condition 1 of Theorem 6. Note that we start from $i = k$ since p_2 would be 0 if $i < k$.*

The condition 2 in Theorem 6 actually guarantees that p is the largest possible value satisfying the condition 1. local_thres *computes such p in a bottom-up strategy. Specifically, we can find that the variable p_1 is monotonic decreasing, and p_2 is monotonic increasing. As a result, $\min(p_1, p_2)$ first monotonically increases to a peak and then monotonically decreases. Line 8 of* local_thres *checks whether the current $\hat{t}(u)$ reaches the peak. Once $\min(p_1, p_2)$ stops increasing, the procedure breaks the iteration and derives the correct $\hat{t}(u)$.*

In line 7 of local_thres, we do not need to compute $k\text{-}prob(u, N_i)$ from scratch in each iteration. Based on Equation 3, we maintain $Pr[deg(u, N_i) = j]$ for $0 \leq j \leq k-1$ in each iteration. The first iteration takes $O(k^2)$ time. In the iteration i with $i > 1$, assume $N_i = N_{i-1} \cup \{v\}$. We have $Pr[deg(u, N_i) = j] = (1 - p_{(u,v)}) \cdot Pr[deg(u, N_{i-1}) = j] + p_{(u,v)} * Pr[deg(u, N_{i-1}) = j-1]$. Therefore, the total time complexity of line 7 is $O(k \cdot Deg(u))$, which is the same as that of computing k-probability of u in $\mathcal{G}$.

Algorithm 5: UCEF-INDEX CONSTRUCTION

```
   Input: An uncertain graph $\mathcal{G}(V, E, p)$
   Output: $UCEF\text{-}Index$ of $\mathcal{G}$
1  compute $core(u)$ for all $u \in V$; [18]
2  for $k \leftarrow k_{max}$ to 1 do
3      foreach $u \in V : core(u) \geq k$ do
4          $\hat{t}(u) = 1$, and mark $u$ as active;
5      while active vertex exists do
6          foreach $u \in V : u$ is active do
7              mark $u$ as inactive;
8              load $N(u)$ from the disk;
9              $\hat{t}_{old} \leftarrow \hat{t}(u)$;
10             $\hat{t}(u) \leftarrow$ local_thres$(u, N(u), k)$;
11             if $\hat{t}(u) = \hat{t}_{old}$ then continue;
12             foreach $v \in N(u) : \hat{t}(u) < \hat{t}(v) \leq \hat{t}_{old}$ do
13                 mark $v$ as active;
       // construct $\eta$-tree for $k$
14     sort vertices of $k$-core in non-increasing order of $\hat{t}$
         values, and write their neighbors accordingly in
         external memory;
15     invoke Algorithm 8 in [20] to construct $\eta$-tree;

1  Procedure local_thres$(u, N, k)$ :
2      sort vertices in $N$ in non-increasing order of $\hat{t}$
         values;
3      $\hat{t}(u) \leftarrow 0$;
4      for $k \leq i \leq |N|$ do
5          $p_1 \leftarrow$ the $i$-th $\hat{t}$ value in $N$;
6          $N_i \leftarrow$ the first $i$ vertices in $N$;
7          $p_2 \leftarrow$ compute $k\text{-}prob(u, N_i)$;
8          if $\hat{t}(u) \geq \min(p_1, p_2)$ then break;
9          $\hat{t}(u) \leftarrow \min(p_1, p_2)$;
10     return $\hat{t}(u)$;
```

6.3 The Algorithm

Based on Theorem 6, we give the pseudocode of UCEF-Construct in Algorithm 5. To derive core numbers of all vertices under the semi-external setting, we adopt the algorithm SemiCore* in [18]. SemiCore* uses $O(n)$ memory space and computes core numbers in several iterations of sequentially reading the external graph. For each integer k, we compute η-thresholds in lines 3–13.

For each vertex u, we maintain a core number $core(u)$, a probability value $\hat{t}(u)$ as an estimation of $\eta\text{-}threshold_k(u)$, and an indicator to represent whether the vertex is active. Lines 3–4 initialize $\hat{t}(u)$ and mark all vertices in the k-core as active. Line 10 computes a new $\hat{t}(u)$ according to the current $\hat{t}$ values of neighbors. Lines 12–13 mark possibly influenced neighbors as active. The η-threshold computation terminates if there is no active vertex. For each vertex u, $\hat{t}(u)$ is always an upper bound of $\eta\text{-}threshold(u)$, which never increases and converges to $\eta\text{-}threshold(u)$ finally. The proof of the algorithmic correctness is similar to that of [18], [23], and we omit the details here.

Note that the condition in line 12 significantly reduces unnecessary active vertices. We explain the rationale as follows. For a vertex u, the computation of $\hat{t}(u)$ of each vertex is based on the neighbors in $N_k^{\hat{t}(u)}(u)$ according to Theorem 6. $\hat{t}(u)$ requires to be updated if $N_k^{\hat{t}(u)}(u)$ changes. Since $\hat{t}(u)$ for any vertex u never increases, we mark a vertex v as active if its neighbor u leaves $N_k^{\hat{t}(v)}(v)$. In line 12, $\hat{t}(v) \leq \hat{t}_{old}$ means that u is in $N_k^{\hat{t}(v)}(v)$ when computing $\hat{t}(v)$, and $\hat{t}(u) < \hat{t}(v)$ means that u leaves $N_k^{\hat{t}(v)}(v)$.

THEOREM 7. *The I/O complexity of computing η-thresholds of all vertices for every possible k is $O(\frac{l \cdot m}{B})$, where l is the total number of iterations, and B is the block size.*

η-Tree Construction. We construct η-trees in lines 14–15. In the in-memory algorithm to construct η-tree (Algorithm 8 in [20]), vertices are processed in a non-increasing order of their η-thresholds. We adopt the same idea here and create a temporary file to arrange neighbors of vertices in k-core in such order. As a result, we can construct the tree by sequentially reading the required vertex neighbors from external memory in only one iteration. The temporary file can be constructed using a traditional external-sorting algorithm with the I/O complexity $O(\frac{m}{B} \log_{\frac{M}{B}} \frac{m}{B})$, where M is the memory size [24]. The semi-external setting allows us to use $O(n)$ memory. Since $n^2 \geq m$, $\log_{\frac{n}{B}} \frac{m}{B}$ can be regarded as a constant, and the I/O complexity of external sorting is reduced to $O(m/B)$. For an integer k, the I/O cost of η-tree construction is bounded by $O(m/B)$, and the overall I/O cost of Algorithm 5 is still $O(\frac{l \cdot m}{B})$.

6.4 Further Optimizations

It is obvious to see that the dominating cost in Algorithm 5 is incurred by computing η-thresholds. We propose several optimizations to reduce the I/O cost of this step and further improve the efficiency of Algorithm 5.

6.4.1 Reducing η-Threshold Estimations

Recall that for each vertex u in Algorithm 5, we initialize $\hat{t}(u)$ by 1, which is a very loose upper bound of the η-threshold for u. We reduce unnecessary η-threshold computations by setting a relatively tighter upper bound. Given an integer $1 < k \leq k_{max}$ and an arbitrary vertex u, we can naturally use $\eta\text{-}threshold_{k-1}(u)$ as an upper bound of $\eta\text{-}threshold_k(u)$ based on Lemma 5. To implement this idea, we adopt a bottom-up strategy which computes η-thresholds from $k = 1$ to $k = k_{max}$. In this way, we set $\hat{t}(u) = \eta\text{-}threshold_{k-1}(u)$ in line 4 of Algorithm 5.

6.4.2 Partial Neighbor Loading

According to Theorem 6, the η-threshold computation only requires the neighbors whose core numbers are not smaller than k. We reduce the I/O cost of the η-threshold computation by only loading necessary neighbors of each vertex. We implement this idea by sorting neighbors of each vertex after computing core numbers in line 1 of Algorithm 5. Specifically, we load neighbors of each vertex from the external memory. We sort the neighbors in non-increasing order of their core numbers and write back to the external memory. The I/O complexity of this step is $O(m/B)$. Given the sorted neighbors of each vertex, in line 8 of Algorithm 5, we sequentially read neighbors one by one from external memory until a neighbor is found with the core number smaller than k. This step reduces the I/O cost of line 8 from $O(|N(u)|/B)$ to $O(|N_k(u)|/B)$.

6.4.3 Vertex Ordering

Compared with the in-memory index construction, Algorithm 5 may perform line 10 several times for each vertex u until $\hat{t}(u)$ converges to $\eta\text{-}threshold_k(u)$ even using a tighter upper bound in Section 6.4.1. Intuitively, $\hat{t}(u)$ will be close to $\eta\text{-}threshold_k(u)$ if $\hat{t}$ values for all neighbors are close to their η-thresholds. A special case is shown as follows.

LEMMA 12. *Given an integer k, assume that all vertices are sorted in a non-decreasing order of their η-thresholds for k, i.e., $\forall u, v \in V, \eta\text{-}threshold_k(u) \leq \eta\text{-}threshold_k(v)$ if $u < v$. Line 10 of Algorithm 5 performs only once in computing $\eta\text{-}threshold_k(u)$ for every vertex u.*

In the case of the lemma, vertices in line 6 are processed in non-decreasing order of η-thresholds. $\hat{t}(u)$ derived in line 10 is exactly $\eta\text{-}threshold_k(u)$ according to Theorem 6. Based on Lemma 12, we aim to improve the efficiency of Algorithm 5 by postponing the η-threshold computations of some vertices if their η-thresholds are relatively large with a high probability. To implement this idea, we sort vertices in external memory after line 1 of Algorithm 5 using several heuristic rules. We first arrange vertices in a non-decreasing order of their core numbers. We break a tie by considering the probability that the vertex u has at least one neighbor. Specifically, given two vertices u and v with $core(u) = core(v)$, we assign u to the front if $Pr[deg(u, \mathcal{G}) \geq 1] < Pr[deg(v, \mathcal{G}) \geq 1]$. The computation of $Pr[deg(u, \mathcal{G}) \geq 1]$ for all vertices u takes $O(m/B)$ I/Os since neighbors of each vertex are required.

We perform an external sorting algorithm to rearrange the graph structure according to the new vertex order, which takes $O(m/B)$ I/Os, similar to the discussion in Section 6.3. Note that the neighbor ordering discussed in Section 6.4.2 can be done as a byproduct in the vertex ordering.

7 EXPERIMENTS

We conducted extensive experiments to evaluate the performance of our proposed solutions. 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, 96GB DDR3-RAM, and a 2TB 7200 RPM SATA III Hard Drive.

Datasets. We used eight publicly-available real-world graphs to evaluate the algorithms. The edge probabilities in the first four datasets come from real-world applications. Krogan is a protein-protein interaction (PPI) network [25]. The edge probability represents the possibility of an interaction between the pair of proteins connected by this edge [26]. Flickr is an online community for sharing photos. The edge probability is the Jaccard coefficient of interest groups two users share [9], [17]. DBLP is a computer science bibliography website. The edge probability is an exponential function based on the number of collaborations [9], [17]. BioMine is a snapshot of the database of the BioMine

TABLE 1: Network statistics

Datasets	$\lvert V\rvert$	$\lvert E\rvert$	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

project [27] containing biological interactions. The edge probability is based on the confidence that the interaction actually exists [9], [17]. The last four datasets are from SNAP (http://snap.stanford.edu/index.html). Web-Google is a web network. Cit-Patents is a citation network. LiveJournal and Orkut are social networks. Edge probabilities are assigned at random between 0 and 1. Detailed statistics of these datasets are summarized in Table 1. The maximum degree (deg_{max}) and the maximum core number (k_{max}) are shown in the last two columns.

Due to space limitation, we omit the evaluations for the algorithms of in-memory index construction. Interested readers can find the details in [20].

7.1 Performance of Query Processing

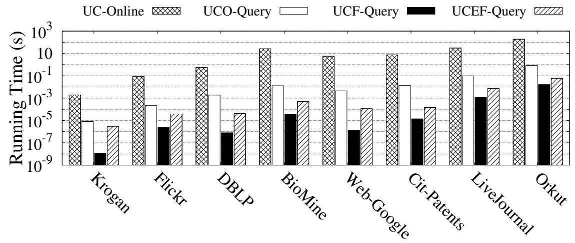


Fig. 5: Query time on different datasets

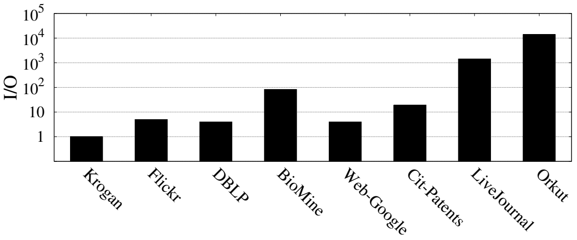


Fig. 6: I/O cost of external query processing

Evaluation-I: Query Performance on Different Datasets. The running time of UC-Online (Algorithm 2), UCO-Query, and UCF-Query with the default parameters $k = 15$ and $\eta = 0.5$ on all datasets are shown in Figure 5. 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 of UCF-Query on Krogan is about 0.012μs, which is the smallest value in all results. Meanwhile, the running times of UCO-Query and UC-Online are 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. We also evaluate the performance of query processing by varying k and η. The details can be found in the conference version [20].

Regarding the external memory setting, the running time of UCEF-Query is shown in the last bar for each dataset in Figure 5, and the corresponding number of I/Os is shown in Figure 6. The only difference between UCEF-Query and UCF-Query is that UCEF-Query loads the index from external memory.

7.2 Performance of External Index Construction

We use UCF-Construct*-EM to represent the naive extension of UCF-Construct*, which loads neighbors of each vertex from external memory. We use UCEF-Construct* to denote our final algorithm for index construction in external memory with all optimizations in Section 6.4.

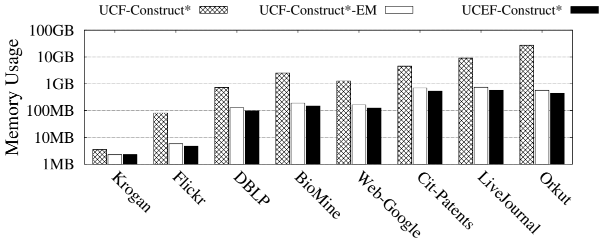


Fig. 7: Memory usage for external index construction

Evaluation-II: Memory Usage. We report the memory usage of UCEF-Construct* with UCF-Construct*-EM and the in-memory algorithm UCF-Construct* as comparisons. We can see a considerable decrease in memory usage from UCF-Construct* to other external algorithms, since we limit the memory usage to $O(n)$. In the largest dataset Orkut, UCEF-Construct* takes about 430MB, while UCF-Construct* takes up to 26GB.

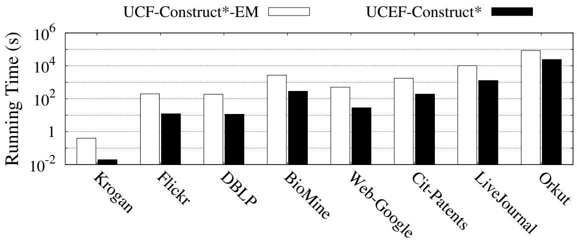


Fig. 8: Time cost for external index construction

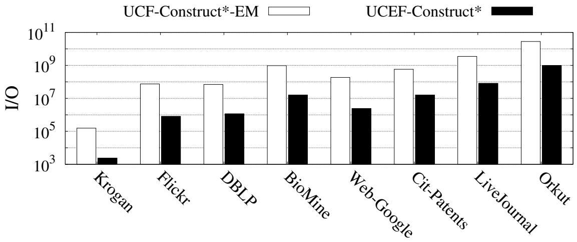


Fig. 9: I/O cost for external index construction

Evaluation-III: External Index Construction. The running time and I/O cost of our final algorithm UCEF-Construct* for external index construction are reported in Figure 8 and Figure 9, respectively. The performance of UCF-Construct*-EM is also reported as a comparison. We can find that the strategy for η-threshold computation in Theorem 6 is effective. In several datasets, UCEF-Construct* is one order of magnitude faster than UCF-Construct*-EM, and the I/O cost of UCEF-Construct* is almost two orders of

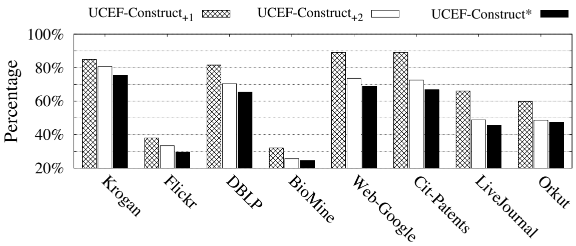


Fig. 10: Speedup for external index construction

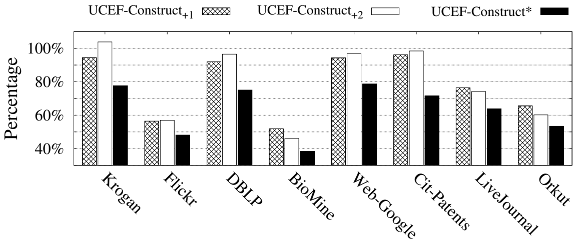


Fig. 11: I/O reduction for external index construction

magnitude smaller than that of UCF-Construct*-EM. For example, in DBLP, UCEF-Construct* and UCF-Construct*-EM take 11s and 189s, respectively.

Evaluation-IV: Optimizations for External Index Construction. We evaluate the effectiveness of optimizations proposed in Section 6.4. We use UCEF-Construct to denote Algorithm 5 without any optimizations in Section 6.4 and record its running time and I/O cost in each dataset. We use UCEF-Construct$_{+1}$ to denote the algorithm with the upper bound optimization in Section 6.4.1. We use UCEF-Construct$_{+2}$ to denote the algorithm with both upper bound optimization in Section 6.4.1 and neighbor ordering optimization in Section 6.4.2. Recall that UCEF-Construct* is the final algorithm with all three optimizations. We record the running time and I/O cost of these algorithms. For each dataset, we compute the percentages that the running time and the I/O cost of these algorithms account for those of UCEF-Construct, respectively. The results are shown in Figure 10 and Figure 11.

We can find that the upper bound optimization is the most effective among them, and the speedup is obvious especially in large datasets. Note that in several small datasets of Figure 11, UCEF-Construct$_{+2}$ takes a little more I/Os than UCEF-Construct$_{+1}$ due to the external sorting of vertex neighbors. However, this optimization reduces a large number of unnecessary neighbors for the η-threshold computation and still achieves a speedup compared to UCEF-Construct$_{+1}$.

8 Related Work

Uncertain Graphs. Many fundamental 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. [28] 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 [29].

K-Core Computations. k-core is first introduced by Seidman [8]. Batagelj and Zaversnik [21] propose a linear algorithm for core decomposition, which is presented in Section 2. I/O efficient algorithms for core decomposition are studied in [18], [22], [30]. Montresor et al. [23] investigate a distributed algorithm for core decomposition. Core decomposition in random graphs is studied in [31], [32], [33], [34]. Additionally, k-core is studied using weighted graphs in [35], directed graphs in [36], dynamic graphs in [37], [38], [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 3. 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.

9 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 presented to speed up the index construction. We also propose an algorithm to construct the index in external memory. The results of extensive performance studies demonstrate the effectiveness of the index-based approach and the efficiency of the query algorithm.

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.*, 2007.

[3] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *JASIST*, 2007.

[4] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa, "Injecting uncertainty in graphs for identity obfuscation," *PVLDB*, 2012.

[5] R. Jin, L. Liu, B. Ding, and H. Wang, "Distance-constraint reachability computation in uncertain graphs," *PVLDB*, 2011.

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

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

[8] S. B. Seidman, "Network structure and minimum degree," *Social networks*, 1983.

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

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

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

[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.

[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*, 2010.

[14] G. D. Bader and C. W. Hogue, "An automated method for finding molecular complexes in large protein interaction networks," *BMC bioinformatics*, 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.

[17] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios, "K-nearest neighbors in uncertain graphs," *PVLDB*, 2010.

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

[19] Z. Zhang, J. X. Yu, L. Qin, and Z. Shang, "Divide & conquer: I/O efficient depth-first search," in *SIGMOD*, T. K. Sellis, S. B. Davidson, and Z. G. Ives, Eds., 2015.

[20] B. Yang, D. Wen, L. Qin, Y. Zhang, L. Chang, and R. Li, "Index-based optimal algorithm for computing k-cores in large uncertain graphs," in *ICDE*, 2019.

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

[22] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo, "K-core decomposition of large networks on a single pc," *PVLDB*, 2015.

[23] A. Montresor, F. De Pellegrini, and D. Miorandi, "Distributed k-core decomposition," *TPDS*, 2013.

[24] D. E. Knuth, *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., 1998.

[25] 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*, 2006.

[26] A. D. Fox, B. J. Hescott, A. C. Blumer, and D. K. Slonim, "Connectedness of ppi network neighborhoods identifies regulatory hub proteins," *Bioinformatics*, 2011.

[27] L. Eronen and H. Toivonen, "Biomine: predicting links between biological entities using network models of heterogeneous databases," *BMC bioinformatics*, 2012.

[28] Y. Gao, X. Miao, G. Chen, B. Zheng, D. Cai, and H. Cui, "On efficiently finding reverse k-nearest neighbors over uncertain graphs," *VLDBJ*, 2017.

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

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

[31] S. Janson and M. J. Luczak, "A simple solution to the k-core problem," *Random Structures & Algorithms*, 2007.

[32] M. Molloy, "Cores in random hypergraphs and boolean formulas," *Random Structures & Algorithms*, 2005.

[33] T. Łuczak, "Size and connectivity of the k-core of a random graph," *Discrete Mathematics*, 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*, 1996.

[35] A. Garas, F. Schweitzer, and S. Havlin, "A k-shell decomposition method for weighted networks," *New Journal of Physics*, 2012.

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

[37] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek, "Streaming algorithms for k-core decomposition," *PVLDB*, 2013.

[38] R.-H. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *TKDE*, 2014.

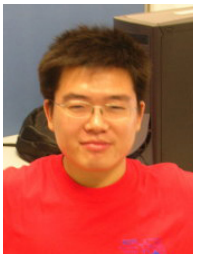
[39] Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin, "A fast order-based approach for core maintenance," in *ICDE*, 2017.

[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*, 2017.

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

Dong Wen reveived the BEng degree from Nankai University in 2013, and the PhD degree from the Faculty of Engineering and Information Technology, University of Technology Sydney in 2019. He is currently a postdoctoral research fellow in the Centre for Artificial Intelligence, University of Technology Sydney. His research interests include I/O efficient graph processing and streaming graph analysis.

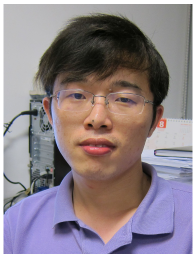
Bohua Yang received the BEng degree from Renmin University of China in 2016. He is currently working toward the PhD degree in the Centre for Artificial Intelligence, University of Technology, Sydney. His major research interests include cohesive subgraph detection and graph traversal algorithms on massive graphs.



Lu Qin received the BEng degree from the Department of Computer Science and Technology, Renmin University of China, in 2006, and the PhD degree from the Department of Systems Engineering and Engineering Management, Chinese University of Hong Kong, in 2010. He is currently an associate professor with the Centre for Artificial Intelligence, University of Technology, Sydney. His research interests include big graph analytics and graph query processing.

Ying Zhang is a professor and ARC Future Fellow at CAI, the University of Technology Sydney (UTS). He received his BSc and MSc degrees in Computer Science from Peking University, and PhD in Computer Science from the University of New South Wales. His research interests include query processing on data stream, uncertain data and graphs. He was an Australian Research Council Australian Postdoctoral Fellowship (ARC APD) holder (2010 – 2013) and ARC DECRA research fellow (2014 – 2016).

Lijun Chang is a Senior Lecturer and ARC Future Fellow in the School of Computer Science at the University of Sydney. He received Bachelor degree from Renmin University of China in 2007, and Ph.D. degree from The Chinese University of Hong Kong in 2011. He worked as a Postdoc and then DECRA research fellow at the University of New South Wales from 2012 to 2017. His research interests are in the fields of big graph (network) analytics, with a focus on designing practical algorithms and developing theoretical foundations for massive graph analysis. He has co-authored two monographs, and published over 50 papers in top venues such as SIGMOD, KDD, PVLDB, ICDE, VLDB Journal, TKDE, and Algorithmica.



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