

Article

Accelerating (k, l, η) -Core Query Processing in Directed Uncertain Graphs

Xian Tang ¹, Guo Chen ² and Junfeng Zhou ^{2,*} 

¹ School of Electronic and Electrical Engineering, Shanghai University of Engineering Science, Shanghai 201620, China; tangxian@sues.edu.cn

² School of Information and Intelligent Science, Donghua University, Shanghai 201620, China; gcheni@tsmc.com

* Correspondence: zhoujf@dhu.edu.cn

Abstract

Uncertain graphs are commonly used to model the uncertain relationships between entities that arise from experimental or measurement errors. In recent years, the analysis of uncertain graphs has attracted significant research attention, with the computation of (k, η) -cores emerging as a fundamental problem. However, existing studies on (k, η) -cores often neglect edge directions, resulting in weak correlations among vertices in the resulting subgraph. To address this limitation, we propose a direction-aware (k, l, η) -core model. Specifically, a (k, l, η) -core is defined as a maximal connected subgraph in which every vertex has a probability of at least η of having in-degree $\geq k$ and out-degree $\geq l$. We first present an online algorithm based on a peeling strategy to compute (k, l, η) -cores. To improve query performance, we develop two indexing mechanisms, DUCS-E and DUCS, that accelerate query processing. DUCS-E stores probability information for all possible (k, l, η) -cores, enabling it to completely avoid redundant computations during query processing, but at the cost of large storage space. To mitigate this issue, we propose the lightweight DUCS index, which stores directional probability information separately, reducing storage overhead while still pruning many irrelevant vertices; however, it requires additional verification. To balance efficiency and storage, we further design a hybrid index that combines the strengths of both approaches. Finally, experimental evaluations on real-world datasets demonstrate the effectiveness of the proposed (k, l, η) -core model as well as the efficiency and scalability of our methods.

Keywords: uncertain graph; directed graph; (k, l, η) -core; index; graph analysis



Academic Editor: Javid Taheri

Received: 7 July 2026

Revised: 3 August 2026

Accepted: 5 August 2026

Published: 7 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

In practical applications, graphs are commonly used to represent relationships among different entities, such as social networks [1,2], transportation networks [3], and biological networks [4,5]. Due to experimental errors or measurement inaccuracies, the relationships between entities often exhibit uncertainty and are typically modeled as uncertain graphs [6–10]. For example, in protein–protein interaction (PPI) networks [11], the probabilities associated with edges indicate the likelihood of interaction between protein molecules. In sensor networks [12], edge probabilities represent the reliability of information transmission between nodes. In social networks [13], edge probabilities reflect the degree of attention between users.

Cohesive subgraph mining is an important method for analyzing uncertain graphs, among which the (k, η) -core is one of the most popular uncertain cohesive subgraph models [14,15]. Compared with other uncertain cohesive subgraph models, such as the

(k, τ) -clique [16,17] and the (k, γ) -truss [3,18–20], the (k, η) -core model is simple and easy to compute. It can serve as a basis for uncertain graph analysis or as an input for other complex operations to simplify computation and improve efficiency, which has garnered widespread attention from researchers [12]. The (k, η) -core is a maximal subgraph that ensures that the probability that the degree of each vertex is at least k is at least η [12].

In practical applications, relationships between entities often exhibit directionality. The existing (k, η) -core model overlooks the directionality of edges, making it incapable of capturing dense subgraphs based on directional constraints. For example, Figure 1a depicts a social network in a real-world application, where the vertices represent users and the edges denote the attention among users. When using the (k, η) -core model without considering directions, only the 2-core subgraph in Figure 1b can be obtained, i.e., $\{v_1, v_2, v_3, v_4, v_5\}$. However, in practical scenarios, relationships between vertices are often unidirectional, as shown in Figure 1a. It can be seen that no other user pays attention to v_3 , and v_3 may be a zombie user (a virtual user manipulated by the software); v_5 does not actively pay attention to other users in this subgraph. Obviously, v_3 and v_5 do not have the same attention preference as other users in the subgraph, and they should not form a 2-core subgraph with v_1, v_2 , and v_4 .

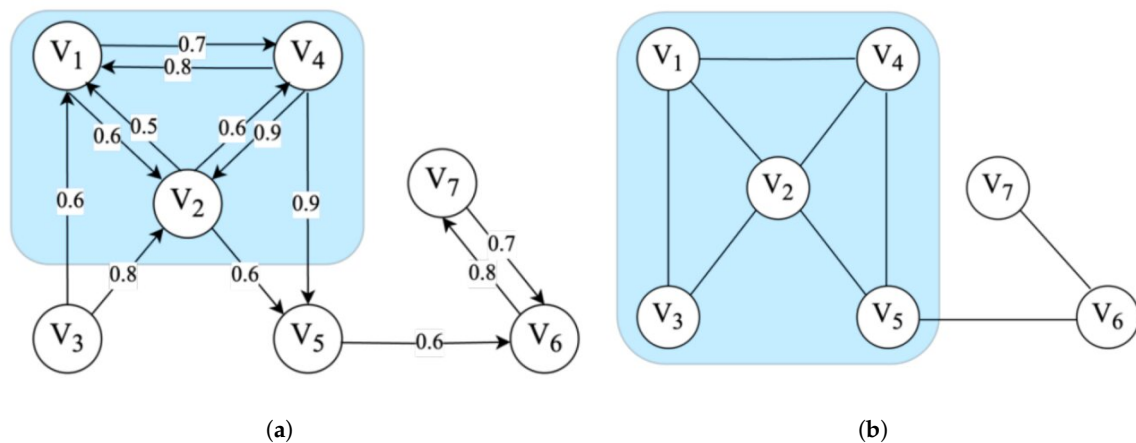


Figure 1. Illustration of directional and non-directional relationships in graphs. (a) $(2, 2, 0.3)$ -core of directed uncertain graph G_d . (b) 2-core of graph G .

To address the above problem, we propose the (k, l, η) -core model to obtain cohesive subgraphs in directed graphs by considering the directions of the edges. Given a directed uncertain graph, a (k, l, η) -core is a maximal connected subgraph that satisfies that the probability that the in-degree of each vertex is at least k and the out-degree of each vertex is at least l is at least η . For instance, given $k = 2$, $l = 2$, and a probability threshold $\eta = 0.3$, the $(2, 2, 0.3)$ -core that satisfies the condition in Figure 1a is $\{v_1, v_2, v_4\}$. Compared to the 2-core in Figure 1b, the (k, l, η) -core exhibits higher interactivity and tighter connections among vertices, excluding zombie users who are not followed by others and those who do not follow anyone within the subgraph.

While previous research has explored directed k -core queries [21–23], these studies did not take into account the probability information on edges. Additionally, existing methods for the (k, η) -core and directed k -core cannot be easily combined to solve the problem of (k, l, η) -core queries, necessitating the design of specialized solutions. Recently, Teng et al. [24] has explored the (k, l, η) -core model in the context of topic-aware most influential community search (TAMICS), their primary focus is on identifying the most influential community under a specific topic vector. To the best of our knowledge, there is no existing work that focuses solely on the fundamental problem of efficient and exact (k, l, η) -

core query processing itself. Our work is orthogonal and complementary to [24]; we provide the first comprehensive study on efficiently answering ad-hoc (k, l, η) -core queries with a suite of indexing solutions, without the additional complexity of influence maximization.

In this paper, we first propose an online query algorithm based on the peeling idea. Similar to the algorithm for determining the k -core on deterministic graphs, it iteratively removes vertices that do not meet the constraints and updates the probability information of associated vertices. This process continues until every vertex in the induced subgraph satisfies the constraints. While this algorithm can solve (k, l, η) -core queries, it requires frequent computation and updating of vertex probability information, leading to lower query efficiency. To address this issue, we propose an index-based method to accelerate the computation of (k, l, η) -cores, called DUCS-E (Directed Uncertain Core Sequence Enhanced). DUCS-E uses a single array to simultaneously maintain the probability information for both the in-degree and out-degree of each vertex. During query processing, it can directly return the results. To address the excessive storage space of the DUCS-E index, we further propose an index that considers the constraints of vertex in-degree and out-degree separately: DUCS (Directed Uncertain Core Sequence). This index uses two arrays to store the subgraph probability information. During query processing, there is no need for frequent computation and updating of vertex probability information in the uncertain graph. It can prune many unsatisfied vertices, but it requires further validation, leading to a less noticeable improvement in efficiency. Finally, to deal with this problem, a hybrid index structure named Hybrid is proposed. The Hybrid index combines the advantages of DUCS and DUCS-E. It uses a single array to store the subgraph probability information for (k, l, η) -cores when parameters k and l are small. When parameters k and l are large, it employs two arrays to separately store subgraph probability information for each direction. Based on different query parameters, it selects an appropriate solving strategy. The Hybrid algorithm ensures efficient query performance while reducing the time and space overhead of index construction.

We clarify that the efficiency claim in this paper refers to the online query processing stage, where the goal is to answer a given (k, l, η) -core query as fast as possible. To achieve this, we shift the heavy computation to an offline preprocessing phase, during which the index structures are constructed. Although the offline index construction may take considerable time on large datasets, it is a one-time investment. Once built, the index enables rapid query responses for arbitrary ad-hoc queries. This design is particularly beneficial in real-world applications where the number of queries is large or the query workload is unpredictable. Our main contributions are summarized as follows:

1. We propose a family of algorithms for (k, l, η) -core computation. The first is a baseline online algorithm. To improve performance, we further propose three index-based algorithms: DUCS-E, DUCS, and Hybrid. The DUCS-E index stores probability information related to all (k, l, η) -cores, achieving the best query performance. However, DUCS-E suffers from a large index size. Considering this, we provide a lightweight index, DUCS, which maintains (k, η) -cores for forward and backward directions, respectively. Although it can reduce computation cost by pruning useless vertices, it still requires verification of the remaining vertices. Finally, we propose the Hybrid index, which utilizes the benefits of both DUCS and DUCS-E to make a trade-off between query performance, index construction time, and index size.
2. We conduct extensive experiments on large real-world datasets. The experimental results indicate that the three index-based approaches achieve a four-order-of-magnitude improvement in query efficiency compared to the online query algorithm.

The rest of this paper is organized as follows. Section 2 introduces basic knowledge and problem definitions. Section 3 presents the online query algorithm. Sections 4–6

provide detailed descriptions of the DUCS-E index, the DUCS index, and the Hybrid index, respectively, along with their associated algorithms. Section 7 primarily analyzes and summarizes the complexity of the various algorithms. Section 8 validates the proposed algorithms through experiments. Section 9 introduces related work. Finally, conclusions are presented.

2. Preliminaries

2.1. Uncertain Graphs

Given a deterministic graph $G = (V, E)$, where V and E represent the sets of vertices and edges, respectively. For a vertex $u \in V$, the set of neighbors of u in G is denoted as $N(u, G) = \{v \in V \mid (u, v) \in E\}$, and the degree of u is $\deg(u, G) = |N(u, G)|$. When the context is clear, we abbreviate the neighbor set and degree as $N(u)$ and $\deg(u)$. For any subset $V' \subseteq V$, the induced subgraph of V' is denoted by $G[V'] = (V', \{(u, v) \in E \mid u, v \in V'\})$.

Definition 1 (k -core). Given a deterministic graph $G = (V, E)$ and a positive integer k , a k -core is a maximal connected induced subgraph $G[V']$ such that every vertex has degree at least k , i.e., for all $u \in V'$, $\deg(u, G[V']) \geq k$.

Definition 2 (Core number). Given a deterministic graph $G = (V, E)$, the core number of a vertex $u \in V$, denoted by $\text{core}(u)$, is the maximum value of k for which u belongs to a k -core.

Let $\mathcal{G} = (V, E, p)$ be an uncertain graph, where p is a function that assigns to each edge $e \in E$ a probability value in $[0, 1]$. For each edge $e = (u, v) \in E$, p_e denotes its probability. The neighbor set and degree of a vertex $u \in V$ in $\mathcal{G}$ are denoted by $\mathcal{N}(u, \mathcal{G})$ and $\text{Deg}(u, \mathcal{G})$, respectively.

Following the literature [12], we assume that the edge probabilities are mutually independent. Under this assumption, the uncertain graph can be analyzed using possible-world semantics. There are $2^{|E|}$ possible instances $G = (V, E')$ with $E' \subseteq E$. The probability of a particular instance graph G is given by

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

Definition 3 ((k, η) -core). Given an uncertain graph $\mathcal{G} = (V, E, p)$, a positive integer k , and a probability threshold $\eta \in [0, 1]$, the (k, η) -core is a maximal connected induced subgraph $\mathcal{G}[V']$ such that for every vertex $u \in V'$, the probability that the degree of u in $\mathcal{G}[V']$ is at least k is not less than η :

$$\forall u \in V', \quad \Pr[\text{Deg}(u, \mathcal{G}[V']) \geq k] \geq \eta.$$

For a vertex v , define $\mathcal{G}_v^{\geq k}$ as the set of all possible world subgraphs of $\mathcal{G}$ in which the degree of v is at least k :

$$\mathcal{G}_v^{\geq k} = \{G \subseteq \mathcal{G} \mid \deg(v, G) \geq k\}.$$

Then, the probability that $\text{Deg}(v, \mathcal{G}) \geq k$ is computed as

$$\Pr[\text{Deg}(v, \mathcal{G}) \geq k] = \sum_{G \in \mathcal{G}_v^{\geq k}} \Pr(G). \quad (2)$$

Definition 4 (η -degree). Given an uncertain graph $\mathcal{G} = (V, E, p)$ and a probability threshold $\eta \in [0, 1]$, the η -degree of a vertex $v \in V$ is the maximum integer k such that $\Pr[\text{Deg}(v, \mathcal{G}) \geq k] \geq \eta$.

From Definition 4, the η -degree can be obtained via

$$\Pr[\text{Deg}(v, \mathcal{G}) \geq k] = 1 - \sum_{i=0}^{k-1} \Pr[\text{Deg}(v, \mathcal{G}) = i]. \quad (3)$$

Reference [12] proposes a dynamic programming (DP) method for computing (k, η) -cores. Using Equation (3) and the DP approach, we can compute $\Pr[\text{Deg}(v, \mathcal{G}) = i]$ for each vertex v . Let $E(v, \mathcal{G}) = \{e_1, e_2, \dots, e_s\}$, where $s = \text{Deg}(v)$, and let $E_s^r(v, \mathcal{G}) = \{e_1, e_2, \dots, e_r\}$ for $r \leq s$. Define $\mathcal{G}_v^r = (V, E \setminus (E(v, \mathcal{G}) \setminus E_s^r(v, \mathcal{G})), p)$ as the subgraph obtained by removing the edges $E(v, \mathcal{G}) \setminus E_s^r(v, \mathcal{G})$. The degree of v in $\mathcal{G}_v^r$ is $\text{Deg}(v, \mathcal{G}_v^r)$. If the degree of v equals i , there are two possibilities: (i) e_s exists and $\text{Deg}(v, \mathcal{G}_v^{s-1}) = i - 1$; (ii) e_s does not exist and $\text{Deg}(v, \mathcal{G}_v^{s-1}) = i$. Let $Y_v(r, i)$ denote the probability that the degree of v is exactly i in the subgraph $\mathcal{G}_v^r \subseteq \mathcal{G}$. The DP recurrence is

$$Y_v(r, i) = p_{e_r} \cdot Y_v(r-1, i-1) + (1 - p_{e_r}) \cdot Y_v(r-1, i). \quad (4)$$

The boundary conditions are

$$\begin{cases} Y(0, 0) = 1, \\ Y(r, -1) = 0, & \text{for } r = 0, \dots, \text{Deg}(v), \\ Y(r, j) = 0, & \text{for } r = 0, \dots, \text{Deg}(v), j = r + 1, \dots, i. \end{cases} \quad (5)$$

After removing an edge $e \in E(v)$, the updated probability that the degree of v equals i in the reduced graph is given by

$$\Pr[\text{Deg}(v \mid E(v) \setminus \{e\}) = i] = \frac{\Pr[\text{Deg}(v) = i] - p_e \Pr[\text{Deg}(v \mid E(v) \setminus \{e\}) = i-1]}{1 - p_e}. \quad (6)$$

For the special case $i = 0$, we have

$$\Pr[\text{Deg}(v \mid E(v) \setminus \{e\}) = 0] = \frac{\Pr[\text{Deg}(v) = 0]}{1 - p_e}.$$

2.2. Directed Uncertain Graphs

When dealing with directed uncertain graphs, we must revisit the previous definitions to incorporate edge directions.

Let $\mathcal{G}_d = (V, E, p)$ be a directed uncertain graph. For a vertex $u \in V$, the out-neighbor set is $\mathcal{N}_{\text{out}}(u, \mathcal{G}_d) = \{v \in V \mid (u, v) \in E\}$ and the in-neighbor set is $\mathcal{N}_{\text{in}}(u, \mathcal{G}_d) = \{v \in V \mid (v, u) \in E\}$. We abbreviate these as $\mathcal{N}_{\text{out}}(u)$ and $\mathcal{N}_{\text{in}}(u)$ when the context is clear. The out-degree and in-degree of u are denoted by $\text{Deg}_{\text{out}}(u, \mathcal{G}_d)$ and $\text{Deg}_{\text{in}}(u, \mathcal{G}_d)$, respectively. The underlying deterministic graph is $G_d = (V, E)$.

Definition 5 (k -core_{in}, l -core_{out}). Given a directed deterministic graph $G_d = (V, E)$ and two positive integers k and l , a k -core_{in} (resp. l -core_{out}) is a maximal connected induced subgraph G'_d on V' such that every vertex has in-degree (resp. out-degree) at least k (resp. l). That is,

$$\forall u \in V', \quad \text{Deg}_{\text{in}}(u, G'_d) \geq k \quad (\text{resp. } \text{Deg}_{\text{out}}(u, G'_d) \geq l).$$

Definition 6 (core_{in}, core_{out}). Given a directed deterministic graph $G_d = (V, E)$, the in-degree core number $\text{core}_{\text{in}}(u)$ is the maximum k such that u belongs to a k -core_{in}. Similarly, the out-degree core number $\text{core}_{\text{out}}(u)$ is the maximum l such that u belongs to an l -core_{out}.

Based on Definitions 5 and 6, we now define the (k, l, η) -core.

Definition 7 ($((k, l, \eta)$ -core). Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, two positive integers k and l , and a probability threshold $\eta \in [0, 1]$, the $((k, l, \eta)$ -core is a maximal connected induced subgraph $\mathcal{G}'_d$ on V' such that every vertex has probability at least η of having in-degree $\geq k$ and out-degree $\geq l$:

$$\forall u \in V', \quad \min(\Pr[\text{Deg}_{in}(u, \mathcal{G}'_d) \geq k], \Pr[\text{Deg}_{out}(u, \mathcal{G}'_d) \geq l]) \geq \eta.$$

Remark 1. The in-degree and out-degree of a vertex are determined by two disjoint edge sets. Under the independent edge probability model, the random variables $\text{Deg}_{in}(v)$ and $\text{Deg}_{out}(v)$ are independent. However, for a vertex to be considered a cohesive part of a directed core, it must play a significant role in both directions. Therefore, we impose constraints on both marginal probabilities. The $\min$ operator is deliberately employed to enforce the “shortboard effect”: the overall reliability of a vertex in the core is limited by its weakest direction. This guarantees that every vertex in the resulting subgraph is statistically significant as both a source and a target of connections, ensuring strong mutual engagement within the community.

Example 1. Consider the directed uncertain graph in Figure 1a. For $k = 2$, $l = 2$, and $\eta = 0.3$, the connected subgraph $\mathcal{G}'_d = \mathcal{G}_d[\{v_1, v_2, v_4\}]$ satisfies the $((2, 2, 0.3)$ -core condition. In this subgraph:

- Vertex v_1 has two in-edges and two out-edges; $\Pr[\text{Deg}_{in}(v_1, \mathcal{G}'_d) \geq 2] = 0.4$, $\Pr[\text{Deg}_{out}(v_1, \mathcal{G}'_d) \geq 2] = 0.42$, and $\min(0.4, 0.42) = 0.4 \geq 0.3$.
- Vertex v_2 : $\Pr[\text{Deg}_{in}(v_2, \mathcal{G}'_d) \geq 2] = 0.54$, $\Pr[\text{Deg}_{out}(v_2, \mathcal{G}'_d) \geq 2] = 0.3$, $\min(0.3, 0.54) = 0.3 \geq 0.3$.
- Vertex v_4 : $\Pr[\text{Deg}_{in}(v_4, \mathcal{G}'_d) \geq 2] = 0.42$, $\Pr[\text{Deg}_{out}(v_4, \mathcal{G}'_d) \geq 2] = 0.72$, $\min(0.42, 0.72) = 0.42 \geq 0.3$.

Moreover, $\mathcal{G}'_d$ is maximal: adding v_3 would give $\Pr[\text{Deg}_{out}(v_3, \mathcal{G}'_d) \geq 2] = 0.48 > 0.3$ but $\Pr[\text{Deg}_{in}(v_3, \mathcal{G}'_d) \geq 2] = 0$, so $\min(0.48, 0) = 0 < 0.3$; thus v_3 does not satisfy the constraint.

Problem Definition. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, two positive integers k and l , and a probability threshold $\eta \in [0, 1]$, return all $((k, l, \eta)$ -cores in $\mathcal{G}_d$.

3. The Online Algorithm

The online query algorithm is based on the peeling idea. It iteratively removes vertices that do not meet the constraints and updates the probability information of the associated vertices until every vertex in the induced subgraph satisfies the constraints, at which point the execution terminates and returns the results. For directed uncertain graphs, however, the in-degree and out-degree of vertices must be considered separately to solve the corresponding core decomposition problem. We introduce the following definitions.

Definition 8 ($((k, \eta)$ -core_{in}, (l, η) -core_{out}). Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$ and a probability threshold $\eta \in [0, 1]$, a $((k, \eta)$ -core_{in} (resp. (l, η) -core_{out}) is a maximal connected induced subgraph $\mathcal{G}'_d$ on vertex set V' such that for every vertex $u \in V'$, the probability that its in-degree (resp. out-degree) is at least k (resp. l) is at least η . That is,

$$\forall u \in V', \quad \Pr[\text{Deg}_{in}(u, \mathcal{G}'_d) \geq k] \geq \eta \quad (\text{resp. } \Pr[\text{Deg}_{out}(u, \mathcal{G}'_d) \geq l] \geq \eta).$$

Definition 9 ((η) -Deg_{in}, (η) -Deg_{out}). Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$ and a probability threshold $\eta \in [0, 1]$, the (η) -Deg_{in} and (η) -Deg_{out} of a vertex $v \in V$ are the maximum integers k and l , respectively, satisfying

$$\Pr[\text{Deg}_{in}(v, \mathcal{G}_d) \geq k] \geq \eta \quad \text{and} \quad \Pr[\text{Deg}_{out}(v, \mathcal{G}_d) \geq l] \geq \eta.$$

Definition 10 (η -core_{in}, η -core_{out}). Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$ and a probability threshold $\eta \in [0, 1]$, the η -core_{in} and η -core_{out} of a vertex $u \in V$ are the maximum values of k and l , respectively, such that u belongs to a (k, η) -core_{in} and an (l, η) -core_{out}.

According to Definitions 9 and 10, together with Equations (3) and (6), we can compute and update the values of η -Deg_{in} and η -Deg_{out} for each vertex in the directed uncertain graph.

The (k, l, η) -core online query algorithm is presented in Algorithm 1. Before checking whether a vertex satisfies the (k, l, η) -core constraint, we iteratively remove vertices that do not meet the structural constraints $\text{Deg}_{\text{out}} < l$ or $\text{Deg}_{\text{in}} < k$ (lines 1–8). Then, we compute η -Deg_{in} and η -Deg_{out} for the remaining vertices using Equation (3) (lines 9–10). The algorithm continues by iteratively removing vertices and their incident edges that do not satisfy the (k, l, η) -core constraint, updating the η -Deg_{in} and η -Deg_{out} of neighboring vertices according to Equation (6) (lines 11–18). Finally, all connected subgraphs that satisfy the constraints are stored in the result set $\mathcal{S}$ (lines 19–22).

Algorithm 1: (k, l, η) -core Decomposition

Input: A directed uncertain graph $\mathcal{G}_d = (V, E, p)$, two integers k and l , and a probability threshold η

Output: All (k, l, η) -cores in $\mathcal{G}_d$

```

1 while  $\exists v \in V$  s.t.  $\text{Deg}_{\text{in}}(v, \mathcal{G}_d) < k$  or  $\text{Deg}_{\text{out}}(v, \mathcal{G}_d) < l$  do
2   foreach  $u \in \mathcal{N}_{\text{out}}(v, \mathcal{G}_d)$  do
3     remove edge  $(v, u)$  from  $\mathcal{G}_d$ ;
4      $\text{Deg}_{\text{in}}(u) \leftarrow \text{Deg}_{\text{in}}(u) - 1$ ;
5   foreach  $u \in \mathcal{N}_{\text{in}}(v, \mathcal{G}_d)$  do
6     remove edge  $(u, v)$  from  $\mathcal{G}_d$ ;
7      $\text{Deg}_{\text{out}}(u) \leftarrow \text{Deg}_{\text{out}}(u) - 1$ ;
8    $V \leftarrow V \setminus \{v\}$ ;
9 compute  $\eta$ -Degin $(v, \mathcal{G}_d)$  for all  $v \in V$ ;
10 compute  $\eta$ -Degout $(v, \mathcal{G}_d)$  for all  $v \in V$ ;
11 while  $\exists v \in V$  s.t.  $\eta$ -Degin $(v, \mathcal{G}_d) < k$  or  $\eta$ -Degout $(v, \mathcal{G}_d) < l$  do
12   foreach  $u \in \mathcal{N}_{\text{out}}(v, \mathcal{G}_d)$  do
13     remove edge  $(v, u)$  from  $\mathcal{G}_d$ ;
14     update  $\eta$ -Degin $(u, \mathcal{G}_d)$ ;
15   foreach  $u \in \mathcal{N}_{\text{in}}(v, \mathcal{G}_d)$  do
16     remove edge  $(u, v)$  from  $\mathcal{G}_d$ ;
17     update  $\eta$ -Degout $(u, \mathcal{G}_d)$ ;
18    $V \leftarrow V \setminus \{v\}$ ;
19  $\mathcal{S} \leftarrow \emptyset$ ;
20 foreach connected component  $\mathcal{G}_d[C]$  in  $\mathcal{G}_d$  do
21    $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$ ;
22 return  $\mathcal{S}$ ;

```

Example 2. Consider the directed uncertain graph shown in Figure 1a. Suppose $k = 1$, $l = 1$, and the probability threshold $\eta = 0.7$ (lines 1–8). First, we compute $\text{Deg}_{\text{in}}(v_3) = 0 < 1$, thus

remove v_3 . The remaining vertices have Deg_{in} and Deg_{out} satisfying the conditions (lines 9–10). Next, we calculate

$$\Pr[\text{Deg}_{in}(v_5) \geq 1] = 1 - \sum_{i=0}^1 \Pr[\text{Deg}_{in}(v_5) = i] = 1 - 0.4 \times 0.1 = 0.96 \geq 0.7,$$

and

$$\Pr[\text{Deg}_{in}(v_5) \geq 2] = 0.6 \times 0.9 = 0.54 < 0.7.$$

Therefore, $0.7\text{-Deg}_{in}(v_5) = 1$. Similarly, we obtain $0.7\text{-Deg}_{out}(v_5) = 0$ and the corresponding values for the remaining vertices (lines 11–18). Although the in-degree probability of v_5 satisfies the condition, its out-degree does not. Hence, we remove v_5 and its incident edges. We then update the out-degree probabilities of v_2 and v_4 , and the in-degree probability of v_6 using Equation (6). After updating, all vertices in the remaining two maximal connected induced subgraphs $C_1 = \{v_1, v_2, v_4\}$ and $C_2 = \{v_6, v_7\}$ satisfy the constraint. We store these final subgraphs that satisfy the $(1, 1, 0.7)$ -core condition in the result set $\mathcal{S} = \{\{v_1, v_2, v_4\}, \{v_6, v_7\}\}$ (lines 19–22).

Theorem 1. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, the time complexity of Algorithm 1 is

$$O\left(|V| + \sum_{v \in V} \left(\max(\eta\text{-Deg}_{in}(v), \eta\text{-Deg}_{out}(v)) \cdot \max(\text{Deg}_{in}(v), \text{Deg}_{out}(v)) \right)\right).$$

Proof. First, Algorithm 1 takes $O(|V|)$ time to iteratively delete vertices whose in-degree or out-degree does not satisfy the structural constraint (lines 1–8). Then, according to [12], the time to compute $\eta\text{-Deg}_{in}$ and $\eta\text{-Deg}_{out}$ for each remaining vertex is $O(\sum_{v \in V} (\eta\text{-Deg}_{in}(v) \cdot \text{Deg}_{in}(v)))$ and $O(\sum_{v \in V} (\eta\text{-Deg}_{out}(v) \cdot \text{Deg}_{out}(v)))$ (lines 9–10). Finally, we need to delete all vertices that do not satisfy the query conditions and iteratively update the $\eta\text{-Deg}_{in}$ and $\eta\text{-Deg}_{out}$ values. According to [12], the time to update these values is $O(\sum_{v \in V} \max(\text{Deg}_{in}(v), \text{Deg}_{out}(v)))$ (lines 11–19). Therefore, the overall time complexity of Algorithm 1 is

$$O\left(|V| + \sum_{v \in V} \left(\max(\eta\text{-Deg}_{in}(v), \eta\text{-Deg}_{out}(v)) \cdot \max(\text{Deg}_{in}(v), \text{Deg}_{out}(v)) \right)\right).$$

□

The (k, l, η) -core online query algorithm produces accurate result sets, but it is highly dependent on the parameter settings, which limits its universality. Although the online algorithm can be optimized using dynamic programming, the cost of computing and updating the η -degree for each vertex remains high.

4. DUCS-E Index

4.1. Index Structure

The online query algorithm repeatedly calculates the probability values of vertices, leading to low query efficiency. To avoid redundant calculations, we consider maintaining the probability information of each vertex.

Definition 11. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$ and two positive integers k and l , the value $\eta_{(k,l)}(v, \mathcal{G}_d)$ for a vertex $v \in V$ is defined as the maximum probability η such that v belongs to a (k, l, η) -core.

Thus, $\eta_{(k,l)}(v, \mathcal{G}_d)$ represents the highest probability threshold for which vertex v satisfies the constraints k and l . Given k, l , and η , if $\eta_{(k,l)}(v, \mathcal{G}_d) \geq \eta$, then v belongs to the (k, l, η) -core.

Lemma 1. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, two positive integers k and l , and probability thresholds $0 \leq \eta \leq \eta' \leq 1$, if there exist subgraphs $\mathcal{G}'_{d_1} = (k, l, \eta)$ -core and $\mathcal{G}'_{d_2} = (k, l, \eta')$ -core, then $\mathcal{G}'_{d_2} \subseteq \mathcal{G}'_{d_1}$.

Proof. Consider a directed uncertain graph $\mathcal{G}_d = (V, E, p)$. Suppose $\mathcal{G}'_{d_1}$ is a (k, l, η) -core and $\mathcal{G}'_{d_2}$ is a (k, l, η') -core with $\eta \leq \eta'$. According to Definition 5, for every $v \in V_{\mathcal{G}'_{d_2}}$,

$$\min(\Pr[\text{Deg}_{\text{in}}(v, \mathcal{G}_d) \geq k], \Pr[\text{Deg}_{\text{out}}(v, \mathcal{G}_d) \geq l]) \geq \eta' \geq \eta.$$

Hence, $\mathcal{G}'_{d_2}$ also satisfies the constraints of a (k, l, η) -core, implying $V_{\mathcal{G}'_{d_2}} \subseteq V_{\mathcal{G}'_{d_1}}$. Therefore, $\mathcal{G}'_{d_2} \subseteq \mathcal{G}'_{d_1}$. $\square$

According to Lemma 1 and Definition 11, we can determine whether a vertex $v \in V$ belongs to a (k, l, η) -core based on the value of $\eta_{(k,l)}(v, \mathcal{G}_d)$. Thus, by storing $\eta_{(k,l)}(v, \mathcal{G}_d)$ for all vertices, we can efficiently answer all possible (k, l, η) -core queries.

The DUCS-E index stores information for all (k, l, η) -core combinations with $k \in [1, k_{\max}]$ and $l \in [1, l_{\max}]$. By Lemma 1, the subgraphs corresponding to different η values are nested. Therefore, we can process vertices in ascending order of η values step by step, avoiding simultaneous storage of all satisfying nodes and thus saving memory.

Example 3. Consider the directed uncertain graph $\mathcal{G}_d$ shown in Figure 1a. The DUCS-E index is illustrated in Figure 2. Take vertex v_1 as an example. For $k = 1$ and $l = 1$, we have $\eta_{(1,1)}(v_1, \mathcal{G}_d) = 0.8$, meaning that v_1 satisfies the constraints of a $(1, 1, 0.8)$ -core. If $\eta > 0.8$, then v_1 does not satisfy the $(1, 1, \eta)$ -core constraints.

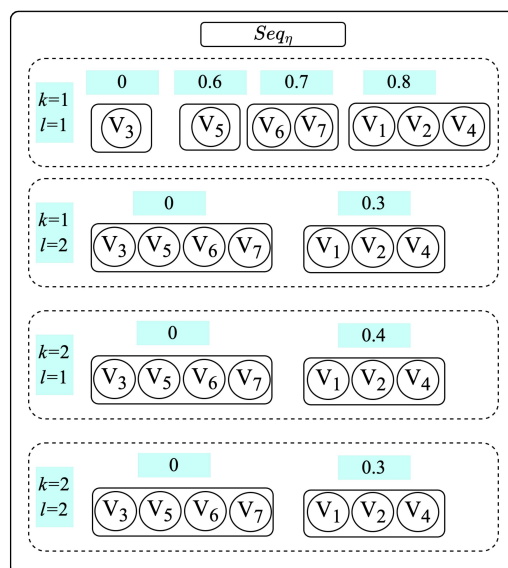


Figure 2. DUCS-E index of $\mathcal{G}_d$.

4.2. Query Processing

The DUCS-E index simultaneously accounts for both in-degree and out-degree constraints of each vertex, eliminating the need to recompute vertex probabilities during query processing.

The DUCS-E index-based query algorithm is presented in Algorithm 2. We first initialize the storage array (line 1). Then, we invoke the DUCS-E index construction algorithm to store all potential (k, l, η) -cores in the array (line 2). Next, we traverse the array to identify vertices satisfying the query conditions (lines 5–7). Finally, we return the maximal connected subgraph induced by these vertices (lines 8–10).

Algorithm 2: DUCS-E-Index Query

Input: An integer k , an integer l , a probability threshold η , and the DUCS-E index of $\mathcal{G}_d = (V, E, p)$
Output: The (k, l, η) -core of $\mathcal{G}_d$

```

1 Initialize  $res \leftarrow \emptyset$ ;
2  $res \leftarrow$  the DUCS-E index of  $\mathcal{G}_d$  for  $(k, l)$ ;
3 Initialize  $\mathcal{S} \leftarrow \emptyset$ ;
4 Initialize  $V' \leftarrow \emptyset$ ;
5 foreach  $v \in V$  do
6   if  $res(v) \geq \eta$  then
7      $V' \leftarrow V' \cup \{v\}$ ;
8 foreach connected component  $\mathcal{G}_d[C] \subseteq \mathcal{G}_d[V']$  do
9    $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$ ;
10 return  $\mathcal{S}$ ;
```

Example 4. Based on the DUCS-E index shown in Figure 2, consider the query parameters $k = 2$, $l = 1$, and $\eta = 0.4$. First, from the stored array we identify the set of vertices $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ that satisfy the conditions $k = 2$ and $l = 1$. Within this set, we then select the subset $\{v_1, v_2, v_4\}$ where the stored probability values are at least 0.4. Finally, we output the maximal connected induced subgraph $\mathcal{G}_d[\{v_1, v_2, v_4\}]$ that satisfies the given constraints.

Theorem 2. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, the time complexity of Algorithm 2 is $O(\sum_{v \in V} \max(\text{Deg}_{\text{out}}(v), \text{Deg}_{\text{in}}(v)) + |V|)$.

Proof. Traversing the index array to find vertices satisfying the conditions takes $O(|V|)$ time. Then, obtaining the vertex sets of each connected component requires $O(\sum_{v \in V} \max(\text{Deg}_{\text{out}}(v), \text{Deg}_{\text{in}}(v)))$ time. Therefore, the total time complexity of Algorithm 2 is $O(\sum_{v \in V} \max(\text{Deg}_{\text{out}}(v), \text{Deg}_{\text{in}}(v)) + |V|)$. $\square$

4.3. Index Construction

According to Definition 11, all information of (k, l, η) -cores for $k \in [1, k_{\max}]$ and $l \in [1, l_{\max}]$ is stored in the sequence array Seq_{η} using a bottom-up approach. Algorithm 3 details the construction of the DUCS-E index.

First, we perform core decomposition on the deterministic graph to obtain $k_{\max}$ and $l_{\max}$ (line 1). Then, for each pair (k, l) , we construct an index covering all relevant information (lines 5–24). For each (k, l) combination, dynamic programming is used to compute $\text{Pr}_{\text{in}}(v)$ and $\text{Pr}_{\text{out}}(v)$ for every vertex v , and we store the minimum in $\text{Pr}(v)$ (lines 8–11). We identify the vertex v with the smallest $\text{Pr}(v)$, delete v and its incident edges (lines 13–15), and iteratively update Pr_{in} and Pr_{out} for the neighbors of v (lines 17–22). The value $\min(\text{Pr}_{\text{in}}(v), \text{Pr}_{\text{out}}(v))$ for vertices satisfying the constraints is stored in the temporary array Pr (lines 23–24). This process continues until all (k, l) pairs are processed, storing $\eta_{(k,l)}(v, \mathcal{G}_d)$ for each vertex. Finally, the DUCS-E index is obtained.

Algorithm 3: DUCS-E-Index Construction

Input: An uncertain graph $\mathcal{G}_d = (V, E, p)$
Output: The DUCS-E index of $\mathcal{G}_d$

```

1  $k_{\max}, l_{\max} \leftarrow$  the maximum  $\text{core}_{\text{in}}$  and  $\text{core}_{\text{out}}$  values among all vertices of  $\mathcal{G}_d$ ;
2 Initialize  $\text{Seq}(v) \leftarrow \emptyset$  for each  $v \in V$ ;
3 Initialize  $\text{Pr}(v) \leftarrow \emptyset, \text{res}(v) \leftarrow \emptyset$  for each  $v \in V$ ;
4  $\text{curP} \leftarrow 0$ ;
5 for  $k \leftarrow 1$  to  $k_{\max}$  do
6   for  $l \leftarrow 1$  to  $l_{\max}$  do
7      $g \leftarrow \mathcal{G}_d$ ;
8     foreach  $v \in V$  do
9       compute  $\text{Pr}_{\text{in}}(v)$ ;
10      compute  $\text{Pr}_{\text{out}}(v)$ ;
11       $\text{Pr}(v) \leftarrow \min(\text{Pr}_{\text{in}}(v), \text{Pr}_{\text{out}}(v))$ ;
12     while  $g$  is not empty do
13        $v \leftarrow \text{argmin}_{u \in V_g} \text{Pr}(u)$ ;
14        $\text{curP} \leftarrow \max(\text{Pr}(v), \text{curP})$ ;
15        $\text{res}(v) \leftarrow \text{curP}$ ;
16        $V_g \leftarrow V_g \setminus \{v\}$ ;
17       foreach  $u \in \mathcal{N}_{\text{in}}(v, g)$  do
18         remove edge  $(u, v)$  from  $g$ ;
19         update  $\text{Pr}_{\text{out}}(u, g)$ ;
20       foreach  $u \in \mathcal{N}_{\text{out}}(v, g)$  do
21         remove edge  $(v, u)$  from  $g$ ;
22         update  $\text{Pr}_{\text{in}}(u, g)$ ;
23       foreach  $u \in \mathcal{N}_{\text{out}}(v, g) \cup \mathcal{N}_{\text{in}}(v, g)$  do
24          $\text{Pr}(u) \leftarrow \min(\text{Pr}(u), \min(\text{Pr}_{\text{in}}(u, g), \text{Pr}_{\text{out}}(u, g)))$ ;
25 return  $\text{Seq}(v)$  for each  $v \in V$ ;

```

Theorem 3. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, the time complexity of Algorithm 3 is

$$O((k_{\max} \cdot l_{\max}) \cdot (\max(k_{\max}, l_{\max}) \cdot (|V| \log |V| + |E|))).$$

Proof. Computing core_{in} and core_{out} for each vertex in the deterministic graph takes $O(\max(|E_{\text{in}}|, |E_{\text{out}}|))$, where $|E_{\text{in}}|$ and $|E_{\text{out}}|$ are the total in-degree and out-degree of $\mathcal{G}_d$ (line 1). According to [12], calculating $\eta\text{-Deg}_{\text{in}}$ and $\eta\text{-Deg}_{\text{out}}$ requires $O(\sum_{v \in V} (\eta\text{-Deg}_{\text{in}}(v) \cdot \text{Deg}_{\text{in}}(v)))$ and $O(\sum_{v \in V} (\eta\text{-Deg}_{\text{out}}(v) \cdot \text{Deg}_{\text{out}}(v)))$ time (lines 8–11). Finding the vertex with the minimum value and deleting it costs $O(|V| \log |V|)$ (lines 13–14). Updating Pr_{in} and Pr_{out} takes $O(\sum_{v \in V} \max(\text{Deg}_{\text{in}}(v), \text{Deg}_{\text{out}}(v)))$ (lines 17–22). Since $\max(\text{Deg}_{\text{in}}(v), \text{Deg}_{\text{out}}(v)) \leq \max(k_{\max}, l_{\max})$, the overall time complexity follows. $\square$

Theorem 4. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, the space complexity of Algorithm 3 is $O(\sum_{v \in V} (\text{core}_{\text{in}}(v) \cdot \text{core}_{\text{out}}(v)))$.

Proof. For each vertex $v \in V$, the storage space $\text{Seq}(v)$ depends on both $\text{core}_{\text{in}}(v)$ and $\text{core}_{\text{out}}(v)$. In the worst case, the space consumed per vertex is $\text{core}_{\text{in}}(v) \cdot \text{core}_{\text{out}}(v)$. Thus, the total space complexity is $O(\sum_{v \in V} (\text{core}_{\text{in}}(v) \cdot \text{core}_{\text{out}}(v)))$. $\square$

5. DUCS Index

5.1. Index Structure

While the query algorithm based on the DUCS-E index significantly improves query efficiency, its index size is prohibitively large because it stores probability thresholds for all combinations of k and l values. We can substantially reduce the index storage by storing probability thresholds for the two directions separately. To this end, based on Definition 11, we first define probability thresholds for directional constraints.

Definition 12 ($\eta_k(v, \mathcal{G}_d)$ and $\eta_l(v, \mathcal{G}_d)$). Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$ and two positive integers k and l , $\eta_k(v, \mathcal{G}_d)$ denotes the maximum η such that v belongs to a (k, η) -core_{in}, and $\eta_l(v, \mathcal{G}_d)$ denotes the maximum η such that v belongs to a (l, η) -core_{out}.

Thus, $\eta_k(v, \mathcal{G}_d)$ (resp. $\eta_l(v, \mathcal{G}_d)$) is the highest probability value for which v can be part of a (k, η) -core_{in} (resp. (l, η) -core_{out}). If for given k (or l) and η we have $\eta_k(v, \mathcal{G}_d) \geq \eta$ (or $\eta_l(v, \mathcal{G}_d) \geq \eta$), then v certainly belongs to the corresponding core.

Lemma 2. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, two positive integers k, l , and probability thresholds $0 \leq \eta \leq \eta' \leq 1$,

- If there exist subgraphs $\mathcal{G}'_{d_1} = (k, \eta)$ -core_{in} and $\mathcal{G}'_{d_2} = (k, \eta')$ -core_{in}, then $\mathcal{G}'_{d_2} \subseteq \mathcal{G}'_{d_1}$;
- If there exist subgraphs $\mathcal{G}'_{d_3} = (l, \eta)$ -core_{out} and $\mathcal{G}'_{d_4} = (l, \eta')$ -core_{out}, then $\mathcal{G}'_{d_4} \subseteq \mathcal{G}'_{d_3}$.

Proof. Consider a directed uncertain graph $\mathcal{G}_d = (V, E, p)$. Suppose $\mathcal{G}'_{d_1}$ is a (k, η) -core_{in} and $\mathcal{G}'_{d_2}$ is a (k, η') -core_{in} with $\eta \leq \eta'$. By Definition 8, for every $v \in V_{\mathcal{G}'_{d_2}}$, we have $\Pr[\text{Deg}_{\text{in}}(v) \geq k] \geq \eta' \geq \eta$. Hence, $\mathcal{G}'_{d_2}$ also satisfies the (k, η) -core_{in} constraints, so $V_{\mathcal{G}'_{d_2}} \subseteq V_{\mathcal{G}'_{d_1}}$ and consequently $\mathcal{G}'_{d_2} \subseteq \mathcal{G}'_{d_1}$. The proof for out-degree cores is analogous. $\square$

According to Lemma 2 and Definition 12, we can test whether a vertex v belongs to a (k, η) -core_{in} or (l, η) -core_{out} using the stored values $\eta_k(v, \mathcal{G}_d)$ and $\eta_l(v, \mathcal{G}_d)$. Thus, we only need to store these values for all vertices to answer all possible queries.

The DUCS index comprises two sequence arrays $\text{Seq}_{\eta}^{\text{in}}$ and $\text{Seq}_{\eta}^{\text{out}}$. They store, for each vertex, all possible values of $\eta_k(v, \mathcal{G}_d)$ and $\eta_l(v, \mathcal{G}_d)$, which correspond to all possible (k, η) -core_{in} and (l, η) -core_{out}. Since k and l are bounded, we only need to consider $1 \leq k \leq k_{\max}$ and $1 \leq l \leq l_{\max}$, where $k_{\max} = \max_{v \in V} \text{core}_{\text{in}}(v)$ and $l_{\max} = \max_{v \in V} \text{core}_{\text{out}}(v)$. Following the nesting property from Lemma 2, the values are stored in increasing order of $\eta_k(v, \mathcal{G}_d)$ and $\eta_l(v, \mathcal{G}_d)$ to reduce redundant storage.

Example 5. Consider the directed uncertain graph $\mathcal{G}_d$ in Figure 1a. The DUCS index is shown in Figure 3. Take vertex v_1 as an example. For $k = 1$, $\eta_1(v_1, \mathcal{G}_d) = 0.9$, meaning v_1 satisfies the $(1, 0.9)$ -core_{in} constraint. If $\eta > 0.9$, v_1 does not satisfy the $(1, \eta)$ -core_{in} constraint. For $l = 1$, $\eta_1(v_1, \mathcal{G}_d) = 0.8$, so v_1 satisfies the $(1, 0.8)$ -core_{out} constraint; if $\eta > 0.8$, it does not. From Figure 3, we see that the $(1, 0.8)$ -core_{in} is $\{v_1, v_2, v_4, v_5, v_6, v_7\}$ and the $(2, 0.3)$ -core_{out} is $\{v_1, v_2, v_3, v_4\}$.

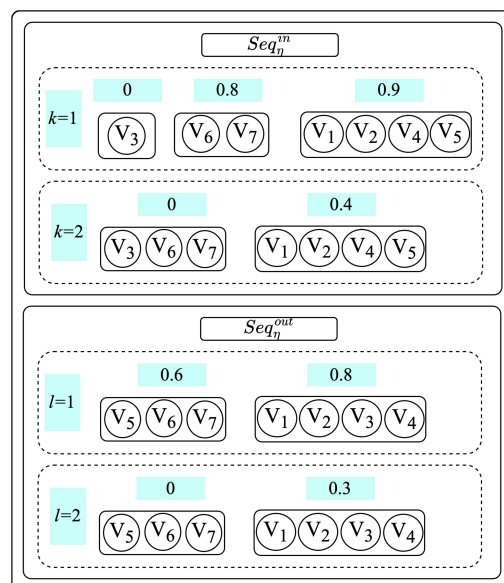


Figure 3. DUCS index of $\mathcal{G}_d$.

5.2. Query Processing

In a directed uncertain graph, edge probabilities may differ between directions for the same vertex. Therefore, we must analyze the inclusion relationships among the (k, l, η) -core, (l, η) -core_{out}, and (k, η) -core_{in}.

Theorem 5. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$ and query parameters k, l, η , suppose there exist subgraphs $\mathcal{G}'_{d_1} = (k, l, \eta)$ -core, $\mathcal{G}'_{d_2} = (k, \eta)$ -core_{in}, and $\mathcal{G}'_{d_3} = (l, \eta)$ -core_{out}. Then:

1. $\mathcal{G}'_{d_1} \subseteq \mathcal{G}'_{d_2}$ and $\mathcal{G}'_{d_1} \subseteq \mathcal{G}'_{d_3}$;
2. $V_{\mathcal{G}'_{d_1}} \subseteq (V_{\mathcal{G}'_{d_3}} \cap V_{\mathcal{G}'_{d_2}})$;
3. $\mathcal{G}'_{d_2}$ and $\mathcal{G}'_{d_3}$ are not necessarily subsets of $\mathcal{G}'_{d_1}$.

Proof. If $\mathcal{G}'_{d_1}$ is a (k, l, η) -core, then for every $v \in V_{\mathcal{G}'_{d_1}}$,

$$\min(\Pr[\text{Deg}_{\text{in}}(v, \mathcal{G}'_d) \geq k], \Pr[\text{Deg}_{\text{out}}(v, \mathcal{G}'_d) \geq l]) \geq \eta.$$

In particular, $\Pr[\text{Deg}_{\text{in}}(v, \mathcal{G}'_d) \geq k] \geq \eta$, so $\mathcal{G}'_{d_1}$ satisfies the (k, η) -core_{in} constraints. Similarly, it satisfies the (l, η) -core_{out} constraints. Thus, $V_{\mathcal{G}'_{d_1}} \subseteq (V_{\mathcal{G}'_{d_3}} \cap V_{\mathcal{G}'_{d_2}})$, establishing (i) and (ii).

Now consider (iii) by contradiction. Suppose $\mathcal{G}'_{d_2}$ (a (k, η) -core_{in}) is a subset of $\mathcal{G}'_{d_1}$ (a (k, l, η) -core). Then, for every $v \in V_{\mathcal{G}'_{d_2}}$,

$$\min(\Pr[\text{Deg}_{\text{in}}(v, \mathcal{G}'_d) \geq k], \Pr[\text{Deg}_{\text{out}}(v, \mathcal{G}'_d) \geq l]) \geq \eta,$$

but the definition of a (k, η) -core_{in} only guarantees $\Pr[\text{Deg}_{\text{in}}(v, \mathcal{G}'_d) \geq k] \geq \eta$, not necessarily the out-degree condition. Hence, $\mathcal{G}'_{d_2}$ may not be contained in $\mathcal{G}'_{d_1}$. The same holds for $\mathcal{G}'_{d_3}$. $\square$

According to Theorem 5, the result obtained directly from the DUCS index is only an upper bound of the true (k, l, η) -core. Therefore, further verification using the online algorithm (Algorithm 1) is required to obtain the exact result.

The DUCS-based query algorithm is presented in Algorithm 4. We first initialize the storage arrays and invoke the DUCS index construction to store vertices satisfying

$\eta_k(v, \mathcal{G}_d) \geq \eta$ and $\eta_l(v, \mathcal{G}_d) \geq \eta$ (lines 1–3). Then, we collect the candidate vertices (lines 4–7). Finally, we call Algorithm 1 to verify the candidate set and obtain the exact result (line 8).

Example 6. Consider $\mathcal{G}_d$ in Figure 1a with $k = 2, l = 1, \eta = 0.4$. The online algorithm yields the connected induced subgraph $\mathcal{G}_d[\{v_1, v_2, v_4\}]$. From the DUCS index in Figure 3, vertex v_5 belongs to both $(2, 0.4)$ -core_{in} and $(1, 0.6)$ -core_{out} (which is a subset of $(1, 0.4)$ -core_{out}), but v_5 does not satisfy the $(2, 1, 0.4)$ -core constraints. The set of vertices satisfying both $(2, 0.4)$ -core_{in} and $(1, 0.4)$ -core_{out} is $V_c = \{v_1, v_2, v_4\}$, a subset of $\{v_1, v_2, v_4, v_5\}$.

Algorithm 4: DUCS-Index Query

Input: An integer k , an integer l , a probability threshold η , and the DUCS index of

$\mathcal{G}_d = (V, E, p)$

Output: The (k, l, η) -core of $\mathcal{G}_d$

```

1 Initialize  $res_{in} \leftarrow \emptyset, res_{out} \leftarrow \emptyset$ ;
2  $res_{in} \leftarrow$  the DUCS index of  $\mathcal{G}_d$  for  $k$ ;
3  $res_{out} \leftarrow$  the DUCS index of  $\mathcal{G}_d$  for  $l$ ;
4 Initialize  $res \leftarrow \emptyset$ ;
5 foreach vertex  $v \in V$  do
6   if  $res_{in}(v) \geq \eta$  and  $res_{out}(v) \geq \eta$  then
7      $res \leftarrow res \cup \{v\}$ ;
8 Apply lines 11–22 of Algorithm 1 on the induced subgraph of  $res$ ;
9 return  $res$ ;
```

Theorem 6. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$ and parameters k, l, η , the time complexity of Algorithm 4 is

$$O\left(|V| + \sum_{v \in V'} \left(\max(\eta \cdot \text{Deg}_{in}(v), \eta \cdot \text{Deg}_{out}(v)) \cdot \max(\text{Deg}_{in}(v), \text{Deg}_{out}(v)) \right)\right),$$

where V' is the set of vertices that require further validation by Algorithm 1.

Proof. Searching the arrays res_{in} and res_{out} for vertices satisfying the conditions takes $O(|V|)$ time (lines 5–7). By Theorem 1, validating the candidate vertex set V' using the online algorithm costs $O(|V| + \sum_{v \in V'} (\max(\eta \cdot \text{Deg}_{in}(v), \eta \cdot \text{Deg}_{out}(v)) \cdot \max(\text{Deg}_{in}(v), \text{Deg}_{out}(v))))$. Hence, the total complexity follows. $\square$

5.3. Index Construction

Following Definition 12, we construct the sequences Seq_{η}^{in} and Seq_{η}^{out} in a bottom-up manner for $k \in [1, k_{\max}]$ and $l \in [1, l_{\max}]$.

Algorithm 5 presents the DUCS index construction. For clarity, we denote $\text{Pr}_{in}(v) = \text{Pr}[\text{Deg}_{in}(v, \mathcal{G}) \geq k]$ and $\text{Pr}_{out}(v) = \text{Pr}[\text{Deg}_{out}(v, \mathcal{G}) \geq l]$. First, we compute core_{in} and core_{out} for each vertex (line 1). Then, we initialize arrays $Seq_{\eta}^{in}(v)$ and $Seq_{\eta}^{out}(v)$ (line 2). Next, we compute $\text{Pr}_{in}(v)$ and $\text{Pr}_{out}(v)$ based on the k -core and l -core decompositions, and use the SEQ procedure to obtain the index arrays (lines 3–10). We then compute $\eta_k(v, \mathcal{G}_d)$ and $\eta_l(v, \mathcal{G}_d)$ for each vertex (lines 12–26). Specifically, we repeatedly remove the vertex v with the smallest Pr_{in} or Pr_{out} , delete its incident edges, and update the probabilities of affected neighbors. Finally, we return the stored values (line 27).

Algorithm 5: DUCS-Index Construction

Input: An uncertain graph $\mathcal{G}_d = (V, E, p)$
Output: DUCS-Index of $\mathcal{G}_d$

- 1 $k_{\max}, l_{\max} \leftarrow$ the largest $core_{in}, core_{out}$ number of all vertices of $\mathcal{G}_d$;
- 2 Initialize $Seq_{\eta}^{in}(v) \leftarrow \emptyset$ and $Seq_{\eta}^{out}(v) \leftarrow \emptyset$ for each $v \in V$;
- 3 **for** $k \leftarrow 1$ **to** $k_{\max}$ **do**
- 4 $g \leftarrow \mathcal{G}_d$;
- 5 **for** $v \in V$ **do**
- 6 compute $Pr_{in}(v)$;
- 7 $Seq_{\eta}^{in}(v) \leftarrow Seq(g, Pr_{in})$;
- 8 **for** $l \leftarrow 1$ **to** $l_{\max}$ **do**
- 9 compute $Pr_{out}(v)$ like Lines 4–6;
- 10 $Seq_{\eta}^{out}(v) \leftarrow Seq(g, Pr_{out})$;
- 11 **return** all $Seq_{\eta}^{in}(v)$ and $Seq_{\eta}^{out}(v)$ for each $v \in V$;
- 12 **Procedure** $Seq(g, Pr)$;
- 13 Initialize $curP \leftarrow 0, res(v) \leftarrow \emptyset$ for each $v \in V$;
- 14 **while** g is not empty **do**
- 15 $v \leftarrow \operatorname{argmin}_{u \in V_g} Pr(u)$;
- 16 $curP \leftarrow \max(Pr(v), curP)$;
- 17 $res(v) \leftarrow curP$;
- 18 $V_g \leftarrow V_g \setminus \{v\}$;
- 19 **if** $Pr = Pr_{in}$ **then**
- 20 **foreach** $u \in \mathcal{N}_{in}(v, g)$ **do**
- 21 remove edge (u, v) from g ;
- 22 update $Pr_{out}(u, g)$;
- 23 **else**
- 24 **foreach** $u \in \mathcal{N}_{out}(v, g)$ **do**
- 25 remove edge (v, u) from g ;
- 26 update $Pr_{in}(u, g)$;
- 27 **return** $res(v)$ of all vertices in g ;

Theorem 7. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, the time complexity of Algorithm 5 is

$$O(k_{\max}^2 |E_{in}| + l_{\max}^2 |E_{out}| + (k_{\max} + l_{\max}) \cdot |V| \log |V|),$$

where $|E_{in}|$ and $|E_{out}|$ are the total in-degree and out-degree of $\mathcal{G}_d$.

Proof. Computing $core_{in}$ and $core_{out}$ takes $O(\max(|E_{in}|, |E_{out}|))$ (line 1). According to [12], calculating $\eta\text{-Deg}_{in}$ and $\eta\text{-Deg}_{out}$ costs $O(\sum_{v \in V} (\eta\text{-Deg}_{in}(v) \cdot \text{Deg}_{in}(v)))$ and $O(\sum_{v \in V} (\eta\text{-Deg}_{out}(v) \cdot \text{Deg}_{out}(v)))$ (lines 5–6 and 8–9). Selecting the minimum value and deleting a vertex costs $O(|V| \log |V|)$ (lines 15–18). Updating Pr_{in} and Pr_{out} for vertices costs $O(\sum_{v \in V} \text{Deg}_{in}(v))$ and $O(\sum_{v \in V} \text{Deg}_{out}(v))$ (lines 20–26). Since $\eta\text{-Deg}_{in}(v) \leq k_{\max}$ and $\text{Deg}_{out}(v) \leq l_{\max}$, the total complexity follows. $\square$

Theorem 8. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, the space complexity of Algorithm 5 is

$$O\left(\sum_{v \in V} (\text{core}_{in}(v) + \text{core}_{out}(v))\right).$$

Proof. For each $v \in V$, we maintain two arrays $Seq_{\eta}^{\text{in}}(v)$ and $Seq_{\eta}^{\text{out}}(v)$ whose lengths are at most $\text{core}_{\text{in}}(v)$ and $\text{core}_{\text{out}}(v)$, respectively. Hence, the total space is bounded by the sum over all vertices of these two core numbers. $\square$

6. Hybrid Index

6.1. Index Structure

As discussed in the previous sections, the DUCS-E index suffers from prohibitively large construction overhead, while the DUCS index requires additional verification, limiting its efficiency gains. Both issues are closely related to the query parameters k and l . To further analyze this, we introduce Lemma 3.

Lemma 3. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, let $V_{(k,l,\eta)}$ denote the vertex set of a (k, l, η) -core. Then:

1. if $k_1 < k_2$, then $V_{(k_2,l,\eta)} \subseteq V_{(k_1,l,\eta)}$;
2. if $l_1 < l_2$, then $V_{(k,l_2,\eta)} \subseteq V_{(k,l_1,\eta)}$.

Proof. For case (i): suppose $\mathcal{G}'_{d_1} = (k_1, l, \eta)$ -core and $\mathcal{G}'_{d_2} = (k_2, l, \eta)$ -core with $k_1 < k_2$. By Definition 5, for every $v \in V_{\mathcal{G}'_{d_2}}$,

$$\min(\Pr[\text{Deg}_{\text{in}}(v, V_{\mathcal{G}'_{d_2}}) \geq k_2 \geq k_1], \Pr[\text{Deg}_{\text{out}}(v, V_{\mathcal{G}'_{d_2}}) \geq l]) \geq \eta.$$

Hence, $\mathcal{G}'_{d_2}$ also satisfies the (k_1, l, η) -core constraints, so $V_{\mathcal{G}'_{d_2}} \subseteq V_{\mathcal{G}'_{d_1}}$, i.e., $V_{(k_2,l,\eta)} \subseteq V_{(k_1,l,\eta)}$. Case (ii) is proved similarly. $\square$

According to Lemma 3, decreasing either k or l enlarges the vertex set of the (k, l, η) -core. Consequently, as shown in Theorem 2, smaller k and l lead to a larger upper bound when using the DUCS index, incurring higher verification cost. Moreover, Theorems 7 and 8 indicate that the DUCS-E index construction is time-consuming and memory-intensive. To overcome these drawbacks, we propose a hybrid index that combines DUCS and DUCS-E.

Example 7. Both DUCS and DUCS-E indices are built for all $k \in [1, k_{\max}]$ and $l \in [1, l_{\max}]$. We choose two threshold parameters ε_{in} and ε_{out} to partition the ranges, as illustrated in Figure 4. Specifically,

$$k \in [1, \varepsilon_{\text{in}}] \cup [\varepsilon_{\text{in}} + 1, k_{\max}], \quad l \in [1, \varepsilon_{\text{out}}] \cup [\varepsilon_{\text{out}} + 1, l_{\max}].$$

For $k \in [1, \varepsilon_{\text{in}}]$ and $l \in [1, \varepsilon_{\text{out}}]$, only the DUCS-E index is stored; for $k \in [\varepsilon_{\text{in}} + 1, k_{\max}]$ and $l \in [\varepsilon_{\text{out}} + 1, l_{\max}]$, only the DUCS index is stored.

Obviously, the Hybrid index is a generalization of the previously proposed DUCS-E and DUCS indexes: by adjusting the thresholds ε_{in} and ε_{out} to their extreme values, Hybrid reduces to DUCS-E when $\varepsilon_{\text{in}} = k_{\max}$ and $\varepsilon_{\text{out}} = l_{\max}$, and reduces to DUCS when $\varepsilon_{\text{in}} = 0$ and $\varepsilon_{\text{out}} = 0$. For intermediate threshold values, Hybrid achieves a tunable balance between query efficiency and index overhead. Thus, DUCS-E and DUCS can be viewed as two extreme points on the time-space trade-off spectrum, with Hybrid providing a practical and flexible solution that covers the entire spectrum.

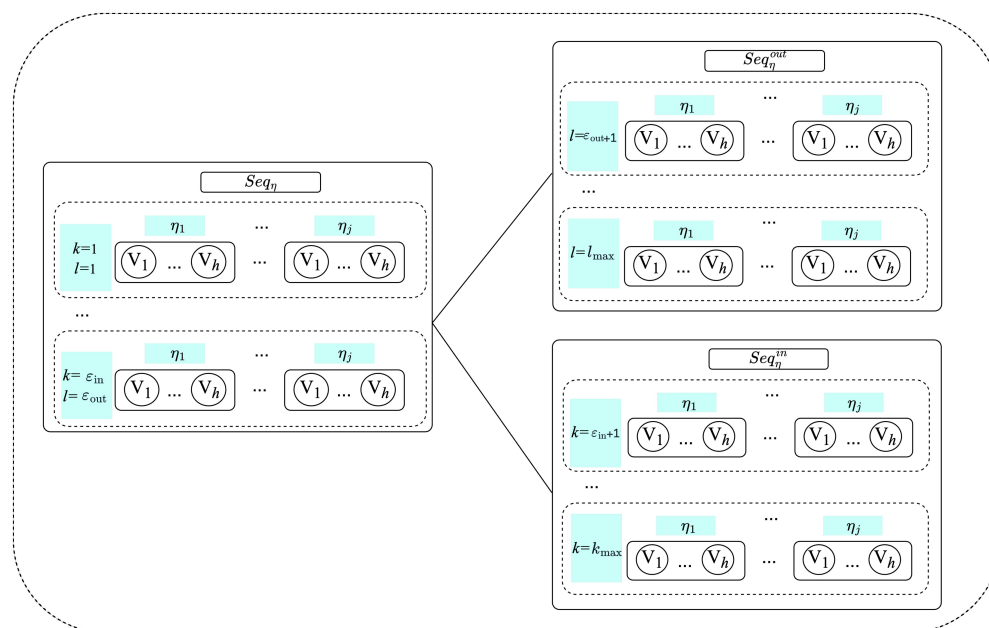


Figure 4. Hybrid index of $\mathcal{G}_d$.

6.2. Query Processing

Given query parameters k, l and thresholds $\epsilon_{in}, \epsilon_{out}$, three query types arise:

- $k \leq \epsilon_{in}$ and $l \leq \epsilon_{out}$;
- $k > \epsilon_{in}$ and $l > \epsilon_{out}$;
- $(k \leq \epsilon_{in} \text{ and } l > \epsilon_{out})$ or $(k > \epsilon_{in} \text{ and } l \leq \epsilon_{out})$.

Different strategies are applied:

- Type (a):** directly use the DUCS-E index to obtain the exact result.
- Type (b):** use the DUCS index to obtain an upper bound set, then apply the online algorithm for verification.
- Type (c):** first use the DUCS-E index to obtain a vertex set containing the result; then use the DUCS index to obtain another vertex set; prune based on containment; finally verify the pruned set with the online algorithm.

Example 8. Consider a type (c) query with $k \leq \epsilon_{in}$ and $l > \epsilon_{out}$. The DUCS index stores subgraphs for $k_1 \geq \epsilon_{in}$, $l_1 \geq \epsilon_{out}$, while the DUCS-E index stores subgraphs for $k_2 < \epsilon_{in}$, $l_2 < \epsilon_{out}$. Thus, there exists $l'_1 = l_1$ such that $(\epsilon_{in}, l'_1, \eta)$ -core $\subseteq (k, l, \eta)$ -core, and $k'_2 = k_2$ such that (k, l, η) -core $\subseteq (k'_2, \epsilon_{out}, \eta)$ -core. As shown in Figure 5, let V , V_1 , and V_2 be the vertex sets of (k, l, η) -core, $(\epsilon_{in}, l'_1, \eta)$ -core, and $(k'_2, \epsilon_{out}, \eta)$ -core, respectively. Then, the result set $V_c = V'_1 \cup V'_2$, where V'_1 is obtained by removing vertices from V_1 that do not satisfy the $(\epsilon_{in}, l'_1, \eta)$ -core constraint (verified online), and V'_2 is obtained by validating $V_2 \setminus V'_1$ online. The case $k > \epsilon_{in}$ and $l \leq \epsilon_{out}$ is handled symmetrically.

The hybrid index query algorithm is presented in Algorithm 6. It first obtains the hybrid index (line 1), then selects the appropriate query method based on the three cases (line 4), and finally returns the connected components of the resulting vertex set (lines 5–7).

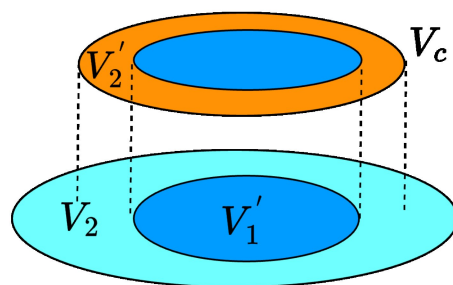


Figure 5. Analysis of type (c) query.

Theorem 9. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, the worst-case time complexity of Algorithm 6 is

$$O\left(\sum_{v \in V'} \max(\eta\text{-Deg}_{in}(v), \eta\text{-Deg}_{out}(v)) \cdot \max(\text{Deg}_{in}(v), \text{Deg}_{out}(v))\right),$$

where V' is the pruned vertex set in type (c) queries.

Proof. For type (a), Theorem 6 gives complexity $O(\sum_{v \in V} \max(\text{Deg}_{out}(v), \text{Deg}_{in}(v)) + |V|)$. For type (b), Theorem 3 gives $O(|V| + \sum_{v \in V'} (\max(\eta\text{-Deg}_{in}(v), \eta\text{-Deg}_{out}(v)) \cdot \max(\text{Deg}_{in}(v), \text{Deg}_{out}(v))))$. For type (c), we first use the DUCS-E query (Theorem 6) to obtain V_2 in $O(\sum_{v \in V} \max(\text{Deg}_{out}(v), \text{Deg}_{in}(v)) + |V|)$. Then, the DUCS index query gives a pruned set V_1' with complexity $O(\sum_{v \in V'} (\max(\eta\text{-Deg}_{in}(v), \eta\text{-Deg}_{out}(v)) \cdot \max(\text{Deg}_{in}(v), \text{Deg}_{out}(v))))$. Finally, verifying $V_2 \setminus V_1'$ online (Theorem 1) costs $O(\sum_{v \in V_2 \setminus V_1'} (\max(\eta\text{-Deg}_{in}(v), \eta\text{-Deg}_{out}(v)) \cdot \max(\text{Deg}_{in}(v), \text{Deg}_{out}(v))))$. Thus, type (c) dominates, yielding the stated bound. $\square$

Algorithm 6: Hybrid-Index Query

Input: Integers k, l , CUR_{in} , CUR_{out} , a probability threshold η , and the hybrid index of $\mathcal{G}_d = (V, E, p)$

Output: The (k, l, η) -core of $\mathcal{G}_d$

- 1 Seq, Seq_{η}^{in} and $Seq_{\eta}^{out} \leftarrow$ the hybrid index of $\mathcal{G}_d$ for (k, l)
 - 2 Initialize $\mathcal{S} \leftarrow \emptyset$
 - 3 Initialize $V_c \leftarrow \emptyset$
 - 4 $V_c \leftarrow \{v \in V \mid v \text{ satisfies the query conditions according to the three cases}\}$
 - 5 **foreach** connected component $\mathcal{G}_d[C] \subseteq \mathcal{G}_d[V_c]$ **do**
 - 6 $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$
 - 7 **return** $\mathcal{S}$
-

6.3. Index Construction

The hybrid index consists of a DUCS-E part for $k \in [1, \varepsilon_{in}]$, $l \in [1, \varepsilon_{out}]$ and a DUCS part for $k \in [\varepsilon_{in} + 1, k_{max}]$, $l \in [\varepsilon_{out} + 1, l_{max}]$. This significantly reduces construction time and space.

Lemma 4. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, if $k_1 < k_2$, $l_1 < l_2$, and $\eta_1 < \eta_2$, then for any vertex v in the (k_2, l_2, η_2) -core, $\eta_{(k_2, l_2)}(v, \mathcal{G}_d) > \eta_1$.

Proof. From the nesting property of k -cores, (k_2, l_2, η_2) -core $\subseteq (k_1, l_1, \eta_2)$ -core. Every vertex v in the latter satisfies $\eta_{(k_1, l_1)}(v, \mathcal{G}_d) \geq \eta_2 > \eta_1$. Since $\eta_{(k_2, l_2)}(v, \mathcal{G}_d) \geq \eta_{(k_1, l_1)}(v, \mathcal{G}_d)$ by definition, the claim follows. $\square$

Yang et al. [14] proposed a top-down index construction exploiting subgraph nesting. Building on this, Lemma 4 allows pruning: when computing (k, l, η) -core, vertices satisfying $\eta_{(k+1, l)}(v, \mathcal{G}_d) > \eta$ can be skipped. Thus, we adopt a top-down approach instead of bottom-up, reducing redundant computations.

Algorithm 7 details the hybrid index construction. First, we obtain $k_{\max}$ and $l_{\max}$ for the deterministic graph G_d (line 1). Then, we initialize thresholds ε_{in} , ε_{out} and the index arrays (lines 2–4). Neighbors are sorted by $\min(\text{core}_{\text{in}}, \text{core}_{\text{out}})$ (line 5). Next, we construct the DUCS-E part (lines 6–10) and the DUCS part (lines 11–18). For DUCS-E, we identify vertices satisfying both in-degree and out-degree structural constraints (line 8), compute Pr_{in} and Pr_{out} (line 9), then iteratively delete the vertex with minimum Pr , updating and storing results (line 10). For DUCS, we sort neighbors separately by core_{in} and core_{out} (line 11), then for each k (lines 12–14) and each l (lines 15–17) we apply the corresponding procedures from Algorithm 3 and Algorithm 5 to build $\text{Seq}_{\eta}^{\text{in}}$ and $\text{Seq}_{\eta}^{\text{out}}$.

Algorithm 7: TopDownHybrid-Index Construction

Input: An uncertain graph $\mathcal{G}_d = (V, E, p)$
Output: Hybrid index of $\mathcal{G}_d$

- 1 $k_{\max}, l_{\max} \leftarrow$ the maximum core_{in} and core_{out} among all vertices of G_d
- 2 Initialize $\varepsilon_{\text{in}}, \varepsilon_{\text{out}}$
- 3 Initialize $\text{Seq}_{\eta}(v) \leftarrow \emptyset$ for each $v \in V$
- 4 Initialize $\text{Seq}_{\eta}^{\text{in}}(v) \leftarrow \emptyset$ and $\text{Seq}_{\eta}^{\text{out}}(v) \leftarrow \emptyset$ for each $v \in V$
- 5 Sort the neighbors of each $u \in V$ by $\min(\text{core}_{\text{in}}, \text{core}_{\text{out}})$
- 6 **for** $k \leftarrow \varepsilon_{\text{in}}$ **to** 1 **do**
- 7 **for** $l \leftarrow \varepsilon_{\text{out}}$ **to** 1 **do**
- 8 $\mathcal{G}'_d(V', E', p) \leftarrow G_d\{u \in V \mid \text{core}_{\text{in}}(u) \geq k, \text{core}_{\text{out}}(u) \geq l\}$
- 9 Algorithm 3, lines 8–11
- 10 Algorithm 3, lines 12–24
- 11 Separately sort the neighbors of each $u \in V$ by core_{in} and by core_{out}
- 12 **for** $k \leftarrow k_{\max}$ **to** $\varepsilon_{\text{in}} + 1$ **do**
- 13 $\mathcal{G}'_d(V', E', p) \leftarrow G_d\{u \in V \mid \text{core}_{\text{in}}(u) \geq k\}$
- 14 Algorithm 5, lines 5–6
- 15 **for** $l \leftarrow l_{\max}$ **to** $\varepsilon_{\text{out}} + 1$ **do**
- 16 $\mathcal{G}'_d(V', E', p) \leftarrow G_d\{u \in V \mid \text{core}_{\text{in}}(u) \geq l\}$
- 17 Algorithm 5, lines 8–10
- 18 **return** all $\text{Seq}_{\eta}(v)$, $\text{Seq}_{\eta}^{\text{in}}(v)$ and $\text{Seq}_{\eta}^{\text{out}}(v)$

Theorem 10. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, the time complexity of Algorithm 7 is

$$O\left(\max(|E_{\text{in}}|, |E_{\text{out}}|) + k_{\max}^2 |E_{\text{in}}| + l_{\max}^2 |E_{\text{out}}| + (k_{\max} + l_{\max}) \cdot |V| \log |V|\right).$$

Proof. The complexity is dominated by the DUCS-E and DUCS index constructions. By Theorem 7, top-down DUCS-E construction costs $O((\varepsilon_{\text{in}} \varepsilon_{\text{out}}) \cdot (\max(\varepsilon_{\text{in}}, \varepsilon_{\text{out}}) \cdot |V| \log |V| + \varepsilon_{\text{in}} |E_{\text{in}}| + \varepsilon_{\text{out}} |E_{\text{out}}|))$. By Theorem 4, DUCS construction costs $O(k_{\max}^2 |E_{\text{in}}| + l_{\max}^2 |E_{\text{out}}| + (k_{\max} + l_{\max}) |V| \log |V|)$. Since $\varepsilon_{\text{in}} \leq k_{\max}$ and $\varepsilon_{\text{out}} \leq l_{\max}$, the total simplifies to the stated bound. $\square$

Theorem 11. Given a directed uncertain graph $\mathcal{G}_d = (V, E, p)$, the space complexity of Algorithm 7 is

$$O\left(\sum_{v \in V} (k_{\max} - \varepsilon_{\text{in}} + l_{\max} - \varepsilon_{\text{out}} + \varepsilon_{\text{in}} \cdot \varepsilon_{\text{out}})\right).$$

Proof. The DUCS index stores information for $k \in [\varepsilon_{\text{in}} + 1, k_{\text{max}}]$ and $l \in [\varepsilon_{\text{out}} + 1, l_{\text{max}}]$. From Theorem 5 (adapted to per-vertex sums), its space complexity is $O(\sum_{v \in V} (k_{\text{max}} - \varepsilon_{\text{in}} + l_{\text{max}} - \varepsilon_{\text{out}}))$. The DUCS-E index stores information for $k \in [1, \varepsilon_{\text{in}}]$ and $l \in [1, \varepsilon_{\text{out}}]$. By Theorem 8, its space complexity is $O(\sum_{v \in V} (\varepsilon_{\text{in}} \cdot \varepsilon_{\text{out}}))$. Summing gives the result. $\square$

7. Algorithm Comparison

We now compare the time and space complexities of the Baseline, DUCS, DUCS-E, and Hybrid algorithms.

Table 1 shows that the Baseline algorithm has the highest query time due to exhaustive computation. DUCS-E is the most efficient, while DUCS and Hybrid trade off some efficiency for lower index size.

Table 1. Comparison of query time across different algorithms.

Algorithm	Time Complexity
Baseline	$O(V + \sum_{v \in V} (\max(\eta\text{-Deg}_{\text{in}}(v), \eta\text{-Deg}_{\text{out}}(v)) \cdot \max(\text{Deg}_{\text{in}}(v), \text{Deg}_{\text{out}}(v))))$
DUCS-E	$O(\sum_{v \in V} \max(\text{Deg}_{\text{out}}(v), \text{Deg}_{\text{in}}(v)) + V)$
DUCS	$O(V + \sum_{v \in V'} (\max(\eta\text{-Deg}_{\text{in}}(v), \eta\text{-Deg}_{\text{out}}(v)) \cdot \max(\text{Deg}_{\text{in}}(v), \text{Deg}_{\text{out}}(v))))$
Hybrid	$O(\sum_{v \in V'} (\max(\eta\text{-Deg}_{\text{in}}(v), \eta\text{-Deg}_{\text{out}}(v)) \cdot \max(\text{Deg}_{\text{in}}(v), \text{Deg}_{\text{out}}(v))))$

Table 2 indicates that DUCS-E incurs the highest construction cost, while DUCS and Hybrid are more scalable, especially Hybrid which balances between the two.

Table 2. Comparison of index construction time across different algorithms.

Algorithm	Time Complexity
DUCS-E	$O((k_{\text{max}} \cdot l_{\text{max}}) \cdot (\max(k_{\text{max}}, l_{\text{max}}) \cdot (V \log V + E)))$
DUCS	$O(k_{\text{max}}^2 E_{\text{in}} + l_{\text{max}}^2 E_{\text{out}} + (k_{\text{max}} + l_{\text{max}}) \cdot V \log V)$
Hybrid	$O(\max(E_{\text{in}} , E_{\text{out}}) + k_{\text{max}}^2 E_{\text{in}} + l_{\text{max}}^2 E_{\text{out}} + (k_{\text{max}} + l_{\text{max}}) \cdot V \log V)$

Table 3 highlights the storage trade-offs. DUCS-E requires the most space, DUCS the least, and Hybrid offers a middle ground, making it suitable for large-scale applications.

Table 3. Index size comparison across different algorithms.

Algorithm	Space Complexity
DUCS-E	$O(\sum_{v \in V} (\text{core}_{\text{in}}(v) \cdot \text{core}_{\text{out}}(v)))$
DUCS	$O(\sum_{v \in V} (\text{core}_{\text{in}}(v) + \text{core}_{\text{out}}(v)))$
Hybrid	$O(\sum_{v \in V} (k_{\text{max}} - \varepsilon_{\text{in}} + l_{\text{max}} - \varepsilon_{\text{out}} + \varepsilon_{\text{in}} \varepsilon_{\text{out}}))$

In summary, DUCS-E provides the fastest queries but at high construction cost and memory usage. DUCS is lightweight but requires additional verification. Hybrid combines the strengths of both, achieving a practical balance between query efficiency, construction time, and index size.

8. Experiments

8.1. Experimental Setup

All experiments were conducted on a Linux server equipped with a 2.10 GHz Intel(R) Xeon(R) Gold 5218R CPU and 1 TB of DDR4 RAM. The algorithms under comparison fall into two main categories: query algorithms and index construction algorithms.

In the experimental evaluation, the Baseline algorithm refers to Algorithm 1. The query algorithm and index construction algorithm based on the DUCS-E index are denoted as Algorithms 2 and 3, respectively. Similarly, the DUCS-based query and construction algorithms are Algorithms 4 and 5, and the Hybrid-based ones are Algorithms 6 and 7.

All algorithms were implemented in C++ and compiled with the g++ compiler using the -O3 optimization level. Experiments that ran for more than 24 h were terminated prematurely.

Evaluation Metrics. (1) (k, l, η) -core query efficiency; (2) index size and construction efficiency; (3) community member similarity comparison; (4) scalability.

Datasets. We use six real-world datasets from SNAP (<https://snap.stanford.edu/> (accessed on 7 July 2025)). Table 4 summarizes the number of vertices, edges, maximum in-degree, maximum out-degree, maximum in-core number, and maximum out-core number for each dataset.

Table 4. Dataset statistics.

Dataset	$ V $	$ E $	$k_{\max}$	$l_{\max}$
wiki-Vote	8297	103,689	19	15
web-Stanford	281,903	2,312,497	41	41
web-Google	875,713	5,105,039	32	32
web-BerkStan	685,230	7,600,595	162	162
soc-Pokec	1,632,803	30,622,564	32	31
soc-LiveJournal1	4,847,571	68,993,773	252	253

- **wiki-Vote:** a voting network on Wikipedia, where nodes represent users and edges represent voting relationships. We assign each edge a uniformly random probability in $[0, 1]$ to indicate the likelihood of one user voting for another.
- **web-Stanford, web-Google, web-BerkStan:** web graphs where nodes represent web pages and edges represent hyperlinks. Each edge is assigned a uniformly random probability in $[0, 1]$ representing the likelihood of a hyperlink from one page to another.
- **soc-Pokec, soc-LiveJournal1:** social networks where nodes represent users and edges represent social connections. Again, we assign a uniformly random probability in $[0, 1]$ to each edge to indicate the probability of a friendship or connection.

8.2. Parameters Setting

Setting ε_{in} and ε_{out} . The key to the Hybrid index is choosing appropriate ε_{in} and ε_{out} . Figure 6 shows how the construction time of the Hybrid index and the query time vary with $\varepsilon_{\text{in}}, \varepsilon_{\text{out}}$. The x -axis represents the percentage of $k_{\max}$ and $l_{\max}$ (e.g., 10% means $\varepsilon_{\text{in}} = 0.1k_{\max}, \varepsilon_{\text{out}} = 0.1l_{\max}$). The left y -axis is the construction time, the right y -axis is the query time for type-(b) queries (both parameters larger than the thresholds). As $\varepsilon_{\text{in}}, \varepsilon_{\text{out}}$ increase, construction time rises while query time falls.

Figure 6a (smaller datasets) shows that query time stabilizes when $\varepsilon_{\text{in}}, \varepsilon_{\text{out}}$ reach 30%. Hence, we set $\varepsilon_{\text{in}} = 0.3k_{\max}, \varepsilon_{\text{out}} = 0.3l_{\max}$. For web-Google, this yields construction time

324 s and query time 0.004 s; increasing to 40% increases construction time to 416 s with negligible improvement in query time.

Figure 6b (larger datasets) indicates that query time stabilizes when ε_{in} , ε_{out} are around 30%. For web-BerkStan, setting them to 32 gives construction time 4049 s and query time 0.004 s; increasing to 30% of k_{max} , l_{max} raises construction time to 15,399 s with no improvement in query time. For soc-LiveJournal1, exceeding 30% leads to construction time over 24 h. Therefore, we set $\varepsilon_{in} = \min\{0.3k_{max}, 30\}$ and $\varepsilon_{out} = \min\{0.3l_{max}, 30\}$ in practice.

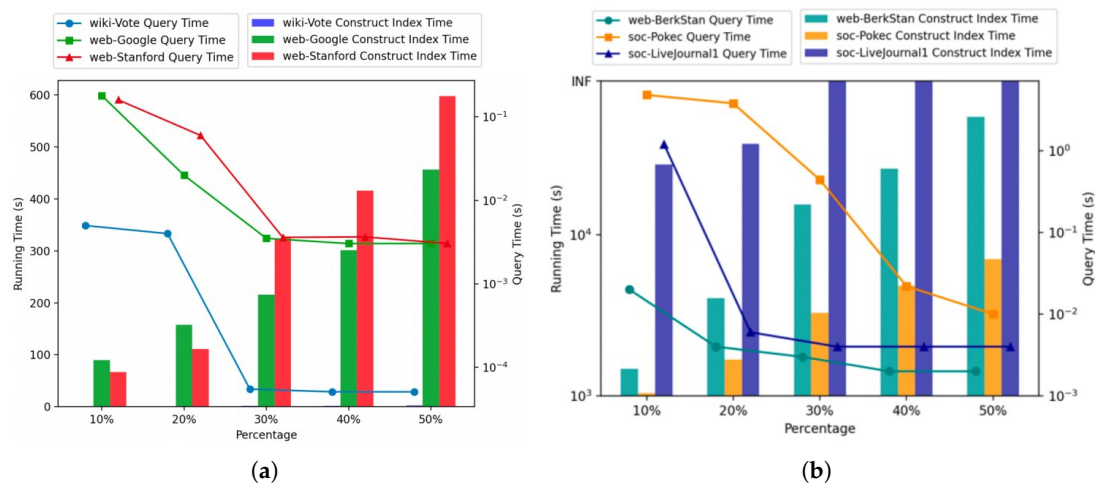


Figure 6. Effects of ε_{in} and ε_{out} on query performance and index construction. (a) Small datasets. (b) Large datasets.

We follow the η settings of [14], namely 0.1, 0.3, 0.5, 0.7, 0.9, with a default value of 0.5. Given ε_{in} and ε_{out} , three types of queries are defined in below, where k and l for each dataset are given in Table 5, due to that different datasets have different k_{max} and l_{max} .

- (a) $k < \varepsilon_{in}$ and $l < \varepsilon_{out}$;
- (b) $k \geq \varepsilon_{in}$ and $l \geq \varepsilon_{out}$;
- (c) $(k < \varepsilon_{in} \text{ and } l \geq \varepsilon_{out}) \text{ or } (k \geq \varepsilon_{in} \text{ and } l < \varepsilon_{out})$.

Table 5. Query parameters k and l settings.

Dataset	Query Type	k	l
wiki-Vote	a	3	4
	b	8	10
	c	4	8
web-Stanford	a	4	8
	b	20	25
	c	5	16
web-Google	a	4	5
	b	15	20
	c	5	15
web-BerkStan	a	5	10
	b	40	40
	c	10	35
soc-Pokec	a	4	5
	b	15	20
	c	5	15
soc-LiveJournal1	a	5	10
	b	40	40
	c	10	35

8.3. Query Efficiency Comparison and Analysis

Effect of η on query efficiency. Figures 7–9 compare the query efficiency of Baseline, DUCS, DUCS-E, and Hybrid for the three query types, varying η while keeping k and l fixed.

Figure 7 shows the query time for type-(a) queries. The Hybrid and DUCS-E algorithms achieve the best performance, being at least four orders of magnitude faster than Baseline. On most datasets, they are at least two orders of magnitude faster than DUCS. For example, on soc-LiveJournal1, Baseline exceeds 50 s, DUCS exceeds 10 s, while Hybrid and DUCS-E both stay below 0.1 s. The high cost of Baseline stems from recomputing vertex probabilities for every query. DUCS is slower because it still requires an online verification step. In contrast, Hybrid and DUCS-E directly return results using the precomputed index.

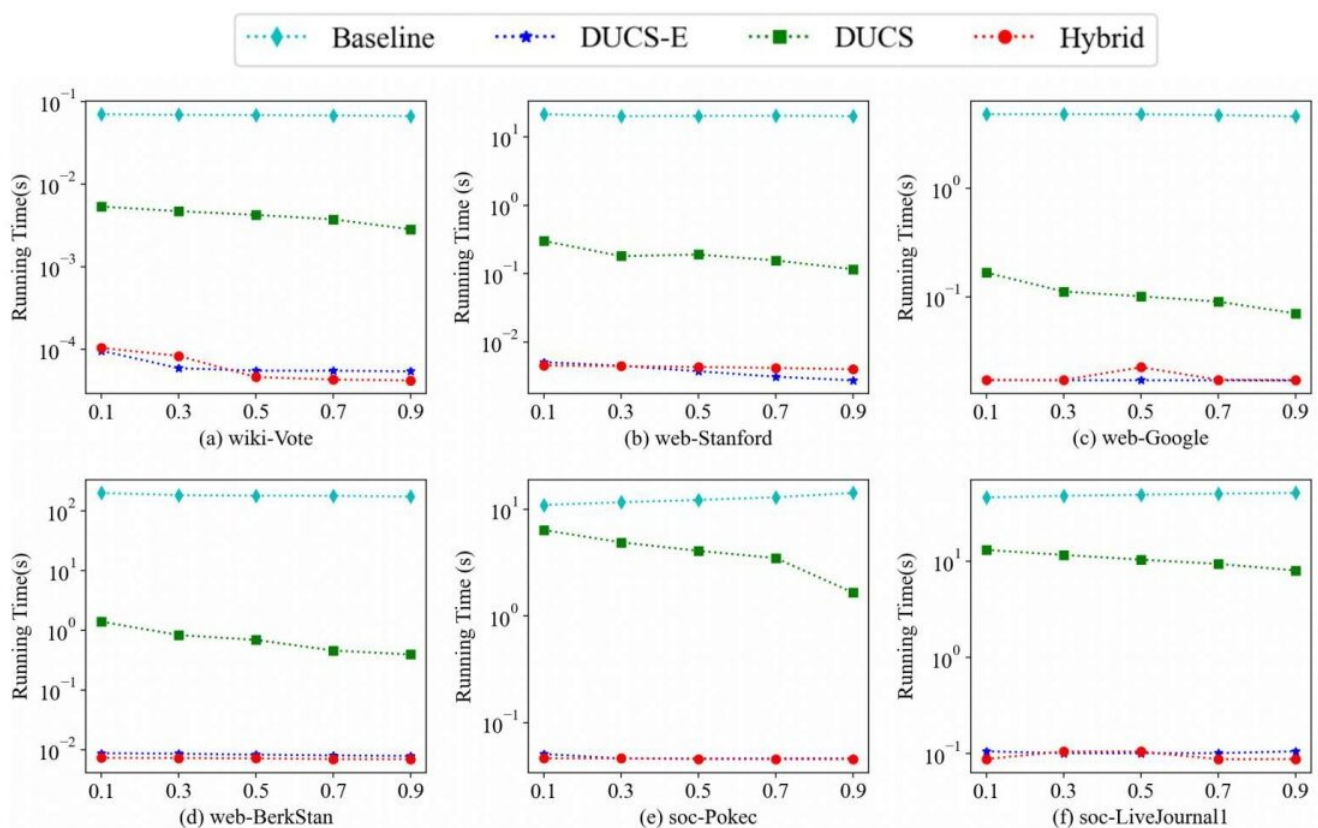


Figure 7. Query time for different η (type-a queries).

Figure 8 presents the query time for type-(b) queries. The DUCS, DUCS-E, and Hybrid algorithms have essentially the same performance, all about five orders of magnitude faster than Baseline. This is because type-(b) queries produce a small result set, minimizing the verification overhead for DUCS and Hybrid. For instance, on wiki-Vote with $k = 8$, $l = 10$, no vertex satisfies the conditions. On denser datasets such as soc-LiveJournal1, DUCS and Hybrid still have query times above 0.1 s, while DUCS-E remains below 0.1 s due to the larger verification cost.

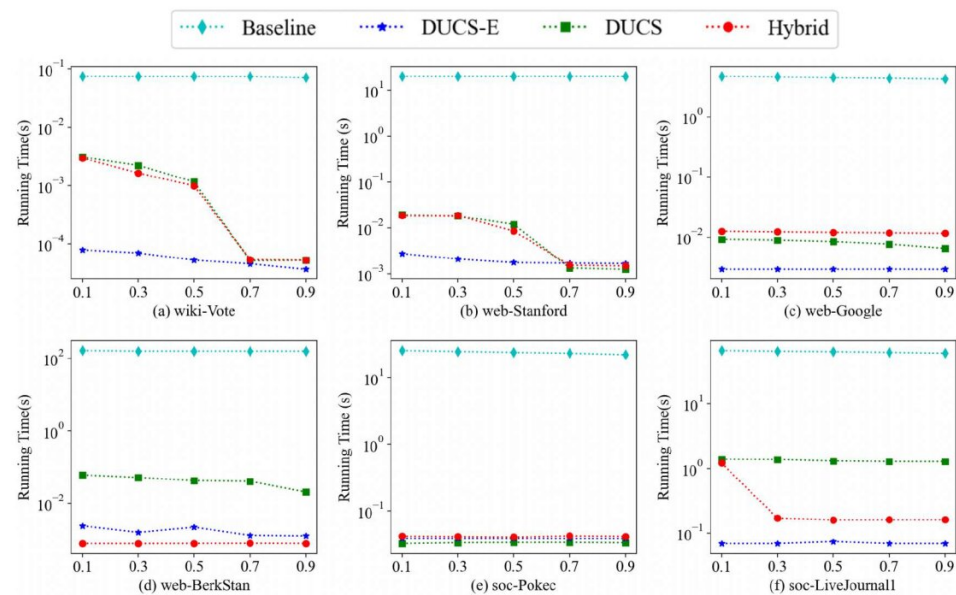


Figure 8. Query time for different η (type-b queries).

Figure 9 shows the query time for type-(c) queries. The DUCS-E algorithm exhibits stable query times. On smaller datasets, the query times of DUCS and Hybrid decrease significantly as η increases. On larger datasets, Hybrid is considerably faster than DUCS, because Hybrid prunes unnecessary vertices before the validation step.

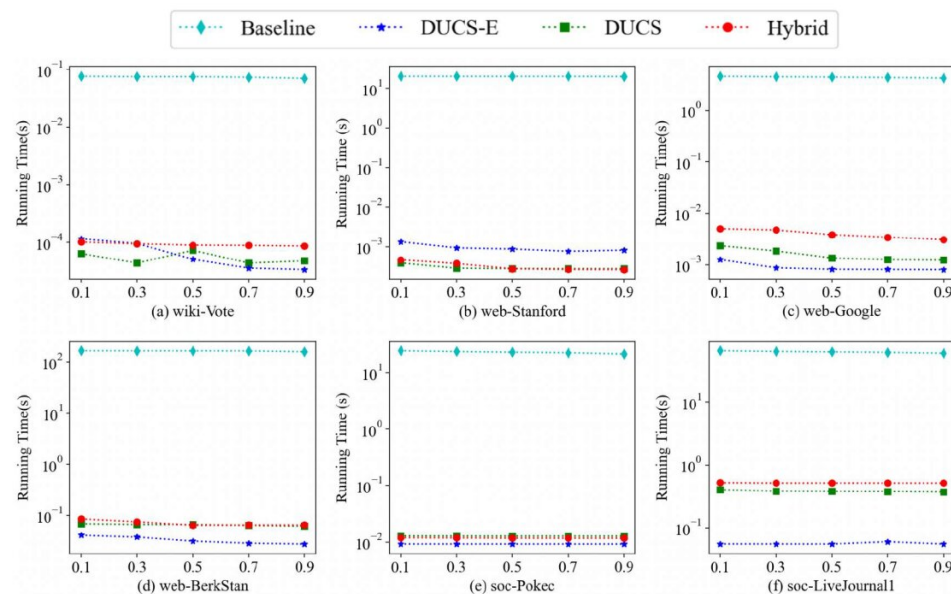


Figure 9. Query time for different η (type-c queries).

Effect of k and l on query efficiency. Figure 10 (varying k) and Figure 11 (varying l) illustrate the impact of k and l on query time, with $\eta = 0.5$.

DUCS-E consistently outperforms the other methods, being three to five orders of magnitude faster than Baseline. For example, on web-BerkStan with $(k, l, \eta) = (10, 10, 0.5)$, Baseline takes 187 s while DUCS-E takes only 7 ms. The query time of Baseline and DUCS-E

remains relatively stable as k or l increases, because these algorithms are less sensitive to parameter changes.

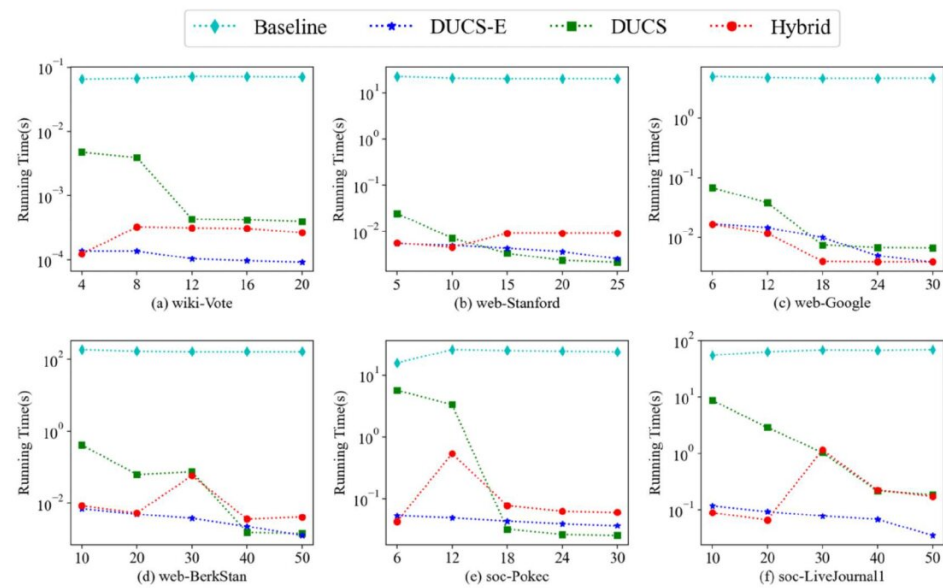


Figure 10. Query time for different k ($\eta = 0.5$).

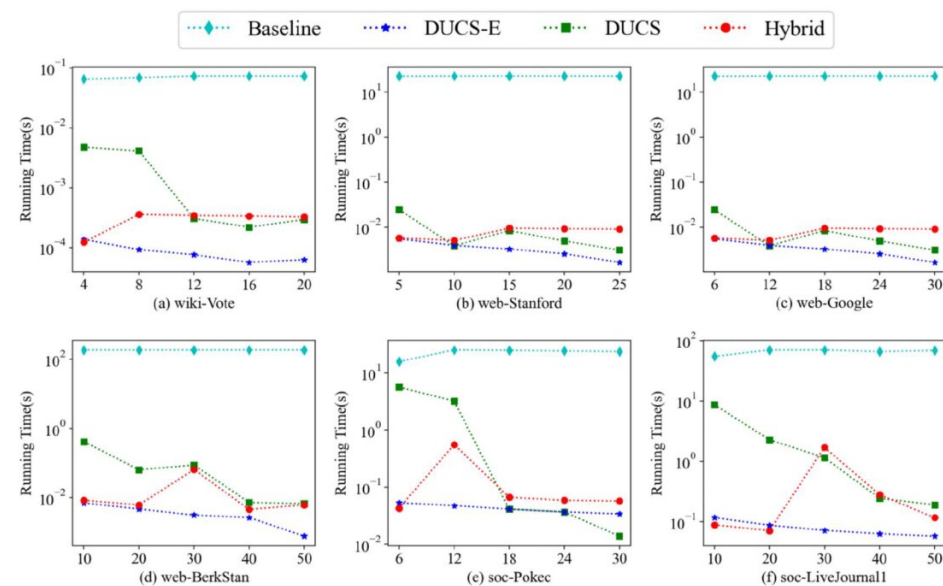


Figure 11. Query time for different l ($\eta = 0.5$).

As k (or l) increases, the query time of DUCS gradually decreases, because the size of the query subgraph shrinks, reducing verification cost. On soc-LiveJournal1, DUCS takes 8.74 s for $(k, l) = (10, 10)$ but only 0.18 s for $(10, 50)$.

The Hybrid algorithm's query time first increases and then decreases. When k and l are below the thresholds $\varepsilon_{in}, \varepsilon_{out}$, the query is type-(a) and results are obtained directly from the index, resulting in low time. When k or l exceeds the thresholds, the query becomes type-(c) and requires verification, increasing the time. As k or l further increases, the number of qualifying vertices drops, and the verification cost falls again. For instance, on soc-Pokec

with $\varepsilon_{in} = \varepsilon_{out} = 10$, Hybrid takes 0.04 s for (6, 6), 0.54 s for (6, 12), and 0.06 s for (6, 30) (no vertices meet the criteria).

In summary, Baseline is unsuitable for efficient query processing. DUCS is generally slower than DUCS-E and Hybrid. DUCS-E and Hybrid achieve the best query efficiency, especially on larger datasets: for type-(a), DUCS-E and Hybrid are optimal; for type-(b), all three index-based algorithms perform well; for type-(c), DUCS-E maintains the most stable and best performance.

8.4. Index Size and Index Construction Time

Index size comparison. From Figure 12a, the DUCS-E index is the largest, followed by Hybrid, and DUCS is the smallest. DUCS-E stores all (k, l) combinations, requiring the most space. DUCS stores only two approximate indices for in-degree and out-degree separately, thus the smallest. Hybrid stores both a partial precise index and partial approximate indices, resulting in moderate space. For soc-LiveJournal1, DUCS-E exceeds 9000 MB, Hybrid uses 4210 MB, and DUCS uses 1006 MB.

Index construction time comparison. Figure 12b shows that DUCS-E has the longest construction time, especially on large datasets (e.g., soc-LiveJournal1 exceeds 24 h). DUCS is the fastest. On wiki-Vote, Hybrid takes only 1.3 s, ten times faster than DUCS-E. On soc-LiveJournal1, Hybrid finishes in 7.8 h, while DUCS-E exceeds 24 h (at least three times slower).

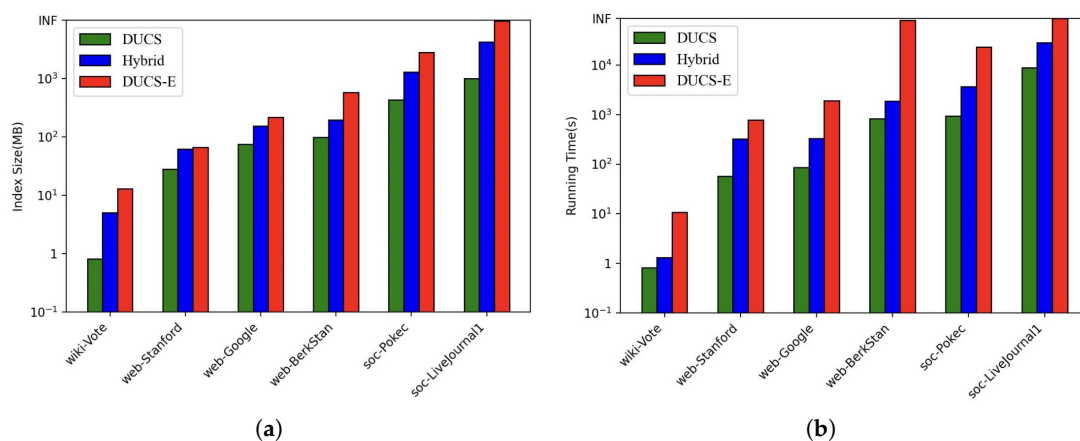


Figure 12. Comparison of index size and construction time. (a) Index size for different datasets. (b) Construction time for different datasets.

8.5. Model Comparison

If edge directions are ignored, a directed graph becomes an undirected graph with fewer edges (bidirectional edges become a single edge). To evaluate effectiveness, we follow [21] and set the (k, η) -core query parameter k to $0.7 \cdot (k + l)$. We choose different (k, l) pairs for different datasets to ensure a fair comparison. We use community member similarity (CMS) as the quality metric.

Community Member Similarity (CMS). A cohesive community should have high internal similarity. We use the Jaccard index to measure similarity between two vertices based on common in-neighbors and out-neighbors. For a vertex set C , we define

$$\text{CMS}(C)^{\text{in}} = \frac{1}{|C|^2} \sum_{u \in C} \sum_{v \in C} \frac{|\text{in}(u) \cap \text{in}(v)|}{|\text{in}(u) \cup \text{in}(v)|}, \quad (7)$$

$$\text{CMS}(C)^{\text{out}} = \frac{1}{|C|^2} \sum_{u \in C} \sum_{v \in C} \frac{|\text{out}(u) \cap \text{out}(v)|}{|\text{out}(u) \cup \text{out}(v)|}. \quad (8)$$

Both measures lie in $[0, 1]$, with larger values indicating higher cohesion. Figure 13 shows that the (k, l, η) -core model yields substantially higher CMS values than the (k, η) -core model on undirected graphs. For example, on soc-LiveJournal1, (k, η) -core gives $\text{CMS}^{\text{in}} = 0.00002$ and $\text{CMS}^{\text{out}} = 0.00001$, while our model gives 0.0018 and 0.0014, respectively—an improvement of over two orders of magnitude.

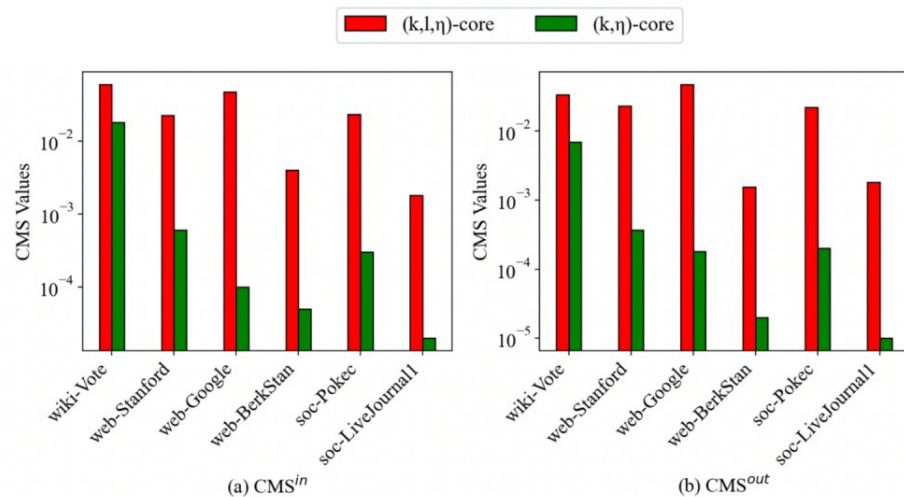


Figure 13. Community member similarity (CMS).

8.6. Scalability Testing

Due to space limitations, we only report scalability tests on the largest dataset, soc-LiveJournal1. As shown in Figure 14, we sampled nodes and edges from 20% to 100% uniformly at random. For node sampling, we take the induced subgraph of the sampled nodes; for edge sampling, we include all incident nodes. The figure shows the index construction time and space consumption for DUCS, DUCS-E, and Hybrid as the graph size increases.

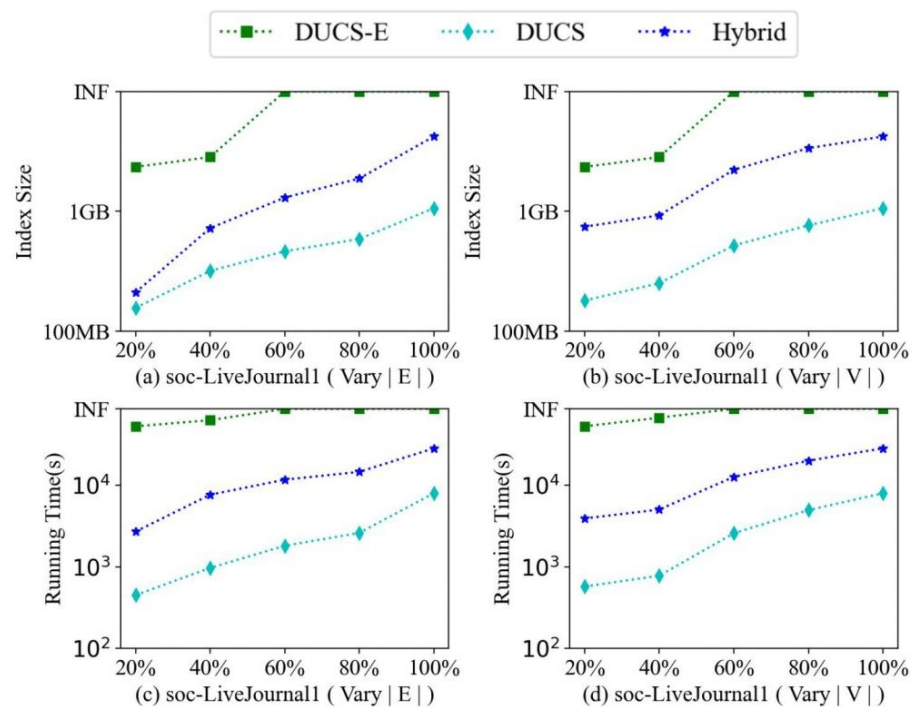


Figure 14. Scalability of index size and index construction time.

All three methods exhibit increasing construction time and space with larger $|V|$ or $|E|$. Hybrid consistently requires less time and space than DUCS-E. These results demonstrate that the proposed Hybrid index construction and DUCS construction scale well.

8.7. Experiments on Software Engineering and Electronics-Related Datasets

Here, we conducted additional experiments on two publicly available datasets, which are related to software engineering and electronics. We present the results separately, due to their relatively small sizes.

The first dataset is the linux kernel function call graph (<https://zenodo.org/record/2652487> (accessed on 18 July 2026)) with 3973 vertices and 30,500 edges, where each vertex represents a C source file in the Linux kernel, and each directed edge (u, v) indicates that source file u calls one or more functions defined in source file v . The direction is naturally irreversible, representing the static dependency relationship between files. The graph captures the inter-file function call dependencies extracted through static analysis of the Linux kernel v4.21 source code. Due to the presence of conditional compilation (e.g., `#ifdef CONFIG_XXX`), the same function call may be included or excluded depending on the compilation configuration, which naturally introduces uncertainty into the dependency graph. The edge probability $p(u, v)$ is computed by counting the frequency of calls from file u to file v in the raw data, and then mapping the frequency to a probability value using a log-scaled transformation. A higher call frequency indicates that the dependency exists under more compilation paths, and thus has a higher probability.

The second dataset is MIT reality mining human contact network (<https://iee-dataport.org/open-access/crawdad-mitreality> (accessed on 18 July 2026)) with 96 vertices and 5078 edges, where each vertex represents a participant (student or faculty member) at MIT Media Lab who carried a Nokia 6600 smartphone equipped with logging software, and each directed edge (u, v) exists if participant u was within Bluetooth range of participant v on at least one day during the 234-day observation period. As physical proximity is symmetric, we assign the same probability to both directions. The dataset captures human physical proximity patterns through Bluetooth device scans, which are representative of wireless sensor networks and IoT communication scenarios. The probability of an edge reflects the frequency and regularity of contact between two individuals over time. The observation period is divided into 234 daily windows. If person A appears within person B 's Bluetooth scan range at least once on a given day, that day is counted as a valid contact day. The edge probability $p(u, v)$ is then computed as the number of valid contact days divided by the total number of days (234). This models the inherent uncertainty in human mobility and interaction patterns.

The experimental results on these two datasets are summarized in Table 6, where parameters are chosen in the same way as Section 8.2. We emphasize that these datasets are considerably smaller than the six large-scale benchmarks (wiki-Vote, web-Stanford, web-Google, web-BerkStan, soc-Pokec, and soc-LiveJournal1) used above. As shown in Table 4 of the main paper, the largest dataset (soc-LiveJournal1) contains over 4.8 million vertices and 68 million edges. Due to the significant scale difference, we present these results separately rather than integrating them into the main experimental section, to maintain the focus on the scalability and efficiency of our methods on large graphs.

Table 6. Average query time and index overhead on additional datasets.

Dataset	Method	Index Size (MB)	Build Time (ms)	Avg. Query Time ($\times 10^{-2}$ ms)
linux	Baseline	–	–	43.20
	DUCS-E	1.09	47.40	2.53
	DUCS	0.36	3.02	42.40
	Hybrid	0.48	7.95	42.20
mit	Baseline	–	–	18.90
	DUCS-E	0.21	552.00	0.71
	DUCS	0.02	0.19	17.00
	Hybrid	0.04	13.00	16.80

The experimental results on both datasets are fully consistent with the main findings reported in this paper:

- **Index-based methods significantly outperform the online baseline:** On linux, DUCS-E achieves a $17\times$ speed-up over Baseline; on mit, DUCS-E achieves a $26\times$ speed-up.
- **DUCS-E achieves the fastest query response:** 2.53×10^{-2} ms on linux and 0.71×10^{-2} ms on mit.
- **DUCS requires the smallest index storage:** 0.36 MB on linux and 0.02 MB on mit.
- **Hybrid provides the best trade-off:** It achieves query times close to DUCS-E (42.20 vs. 2.53 on linux; 16.80 vs. 0.71 on mit) while requiring significantly less storage than DUCS-E (0.48 MB vs. 1.09 MB on linux; 0.04 MB vs. 0.21 MB on mit).

8.7.1. Implications for Software Engineering Applications

The linux dataset demonstrates the practical applicability of our methods to software engineering scenarios:

- **Dependency Analysis:** The (k, l, η) -core can identify a core set of mutually dependent source files in large software systems. Files with both high in-degree (many files depend on them) and high out-degree (they depend on many others) represent the most critical and tightly coupled components.
- **Conditional Compilation Uncertainty:** The edge probabilities derived from conditional compilation frequencies naturally model the uncertainty in build-time dependencies. Our index-based methods enable fast, on-the-fly identification of stable dependency cores under different compilation configurations.
- **Maintenance Prioritization:** By querying different (k, l, η) parameter combinations, software engineers can identify modules that are critical for system stability and should be prioritized for testing, refactoring, and security auditing.

8.7.2. Implications for Electronics/Sensor Network Applications

The mit dataset validates the applicability to wireless sensor networks and IoT scenarios:

- **Contact Pattern Analysis:** The (k, l, η) -core can identify stable and reliable communication clusters in mobile ad-hoc networks, where contacts are probabilistic and time-varying.
- **Network Reliability:** The edge probabilities reflect communication reliability or contact frequency. Our methods enable fast identification of subgraphs where both sending and receiving capabilities exceed desired thresholds with high probability, which is crucial for routing and topology control.
- **Scalability to Larger Sensor Networks:** Although the mit dataset is small (96 nodes), the consistency of our results with large-scale graph experiments suggests that the proposed methods can scale to larger IoT deployments.

These results further strengthen the generalizability of our contributions and confirm the relevance of our work to the software engineering and electronics communities.

8.8. Discussion

8.8.1. Lazy Caching vs. Exhaustive Precomputation

We discuss an alternative strategy suggested: lazy caching (memoization), where the result of a (k, l, η) -core query is computed only upon its first occurrence and cached for subsequent queries. This approach can be efficient when the number of distinct queries is small or the query workload is highly repetitive. However, it has several limitations compared to our exhaustive precomputation strategy.

First, in the worst case where all possible (k, l) combinations are queried at least once, lazy caching degenerates to the online algorithm and incurs the same total computation cost as running all queries from scratch. In contrast, DUCS-E guarantees constant-time response for any query after a single offline construction phase, regardless of the query workload.

Second, the Hybrid index proposed in this paper already bridges the gap between exhaustive precomputation and on-demand computation. By selectively storing DUCS-E for small k, l and DUCS for large k, l , Hybrid avoids the high storage cost of full precomputation while still providing fast query processing for most practical scenarios.

Third, implementing and benchmarking a memoization strategy would require defining a specific query workload distribution (e.g., Zipfian, uniform, or repeated patterns), which is highly application-dependent and may not lead to generally applicable conclusions. Since the focus of this paper is on the algorithmic design of exact (k, l, η) -core query processing, we believe that the systematic comparison among the online algorithm, DUCS, DUCS-E, and Hybrid is sufficient to demonstrate the effectiveness and efficiency of our proposals.

8.8.2. Index Trade-Offs

We highlight the relationship among the three indexing structures from an empirical perspective. As shown in Figures 7–10, DUCS-E consistently achieves the fastest query response time across all settings, at the cost of the largest index size and longest construction time. In contrast, DUCS requires significantly less storage and construction overhead but incurs higher query latency due to the need for online verification. The Hybrid index, by tuning ε_{in} and ε_{out} , seamlessly interpolates between these two extremes, offering a flexible and practical solution that balances query efficiency and index overhead. This confirms that DUCS-E and DUCS represent the upper and lower bounds of both query time and index size, with Hybrid falling in between and providing the best overall trade-off for most application scenarios.

9. Related Work

This section briefly reviews related work on core decomposition in deterministic and uncertain graphs.

9.1. Core Decomposition on Deterministic Graphs

The concept of k -core was introduced by Seidman [25] as a maximal subgraph where each vertex has degree at least k . Core decomposition aims to find all k -cores.

k -core decomposition on undirected graphs. Many algorithms have been proposed, notably peeling-based methods [26–28] that iteratively remove minimum-degree vertices. Parallel and distributed solutions have also been developed, e.g., using MapReduce [27,28] or Spark [29,30].

D-core decomposition on directed graphs. To account for edge direction, the directed core (D-core) or (k, l) -core model was proposed [21,22,31]. Peeling-based algorithms remove vertices that do not satisfy in-degree or out-degree constraints. Later, a distributed D-core decomposition algorithm was introduced [32] using the concept of skyline coreness and H-index to improve efficiency.

9.2. Core Decomposition on Uncertain Graphs

Bonchi et al. [12] first defined the (k, η) -core in uncertain graphs, a maximal subgraph where each vertex has probability at least η of having degree $\geq k$. They proposed a peeling algorithm based on η -degree. Esfahani et al. [33] accelerated the computation using the Lyapunov central limit theorem. Yang et al. [14] designed an index-based optimal algorithm for (k, η) -core queries. Wen et al. [34] extended the index to external memory. Dai et al. [15] addressed precision issues and proposed an exact algorithm.

Existing work on core decomposition for directed graphs does not consider uncertainty, and existing probabilistic core decomposition ignores edge directions. Therefore, none of these methods can be directly applied to (k, l, η) -core queries, motivating our specialized solution.

9.3. Topic-Aware Most Influential Community Search.

Teng et al. [24] recently introduced the TAMICS problem, which also utilizes the (k, l, η) -core model for cohesiveness. However, their work differs from ours in two fundamental aspects. First, their goal is to find the community with the maximum influence score for a given topic, whereas we aim to answer arbitrary (k, l, η) -core queries efficiently. Second, their indexing strategies (TUC-list and TIE-tree) are tailored for influence and topic estimation, while our proposed DUCS, DUCS-E, and Hybrid indexes are specifically designed to optimize core computation and verification. Therefore, our work provides the first dedicated solution for the core problem of efficient (k, l, η) -core query processing.

9.4. Other Related Work

Beyond (k, η) -core, other problems have been studied in uncertain graphs, including influence community search [35], butterfly counting and bitruss decomposition on bipartite graphs [17,36], maximal clique mining [16], reliable clustering [37,38], and k -truss computation [19,20,39].

10. Conclusions

In this paper, we presented the first in-depth study on efficient (k, l, η) -core queries in directed uncertain graphs. Distinct from the concurrent work on TAMICS [24], which focuses on topic-aware influence maximization, our work establishes the foundational building blocks for core decomposition in this setting, providing a family of algorithms and indexes that enable fast, exact core queries. We first present an online query algorithm based on the (k, l, η) -core decomposition. To further optimize query efficiency, we develop two indexing structures, the DUCS-E index and the DUCS index, leading to the DUCS-E Query and DUCS Query algorithms. Furthermore, to enhance query efficiency while reducing index overhead, we propose a hybrid index structure called the Hybrid index, which combines the advantages of DUCS and DUCS-E. The Hybrid index ensures efficient query processing while lowering index construction time and space consumption. Experimental results demonstrate that the index-based query algorithms achieve a four-order-of-magnitude improvement in query speed compared to the online query algorithm, validating the effectiveness and scalability of our proposed methods.

As lazy caching could be beneficial in environments with skewed query distributions, an interesting direction for future work is extending our indexing framework to support workload-aware caching and dynamic index updates.

Author Contributions: Conceptualization, X.T. and G.C.; methodology, X.T.; software, G.C.; validation, J.Z. and X.T.; formal analysis, J.Z.; investigation, X.T.; resources, G.C.; data curation, G.C.; writing—original draft preparation, X.T.; writing—review and editing, J.Z.; visualization, G.C.; supervision, J.Z.; project administration, X.T.; funding acquisition, J.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported by grants from the National Natural Science Foundation of China (No.: 62372101).

Institutional Review Board Statement: This work is not applicable for human or animal studies.

Data Availability Statement: All datasets used in this study are publicly available and discussed in Section 8.

Conflicts of Interest: We declare that we do not have any commercial or associative interests that represent conflicts of interest in connection with the work submitted.

References

- Goyal, A.; Bonchi, F.; Lakshmanan, L.V.S. Learning influence probabilities in social networks. In *Proceedings of the Third International Conference on Web Search and Web Data Mining, WSDM 2010, New York, NY, USA, February 4–6, 2010*; Davison, B.D., Suel, T., Craswell, N., Liu, B., Eds.; ACM: New York, NY, USA, 2010; pp. 241–250. <https://doi.org/10.1145/1718487.1718518>.
- Cui, W.; Xiao, Y.; Wang, H.; Wang, W. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*; ACM: New York, NY, USA, 2014; pp. 991–1002.
- Sun, Z.; Huang, X.; Xu, J.; Bonchi, F. Efficient probabilistic truss indexing on uncertain graphs. In *Proceedings of the Web Conference 2021*; ACM: New York, NY, USA, 2021; pp. 354–366.
- Brohee, S.; Van Helden, J. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinform.* **2006**, *7*, 1–19.
- Bader, G.D.; Hogue, C.W. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinform.* **2003**, *4*, 1–27.
- Duong, K.; Nguyen, T. Mining highly reliable dense subgraphs from uncertain weighted graphs for protein-protein interaction networks. *PeerJ Comput. Sci.* **2025**, *11*, e3304.
- Zhang, Q.; Chu, L.; Zhao, Z.; Pei, J. Finding antagonistic communities in signed uncertain graphs. *IEEE Trans. Knowl. Data Eng.* **2025**, *37*, 655–669.
- Cui, H.; Chang, A.; Zhu, J.; Liu, R.; Hu, K. A vertex partitioning algorithm for large-scale uncertain graphs. *Concurr. Comput. Pract. Exp.* **2026**, *38*, e70580.
- Hu, R.; Kanani, D.H.; Zhang, J. Computing the center of uncertain points on cactus graphs. *Theor. Comput. Sci.* **2026**, *1067*, 115761.
- Xu, H.; Zhang, J. The k-center problem of uncertain points on graphs. *Inf. Process. Lett.* **2026**, *193*, 106621.
- Mehmood, Y.; Bonchi, F.; García-Soriano, D. Spheres of influence for more effective viral marketing. In *Proceedings of the 2016 International Conference on Management of Data*; ACM: New York, NY, USA, 2016; pp. 711–726.
- Bonchi, F.; Gullo, F.; Kaltenbrunner, A.; Volkovich, Y. Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; ACM: New York, NY, USA, 2014; pp. 1316–1325.
- Kawahigashi, H.; Terashima, Y.; Miyauchi, N.; Nakakawaji, T. Modeling ad hoc sensor networks using random graph theory. In *Proceedings of the Second IEEE Consumer Communications and Networking Conference, 2005. CCNC. 2005*; IEEE: Piscataway, NJ, USA, 2005; pp. 104–109.
- Yang, B.; Wen, D.; Qin, L.; Zhang, Y.; Chang, L.; Li, R.H. Index-based optimal algorithm for computing k-cores in large uncertain graphs. In *Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE)*; IEEE: Piscataway, NJ, USA, 2019; pp. 64–75.
- Dai, Q.; Li, R.; Wang, G.; Mao, R.; Zhang, Z.; Yuan, Y. Core decomposition on uncertain graphs revisited. *IEEE Trans. Knowl. Data Eng.* **2023**, *35*, 196–210.
- Li, R.H.; Dai, Q.; Wang, G.; Ming, Z.; Qin, L.; Yu, J.X. Improved algorithms for maximal clique search in uncertain networks. In *Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE)*; IEEE: Piscataway, NJ, USA, 2019; pp. 1178–1189.

17. Mukherjee, A.P.; Xu, P.; Tirthapura, S. Mining maximal cliques from an uncertain graph. In *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering*; IEEE: Piscataway, NJ, USA, 2015; pp. 243–254.
18. Esfahani, F.; Wu, J.; Srinivasan, V.; Thomo, A.; Wu, K. Fast Truss Decomposition in Large-scale Probabilistic Graphs. In *Proceedings of the International Conference on Extending Database Technology*; OpenProceedings.org: Konstanz, Germany, 2019.
19. Sun, Z.; Huang, X.; Xu, J.; Bonchi, F.; Chang, L. Probabilistic truss decomposition on uncertain graphs: Indexing and dynamic maintenance. *ACM Trans. Database Syst.* **2025**, *50*, 6.
20. Xing, B.; Tan, Y.; Zhou, J.; Du, M. Truss community search in uncertain graphs. *Knowl. Inf. Syst.* **2024**, *66*, 7739–7773.
21. Fang, Y.; Wang, Z.; Cheng, R.; Wang, H.; Hu, J. Effective and efficient community search over large directed graphs. *IEEE Trans. Knowl. Data Eng.* **2018**, *31*, 2093–2107.
22. Zhang, J.; Wang, C.; Wang, J. Who proposed the relationship? recovering the hidden directions of undirected social networks. In *Proceedings of the 23rd international Conference on World Wide Web*; ACM: New York, NY, USA, 2014; pp. 807–818.
23. Giatsidis, C.; Thilikos, D.M.; Vazirgiannis, M. D-cores: Measuring collaboration of directed graphs based on degeneracy. *Knowl. Inf. Syst.* **2013**, *35*, 311–343.
24. Teng, L.; Wang, Y.; Lin, Z.; Yu, F. Topic-aware most influential community search in social networks. *Neurocomputing* **2025**, *638*, 130173.
25. Seidman, S.B. Network structure and minimum degree. *Soc. Netw.* **1983**, *5*, 269–287.
26. Khaouid, W.; Barsky, M.; Srinivasan, V.; Thomo, A. K-core decomposition of large networks on a single PC. *Proc. VLDB Endow.* **2015**, *9*, 13–23.
27. Batagelj, V.; Zaversnik, M. An $o(m)$ algorithm for cores decomposition of networks. *arXiv* **2003**, arXiv:cs/0310049.
28. Cheng, J.; Ke, Y.; Chu, S.; Özsu, M.T. Efficient core decomposition in massive networks. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*; IEEE: Piscataway, NJ, USA, 2011; pp. 51–62.
29. Esfandiari, H.; Lattanzi, S.; Mirrokni, V. Parallel and streaming algorithms for k-core decomposition. In *Proceedings of the International Conference on Machine Learning*; PMLR: Cambridge, MA, USA, 2018; pp. 1397–1406.
30. Dasari, N.S.; Desh, R.; Zubair, M. ParK: An efficient algorithm for k-core decomposition on multicore processors. In *Proceedings of the 2014 IEEE International Conference on Big Data (Big Data)*; IEEE: Piscataway, NJ, USA, 2014; pp. 9–16.
31. Li, Z.; Fang, Y.; Liu, Q.; Cheng, J.; Cheng, R.; Lui, J.C. Walking in the cloud: Parallel simrank at scale. *Proc. VLDB Endow.* **2015**, *9*, 24–35.
32. Liao, X.; Liu, Q.; Jiang, J.; Huang, X.; Xu, J.; Choi, B. Distributed d-core decomposition over large directed graphs. *arXiv* **2022**, arXiv:2202.05990.
33. Esfahani, F.; Srinivasan, V.; Thomo, A.; Wu, K. Efficient Computation of Probabilistic Core Decomposition at Web-Scale. In *Proceedings of the International Conference on Extending Database Technology*; OpenProceedings.org: Konstanz, Germany, 2019.
34. Wen, D.; Yang, B.; Qin, L.; Zhang, Y.; Chang, L.; Li, R.H. Computing k-cores in large uncertain graphs: An index-based optimal approach. *IEEE Trans. Knowl. Data Eng.* **2020**, *34*, 3126–3138.
35. Luo, W.; Zhou, X.; Li, K.; Gao, Y.; Li, K. Efficient influential community search in large uncertain graphs. *IEEE Trans. Knowl. Data Eng.* **2021**, *35*, 3779–3793.
36. Zhou, A.; Wang, Y.; Chen, L. Butterfly counting and bitruss decomposition on uncertain bipartite graphs. *VLDB J.* **2023**, *32*, 1013–1036.
37. Han, K.; Gui, F.; Xiao, X.; Tang, J.; He, Y.; Cao, Z.; Huang, H. Efficient and effective algorithms for clustering uncertain graphs. *Proc. VLDB Endow.* **2019**, *12*, 667–680.
38. Liu, L.; Jin, R.; Aggarwal, C.; Shen, Y. Reliable clustering on uncertain graphs. In *Proceedings of the 2012 IEEE 12th International Conference on Data Mining*; IEEE: Piscataway, NJ, USA, 2012; pp. 459–468.
39. Zou, Z.; Zhu, R. Truss decomposition of uncertain graphs. *Knowl. Inf. Syst.* **2017**, *50*, 197–230.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.