




# Core Decomposition, Maintenance and Applications

Feiteng Zhang, Bin Liu<sup>(✉)</sup>, and Qizhi Fang

School of Mathematical Sciences, Ocean University of China, Qingdao 266100,  
Shandong, People's Republic of China  
binliu@ouc.edu.cn

**Abstract.** Structures of large graphs have attracted much attention in recent years, including  $k$ -clique,  $k$ -core,  $k$ -truss,  $k$ -club, to name just a few. These structures can help detect the most cohesive or most influential subgraphs of social networks and other massive graphs. In this survey, we summarize the research on  $k$ -core, which is the maximal connected subgraph of a graph and the degree for each vertex is equal to or greater than  $k$ . We will address the core decomposition problem, the core maintenance problem, and a few applications of  $k$ -core.

**Keywords:**  $K$ -core · Core decomposition · Core maintenance

## 1 Introduction

In many fields, relationships between entities is ubiquity and graph is a suitable model to depict entities and their relationships. For example, in a telecommunication record, a vertex represents a person and an edge between two vertices represents that the two persons have communicated. In a data science model, a graph may consist of millions of vertices and edges, and may evolve over time. Detecting and analyzing the structures of large graphs have therefore become important.

Capturing the structures, such as  $k$ -core,  $k$ -clique,  $k$ -truss,  $k$ -club, and cohesive communities of graphs, has attracted much attention. Such structures have been used widely to find densely connected regions in a graph, analyze topological structures of the internet, and identify the most influential spreaders, among other things. A  $k$ -core of a graph plays a significant role in analyzing networks. Determining all  $k$ -cores in a static graph, which is called the *core decomposition* problem, can be solved in linear time [2]. In this survey, we summarize some of the major contributions to the *core decomposition* problem and

---

This research was supported in part by the National Natural Science Foundation of China (11971447, 11871442), the Natural Science Foundation of Shandong Province of China (ZR2017QA010), and the Fundamental Research Funds for the Central Universities (201964006).

the *core maintenance* problem. We also address a few applications of these two problems.

The rest of this paper is organized as follows: In Sect. 2, we present a few basic definitions. In Sect. 3 we describe algorithms for solving the core decomposition problem, including a linear-time algorithm [2], distributed algorithms [6, 7, 10, 20], external-memory algorithms [4], semi-external model [14, 15],  $k$ -core on uncertain graphs [11, 12], and structure detecting algorithms for adding some constraints to  $k$ -core (such as adding a radius or dual graph). We then present a few core maintenance algorithms in Sect. 4, including streaming algorithms [16–18], distributed algorithms [19], parallel algorithms [22, 23], and order-based algorithms [21]. In Sect. 5, we provide a few applications of the core decomposition problem and the core maintenance problem.

## 2 Basic Definitions and a Problem Statement

The definition of  $k$ -core was first presented in 1983 by Seidman [1], which has played a significant role in describing structures of social networks. For an undirected graph  $G = (V, E)$ , where  $V$  is the set of vertices and  $E$  is the set of edges, the degree of a vertex  $u \in V$  is the number of incident edges of  $u$  in  $G$ , denoted by  $d_G(u)$ . We define  $\delta(G) = \min\{d_G(u) : u \in V\}$ . Next we introduce some definitions and properties.

**Definition 1.** *Let  $H$  be a connected subgraph of  $G = (V, E)$ . If  $\delta(H) \geq k$ , where  $k$  is a non-negative integer, then  $H$  is called a seed  $k$ -core of  $G$ . Furthermore, if  $H$  satisfies the maximality, i.e., there is no other seed  $k$ -core  $H'$  contains  $H$ , then  $H$  is called a  $k$ -core of  $G$ .*

Suppose that  $H$  is a  $k$ -core that contains a vertex  $u$ . Then  $H$  is the unique  $k$ -core that contains  $u$ , denoted by  $H_k^u$ . Otherwise, there must have another  $k$ -core  $Q$  containing  $u$ . Since  $H \cup Q$  is also a  $k$ -core containing  $u$  and  $H \cup Q \supset H$ , it contradicts to the maximality of  $H$ . On the other hand, if  $u$  has a  $k$ -core  $H_k^u$  with  $k \geq 1$ , then a  $(k - 1)$ -core  $H_{k-1}^u$  must exist and  $H_{k-1}^u \supseteq H_k^u$ .

**Definition 2.** *For a vertex  $u$  in  $G = (V, E)$ , the  $K$  value (core number) of  $u$ , denoted by  $K(u)$ , is the largest  $k$ , such that there exists a  $k$ -core containing  $u$ . The max- $k$ -core of  $u$  in  $G$ , denoted by  $H^u$ , is the  $k$ -core with  $k = K(u)$ .*

An example is presented in Fig. 1, illustrating  $k$ -cores of a graph. The vertex  $u$  is contained in a 1-core and a 2-core. The max- $k$ -core of  $u$  is a 2-core and  $K(u) = 2$ .

**Problem Statement.** For a graph  $G = (V, E)$ , without lose generality, it is assumed that  $G$  is connected. We want to calculate the  $K$  value of every vertex in  $V$ , which is equivalent to finding all  $k$ -cores with different values of  $k$  in  $G$ . This problem is known as the *core decomposition* problem. An ideal  $O(m)$ -time algorithm was proposed in 2003 [2], where  $m = |E|$ . Built on this initial success, a number of other algorithms have been devised, including distributed algorithms

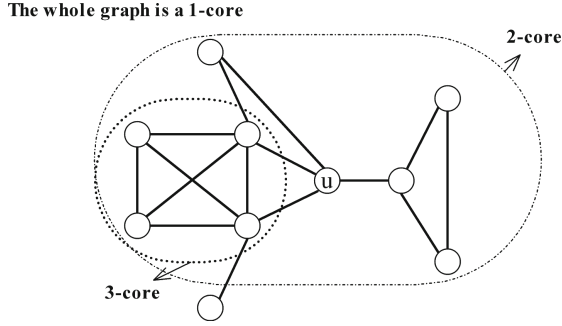


Fig. 1. An example of  $k$ -core

[6,7,10,20], external-memory algorithms [4], semi-external model [14,15], and  $k$ -core on uncertain graphs [11,12].

The core decomposition problem is formulated on a static graph. However, a lot of graphs in applications are evolving over time, inserting or removing edges or vertices. To obtain  $k$ -core structures of the new graphs, it needs to update the  $K$  values, which is known as the *core maintenance* problem. Although the  $O(m)$  algorithm can be used to the new graph to update the new  $K$  values, it will incur too much time. Since only a small portion of vertices would need their  $K$  values updated, detecting the subgraph that contains all vertices whose  $K$  values need to be updated can help improve the efficiency of this process. This has motivated the work on designing streaming algorithms [16–18], distributed algorithms [19], parallel algorithms [22,23] and order-based algorithms [21].

### 3 Core Decomposition

#### 3.1 A Linear-Time Algorithm

In 2003, Batagelj and Zaversnik [2] presented a linear-time algorithm for the core decomposition problem. This is the first algorithm for the core decomposition problem to reach be linear time. For a connected graph  $G = (V, E)$ , if  $\delta(G) \geq k$ , then  $G$  itself is a  $k$ -core. Based on the degrees of vertices, all  $k$ -cores can be detected. In fact, if we recursively delete all vertices whose degrees are less than  $k$ , then the remaining graph is a combination of some  $k$ -cores.

Algorithm 1 is the algorithm presented in [2]. The vertices need to be ordered in a nondecreasing order by their degrees after computing them. Since all degrees of vertices are bounded integers, bucket sort can be used to order them and the time complexity is  $O(n)$ , where  $n = |V|$ . Testing from vertex  $u_1$  of the smallest degree whether  $K(u_1) = d(u_1)$  and subtract 1 from  $d(w)$  if  $(u_1, w) \in E$  and  $d(w) > d(u_1)$ . Do the same operation on the remaining vertices recursively to compute the  $K$  values of all vertices. In the worst case, we traverse all edges at most once, so the time complexity is  $O(m)$ . Based on this approach, researchers

---

**Algorithm 1.**  $O(m)$  Algorithm for Core Decomposition

---

**Require:** Graph  $G = (V, E)$ **Ensure:**  $K(u)$ , for each  $u \in V$ 

```

1: Compute the degree  $d(u)$  in  $G$  for each  $u \in V$ 
2: Order the vertices in a non-decreasing order by their degrees
3: for each  $u \in V$  in order do
4:    $K(u) \leftarrow d(u)$ 
5:   for each  $(u, w) \in E$  do
6:     if  $K(u) < d(w)$  then
7:        $d(w) \leftarrow d(w) - 1$ 
8:       Reorder the rest vertices in  $V$  by their degrees
9:     end if
10:  end for
11: end for
12: return  $K(u)$ 

```

---

have devised algorithms for solving the core decomposition problem and the core maintenance problem.

A *vertex property function* is  $p(v, C)$  with real values, where  $v \in V$  and  $C \subseteq V$ . We say  $p(v, C)$  is monotone, if for each  $C_1, C_2$  with  $C_1 \subseteq C_2$ ,  $p(v, C_1) \leq p(v, C_2)$ . Listed below are examples of vertex property functions [3]:

$$\begin{aligned}
 p_1(v, C) &= \deg(v, C), \\
 p_2(v, C) &= \text{indeg}(v, C), \\
 p_3(v, C) &= \text{outdeg}(v, C), \\
 p_4(v, C) &= \text{indeg}(v, C) + \text{outdeg}(v, C).
 \end{aligned} \tag{1}$$

Batagelj et al. [3] generalized of the notion of core using vertex property functions. Algorithms for the cores of their generalization are similar to determining  $k$ -cores by degrees in Algorithm 1, testing from the vertex with smallest  $p(v, C)$ . They showed that if the vertex property function is monotone, the time complexity of determining the corresponding cores is  $O(m \cdot \max(\Delta, \log n))$ , where  $\Delta$  is the maximum degree,  $m = |E|$ , and  $n = |V|$ .

### 3.2 External-Memory Algorithms and I/O Efficient Algorithms

The  $O(m)$ -time algorithm [2] assumes that the entire graph can be loaded in the main memory and for random access. Thus, this algorithm is not suitable for a very large graph encountered in practice that exceeds the capacity of the underlying memory of a computer. An *External-Memory algorithm* for the core decomposition problem was designed [4] to take care of large graphs efficiently. Furthermore, for the networks that can be kept in the main memory, the external-memory algorithm can obtain comparable results as in-memory algorithms. Compared to the traditional *bottom-up* algorithm, the external-memory algorithm uses a novel *top-down* approach which detects  $k$ -cores recursively for

$k$  values from large to small. By removing the vertices in the  $k$ -cores we have detected, the  $I/O$  cost and search space can be reduced. The entire external-memory algorithm is divided into three parts: (1) Divide the whole graph into several subgraphs so that each subgraph can be loaded in the main memory and an efficient partition algorithm can be devised to scan the graph  $G$  only once. (2) Estimate the upper bound on  $K$  values of vertices in each subgraph and refine it progressively. (3) Use the *top-down* core decomposition algorithm recursively to determine the  $K$  value of every vertex in  $G$ . If the graph cannot be stored in the main memory, the algorithm needs  $O(k_{max})$  scans of the graph, where  $k_{max} = \max\{K(u) : u \in G\}$ . As a result, the external-memory algorithm determines the  $K$  values of all vertices in  $O(k_{max}(m+n))$  time, with  $O(\frac{k_{max}(m+n)}{B})$   $I/O$  space and  $O(\frac{m+n}{B})$  disk block space in the worst case, where  $m = |E|$  and  $n = |V|$ .

To design an  $I/O$  efficient core decomposition algorithm, Wen et al. presented a *Semi-External* model in [14] and [15], which only stores the information of vertices to the main memory and the information of edges on the disk of the underlying computer. The semi-external algorithm stores  $K$  values in the main memory and updates  $K$  values iteratively on the edges that are scanned. Their algorithms can also be used to the core maintenance problem for edge removing. In particular, they first devised  $I/O$  efficient core maintenance algorithms for edge inserting, degeneracy order computation and maintenance algorithms, respectively, under the semi-external model. As a result, the space complexity, the time complexity, and the  $I/O$  complexity of the  $I/O$  efficient core decomposition algorithm are, respectively,  $O(n)$ ,  $O(l(m+n))$  and  $O(\frac{l(m+n)}{B})$ , where  $l$  is the number of iterations and is often small in practice.

### 3.3 Distributed Algorithms

A distributed algorithm is desirable when a graph is too large to store in a single host or the description of the graph is inherently distributed in multiple hosts. Montresor et al. [6] propose distributed core decomposition algorithms to solve it. They considered two computation models:

- *One-to-one* model. One computational unit, namely one host, is associated with one vertex in the graph. Thus the information can be diffused directly through edges between two nodes.
- *One-to-many* model. One computational unit is associated with a set of vertices in the graph. Information diffuses between vertex sets in this model, which is suitable to the situation for the graph inherently distributed in multiple hosts.

At the beginning, use the degrees of vertices as the upper bounds of their  $K$  values. Hosts diffuse their information to their neighbors, then neighbors update their upper bounds of  $K$  values recursively until the upper bounds of  $K$  cannot be updated. The final upper bounds are the  $K$  values of vertices and can be reached with at most  $O(n)$  iterations, where  $n = |V|$ . Distributed algorithms are

implemented on GraphChi and Webgraph, and can be extended to large datasets on a single PC [8].

Mandal et al. [10] proposed a distributed core decomposition algorithm, called *Spark-kCore*, to run on a Spark cluster computing platform. Using a think-like-a-vertex paradigm and a message passing paradigm for the core decomposition problem, Spark-kCore algorithms can reach the target with reduced *I/O* cost.

### 3.4 Core Decomposition on Uncertain Graphs

**Definition 3.** A graph  $G = (V, E, p)$  is an uncertain graph or a probabilistic graph, where  $V$  is the set of vertices,  $E$  is the set of edges, and each  $(u, v) \in E$  is assigned a probability  $p(u, v)$ ,  $p \in (0, 1]$ .

Uncertain graphs have arisen in many fields. For instance, a vertex represents gene and edge represents interactions among genes. Since the interactions among them are derived through noisy and error operation in experiments, edges are existing in a probability. To solve the problem that can the core decomposition problem of uncertain graphs be solved by an efficient approach, Bonchi et al. [11] propose some algorithms for this problem. They introduce the definition of  $(k, \eta)$ -core  $H = (C, E|C, p)$ .  $H$  is a maximal subgraph of  $G = (V, E, p)$  satisfying  $\delta(H) \geq k$  and every vertex in  $H$  has probability no less than  $\eta$ , i.e.,  $Pr[d_H(v) \geq k] \geq \eta$ , where  $v \in C$  and  $\eta \in [0, 1]$  is a threshold representing the level of certainty of the cores. The algorithms are like the  $O(m)$ -time algorithm for the core decomposition problem on deterministic graphs [2]: computing the initial  $\eta$ -degrees by a novel efficient dynamic-programming approach, removing the vertex with smallest  $\eta$ -degree and updating  $\eta$ -degrees recursively. However, it may have exponential time complexity when computes and updates  $\eta$ -degrees. Bonchi et al. devise an efficient dynamic-programming method to overcome it. As a result, the complexity of computation  $(k, \eta)$ -core is  $O(m\Delta)$ , where  $m = |E|$  and  $\Delta$  is the maximum  $\eta$ -degree.

Peng et al. [12] proposed a different probabilistic  $k$ -core model on uncertain graphs, named  $(k, \theta)$ -core where  $\theta$  is a probability threshold, basing on the well-known possible world semantics. In a fundamental uncertain graph  $G = (V, E, p)$ , where  $p \in (0, 1]$  and  $G' = (V, E')$  is a deterministic subgraph with probability

$$Pr(G') = \prod_{e \in E'} p(e) \prod_{e \in E \setminus E'} (1 - p(e)). \quad (2)$$

Then they defined

$$p(u) = \sum_{G' \in \mathcal{G}} p(G') I_{G'}(k, u) \quad (3)$$

as the probability of  $u$  contained in a  $k$ -core of  $G$ , where  $I_{G'}(k, u)$  is an indicator function. If  $u$  is contained in a  $k$ -core of  $G'$ ,  $I_{G'}(k, u) = 1$ , otherwise  $I_{G'}(k, u) = 0$ . They [12] showed that solve the problem of finding all  $u$  with  $p(u) \geq \theta$ , i.e., finding a  $(k, \theta)$ -core of  $G$  is NP-hard. They proposed a sampling-based method

to find a  $(k, \theta)$ -core, and used pruning techniques to reduce the candidate size and a novel membership check algorithm to speed up the computation.

### 3.5 Core Decomposition Under Additional Constrains

We discuss two constrains. One is adding a attribute to a graph and detecting the maximal seed  $k$ -core in which any two vertices are linked by a relationship [26, 27], the other is finding  $k$ -connected cores in large dual networks [25].

In the real world, some graphs are given attributes, as explained in [27]. The authors of [27] introduced  $(k, r)$ -core  $H$  which is a maximal subgraph such that  $\delta(H) \geq k$  and any two vertices in  $H$  should satisfy an attribute about a threshold  $r$ . Finding a maximal  $(k, r)$ -core and a maximum  $(k, r)$ -core are both NP-hard. In [26], Wang et al. added a spatial constrain and asked a new question of finding a maximal subgraph  $H$  with  $\delta(H) \geq k$  in a radius-bounded area. Then they explored three algorithms to find a  $(k, r)$ -core where  $k$  is the minimum degree of the core and  $r$  is the radius. These algorithms are triple-vertex-based paradigm, binary-vertex-paradigm, and a paradigm based on rotating circles. Finding radius-bounded  $(k, r)$ -core can be solved in polynomial time.

A dual graph contains a physical graph and a conceptual graph with the same vertices. Yue et al. [25] formulated a  $k$ -connected core ( $k$ -CCO) model, which is a  $k$ -core in the conceptual graph and is connected in the physical graph. For a fixed  $k$ , they designed a polynomial-time peeling-style algorithm to detect all  $k$ -CCOs in a dual graph in  $O(hm)$  time, where  $m$  is the number of edges in the conceptual graph and  $h$  is a value bounded by the number of vertices of the dual graph. Then they designed *bottom-up* and *top-down* algorithms to detect maximum  $k$ -CCOs and a binary search algorithm to speedup these algorithms. Finally, they designed an index structure to detect a  $k$ -CCO containing a set of query vertices. Basing on the index structure, they presented an efficient query-processing algorithm and a polynomial-time index construction algorithm. The size of index is bounded by  $O(n)$  and the time complexity of their query-processing algorithm is  $O(m)$ .

## 4 Core Maintenance

How to find the  $k$ -cores and determine  $K$  values of all vertices in dynamic graphs that evolve over time is an important problem. This is the core maintenance problem of graphs. Let  $V^*$  denote the set of vertices whose  $K$  values will change if the graph  $G = (V, E)$  changes. The main idea of the core maintenance problem is to find a subgraph  $H \subseteq G$  (or a subset  $V'$  of  $V$ ) which contains  $V^*$ , prune the vertices with unchanged  $K$  values in  $H$ , and update the  $K$  values for the vertices in  $V^*$ .

### 4.1 Streaming Algorithms

Sarıyüce et al. [16, 17] presented three streaming algorithms, called *Subcore Algorithm*, *Purecore Algorithm*, *Traversal Algorithm*, to solve the core maintenance problem. Under the assumption that only one edge is inserted to or

removed from a graph  $G = (V, E)$  each time, they showed that  $|K(w) - K'(w)| \leq 1$  for each vertex  $w \in V$ , where  $K'(w)$  is the new  $K$  value of  $w$  after inserting or removing. Furthermore, if there is a vertex whose  $K$  value changes after inserting or removing an edge  $e = (u, v)$  with  $K(u) \leq K(v)$ , then  $K(u)$  must change and we say that  $u$  is the root  $r$ . In the insertion case, if a vertex  $w$  has  $K'(w) - K(w) = 1$ , then  $w$  is connecting to  $r$  via a path in which all the vertices have  $K = K(r)$  and  $K' = K(r) + 1$  in  $G + e$ . In the deletion case, if a vertex  $w$  has  $K(w) - K'(w) = 1$ , then  $w$  is connecting to  $r$  via a path in which all the vertices have  $K = K(r)$  and  $K' = K(r) - 1$  in  $G$ . Using these properties, they showed how to find a set of vertices, *subcore* of  $r$ , that contains  $V^*$ . Next, they used *current degree* ( $cd$ ) of each vertex as a criterion to judge whether a vertex in the subcore will have its  $K$  value changed. Define the  $cd$  value of a vertex  $w$ , denoted by  $cd(w)$ , as the number of adjacent vertices  $w'$  satisfying  $K(w') \geq K(w)$ . If a vertex  $w$  has  $cd(w) \leq K(r)$  in  $G + e$  in the insertion case, then  $K(w)$  will not change and we delete  $w$  from the subcore of  $r$ . Since  $w$  cannot help  $w'$  to have a high  $K$  value, then  $cd(w')$  should decrease by 1, where  $(w, w') \in E + e$ ,  $cd(w') > cd(w)$ , and  $w'$  on the subcore of  $r$ . Do the same operation recursively until all remaining vertices on the subcore have  $cd > K(r)$  and the set of remaining vertices is  $V^*$ . If a vertex  $w$  has  $cd(w) < K(r)$  in  $G - e$  in the deletion, then  $K(w)$  will change. Updating the  $cd$  values of the remaining vertices by the same method in the insertion case recursively until all the remaining vertices on subcore have  $cd \geq K(r)$ , we can get the set of vertices which have been deleted is  $V^*$  in the deletion case. This is the subcore algorithm, and the time complexity and the space complexity are both  $O(m)$ , where  $m = |E|$ .

Then, they defined three constrains: the *MCD* value, the *PCD* value, and the *RCD* value of a vertex, which are used to judge whether the vertex will have its  $K$  value increased. In other words, if a vertex  $w$  will have its  $K$  value increased, then the *MCD* value, the *PCD* value, and the *RCD* value are both greater than  $K(w)$ . Define the *MCD* value of a vertex  $w$ , denoted by  $MCD(w)$ , as the number of adjacent vertices with  $K \geq k(w)$ . By using the *MCD* value as a constrain, the subcore can be downsized to the *purecore* based on which they devised a *Purecore Algorithm*. The process of the purecore algorithm is the same as the subcore algorithm except  $V'$  is smaller and purecore algorithm only can be used to the insertion case. The time complexity and the space complexity of the purecore algorithm are both  $O(m)$ . On the base of knowing *MCD* values, we can get a smaller  $V'$  by the *PCD* values of vertices. And the *PCD* value of a vertex  $w$ , denoted by  $PCD(w)$ , is defined as the number of adjacent vertices  $w'$  with  $K(w') > K(w)$  or  $(K(w') = K(w)$  and  $MCD(w') > K(w)$ ). Next, the *Traversal Algorithm* is proposed, which uses the depth-first-search to find a smaller subgraph than the purecore algorithm and the subcore algorithm. Because the algorithm will not continue to search when searches a vertex which cannot have its  $K$  value changed. The time complexity and the space complexity are  $O(m)$  and  $O(n)$  respectively, where  $n = |V|$ . The *RCD* value of a vertex  $w$ , specifically denoted by  $RCD(w, n)$ ,  $n \geq 0$ , is a generalization of the *MCD* value and the *PCD* value, where  $RCD(w, 1) = MCD(w)$  and  $RCD(w, 2) = PCD(w)$ . Define  $RCD(w, n)$ ,  $n \geq 0$ , as the number of adjacent vertices  $w'$  with



$K(w') > K(w)$  or ( $K(w') = K(w)$  and  $RCD(w', n - 1) > K(w)$ ). Thus, using the  $RCD(w, n)$  values as a constrain, we can get a smaller  $V'$  and use more calculation with the increasing of  $n$ .

In [18], Li et al. found the same properties that the vertices whose  $K$  values will increased have as that in [16, 17], and designed efficient core maintenance algorithms independently. They used *Color*, *RecolorInsert*, *UpdateInsert* to find  $V'$ , prune vertices, and update  $K$  values, respectively. In particular, they presented two pruning techniques to find a smaller  $V'$ .

## 4.2 Distributed Algorithms

Since graphs are growing too fast to disposed on a single server, Aksu et al. presented distributed algorithms for the core maintenance problem [19]. The idea is executing against the partitioned graph in parallel and taking advantage of  $k$ -core properties to reduce unnecessary computation. In particular, they defined  $G = (V, E, M[V, E], C[V, E])$  and  $N_G^k(v) = |\{w | (w, v) \in E, d_G(w) \geq k\}|$ , where  $V$  is the set of vertices,  $E$  is the set of edges,  $M[V, E]$  is the structured metadata, and  $C[V, E]$  is the unstructured context. First, they developed a distributed  $k$ -core construction algorithm by the method that prune the vertices with  $N_G^k < k$  recursively. Second, they developed a new  $k$ -core maintenance algorithm which intends to update the previous  $k$ -core for a certain  $k$  after the change of a graph. They used a pruning technique to limit the scope of  $k$ -core update after insertion or deletion. For a graph  $G$ , there is a  $k$ -core  $G_k = (V_k, E_k)$  in it. Inserting an edge  $e = (u, v)$ , if both  $u, v \in V_k$ , do not change  $G_k$ ; if  $u$  or  $v$  or both are not in  $V_k$ , then the subgraph consisting of vertices in  $\{w | w \in V, d_G(w) \geq k, N_G^k(w) \geq k\}$ , where every vertex in it is reachable from  $u$  or  $v$ , may need to be updated to include additional vertices into  $G_k$ . Removing an edge  $e = (u, v)$ , if  $(u, v)$  is not in  $E_k$ , do not change  $G_k$ ; if  $(u, v) \in E_k$ , then the subgraph consisting of  $\{w | w \in V, d_G(w) \geq k, N_G^k(w) \geq k\}$ , where every vertex is reachable from  $u$  or  $v$ , may need to be updated to remove additional vertices from  $G_k$ . In the end, they proposed a batch  $k$ -core maintenance, which accumulates data updates and refreshes  $k$ -core in a batch bundles up expensive graphs traversals and can speed up the time of updating, compared to maintaining each update incrementally.

## 4.3 Parallel Algorithms

Previous work mainly focuses on inserting or removing one edge or vertex at a time. In [22] and [23], the authors presented parallel algorithms for the core maintenance problem when multiple edges or vertices are inserted or removed. The parallel algorithms can make a better use of make use of computation power and avoid extra overhead when inserting or removing one at a time.

Considering the case of inserting multiple edges, Wang et al. [22] discovered a structure named *superior edge set* that can update  $K$  values in parallel. Given a graph  $G = (V, E)$ , an edge  $e = (u, v) \in E$  is a *superior edge* of  $u$  if  $K(u) \leq K(v)$ ; define  $K(e) = K(u)$ . Define the  $k$ -*superior edge set* as  $E_k = \{e_1, e_2, \dots, e_p\}$  such

that  $K(e_i) = k$  and each vertex in  $V$  is incident with at most one superior edge. Then the *superior edge set*

$$\varepsilon_q = E_{k_1} \cup E_{k_2} \cup \dots \cup E_{k_q} \tag{4}$$

is a set of edges made up of several  $k$ -superior edge sets with different  $k$  values. Insert  $\varepsilon_q$ , then  $K'(w) - K(w) \leq 1$  for each  $w \in V$ . Next, the set of vertices whose  $K$  values will increase is determined by inserting a superior edge set. Define the *Superior Degree* of a vertex  $u$ ,  $u \in V$ , as follows:

$$SD(u) = |\{w|(u, w) \in E \cup \varepsilon_q, K(w) \geq K(u)\}|. \tag{5}$$

Define the *Constraint Superior Degree* of a vertex  $u$  by

$$CSD(u) = |\{w|(u, w) \in E \cup \varepsilon_q, K(w) > K(u) \text{ or } K(w) = K(u) \wedge SD(w) > K(u)\}|. \tag{6}$$

For a vertex  $w$ , if  $w$  satisfies these conditions: (1)  $CSD(w) > K(w)$ ; (2)  $w$  is connected to a vertex  $u$  with  $K(u) = K(w)$  by a path whose vertices have  $K = K(u)$  in  $G + \varepsilon_q$ ; (3) a superior edge of  $u$  is contained in  $\varepsilon_q$ , then  $w$  may have its  $K$  value increased. (The  $K$  values of  $u$  and  $w$  are those in  $G$ .)

Jin et al. [23] developed the parallel algorithms in [22]. They showed that if the inserted or removed edges constitute a matching, the core number update with respect to each inserted or removed edge can be handled in parallel. Meanwhile, they added parallel core maintenance algorithms for the deletion case. If a matching is inserted to or removed from graph  $G = (V, E)$ , then  $|K'(w) - K(w)| \leq 1$  for each  $w \in V$ . Since the parallel algorithms can operate a matching other than one edge in an iteration, the number of iterations needed is only  $\Delta_c + 1$  where  $\Delta_c$  is maximum number of inserted or removed edges connecting to a vertex, which will substantially reduce the time cost over inserting or removing just one edge at a time.

#### 4.4 Order-Based Algorithms

The  $O(m)$  algorithm for the core decomposition problem [2] will produce a vertices' sequence when it removes vertices recursively to determine  $K$  values. This vertices' sequence is a  $k$ -order. Based on the  $k$ -order, Zhang et al. [21] proposed a novel order-based approach for the core maintenance problem for both insertion and deletion. Meanwhile, they pointed the drawbacks of the existing traversal algorithms in [17], which need a large search space to find  $V^*$  and high overhead to maintain  $PCD$  and  $MCD$ .

For any pair of vertices  $u, v$  in a graph  $G = (V, E)$ , let  $u \preceq v$  denote  $K(u) < K(v)$  or  $K(u) = K(v)$  but  $u$  is removed before  $v$  in the  $O(m)$  algorithm for the core decomposition problem. In other words,  $u$  is in front of  $v$  in a  $k$ -order. We can summarize that a  $k$ -order,  $(v_1, v_2, \dots, v_n)$ , for each vertex in graph  $G$ , has transitivity, i.e., if  $v_h \preceq v_j$  and  $v_j \preceq v_i$ , then  $v_h \preceq v_i$ . After inserting or removing an edge, the  $k$ -order need to be updated and the new  $k$ -order is also

a removing sequence produced from the graph after the changing by using the core decomposition algorithm in [2]. Consider inserting or removing one edge at a time, the  $K$  values of all vertices change at most 1. Define the *remaining degree* of a vertex  $u$  in  $G$ , denoted by  $deg^+(u)$ ,

$$deg^+(u) = |\{v | (u, v) \in E, u \preceq v\}|. \quad (7)$$

Let  $O_k$  denotes the sequence of vertices in  $k$ -order with  $K = k$ , and we get a sequence  $O_0 O_1 O_2 \dots$ , where  $O_i \preceq O_j$  if  $i < j$ . Then the order ( $\preceq$ ) on  $O_0 O_1 O_2 \dots$  is a  $k$ -order iff  $deg^+(u) \leq k$  for every vertex  $u$  in  $O_k$  for each  $k$ . Insert  $(u, v)$ ,  $u \in O_k$  and  $u \preceq v$ , if a vertex  $w \in O_l$ , and  $l > k$  or  $l < k$ , then  $w$  is not in  $V^*$ ; if  $w \in O_k$  but  $w \preceq u$ , then  $w$  is not in  $V^*$ ; if  $w \in O_k$ ,  $u \preceq w$  and there is a path  $w_0, w_1, w_2, \dots, w_t$  such that  $w_0 = u$ ,  $w_t = w$ ,  $(w_i, w_{i+1}) \in E$  and  $w_i \preceq w_{i+1}$  for  $0 \leq i < t$ , then  $w$  may in  $V^*$ . They used the similar idea in the traversal algorithm in [17] to remove an edge, but they used the maintaining  $k$ -order method instead of using the  $PCD$  values. Define the *candidate degree* of a vertex  $w$  in  $O_k$ , denoted by  $deg^*(w)$ , as follows

$$deg^*(w) = |\{w' | (w, w') \in E, w' \preceq w \wedge w' \text{ is a potential candidate of } V^*\}|. \quad (8)$$

Then  $deg^*(w) + deg^+(w)$  is a criterion to judge whether a vertex  $w$  will be in  $V^*$ , where  $w \in O_k$ . Specifically, if  $deg^*(w) + deg^+(w) \leq k$ , then  $w$  is not in  $V^*$ . Otherwise,  $w$  is a potential vertex in  $V^*$ . Finally, they designed OrderInsert and OrderRemoval algorithms for edge inserting and removing respectively.

## 5 Applications

$K$ -core is a critical structure of graphs. It is used to depict the properties (e.g., cohesiveness, centrality, sustainability), and has been applied to a variety of fields: including detecting communities, analyzing the structures of large networks, finding the most influential subgraphs or vertices, helping to find other structures ( $k$ -clique, etc.), large-scale networks fingerprinting and visualization, dealing with problems in bioinformatics, and analyzing software bugs, to name a few.

Seidman et al. [1] introduced the notion of  $k$ -core to measure network cohesion and density, which is the first application of  $k$ -core. For the resilience of core, randomly deleting edges or vertices can destroy the core resilience, and then destroy the graph resilience [28]. Identifying the most influential spreaders is a significant issue in understanding the dynamics of information diffusion in large scale networks. Bae et al. [30] proposed a novel measure *coreness centrality* to estimate the spreading influence of a node in a network by using its  $k$ -shell indices. Rossi et al. [31] further refined the nodes by  $k$ -truss, which have a stronger influential ability locating in cores. To find the most influential part of a graph, Li et al. [29] introduced a novel community model called  *$k$ -influential community* based on  $k$ -core to capture the influence of a community. When it comes to finding a certain community, Papadopoulos et al. [34] used the method

for the core decomposition problem to detect communities in large networks of social media. Nasir et al. [35] used the methods of the core decomposition problem and the core maintenance problem to find the *top-k* densest subgraphs in networks. Alduaiji et al. [32] used *k*-core to detect communities to find the subgraphs and their impact on users' behavior in twitter communities, then they find that community members intend to share positive tweets than negative increasing over time.

In recently years, *k*-core has been widely used in the fields of software engineering, and bioinformatics. Qu et al. [36] used the core decomposition problem on class dependency networks to analyze software bugs. Cheng et al. [37] proposed a method to find cluster subgraphs made of *k*-core and *r*-clique, which is used to gene networks. Ma and Balasundaram [38] focused on a change-constrained version of the minimum spanning *k*-core problem under probabilistic edge failures. This can help telecommunication networks design, airline networks configuration and freight distribution planning. Alvarez-Hamelin et al. [39] proposed a general visualization algorithm to compare different networks using the method of the core decomposition problem, and a visualization tool to find specific structures of a network.

## References

1. Seidman, S.B.: Network structure and minimum degree. *Soc. Netw.* **5**(3), 269–287 (1983)
2. Batagelj, V., Zaversnik, M.: An  $O(m)$  algorithm for cores decomposition of networks. In: *The Computing Research Repository (CoRR)*. [arXiv:cs.DS/0310049](https://arxiv.org/abs/cs/0310049) (2003)
3. Batagelj, V., Zaveršnik, M.: Fast algorithms for determining (generalized) core groups in social networks. *Adv. Data Anal. Classif.* **5**(2), 129–145 (2011)
4. Cheng, J., Ke, Y., Chu, S., Özsu, M.T.: Efficient core decomposition in massive networks. In: *27th International Conference on Data Engineering (ICDE)*, pp. 51–62. IEEE, Hannover (2011)
5. Garas, A., Schweitzer, F., Havlin, S.: A *k*-shell decomposition method for weighted networks. *New J. Phys.* **14**(8), 083030 (2012)
6. Montresor, A., De Pellegrini, F., Miorandi, D.: Distributed *k*-core decomposition. *Trans. Parallel Distrib. Syst.* **24**(2), 288–300 (2012)
7. Jakma, P., Orczyk, M., Perkins, C.S., Fayed, M.: Distributed *k*-core decomposition of dynamic graphs. In: *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, pp. 39–40. ACM, Nice (2012)
8. Khaouid, W., Barsky, M., Srinivasan, V., Thomo, A.: *K*-core decomposition of large networks on a single PC. *Proc. VLDB Endow.* **9**(1), 13–23 (2015)
9. Govindan, P., Wang, C., Xu, C., Duan, H., Soundarajan, S.: The *k*-peak decomposition: mapping the global structure of graphs. In: *Proceedings of the 26th International Conference on World Wide Web*, pp. 1441–1450. International World Wide Web Conferences Steering Committee, Perth (2017)
10. Mandal, A., Al Hasan, M.: A distributed *k*-core decomposition algorithm on spark. In: *2017 IEEE International Conference on Big Data (Big Data)*, pp. 976–981. IEEE, Boston (2017)

11. 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, pp. 1316–1325. ACM, New York (2014)
12. Peng, Y., Zhang, Y., Zhang, W., Lin, X., Qin, L.: Efficient probabilistic k-core computation on uncertain graphs. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE), pp. 1192–1203. IEEE, Paris (2018)
13. Tripathy, A., Hohman, F., Chau, D.H., Green, O.: Scalable K-core decomposition for static graphs using a dynamic graph data structure. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 1134–1141. IEEE, Seattle (2018)
14. Wen, D., Qin, L., Zhang, Y., Lin, X., Yu, J.X.: I/o efficient core graph decomposition at web scale. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp. 133–144. IEEE, Helsinki (2016)
15. Wen, D., Qin, L., Zhang, Y., Lin, X., Yu, J.X.: I/O efficient core graph decomposition: application to degeneracy ordering. *IEEE Trans. Knowl. Data Eng.* **31**(1), 75–90 (2018)
16. Saryüce, A.E., Gedik, B., Jacques-Silva, G., Wu, K.L., Çatalyürek, Ü.V.: Streaming algorithms for k-core decomposition. *Proc. VLDB Endow.* **6**(6), 433–444 (2013)
17. Saryüce, A.E., Gedik, B., Jacques-Silva, G., Wu, K.L., Çatalyürek, Ü.V.: Incremental k-core decomposition: algorithms and evaluation. *VLDB J. Int. J. Very Large Data Bases* **25**(3), 425–447 (2016)
18. Li, R.H., Yu, J.X., Mao, R.: Efficient core maintenance in large dynamic graphs. *IEEE Trans. Knowl. Data Eng.* **26**(10), 2453–2465 (2013)
19. Aksu, H., Canim, M., Chang, Y.C., Korpeoglu, I., Ulusoy, Ö.: Distributed k-core view materialization and maintenance for large dynamic graphs. *IEEE Trans. Knowl. Data Eng.* **26**(10), 2439–2452 (2014)
20. Aridhi, S., Brugnara, M., Montresor, A., Velegrakis, Y.: Distributed k-core decomposition and maintenance in large dynamic graphs. In: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, pp. 161–168. ACM, Irvine (2016)
21. Zhang, Y., Yu, J.X., Zhang, Y., Qin, L.: A fast order-based approach for core maintenance. In: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 337–348. IEEE, San Diego (2017)
22. Wang, N., Yu, D., Jin, H., Qian, C., Xie, X., Hua, Q.S.: Parallel algorithm for core maintenance in dynamic graphs. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 2366–2371. IEEE, Atlanta (2017)
23. Jin, H., Wang, N., Yu, D., Hua, Q.S., Shi, X., Xie, X.: Core maintenance in dynamic graphs: a parallel approach based on matching. *IEEE Trans. Parallel Distrib. Syst.* **29**(11), 2416–2428 (2018)
24. Bonchi, F., Gullo, F., Kaltenbrunner, A.: Core Decomposition of Massive, Information-Rich Graphs. In: Alhajj, R., Rokne, J. (eds.) *Encyclopedia of Social Network Analysis and Mining*. Springer, New York (2018). [https://doi.org/10.1007/978-1-4939-7131-2\\_110176](https://doi.org/10.1007/978-1-4939-7131-2_110176)
25. Yue, L., Wen, D., Cui, L., Qin, L., Zheng, Y.: K-connected cores computation in large dual networks. In: Pei, J., Manolopoulos, Y., Sadiq, S., Li, J. (eds.) *DASFAA 2018*. LNCS, vol. 10827, pp. 169–186. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-91452-7\\_12](https://doi.org/10.1007/978-3-319-91452-7_12)
26. Wang, K., Cao, X., Lin, X., Zhang, W., Qin, L.: Efficient computing of radius-bounded k-cores. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE), pp. 233–244. IEEE, Paris (2018)

27. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: When engagement meets similarity: efficient  $(k, r)$ -core computation on social networks. *Proc. VLDB Endow.* **10**(10), 998–1009 (2017)
28. Laishram, R., Sariyüce, A.E., Eliassi-Rad, T., Pinar, A., Soundarajan, S.: Measuring and improving the core resilience of networks. In: *Proceedings of the 2018 World Wide Web Conference*, pp. 609–618. International World Wide Web Conferences Steering Committee, Lyon (2018)
29. Li, R.H., Qin, L., Yu, J.X., Mao, R.: Finding influential communities in massive networks. *VLDB J. Int. J. Very Large Data Bases* **26**(6), 751–776 (2017)
30. Bae, J., Kim, S.: Identifying and ranking influential spreaders in complex networks by neighborhood coreness. *Phys. A Stat. Mech. Appl.* **395**, 549–559 (2014)
31. Rossi, M.E.G., Malliaros, F.D., Vazirgiannis, M.: Spread it good, spread it fast: identification of influential nodes in social networks. In: *Proceedings of the 24th International Conference on World Wide Web*, pp. 101–102. ACM, Florence (2015)
32. Alduaiji, N., Datta, A.: An empirical study on sentiments in twitter communities. In: *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 1166–1172. IEEE, Singapore (2018)
33. Barbieri, N., Bonchi, F., Galimberti, E., Gullo, F.: Efficient and effective community search. *Data Min. Knowl. Disc.* **29**(5), 1406–1433 (2015)
34. Papadopoulos, S., Kompatsiaris, Y., Vakali, A., Spyridonos, P.: Community detection in social media. *Data Min. Knowl. Disc.* **24**(3), 515–554 (2012)
35. Nasir, M.A.U., Gionis, A., Morales, G.D.F., Girdzijauskas, S.: Fully dynamic algorithm for top- $k$  densest subgraphs. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1817–1826. ACM, Singapore (2017)
36. Qu, Y., et al.: Using K-core decomposition on class dependency networks to improve bug prediction model's practical performance. *IEEE Trans. Softw. Eng.* **1** (2019). <https://doi.org/10.1109/TSE.2019.2892959>
37. Cheng, Y., Lu, C., Wang, N.: Local  $k$ -core clustering for gene networks. In: *2013 IEEE International Conference on Bioinformatics and Biomedicine*, pp. 9–15. IEEE, Shanghai (2013)
38. Ma, J., Balasundaram, B.: On the chance-constrained minimum spanning  $k$ -core problem. *J. Global Optim.* **74**(4), 783–801 (2019)
39. Alvarez-Hamelin, J.I., Dall'Asta, L., Barrat, A., Vespignani, A.: Large scale networks fingerprinting and visualization using the  $k$ -core decomposition. In: *Advances in Neural Information Processing Systems*, pp. 41–50 (2006)
40. Eppstein, D., Löffler, M., Strash, D.: Listing all maximal cliques in sparse graphs in near-optimal time. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) *ISAAC 2010*. LNCS, vol. 6506, pp. 403–414. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-17517-6\\_36](https://doi.org/10.1007/978-3-642-17517-6_36)