mathematics

MDPI

Article

Edge-Based Minimal *k*-Core Subgraph Search

Ting Wang , Yu Jiang *, Jianye Yang * and Lei Xing

Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou 510700, China; gianttina@163.com (T.W.); 2112233112@e.gzhu.edu.cn (L.X.)
* Correspondence: jiangyu@gzhu.edu.cn (Y.J.); jyyang@gzhu.edu.cn (J.Y.)

Abstract: In social networks, *k*-core is commonly used to measure the stability of a network. When a user in a *k*-core leaves the network, other users may follow the user to leave. Hence, maintaining a key user is important to keep the stability of a network. It is known that an edge between two users models the relationship between the two users. In some scenarios, maintaining a relationship comes at a cost. Therefore, selectively in maintaining the relationships between users is crucial. In this paper, we for the first time conceive the concept of an edge-based minimal *k*-core model. An edge-based minimal *k*-core is a *k*-core with a minimal number of edges. In other words, removing any edge in an edge-based minimal *k*-core would make it not be a *k*-core any more. Based on this model, we proposed two problems, namely, an edge-based minimal *k*-core subgraph search (EMK-SS) and an edge-based minimal *k*-core subgraph search with a query node *q* (EMK-*q*-SS). Given a graph *G*, an integer *k*, and a query node (a key user) *q*, the EMK-*q*-SS problem is to find all the edge-based minimal *k*-cores containing the query node *q*, and the EMK-SS problem is to find all the edge-based minimal *k*-cores. We also theoretically prove that the two problems are both NP-complete. To deal with the proposed problems, we design two novel algorithms, namely the edge deletion algorithm and edge extension algorithm. Further, a graph partitioning technique is employed to speed up the computation. Comprehensive experiments on synthetic and real networks are conducted to demonstrate the effect and efficiency of our proposed methods.

Keywords: data management; subgraph search; *k*-core

MSC: 05C30

check for updates

Citation: Wang, T.; Jiang, Y.; Yang, J.; Xing, L. Edge-Based Minimal *k*-Core Subgraph Search. *Mathematics* **2023**, *11*, 3407. https://doi.org/10.3390/math11153407

Academic Editor: Ruo-Wei Hung

Received: 27 June 2023
Revised: 26 July 2023
Accepted: 28 July 2023
Published: 4 August 2023

1. Introduction

As a natural choice for modeling the relationship between entities, graphs have been used in many real-world applications, such as online social networks [1,2], road networks [3,4], and financial transaction networks [5], etc. Recently, there have been a string of works devoted to analysing graph data [6]. Among them, the *k*-core-based cohesive subgraph search has been studied intensively [7–12]. Essentially, a *k*-core is a subgraph where each vertex has at least *k* neighbors in the subgraph. It has been shown that the *k*-core subgraph can be computed in linear time by recursively removing the vertices with neighbor nodes less than *k* in the graph [13]. Due to the computation efficiency, *k*-core has been adopted in many cohesive subgraph-mining-related applications [8,10,11,14–16].

In the literature, the problem of minimum *k*-core computation has been studied, which aims to find a subgraph with the minimum number of vertices satisfying the *k*-core constraint, i.e., each vertex has at least *k* neighbors. The minimum *k*-core has many applications [17]. For example, in online commercials, it can be used for advertisement delivery with a limited budget [18]. In social networks, the minimum *k*-core has been employed for product recommendation [18]. In biology, the minimum *k*-core is capable of modeling the cell assembly [14,15].

Motivation. Although the minimum *k*-core model is useful in many real applications, it has its own limitations. Under some circumstances, the network stability is expensive to maintain, because the relationship between two users are maintained at a cost [19]. In this paper, we advocate the problem of the edge-based minimal k-core search. The edge-based minimal k-core is defined as a k-core in which every edge in the k-core is indispensable. In other words, if any edge in this subgraph is missing, the subgraph is not a k-core any more. For example, for the graph G shown in Figure 1a, we have three different edge-based minimal k-cores, which are shown in Figure 1b–d, respectively. The edge-based minimal k-core plays an important role in many other related problems. Below is a concrete example.

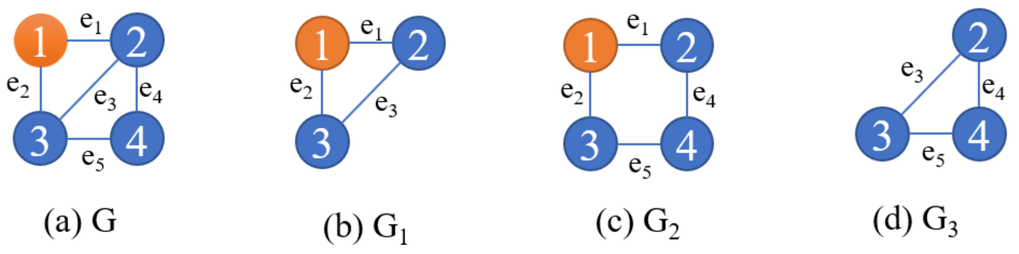


Figure 1. Example of edge-based minimal 2-core.

Example 1 (Group Recommendation). *It is a common practice to encourage group members to participate in social networks by leveraging the positive influence of friends from the same group. In some applications, such as group recommendations with query vertices (for example, specific users), small k-core subgraphs may be preferred, as large k-core subgraphs may contain many vertices unrelated to the query vertices of online social platforms. The system can recommend items to users based on the information from social groups containing users. Usually, items have been adopted by certain users, among these users there exists interactions. Therefore, on one hand, k-core is a promising model to maintain the structural stability of group members. On the other hand, since it comes at a cost to maintain the relationships between group members, the service provider would like to maintain as small a number of relationships as possible. To achieve the two targets, our edge-based minimal k-core model provides a good solution.*

Challenges. The problem of edge-based minimal k-core containing the query node q search is computationally challenging. Given an undirected graph $G = (V, E)$, a straightforward solution is to enumerate all 2^m combinations of subgraphs assuming the number of edges in the graph G is m and verify whether each of them is an edge-based minimal k-core containing the query node q. Clearly, this problem is cost-prohibitive, because the number of edge-based minimal k-cores containing the query node q might be exponential to the number of edges. This shows that our problem is even more difficult than other related search problems, such as the minimum k-core containing the query node q search, which is NP-hard [18]. For example, as is shown in Figure 1, let $k = 2$ and q be node 1, graph G_1 has the smallest-sized vertex number, thus it is a minimum k-core with q. Meanwhile, G_1 is also an edge-based minimal k-core containing q in G. However, though graph G itself does not have the smallest-sized vertex number, G_2 is able to be found as an edge-based minimal 2-core containing node 1 by deleting edge e_3 from graph G. Since the concept of "minimal" for the edge-based minimal k-core is the structural minimum, not simply defined as the minimum number of nodes or edges, the problem of edge-based minimal k-core might involve extra computation.

Contributions. In order to cope with the computation challenges, we propose efficient query processing techniques. In the field of database research, algorithms which combine backtracking with branch-and-bound techniques are widely adopted to tackle graph-based problems, such as maximal biclique enumeration [20,21]. Inspired by these, we propose a competitive branch-and-bound baseline method, namely, the edge deletion algorithm, which explores the search space in a depth first manner. Specifically, the edge deletion algorithm maintains a k-core structure and recursively deletes the candidate edges from the k-core structure to generate edge-based minimal k-cores containing the query node q.

We also propose another baseline named the edge extension algorithm, which searches the edge-based minimal k-cores containing the query node q following a bottom–top manner using a depth first search. More specifically, the edge extension algorithm maintains a partial edge-based minimal k-core containing q and recursively adds the candidate edges to the partial solution to generate results.

To improve the performance of the two baselines, we propose new efficient pruning techniques, such as setting the status of edges to compact the search space, size pruning to terminate search branches early, and edge ordering to avoid redundant computation. To further speed up the computation, we propose a graph partitioning technique. In particular, the graph partitioning technique partitions the graph first to reduce the edges in each iteration, and then, we execute the algorithms concurrently and independently on each partition. Intuitively, this parallel computation paradigm can accelerate the computation linearly to the number of partitions.

Our principal contributions are summarized as follows.

- We conceive the concept of the edge-based minimal k-core to model a special subgraph with a minimal number of edges while remaining a k-core, and put forward two related subgraph search problems, i.e., the edge-based minimal k-core subgraph search problem (EMK-SS) and the edge-based minimal k-core with a query node q subgraph search problem (EMK-q-SS). The EMK-SS problem is to find all the edge-based minimal k-cores, and the EMK-q-SS problem is to find all the edge-based minimal k-cores containing the query node q in a k-core graph. We prove also that the two problems are NP-complete.
- Two algorithms are proposed to solve the EMK-q-SS problem by deleting edges from the candidate set iteratively until one EMK-q is obtained and adding edges to the candidate set iteratively until one EMK-q is obtained, respectively.
- We conduct comprehensive experiments on generated graphs and real graphs to explore the efficiency of the above two algorithms and the speed-up ratio using a graph partition technique. The results show that the two algorithms work effectively and graph partition works well on saving running time and speeds the search process up by up to four orders of magnitude.

2. Preliminaries

In this section, we introduce the concept of a k-core subgraph, an edge-based minimal k-core subgraph, and the edge-based minimal k-core subgraph search problem.

2.1. *k-Core Subgraph*

Let $G(V, E)$ be an undirected and unweighted graph, where V represents the set of vertices (nodes) and E represents the set of edges. We use $n = |V|$ and $m = |E|$ to denote the number of nodes and the number of edges, respectively. Let $N(u, G)$ denote the set of adjacent vertices of node u. We denote $|N(u, G)| = deg(u, G)$ is the degree of the node u in G. Given a subgraph S, we denote its vertices as $V(S)$. The number of vertices in S is shown as $|S|$. When the context is clear, $N(u, G)$, $deg(u, G)$, and $V(S)$ are simplified to $N(u)$, $deg(u)$, and S, respectively. We use d_{max} to denote the largest vertex degree in the graph.

Given a graph G and an integer k, a k-core subgraph S of G is the maximal connected subgraph in which the degree of every vertex in S is larger than or equal to k. In order to simplify the expression, later we use k-core to denote the k-core subgraph. For a node u in G, if it exists in k-core but cannot be contained in $(k + 1)$-core, we say the core number of node u is k and denote it as $core(u, G)$. If the context is clear, we abbreviate this as $core(u)$. The core number of a graph G is the biggest one among the core numbers of all nodes in the graph and is marked as $core(G)$.

2.2. The Edge-Based Minimal k-Core Subgraph

Given a graph G and an integer k, an edge-based minimal k-core subgraph S is a k-core in which the deletion of any edge will make k-core vanish in S. In other words, there is no k-core in S if any edge is deleted.

Example 2. *Given a graph G and $k = 2$, Figure 1 shows all the edge-based minimal 2-cores in graph G. G_1, G_2, and G_3 are three edge-based minimal 2-core subgraphs on G, as the absence of any edge in these subgraphs will result in the disappearance of 2-core in them, which fits with the concept of edge-based minimal k-core.*

2.3. Problem Definition

Problem 1. ***Edge-based minimal k-core subgraph search problem (EMK-SS).*** *Given a graph G, an integer k, and a query node q, the edge-based minimal k-core subgraph search problem is to find all the possible edge-based minimal k-core subgraphs.*

Problem 2. ***Edge-based minimal k-core with a query node q subgraph search problem (EMK-q-SS).*** *Given a graph G, an integer k, and a query node q, the edge-based minimal k-core subgraph search problem is to find all the possible edge-based minimal k-core subgraphs that include the query node q.*

Example 3. *In Figure 1, given a graph G and $k = 2$, we set node 1, which is painted orange, as the query node, G_1 and G_2 are edge-based minimal 2-core subgraphs on G including the query node since every edge in the subgraphs is critical for edge-based minimal 2-core construction.*

2.4. Problem Complexity

Theorem 1. *The edge-based minimal k-core subgraph search problem is NP-complete.*

Proof of Theorem 1. The decision version of this problem is as follows:

Instance: Given a graph G and integer s $\leq |V|$.

Question: Does G has a k-core with edge number s?

Obviously, given a solution of the problem, a nondeterministic Turing machine checks if the choice is true in polynomial time. Therefore, the edge-based minimal k-core subgraph search problem belongs to NP. Furthermore, if we restrict the problem by considering only instances in which the number of edges of the k-core is s, then the original problem can be transformed into a clique problem [22]. Hence, the edge-based minimal k-core subgraph search problem is NP-complete.

The problem of finding all edge-based minimal k-cores also requires graphical enumeration, which refers to the art of counting the number of graphs with a specific property. Note that for some problems. to count the number of graphs with a given property is harder than to determine if there exists a graph that satisfies such a property. For instance, "Given a graph G and a fixed value $k > 0$, how many distinct k-cores are there for G?" is not a trivial problem and it empirically depends on the density of the graph. Enumeration problems associated with NP-complete problems are NP-hard [22]. This is true since the enumeration version of the problem must be at least as hard as the decision version of the problem. Hence, the enumeration of edge-based minimal k-cores is NP-hard. $\square$

Theorem 2. *The edge-based minimal k-core subgraph containing the query node q search problem is NP-complete.*

Proof of Theorem 2. It is clear that the EMK-q-SS problem belongs to NP. Now we show that the EMK-q-SS problem is NP-complete by reducing the problem of finding the maximum clique that contains q (MC-q) to it. MC-q is proved to be NP-complete in reference [23]. Let $G(V, E)$ be a graph, and let q be the query vertex. Consider the decision problem that corresponds to the optimization problem of MC-q: Determine whether G has a clique of at least $k(k+1)/2$ edges that contains q. We can construct the following decision problem for EMK-q-SS. Determine whether a solution $H \subseteq E$ exists such that $|H| = (k-1)k/2$ and it satisfies the EMK-q conditions: (i) q is an endpoint for some edges in H; (ii) $G[H]$ is connected; and (iii) $deg(v) \geq k-1$ for any vertex v in $G[H]$. If H is the answer, then apparently $G[H]$ is a clique, as any node in H has degree $\geq k-1$. □

3. Edge Deletion Algorithm

Inspired by [15], we propose a backtrack algorithm for the problem. The main idea of the algorithm is to delete edges one by one in G until we obtain the edge-based minimal k-core. In order to accelerate the process, we first prune the original graph G to a k-core using a state-of-the-art algorithm [13] for core decomposition. In the process of obtaining the result, we check if the query node is in the result set in each recursive call. The pseudocode is shown in Algorithm 1.

Algorithm 1: Edge Deletion Algorithm.

```
Input  : an undirected and unweighted graph $G$, degree constraint $k$, a query
         node $q$
Output : edge-based minimal $k$-core
1 $R \leftarrow k$-core pruning on $G$;
2 $result \leftarrow R$;
3 for each node $u \in V(R)$ do
4     $deg(u) \leftarrow$ degree of $u$;
5 end
6 Deletion($R, result, 1, k, q, deg$);
```

Firstly we compute the k-core in graph G. Then, we use a backtrack algorithm described in Algorithm 2 to carry out the edge deletion operation. In Algorithm 2, we use set *result* to record the partial solution, and set *deg* is used to record the degrees of all nodes. In the beginning of each recursive call, we check if the query node is in the partial solution, if not, we just return (lines 1–3). We will decide whether degrees of nodes in *result* are all less than k and not enough to support a k-core structure and decide if it is time to check whether the partial solution is an edge-based minimal k-core or not (line 4). If the answer is yes, then we will use the edge-based minimal k-core judge algorithm described in Algorithm 3 to check if the partial solution *result* is exactly an edge-based minimal k-core and report the solution as an edge-based minimal k-core if there is no problem in the check process. If the nodes in *result* are exactly an edge-based minimal k-core, we will report it and return (lines 5–8). In the main iteration, we delete one edge and update the degree of the two endpoints of the deleted edge before stepping into the next backtrack process (lines 12–16). When we finally backtrack to the current iteration, we resume the deleted edge and the degree of its endpoints (lines 18–21).

The algorithm shown in Algorithm 3, named isMinimal, that we apply to determine the edge-based minimal k-core works as follows. According to the definition of the edge-based minimal k-core, we delete one of the edges in the k-core and check if there is no k-core in the input subgraph any more. If there is still a k-core, then the subgraph we obtained is not an edge-based minimal k-core. We do this until all the edges have experienced being deleted to be checked. Note that if there is a node in the result set with degree equal to k, then we only need to delete one of its adjacent edges to check if the edge-based minimal k-core is decomposed.

Algorithm 2: Deletion($R, result, i, k, q, deg$).

Input : a k-core R, a set $result$, an edge number i, degree constraint k, a query node q, a nodes degree set deg

Output: edge-based minimal k-core

```
if $q$ is not in $result$ then
    return;
end
if $deg(u, result) \geq k$ for every node $u$ in result then
    if isMinimal($result$) then
        Report $result$ as an edge-based minimal $k$-core;
        return;
    end
end
else
    for each edge $e \in E(R)$ do
        $e_i \leftarrow$ the edge numbered $i$;
        $result \leftarrow result - e_i$;
        subtract 1 from the degree of each endpoint of $e_i$ separately;
        Deletion($R, result, i+1, k, q, deg$);
        $result \leftarrow result \cup e_i$;
        add 1 to the degree of each endpoint of $e_i$ separately;
    end
end
```

Algorithm 3: isMinimal.

Input : an edge set E, degree constraint k

Output: true or false

```
for each edge $e \in E(result)$ do
    $edges \leftarrow E(result) - e$;
    Core decomposition for subgraph induced by $edges$;
    for each endpoint $u$ of edge $\in$ edges do
        if $core(u) \geq k$ then
            return false;
        end
    end
end
return true;
```

Correctness. As we can see, in Algorithms 1 and 2, we will choose one edge to delete in each recursion. In the later recursion, the edge to be deleted is ordered by index after the current recursion. Backtracking occurs when no edges can be deleted any more to destroy the k-core structure or an edge-based minimal k-core is found, since even if the search goes on to the next recursion and still keeps going until there is no edge to be chosen, there will be no more edge-based minimal k-core to be found. When backtracking to the current recursion, we will choose an edge that has not been processed in the previous level as well as the same level in the search tree. Therefore, the solutions are complete and not repeated with one another.

Example 4. *Figure 2 shows the procedure of employing the edge deletion algorithm to search edge-based minimal 2-cores containing node 1 in the given graph G. Each edge is represented as a letter "e" and a subscript, which is also the process order. For example, edge 1 is denoted as* e_1

and is to be processed at the first step. As shown in Figure 2, if edge 1 is selected to be deleted, then the query node q is not contained in the result set, so the program returns to the previous step, in which edge 1 is restored. In accordance with loop order, the next edge to be deleted is edge 2, q is not contained in the result set too after removing edge 2. Next, edge 3 is selected to be deleted, this time, an edge-based minimal k-core containing node 1 is obtained. Another edge-based minimal k-core containing node 1 occurs when edge 4 is removed and soon edge 5 is deleted. We return to the state when edge 4 is deleted and find there are no edges left to be processed. As a result, we return to the original state and choose edge 5 to be deleted. As no edges remain as candidates to be chosen, we judge the current subgraph is not satisfying and return. The algorithm is finished.

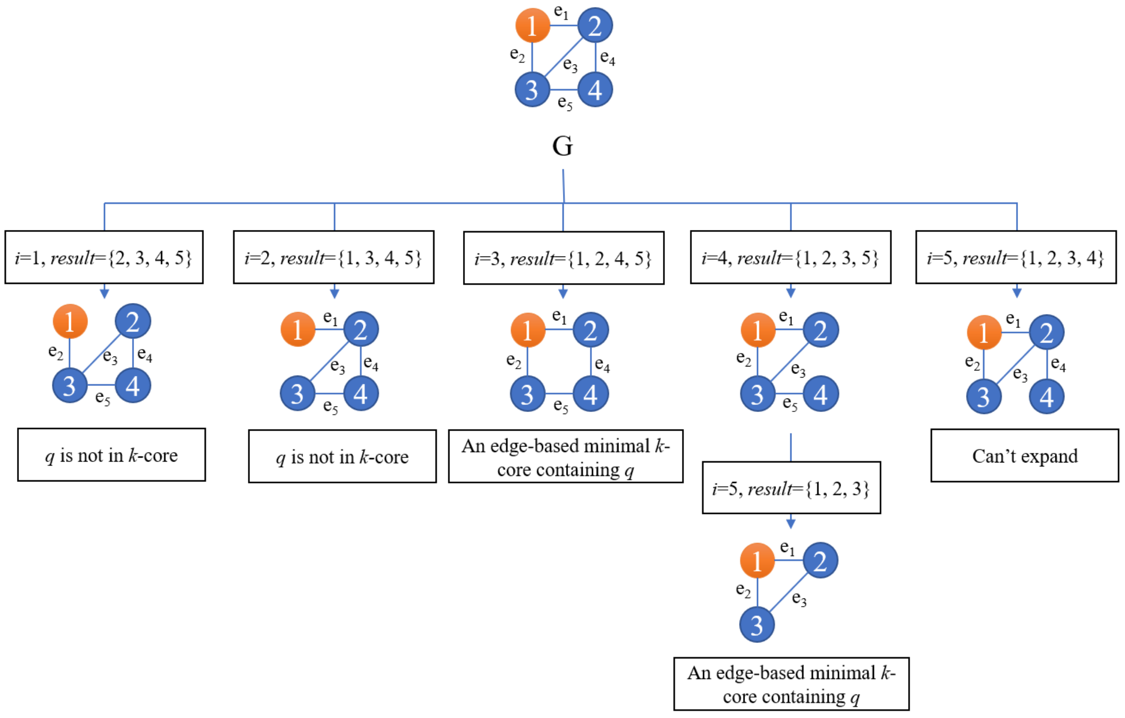


Figure 2. Example of edge deletion algorithm. Given a graph G, $k = 2$, when the query node is node 1, the root of the tree contains k-core of the entire graph and the leaves contain edge-based minimal k-core containing node 1 or extensions that make the algorithm backtrack. The set *result* and variables k and q follow the definitions in Algorithm 2.

Pruning. It is apparent that if the degree of a node in k-core is equal to k, then once the node loses one of its adjacent edges, it will not satisfy the degree constraint of k-core any more. For instance, in Figure 2, when edge 4 is deleted, edge 5 has to be chosen to be deleted, for node 4 is not contained in 2-core definitely and not contained in edge-based minimal 2-core naturally.

Therefore, we reindex the adjacent edges of nodes with degree in k-core equal to k to be in the same status. If we delete one adjacent edge of this kind of node, other adjacent edges of this node will be deleted too.

Complexity. The main time cost is produced by the algorithm deletion shown in Algorithm 2, so we only discuss the complexity of deletion. Suppose the number of edges in k-core is m_k. We start to delete from the first edge, in the first recursive call we have m_k different choices of edges from k-core in the worst case, in the second recursive call we have $m - 1$ edges to choose from. As the function goes on, we delete edges until we obtain the result we want, or we have no edges to delete. Hence, in the worst case there are m_k nested loops and the time complexity is given by $T(m_k) = m_k * (m_k - 1) * (m_k - 2) * ... * 2 * 1 = O(m_k!)$ if we assume that in each recursive call the processing time is $O(1)$. In fact, in each recursive call, we need to compute the core number of vertices in the set *result* in order to check if the current graph induced by edges in *result* is an edge-based minimal k-core. This is achieved with the help of the core decomposition method, whose time complexity is $O(m_k)$ (suppose $m_k > n$) [13]. Notice that in the check function *isMinimal* we need to delete each edge and compute core number separately. So the total complexity is $O(m_k^2 * m_k!)$.

Considering the additional edge reindex operation, the new complexity will be $O(m' * m_k * m_k!)$ if we denote the number of edges after reindexing as m'.

4. Edge Extension Algorithm

In this section, we will introduce the edge extension algorithm and introduce an optimization technique which helps to improve the performance of the proposed algorithms.

4.1. Edge Extension Algorithm

Inspired by [18], we propose another backtrack algorithm following a bottom–top method. Explicitly, we add edges one by one to the result set from the query nodes until the required subgraph is obtained. Similar to the edge deletion algorithm, firstly we obtain k-core of G before the backtracking. Secondly, we process the nodes with degree equal to k in k-core at once. Then, as shown in Algorithm 5, once all the vertices in set *result* have degree larger than or equal to k, we use isMininal to test if the subgraph induced by edges in *result* is an edge-based minimal k-core and report it if the answer is yes before return. The pseudocode is shown in Algorithms 4 and 5.

Firstly, we compute the k-core in graph G (line 1). Then, we use a backtrack algorithm described in Algorithm 5 to conduct the edge deletion operation. In Algorithm 5, we use set *result* to record the partial solution, and set *deg* is used to record the degree of all nodes. At the beginning of each recursive call we will decide whether all the nodes in *result* have degree larger than or equal to k or not (line 4). If the answer is yes, then we will use the edge-based minimal k-core judge algorithm described in Algorithm 3 to check if the partial solution *result* is exactly an edge-based minimal k-core and report the solution as an edge-based minimal k-core if there is no problem in the check process. No matter what the answer is, we will return after the judgment since there is no need to explore extra branches (line 5–line 8). In the main iteration, we add one edge to the partial solution and update the degree of its two endpoints before stepping into the next backtrack process (line 12–line 16). When we finally backtrack to the current iteration, we resume the added edge and the degree of its two endpoints (line 18–line 21).

Example 5. *Figure 3 shows the procedure of employing the edge extension algorithm to search for edge-based minimal 2-cores containing node 1 in a given graph G. First, we add edge 1 into the result set, and thus, edge 2 is added as node 1 needs to have at least 2 neighbors to be contained in a 2-core. Then, we add edge 3 to the result set, there is an edge-based minimal 2-core containing node 1 to be reported. When we add edge 4 rather than edge 3 to the result set, another edge-based minimal 2-core occurs with the participation of edge 5. We go back to the initial graph and process other edges. However, there are no subgraphs which can guarantee that node 1 is contained in 2-core. As a result, the algorithm reports two edge-based minimal 2-cores in the graph.*

Algorithm 4: Edge Extension Algorithm.

```
Input  : an undirected and unweighted graph $G$, degree constraint $k$, a query
         node $q$
Output : edge-based minimal $k$-core
1 $R \leftarrow k$-core pruning on $G$;
2 $result \leftarrow \emptyset$;
3 for each node $u \in V(R)$ do
4     $deg(u) \leftarrow 0$;
5 end
6 Extension($R, result, 1, k, q, deg$);
```

Algorithm 5: Extension.

```
Input  : a $k$-core $R$, a set $result$, an edge number $i$, degree constraint $k$, a query
         node $q$, a degree set $deg$ of all nodes in $R$
Output: edge-based minimal $k$-core
1  if $q$ is not in result then
2      return;
3  end
4  if $deg(u, result) \geq k$ for every node $u$ in result then
5      if isMinimal($result$) then
6          Report $result$ as an edge-based minimal $k$-core;
7          return;
8      end
9  end
10 else
11     for each edge $e \in E(R)$ do
12         $e_i \leftarrow$ the edge numbered $i$;
13         $result \leftarrow result \cup e_i$;
14         add 1 to the degree of each endpoint of $e_i$ separately;
15         Extension($R, result, i + 1, k, q, deg$);
16         $result \leftarrow result - e_i$;
17         subtract 1 from the degree of each endpoint of $e_i$ separately;
18     end
19 end
```

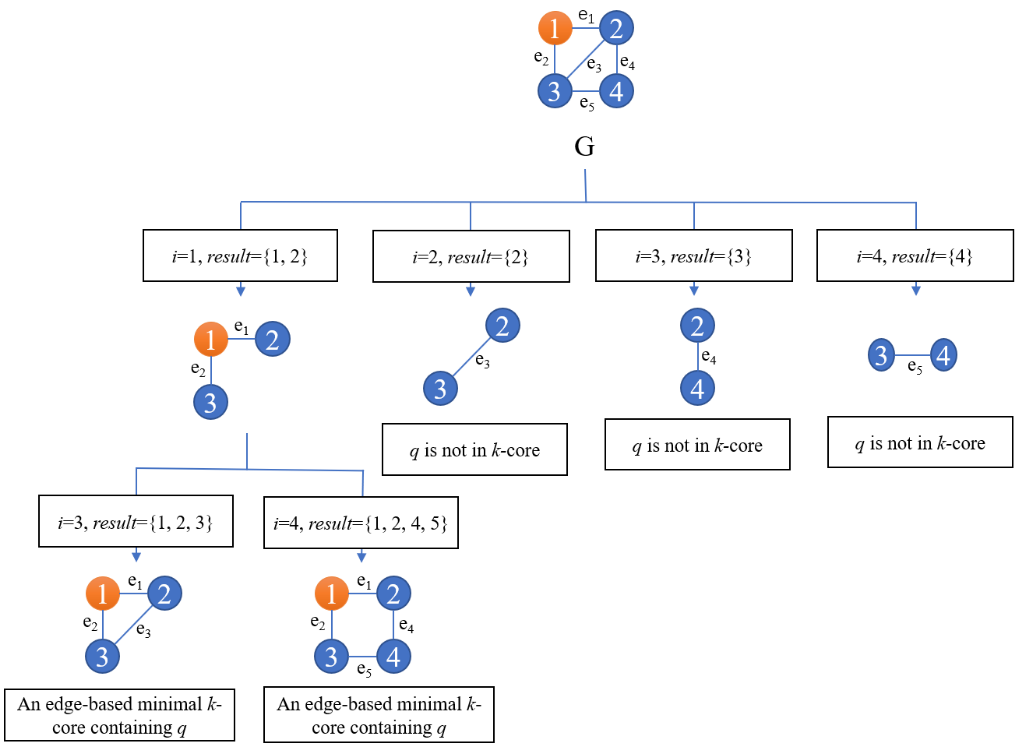


Figure 3. Example of the edge extension algorithm. Given a graph G, $k = 2$, when the query node is node 1, the root of the tree contains k-core of the entire graph and the leaves contain edge-based minimal k-core or extensions that make the algorithm backtrack. The set *result* and variables k and q follow the definitions in Algorithm 2.

Complexity. In each of the recursive calls we will add one edge to set *result* until the degree condition is required before deciding whether an edge-based minimal k-core is obtained. Similar to the edge deletion algorithm, the complexity of the the edge extension algorithm is $T(m_k) = O(m_k^2 * m_k * (m_k - 1) * \dots * M$, where M represents the remaining number of edges in the candidate set. It is obvious that M is 0 in the worst case when the input k-core is exactly an edge-based minimal k-core. Therefore, the final complexity is also $O(m_k^2 * m_k!)$ as for the edge deletion algorithm.

4.2. Optimization

In this part, we will propose one optimization for the proposed algorithms.

Graph Partitioning

In the real world, a graph is usually sparse and is composed of many components. There may exist a kind of node named a cut point that connects two or more connected subgraphs, which means the subgraphs have only one overlap which is exactly the node. In this case, a connected k-core subgraph is able to be decomposed into parts that have at most one overlap (a node) with each other.

Since the complexity of algorithms for edge-based minimal k-core search is tightly related to m, when the input edges are reduced, the speed of the algorithms will improve a lot. As a result, we use Tarjan's strongly connected components algorithm to cut the graphs into blocks through cut points in the graph first, and then take each block as input to the algorithms separately. We explain the reasonability of this technique using the following example.

Example 6. *As shown in Figure 4, graph G is a 2-core but not an edge-based minimal 2-core. It is not necessary to obtain the edge-based minimal 2-core starting from the whole k-core. Partitioning the graph G first will reduce the number of edges we need to process. The process of partitioning does not influence the correctness of the result of the edge-based minimal k-core search, since an edge-based minimal k-core on one side of a cut point contains no edges on another side of the cut point. In Figure 4, the cut point is node 3, G_1 and G_2 both exist on only one side of node 3.*

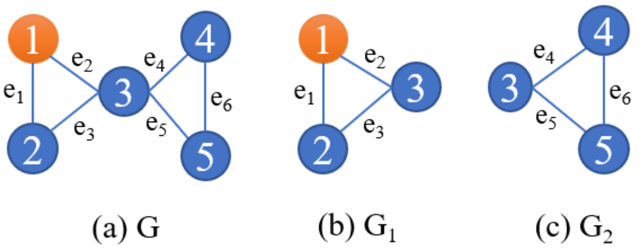


Figure 4. Example of graph partitioning.

5. Experiment

In this section, we evaluate the effectiveness and efficiency of both proposed algorithms and the optimization technique.

5.1. Experimental Setting

Datasets. We deploy 13 datasets to check the efficiency of the algorithms. The datasets can be classified into two types, one is generated datasets, the other is real datasets. The statistics of the two groups of datasets are shown in Table 1 in ascending order of the number of edges separately.

Given the number of nodes, we generate graphs by offering different possibilities of generating an edge between every two nodes. For instance, 40_0.05_3 is a generated dataset with 40 nodes and the edge generating probability is 0.05. Moreover, the core number of the whole graph is 3. d_{avg} is the average degree of the graph, and k_{max} is the largest vertex core number in the graph. In order to support the performance analysis of the proposed two algorithms, we have generated synthetic datasets such that for each pair of parameters, which is the probability of generating edges between any two nodes and the core number of the graph, there are 10 different datasets. We adopted the average value of the results as the final performance of the two algorithms to support the performance analysis of both algorithms.

Table 1. Statistics of datasets.

	Datasets	**m**	**n**	d_{avg}	k_{max}
Generated	10_0.01_2	3.4	10	0.68	2
	10_0.05_2	6.1	10	1.22	2
	10_0.1_3	10	10	2	3
	10_0.15_3	12.8	10	2.56	3
	10_0.2_4	17.8	10	3.56	4
	10_0.25_4	19.9	10	3.98	4
	10_0.3_5	23.3	10	4.66	5
	10_0.35_5	25.3	10	5.06	5
Real	Reptilia-tortoise-network-pv-2012	26	20	2.6	2
	Aves-weaver-social-04	27	18	3	4
	Aves-weaver-social-00	28	18	3.1	4
	Reptilia-tortoise-network-sg	34	30	2.3	3
	Reptilia-tortoise-network-lm-1999	38	24	3.2	3
	Bcspwr01	46	39	2.4	2
	Reptilia-tortoise-network-cs-2010	123	154	1.6	3

The real datasets are all from the NetworkRepository http://networkrepository.com (accessed on 10 March 2023). Since the name of most of the real datasets are too long to display, in later sections we transform the names of the real datasets to another name related to the number of their edges. For example, Bcspwr01 is renamed as e48.

Environment. Our experiments are performed on a computer with a 14-core CPU (12th Gen Intel(R) Core(TM) i712700H 2.30 GHz) and 16 GB memory. All algorithms are implemented in C++.

5.2. Effectiveness of Algorithms

In this part, we report the number of edge-based minimal k-cores containing the query node in all the datasets. The input integer k ranges from 2 to the core number of the graph. As query node q is a random node given by users, we suppose q is in the densest part of the graph in order to perform the experiment in the worst case. As a consequence, we set a node in the k_{max}-core in the graph as query node q during the whole experiment. Table 2 shows the number of edge-based minimal k-cores containing the query node q using the algorithms in this paper. The algorithms are able to compute the right number of edge-based minimal k-cores in each dataset because of the theoretical correctness of the algorithms.

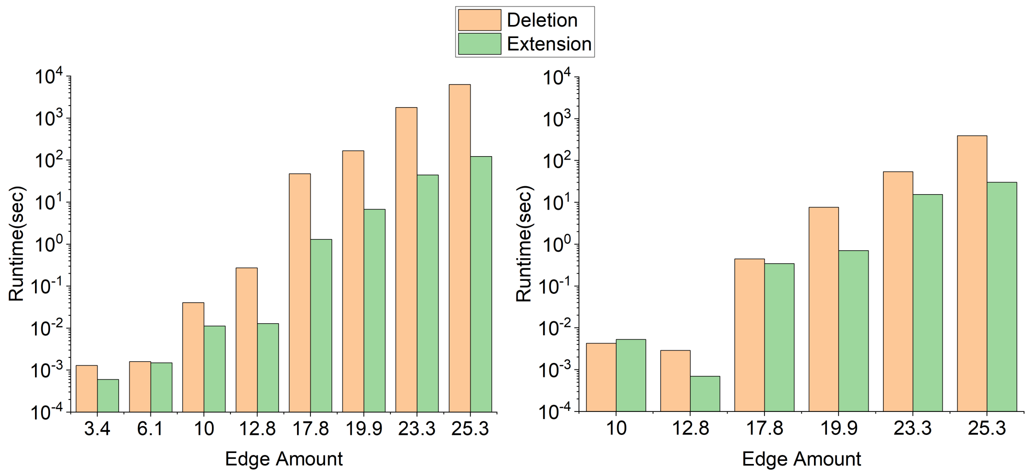
Table 2. Statistics of results.

Datasets		k	Amount
Generated	10_0.01_2	2	1
	10_0.05_2	2	1.2
	10_0.1_3	2	11.7
		3	1
	10_0.15_3	2	21.1
		3	1
	10_0.2_4	2	159.7
		3	82.6
		4	2
	10_0.25_4	2	387.5
		3	357.5
		4	1.9
	10_0.3_5	2	984.6
		3	2236.3
		4	74.2
		5	1.4
	10_0.35_5	2	1784
		3	8296.4
		4	361.2
		5	1.9
Real	Reptilia-tortoise-network-pv-2012	2	16
		3	1
	Aves-weaver-social-04	2	46
		3	19
		4	1
	Aves-weaver-social-00	2	30
		3	19
		4	1
	Reptilia-tortoise-network-sg	2	6
		3	1
	Reptilia-tortoise-network-lm-1999	2	562
		3	42
	Bcspwr01	2	29
	Reptilia-tortoise-network-cs-2010	2	6
		3	1

5.3. *Performance of Algorithms*

In the following, we report the runtime performance of the proposed two algorithms.

Varying Average Degree. We compare the running times of the edge deletion algorithm and the edge extension algorithm on six synthetic datasets through fixing k while varying the average degree, which is also the number of edges of datasets. The results are shown in Figure 5. We display the running time for k equal to 2 and 3 separately. In general, the two algorithms spend more time as the number of edges becomes larger. The edge deletion algorithm always runs slower than the edge extension algorithm. Furthermore, the speed difference between the two algorithms becomes larger when the average degree of the graph becomes large. The running time of the edge extension algorithm can be more than 10 orders of magnitude faster than the edge deletion algorithm. This may be explained by the vast search space of the edge deletion algorithm, especially when the edges in the graph are too many for processing and judging. However, the edge extension algorithm will return at the point of an edge-based minimal *k*-core occurring and does not need to process the left edges.

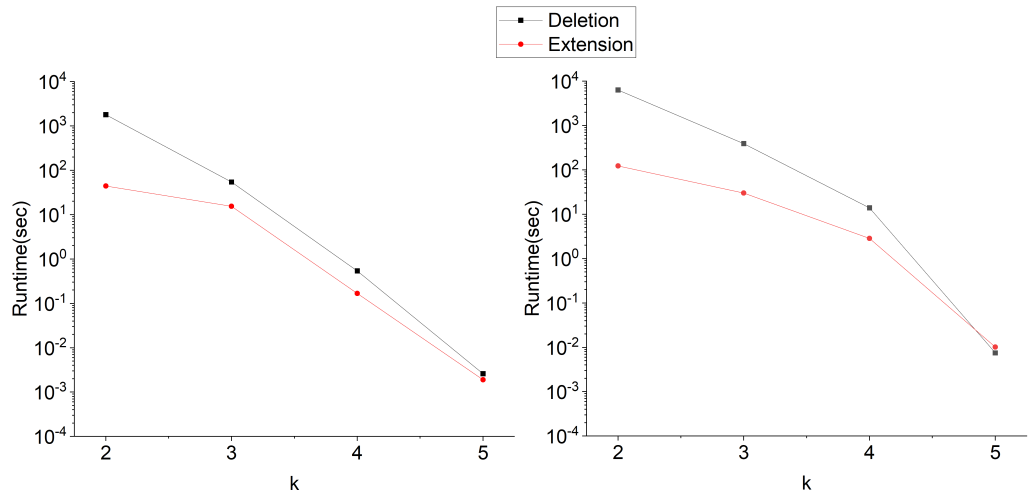


(a) k=2 (b) k=3

Figure 5. Comparison of the two algorithms.

Varying ***k*****.** We fix the dataset and vary the value of *k* to observe the change in runtime as *k* changes. Since when *k* is equal to 0 the edge-based minimal *k*-core is the query node itself, and when *k* is equal to 1 the edge-based minimal *k*-cores are formed from all the adjacent edges of the query node and another endpoint of these edges, it is convenient to find edge-based minimal 0-core and 1-core. The results of the experiment show that for all the datasets, when *k* is equal to 0 or 1, the runtime is nearly 0 s. Therefore, we cancel the display when *k* is 0 or 1 to present more representative results. We control *k* to vary linearly from 2 to 5.

The results are shown in Figure 6. Figure 6a,b show the runtime of the edge deletion algorithm and the edge extension algorithm on 10_0.3_5 and 10_0.4_5. As *k* becomes larger, the runtime becomes shorter rapidly in both algorithms. Note that the edge deletion algorithm cannot finish in ten thousand seconds on 10_0.4_5 when *k* is equal to 2. Furthermore, as *k* becomes greater, the time gap between the two algorithms becomes smaller. As we mentioned before, the time complexity depends on the number of edges we need to process. When *k* is smaller, the number of edges that are needed to be processed is more, hence the time cost is more expensive.



(a) 10_0.3_5 (b) 10_0.35_5

Figure 6. Varying *k* in the algorithms.

Optimization Performance. We choose the edge extension algorithm, which runs faster on the generated datasets, to conduct graph partitioning optimization and experiments on real datasets, fixing k equal to 2. From Figure 7, we can observe that the optimization is efficient and speeds up the running time on all the datasets. The running time has improved by up to four orders of magnitudes using the graph partition method. This is because partitioning the graph first and then computing the edge-based minimal k-cores in each subgraph contributes a lot to the time reduction.

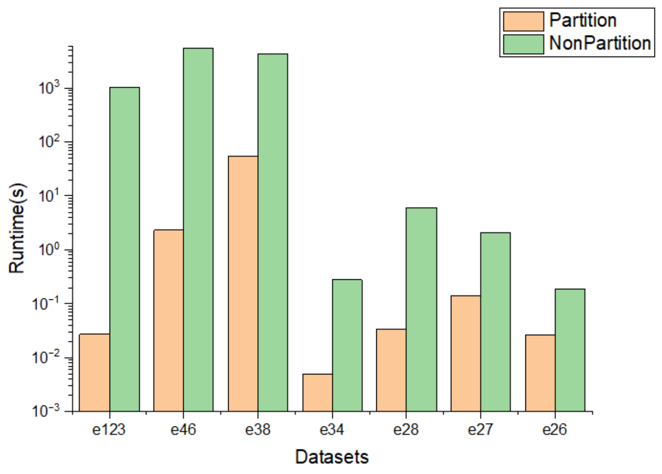


Figure 7. Efficiency of optimization on the edge extension algorithm.

6. Conversion

In this section, we will show that the EMK-SS problem can be easily converted from the EMK-q-SS problem. As we mentioned before, the EMK-SS problem aims to find all the edge-based minimal k-cores in a given graph, while the EMK-q-SS problem intends to find all the edge-based minimal k-cores containing the query node q in the graph. Obviously, the solution objective of the EMK-SS problem covers that of the EMK-q-SS problem. Therefore, we can obtain all the edge-based minimal k-cores containing the query node q by removing the edge-based minimal k-cores which do not contain the query node q after solving the EMK-SS problem. But how can we obtain the answers of the EMK-SS problem when we have the answers of EMK-q-SS?

Firstly, we find all the edge-based minimal k-cores containing the query node q by solving the EMK-q-SS problem given a node in the graph as a query node. Through setting every node in the graph as the query node for one time, we can obtain all the edge-based minimal k-cores in the graph. However, there still exist many edge-based minimal k-cores which repeat one another for two different query nodes and are probable to be in the same edge-based minimal k-cores. Hence, the last thing we should do is remove the redundant duplicate edge-based minimal k-cores in order that we have all the unique edge-based minimal k-cores. So far we have obtained the answers of EMK-q-SS from the answers of EMK-SS.

7. Related Works

The k-core computation problem is a fundamental problem in graph data mining and was first introduced by Seidman [24]. Seidman also derived an algorithm to obtain k-core of a given graph by removing the nodes with degree less than k recursively. k-core can be used in various applications, such as specific organization function prediction or understanding biology [25–28], maintaining stability in mutualistic ecosystems [29], internet graph evaluation at the autonomous system level in computer science [30,31], to measure users' influence in social networks [9,32–34], determining nodes' impact in information spreading [35–37], and community exploration in community detection [38–40]. Batagelj and Zaversnik [13] designed a linear in-memory algorithm to derive core numbers of vertices in a graph using a variant of bin-sort. In order to provide core decomposition for large graphs that are stored in secondary storage, I/O efficient algorithms for core decomposition have been proposed [41,42].

The minimum k-core search problem is attractive in k-core problems [10,15], for it is able to offer a reasonable strategy when the budget is limited in applications such as advertisement delivery. In [15], minimum k-cores are used to model cell assemblies and are obtained by using a backtrack search tree. Ref. [18] aims at finding the smallest k-core containing the given query nodes and delivers an approximate method which extends the partial solution from the query nodes.

Some researchers have studied the problem of minimum k-core in variant models. In [12], the authors try to obtain the minimum k-core after finding b edges to delete in the graph given budget b. The problem definition has been developed into two ways, one is to obtain the remaining graph with the least nodes, the other is to obtain the remaining graph with the least edges after b edges have been removed [43].

8. Conclusions and Future Work

In this paper, we investigate the edge-based minimal k-core subgraph search problem. We formally define the problem and prove its NP-hardness. Due to the hardness of the problem, we first propose an algorithm named the edge deletion algorithm to obtain all the answers. At the same time, we have another algorithm which starts the result set from the query node and extends to the answers. To further accelerate the two methods, graph partitioning techniques are utilized. Lastly, comprehensive experiments on generated networks and real datasets are conducted to demonstrate the effect of the proposed methods. The results show that the edge extension algorithm performs better than the edge deletion algorithm and our optimization technique outperforms the edge extension algorithm significantly.

Although the proposed edge deletion algorithm and edge extension algorithm have limitations on the running time, they may inspire efficient parallel and distributed solutions to further speed up the algorithms. Moreover, as the exact algorithm of the NP-hard problem has an expensive computation cost, approximation algorithms which are more scalable are expected to be proposed. In addition, it is also interesting to explore a variant of the EMK-SS problem and the EMK-q-SS problem. We can find all the edge-based minimal k-cores containing multiple query nodes to take multiple users into consideration at the same time.

Author Contributions: Conceptualization, T.W., Y.J. and J.Y.; methodology, T.W.; validation, T.W. and L.X.; formal analysis, T.W.; investigation, T.W.; writing—original draft preparation, T.W.; writing—review and editing, T.W., Y.J. and J.Y.; visualization, T.W.; supervision, Y.J. and J.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported in part by the National Key Research and Development Program of China (No. 2018YFB1800701), the PCL Project (No. PCL2021A02), the Cultivation Project of PZL (No. PZL2022KF0013), and the Natural Science Foundation of Guangdong Province (No. 2023A1515011655).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, H.; Zhang, X.; Zhao, C. Explaining social events through community evolution on temporal networks. *Appl. Math. Comput.* **2021**, *404*, 126148. [CrossRef]
2. Fang, A. The influence of communication structure on opinion dynamics in social networks with multiple true states. *Appl. Math. Comput.* **2021**, *406*, 126262. [CrossRef]
3. Wu, N.; Zhao, X.W.; Wang, J.; Pan, D. Learning effective road network representation with hierarchical graph neural networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Virtual, 6–10 July 2020; pp. 6–14.

4. Guo, K.; Hu, Y.; Qian, Z.; Liu, H.; Zhang, K.; Sun, Y.; Gao, J.; Yin, B. Optimized graph convolution recurrent neural network for traffic prediction. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 1138–1149. [CrossRef]
5. Kurshan, E.; Shen, H.; Yu, H. Financial crime & fraud detection using graph computing: Application considerations & outlook. In Proceedings of the 2020 Second International Conference on Transdisciplinary AI (TransAI), Irvine, CA, USA, 21–23 September 2020; pp. 125–130.
6. Cicerone, S.; Di Stefano, G.; Klavžar, S. On the mutual visibility in Cartesian products and triangle-free graphs. *Appl. Math. Comput.* **2023**, *438*, 127619. [CrossRef]
7. 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, New York, NY, USA, 24–27 August 2014; pp. 1316–1325.
8. Bhawalkar, K.; Kleinberg, J.; Lewi, K.; Roughgarden, T.; Sharma, A. Preventing unraveling in social networks: The anchored k-core problem. *SIAM J. Discret. Math.* **2015**, *29*, 1452–1475. [CrossRef]
9. Zhang, F.; Zhang, Y.; Qin, L.; Zhang, W.; Lin, X. Finding critical users for social network engagement: The collapsed k-core problem. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31.
10. Mikesell, D.; Hicks, I.V. Minimum k-cores and the k-core polytope. *Networks* **2022**, *80*, 93–108. [CrossRef]
11. Kroger, S. The Maximum Anchored k-Core Problem: Mixed Integer Programming Formulations. Ph.D. Thesis, Rice University, Houston, TX, USA, 2022.
12. Zhu, W.; Chen, C.; Wang, X.; Lin, X. K-core minimization: An edge manipulation approach. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, Torino, Italy, 22–26 October 2018; pp. 1667–1670.
13. Batagelj, V.; Zaversnik, M. An o (m) algorithm for cores decomposition of networks. *arXiv* **2003**, arXiv:cs/0310049.
14. Kong, Y.X.; Shi, G.Y.; Wu, R.J.; Zhang, Y.C. k-core: Theories and applications. *Phys. Rep.* **2019**, *832*, 1–32. [CrossRef]
15. Wood, C.I.; Hicks, I.V. The minimal k-core problem for modeling k-assemblies. *J. Math. Neurosci. (JMN)* **2015**, *5*, 14. [CrossRef]
16. Zhang, F.; Xie, J.; Wang, K.; Yang, S.; Jiang, Y. Discovering key users for defending network structural stability. *World Wide Web* **2022**, *25*, 679–701. [CrossRef]
17. Li, Z.; Hui, P.; Zhang, P.; Huang, J.; Wang, B.; Tian, L.; Zhang, J.; Gao, J.; Tang, X. What happens behind the scene? Towards fraud community detection in e-commerce from online to offline. In Proceedings of the WWW'21: The Web Conference 2021, Ljubljana, Slovenia, 19–23 April 2021; pp. 105–113.
18. Li, C.; Zhang, F.; Zhang, Y.; Qin, L.; Zhang, W.; Lin, X. Efficient progressive minimum k-core search. *Proc. VLDB Endow.* **2020**, *13*, 362–375. [CrossRef]
19. Ravald, A.; Grönroos, C. The value concept and relationship marketing. *Eur. J. Mark.* **1996**, *30*, 19–30. [CrossRef]
20. Abidi, A.; Zhou, R.; Chen, L.; Liu, C. Pivot-based Maximal Biclique Enumeration. In Proceedings of the IJCAI'20: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, Yokohama, Japan, 7–15 January 2020; pp. 3558–3564.
21. Zhang, Y.; Phillips, C.A.; Rogers, G.L.; Baker, E.J.; Chesler, E.J.; Langston, M.A. On finding bicliques in bipartite graphs: A novel algorithm and its application to the integration of diverse biological data types. *BMC Bioinform.* **2014**, *15*, 1–18. [CrossRef]
22. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W. H. Freeman and Company: New York, NY, USA, 1980.
23. 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, Snowbird, UT, USA, 22–27 June 2014; pp. 991–1002.
24. Seidman, S.B. Network structure and minimum degree. *Soc. Netw.* **1983**, *5*, 269–287. [CrossRef]
25. Altaf-Ul-Amine, M.; Nishikata, K.; Korna, T.; Miyasato, T.; Shinbo, Y.; Arifuzzaman, M.; Wada, C.; Maeda, M.; Oshima, T.; Mori, H.; et al. Prediction of protein functions based on k-cores of protein-protein interaction networks and amino acid sequences. *Genome Inform.* **2003**, *14*, 498–499.
26. Luo, F.; Li, B.; Wan, X.F.; Scheuermann, R.H. Core and periphery structures in protein interaction networks. *BMC Bioinform.* **2009**, *10*, 1–11. [CrossRef]
27. Schwab, D.J.; Bruinsma, R.F.; Feldman, J.L.; Levine, A.J. Rhythmogenic neuronal networks, emergent leaders, and k-cores. *Phys. Rev. E* **2010**, *82*, 051911. [CrossRef]
28. Van Den Heuvel, M.P.; Sporns, O. Rich-club organization of the human connectome. *J. Neurosci.* **2011**, *31*, 15775–15786. [CrossRef]
29. Morone, F.; Del Ferraro, G.; Makse, H.A. The k-core as a predictor of structural collapse in mutualistic ecosystems. *Nat. Phys.* **2019**, *15*, 95–102. [CrossRef]
30. Gaertler, M.; Patrignani, M. Dynamic analysis of the autonomous system graph. In Proceedings of the IPS 2004, International Workshop on Inter-domain Performance and Simulation, Budapest, Hungary, 22–23 March 2004; pp. 13–24.
31. Alvarez-Hamelin, I.; Dall'Asta, L.; Barrat, A.; Vespignani, A. k-core decomposition: A tool for the analysis of large scale Internet graphs. *arXiv* **2005**, arXiv:cs.NI/0511007.
32. Brown, P.; Feng, J. Measuring user influence on twitter using modified k-shell decomposition. In Proceedings of the International AAAI Conference on Web and Social Media, Barcelona, Spain, 17–21 July 2011; Volume 5, pp. 18–23.
33. Pei, S.; Muchnik, L.; Andrade, J.S., Jr.; Zheng, Z.; Makse, H.A. Searching for superspreaders of information in real-world social media. *Sci. Rep.* **2014**, *4*, 5547. [CrossRef] [PubMed]
34. Liu, Y.; Tang, M.; Zhou, T.; Do, Y. Core-like groups result in invalidation of identifying super-spreader by k-shell decomposition. *Sci. Rep.* **2015**, *5*, 1–8. [CrossRef] [PubMed]

35. Miorandi, D.; De Pellegrini, F. K-shell decomposition for dynamic complex networks. In Proceedings of the 8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, Avignon, France, 31 May–4 June 2010; pp. 488–496.
36. Garas, A.; Schweitzer, F.; Havlin, S. A k-shell decomposition method for weighted networks. *New J. Phys.* **2012**, *14*, 083030. [CrossRef]
37. Xiong, X.; Hu, Y. Research on the dynamics of opinion spread based on social network services. *Acta Phys. Sin.* **2012**, *61*, 150509. [CrossRef]
38. Giatsidis, C.; Thilikos, D.M.; Vazirgiannis, M. Evaluating cooperation in communities with the k-core structure. In Proceedings of the 2011 International conference on advances in social networks analysis and mining, Kaohsiung, Taiwan, 25–27 July 2011; pp. 87–93.
39. Pinar, A.; Kolda, T.G.; Peng, C. *Accelerating Community Detection by Using k-Core Subgraphs*; Technical Report; Sandia National Lab. (SNL-CA): Livermore, CA, USA, 2014.
40. Li, R.H.; Qin, L.; Yu, J.X.; Mao, R. Influential community search in large networks. *Proc. VLDB Endow.* **2015**, *8*, 509–520. [CrossRef]
41. 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, Hannover, Germany, 11–16 April 2011; pp. 51–62.
42. Wen, D.; Qin, L.; Zhang, Y.; Lin, X.; Yu, J.X. I/O efficient core graph decomposition at web scale. In Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering (ICDE), Helsinki, Finland, 16–20 May 2016; pp. 133–144.
43. Chen, C.; Zhu, Q.; Sun, R.; Wang, X.; Wu, Y. Edge manipulation approaches for k-core minimization: Metrics and analytics. *IEEE Trans. Knowl. Data Eng.* **2021**, *35*, 390–403. [CrossRef]