



VEK: a vertex-oriented approach for edge k -core problem

Zhongxin Zhou¹ · Wenchao Zhang² · Fan Zhang³ · Deming Chu⁴ · Binghao Li⁵

Received: 1 February 2021 / Revised: 7 May 2021 / Accepted: 1 June 2021 /

Published online: 17 June 2021

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract

The stability of a network has been widely studied as an important indicator of network status, e.g., reliability and activity. A popular model for measuring the (structural) stability of a network is k -core, the maximal induced subgraph in which every vertex has at least k neighbors in the subgraph. As the size of k -core well estimates the stability of a network, the edge k -core problem is studied to enhance network stability: given a graph G , an integer k and a budget b , add b edges to non-adjacent vertex pairs in G such that the number of vertices in the k -core is maximized. The state-of-the-art solution is a greedy algorithm (named EKC) by finding a promising vertex pair at each iteration, while the algorithm is still not efficient enough on large graphs. In this paper, we propose a novel vertex-oriented heuristic algorithm (named VEK), with a well-designed scoring function to guide the search order. Effective optimization techniques are proposed to prune unpromising candidates and reuse the intermediate results. The experiments on 9 real-life datasets demonstrate the runtime of our proposed VEK is faster than the state-of-the-art algorithm EKC by 1-3 orders of magnitude.

Keywords Graph processing · Social network · K-core · Network editing

1 Introduction

Graph is a ubiquitous structure for modeling networks in various areas including social networks [24, 40, 41, 43], P2P networks [7], biological networks [1], web networks [39], etc. In a graph, each vertex represents an entity, and each edge represents the relationship between two entities. To recognize groups of well-connected vertices, a variety of cohesive subgraph models are proposed such as clique [4], k -core [28], k -truss [10], and k -ecc [5]. The model of k -core is well-studied for its elegant structure and efficient algorithms to

This article belongs to the Topical Collection: *Special Issue on Large Scale Graph Data Analytics*
Guest Editors: Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang

Zhongxin Zhou and Wenchao Zhang are the joint first authors.

✉ Fan Zhang
zhangf@gzhu.edu.cn

Extended author information available on the last page of the article.

compute [31]. It is defined as a maximal subgraph in which every vertex has at least k neighbors (adjacent vertices) in the subgraph. There are various applications based on the model of k -core such as community discovery [24], network engagement analysis [22], finding influential spreaders [17], social contagion [34], finding molecular complexes [1], etc.

Assume each entity incurs a cost (e.g., k) to remain engaged in a network but receives a benefit proportional to (e.g., equal to) the number of engaged neighbors (adjacent vertices), the k -core corresponds to the natural equilibrium of this engagement model [3]. Thus, the size (the number of vertices) of k -core in a network well estimates the (structural) stability of this network. In order to improve the stability of a network, current studies focus on enlarging the size of k -core such that the vertices in the network are better connected [42]. Considering the linking of non-adjacent vertices, the edge k -core problem is studied recently [46]: given a graph G , an integer k and a budget b , add b edges to non-adjacent vertex pairs in G such that the size of k -core is maximized.

Example 1 Consider a graph G in Figure 1(a), the coreness of each vertex is marked with a value. The whole graph G forms a 1-core because every vertex has at least one neighbor in G , and the subgraph induced by vertices v_1, v_2, v_3 and v_4 is a 3-core. Assume that $k = 3$ and $b = 2$, the edge k -core problem aims to maximize the 3-core by adding two new edges into graph G . One of the optimal results is to add (v_5, v_9) and (v_7, v_8) . Figure 1(b) depicts the modified graph G' . By adding only two edges, we can get a larger 3-core induced by the vertices in G except v_{10} and v_{11} .

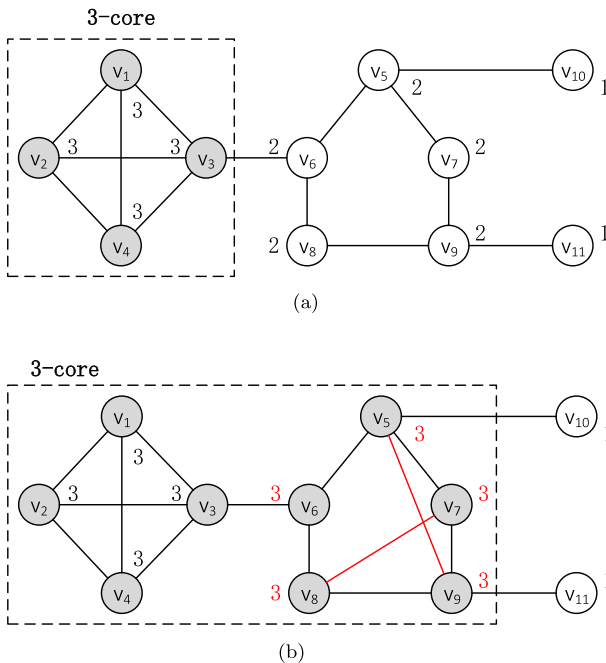


Figure 1 An example of Edge k -core problem. Here, $k = 3$ and $b = 2$. By adding two new edges (v_5, v_9) and (v_7, v_8) , the new 3-core is maximized with five more vertices $\{v_5, v_6, v_7, v_8, v_9\}$

Applications As a fundamental problem, the edge k -core can be used in many applications, e.g., connection construction in telecom networks, link recommendation for a group of webs, etc. Some examples are as follows.

- *Peer-to-Peer (P2P) Networks.* For any machine (vertex) to steadily benefit from a P2P network, they should be connected to at least a certain number (e.g., k) of other machines for data exchange. The edge k -core problem can figure out which connections (edges) shall be established so that many devices can successfully retrieve the resources from others and share the data with others [7]. Besides, in mobile P2P network, the edge k -core problem may also benefit the users getting resources from surrounding users instead of accessing the central cloud for low-latency connection [45].
- *Social Networks.* In current online social platforms, there are various user groups based on different themes, e.g., for organizing a hiking or an online game. To improve the engagement of users in a group, it is critical to promote the social connections of group members. The edge k -core problem can be used to recommend key connections such that user engagement of a group is largely improved.
- *Online Game.* In multiplayer games such as “Death Stranding”, users share resources based on social connections. Due to the importance of connection in the game experience, every player must establish connections with enough players, i.e., a threshold of k players. In this regard, the edge k -core can be used to find key connections and enhance the player experience.

Challenges The main challenge of edge k -core problem stems from the huge number of candidate edges (non-adjacent vertex pairs), because the real-life networks are usually quite sparse. Although the k -core can be computed in $O(m)$ time by recursively removing the vertices with degree less than k , a straightforward exact solution has to exhaustively compute the size of k -core for every possible candidate edge set with size b . It is cost-prohibitive even if b is very small. From the theoretical side [46], the edge k -core problem is proven to be NP-hard when $k \geq 3$. The problem is even hard to have an approximate solution, as it cannot be approximated in polynomial time within a ratio of $(1 - 1/e + \epsilon)$ for $k \geq 3$ and any $\epsilon > 0$, unless $P = NP$.

The state-of-the-art algorithm (named EKC) for the edge k -core problem is proposed in [46]. In each of the b iterations, EKC greedily finds a best pair of vertices such that their connection leads to the largest k -core for current iteration. The result quality (the size of k -core with added edges) of the greedy solution (EKC) is much better than other heuristics, as evaluated in [46]. Although many unpromising candidates are pruned by the optimizations in EKC, the remaining candidate edges are still too many to enable an efficient solution on large graphs. For a graph with about 2 million edges, EKC costs more than 1000 seconds on a commercial machine when $k = 20$ and $b = 5$, as reported in [46]. Consequently, it is crucial to carefully design a novel heuristic with much less time cost while the result quality is basically not sacrificed.

Our solution As the number of vertices is much fewer than the number of edges even for sparse graphs, a vertex-oriented approach can be much more efficient than the greedy EKC. That is, we consider to find the results (the edges to be inserted) based on the idea of preserving outside vertices (through edge additions) in k -core. We first precisely locate the candidate vertices outside of k -core which are worthwhile to be preserved (i.e., anchored). A well-designed scoring function is proposed to select most promising vertices such that the k -core is enlarged by a few new edges. In order to further reduce the addition cost (the number

of edges need to be added for vertex anchoring), we design a budget rebate strategy to adjust the added edges incident to different anchored vertices. The k -core computation with added edges in [46] is improved by using better data structure. We also introduce an optimization to reuse intermediate results. Moreover, the VEK algorithm runs in $O(b \cdot n \cdot m)$ time, which is superior to the state-of-the-art method EKC [46]. As verified in our experiments, our proposed VEK is much faster than EKC while their result qualities are similar.

Contributions Our main contributions in this paper are as follows:

- We analyze the limitations of the state-of-the-art algorithm EKC for the edge k -core problem, and propose a novel vertex-oriented approach VEK to achieve better performance.
- Several optimization techniques are proposed to improve the result quality of VEK and the efficiency, including a well-designed scoring function for vertex selection, a budget rebate strategy, a result-reusing method, etc.
- Our experiments on 9 real datasets demonstrate our proposed VEK is faster than the state-of-the-art EKC by 1-3 orders of magnitude, and the result qualities of them (the sizes of k -core with b inserted edges) are similar.

Roadmap The rest of the paper is organized as follows. Section 2 formally defines the problem, and introduces the existing solution. We then propose our VEK algorithm in Section 3. The experiments are conducted in Section 4. Section 5 surveys the related works. Finally, Section 6 concludes the paper.

2 Preliminaries

2.1 Problem definition

Let $G = (V, E)$ be an undirected graph, where V and E are the set of vertices and edges, respectively. We denote $n = |V|$, $m = |E|$ and assume $m > n$. Given a subgraph S , the neighbor set of u is denoted by $N(u, S)$. The degree of u in S , i.e., $|N(u, S)|$, is denoted by $deg(u, S)$. We may omit the target graph in notations when the context is clear, e.g., we use $deg(u)$ instead of $deg(u, G)$. Table 1 summarizes the notations in this paper.

The k -core of a graph is the maximal subgraph where every vertex has at least k neighbors in the subgraph. We use $C_k(G)$ (or C_k) to denote the k -core of the original graph G .

Definition 1 (k -Core) Given a graph G , a subgraph S is the k -core of G , denoted by C_k , if (i) $deg(u, S) \geq k$ for every vertex $u \in S$; and (ii) S is maximal, i.e., any supergraph $S' \supset S$ is not a k -core.

The k -core can be obtained by recursively removing every vertex with degree less than k in the graph, with a time complexity of $O(m)$ [2]. The *coreness* of a vertex u , denoted by $cn(u)$, is the largest k such that u is in the k -core, i.e., $\max_{k:u \in C_k} \{k\}$. We use H_k to denote the k -shell of G , which is the set of vertices with coreness equals to k , i.e., $H_k = \{u \mid cn(u) = k\}$.

If we add an edge between a pair of vertices which are not adjacent, the k -core of the graph may be enlarged, which is named the edge k -core. The added new edge is called an anchor or anchor edge. The set of added edges, i.e., the anchor set, is denoted by E_A . We

Table 1 Summary of Notations

Notation	Definition
G	an unweighted and undirected graph
$u, v; e, (u, v)$	a vertex in G ; an edge in G
$m; n$	the number of edges in G ; the number of vertices in G
$N(u, G)$	the set of adjacent vertices (neighbors) of u in G
$deg(u, G)$	the number of adjacent vertices of u in G
S	a subgraph of G
$V(S); E(S)$	the vertex set of S ; the edge set of S
$cn(u)$	the coreness of the vertex u
$C_k(G); H_k(G)$	the k -core of G ; the k -shell of G
$k; b$	the degree constraint; the anchor budget
E_A	a set of anchor edges
G_A	the graph $G + E_A$
$\mathcal{F}(E_A)$	the followers of the anchor edge set E_A in G
\mathcal{L}	the onion layers of G [42]

use G_A to denote G with E_A added, i.e., $G_A = (V, E \cup E_A)$. In this paper, we say anchor, add, or insert an edge interchangeably, e.g., an inserted edge is also called an anchored edge. The edges, which may be inserted into the graph, are called candidate anchor edges.

Definition 2 (Edge k -Core) Given a graph $G = (V, E)$ and a set of anchor edges E_A , the edge k -core, denoted by $C_k(G_A)$, is the k -core of G_A .

After adding the anchor edges E_A , some vertices not in $C_k(G)$ may follow the anchor edges to join the edge k -core. We use the followers of E_A to denote these vertices.

Definition 3 (Follower Set) Given a graph $G = (V, E)$ and a set of anchor edges E_A , the follower set of E_A , denoted by $\mathcal{F}(E_A)$, is the set of vertices that are in the edge k -core but not in the k -core, i.e., $\mathcal{F}(E_A)$ is the vertex set of $C_k(G_A) - C_k(G)$.

The number of followers reflects the importance of the corresponding anchor edges. Given a graph G and a set E_A , the vertex set of $C_k(G_A)$ is same to the vertex set of $C_k(G) \cup \mathcal{F}(E_A)$.

Problem statement Given a graph G , a degree constraint k and an edge budget b , the edge k -core problem aims to find a set E_A of b edges in $\binom{V}{2} \setminus E$ such that the number of followers of E_A is maximized, i.e., $|\mathcal{F}(E_A)|$ is maximized.

Problem hardness Given a set of anchor edges E_A , the edge k -core $C_k(G_A)$ can be obtained in $O(m)$ time by computing the k -core of G_A . However, it is challenging to find the optimal E_A , due to the large volume of candidate edges. Chitnis and Talmon [7] show that the edge k -core problem is polynomial-time solvable for $k \leq 2$ and W[1]-hard. They also propose an algorithm for graphs with bounded tree-width. In [46], the problem is proved to be NP-hard for $k \geq 3$. Besides, the problem cannot be approximated in polynomial time within a ratio of $(1 - 1/e + \epsilon)$ for $k \geq 3$ and any $\epsilon > 0$, unless $P = NP$.

2.2 EKC algorithm: state-of-the-art

EKC [46] is the state-of-the-art approach for the edge k -core problem on general graphs, and Algorithm 1 outlines its pseudo-code. E_A is the result set of anchor edges, while E_{cand} is a set of candidate anchor edges. We use $\mathcal{F}(e, G_A)$ to denote the followers of e , after adding it to E_A .

Algorithm 1: EKC(G, k, b).

Input : G : a graph, k : degree constraint, b : edge budget.
Output: E_A : a set of anchor edges

```

1  $E_A \leftarrow \emptyset$ ;
2 while  $|E_A| < b$  do
3    $E_{cand} \leftarrow \text{GetCandidates}(G_A, k)$ ;
4   foreach  $e \in E_{cand}$  do
5      $\mathcal{F}(e, G_A) \leftarrow \text{FindFollower}(e, G_A)$ ;
6      $E_{cand} \leftarrow \text{PruneCandidates}(E_{cand})$ ;
7    $E_A \leftarrow E_A \cup e^*$ , where  $e^*$  has the highest  $|\mathcal{F}(e^*, G_A)|$  in Line 1;
8 return  $E_A$ 

```

Algorithm 1 iteratively finds the anchor edge with the most followers until E_A contains b edges. Line 1–1 selects a new anchor edge in each round. Specifically, we first compute the candidate edges E_{cand} (Line 1), then we find the followers of each candidate edge (Line 1) while pruning the candidate edge set (Line 1), and finally insert into E_A the edge e^* that incurs the most followers (Line 1). The time complexity of EKC is $O(b \cdot n^2 \cdot m)$ and it cannot scale to large graphs.

The techniques in EKC are summarized as follows. $\text{GetCandidates}(G_A, k)$ returns the candidate edges E_{cand} base on the $(k - 1)$ -shell structure and a onion-layer based pruning, to reduce the size of E_{cand} . $\text{FindFollower}(e, G_A)$ returns the followers incurred by adding e to E_A , using an onion-layer based follower computation optimization proposed in [42]. $\text{PruneCandidates}(E_{cand})$ prunes the candidate edge set according to the followers of e . Throughout the iterations, the algorithm reuses the intermediate results of connected components.

3 Vertex-oriented solution

In this section, we present our proposed vertex-oriented algorithm VEK. Section 3.1 introduces our basic idea of VEK. Section 3.2 prunes the search space of candidate pivots. Section 3.3 presents a scoring function that can choose a promising pivot vertex, and Section 3.4 enhances this scoring function by two edge budget saving strategies. Finally, Section 3.5 combines all the techniques of VEK together.

3.1 Basic ideas

Despite adopting several optimization techniques, EKC runs in $O(b \cdot n^2 \cdot m)$ time, which cannot scale to large graphs. The time cost comes from an enormous set of candidate edges whose worst-case size is n^2 . Motivated by this, we develop a vertex-oriented heuristic that considers the potential of each vertex rather than that of each edge. We evaluate the potential

of a vertex by anchor score (Section 3.3), which aims to introduce more vertices into the k -core at a lower cost. In each iteration, we select a pivot with the highest anchor score from the pruned search space (Section 3.2), and find the anchor edges that can bring the pivot into the k -core. These anchor edges are returned as an approximation solution of the edge k -core problem. Section 3.4 further designs two strategies that optimize the quality of anchor score.

Definition 4 (Pivot) Given a graph $G = (V, E)$ and a set of anchor edges E_A , the pivot vertex is the vertex with highest anchor score in $G' = (V, E + E_A)$.

3.2 Candidate pivot pruning

By connecting the anchor edges between a pivot and the k -core, we add the pivot into the edge k -core along with its followers. To select a pivot that brings in followers and enlarges the edge k -core, we derive the necessary conditions for a vertex to be the follower of a pivot.

In the following theorem, u is a pivot and E_A is an anchor edge set that (i) bridges u and the k -core $E_A = \{(v, u) \mid v \in C_k\}$, and (ii) brings u into the edge k -core $u \in C_k(G_A)$. We have that u 's follower set $\mathcal{F}(u) = C_k(G_A) \setminus C_k$ is contained in the $(k - 1)$ -shell.

Theorem 1 Given a graph G , a vertex u , and a set of anchor edges $E_A = \{(v, u) \mid v \in C_k\}$ such that $u \in C_k(G_A)$, the followers of u are all from the $(k - 1)$ -shell, i.e., $\mathcal{F}(u) \subseteq H_{k-1}$.

Proof Proof by contradiction. Assume there is a follower v of u that is not in the $(k - 1)$ -shell. We have $cn(v) < k - 1$, because a follower must have coreness less than k and $cn(v) \neq k - 1$ ($v \notin H_{k-1}$).

According to the definition of k -core, for each $x \in C_k(G_A)$, we have $deg(x, C_k(G_A)) \geq k$.

After removing u and E_A from $C_k(G_A)$, we have $deg(x, C_k(G_A) \setminus u) \geq k - 1$ for each vertex $x \in C_k(G_A) \setminus u$, because all edges in E_A are incident to u , and the degree of x decreases by at most 1. So, $C_k(G_A) \setminus u$ is a $(k - 1)$ -core according to the definition. We have $cn(v) \geq k - 1$ since $v \in C_k(G_A) \setminus u$, which contradicts $cn(v) < k - 1$. \square

Let $u \in V \setminus C_k$ be a pivot we select. Theorem 1 states that u can bring in new followers only if its neighbors are from the $(k - 1)$ -shell. Thus, we select the candidate pivots from the neighbors of the $(k - 1)$ -shell, namely $N(H_{k-1})$.

Assume we have already selected E_A as anchor edges, the theorem can also be applied. The candidate pivot set would be $N(H_{k-1}(G_A))$, by regarding G_A as the original graph in Theorem 1.

3.3 Anchor score evaluation

In this section, we introduce anchor score that evaluates the potential of a vertex to be a pivot. Intuitively, we aim to add more followers using as few anchor edges as possible. The anchor score of a vertex u is proportional to the number of u 's followers $|\mathcal{F}(u)|$ (the vertices added to the edge k -core) and in inverse proportion to the *anchor cost* of u , i.e., how many anchor edges we must use.

The anchor cost of a vertex u measures how many anchor edges are needed to bring u into the edge k -core. The following theorem not only gives the minimum value for the anchor cost of a pivot u , but also ensures there is always a set of anchor edges satisfying this lower bound.

Theorem 2 Given a graph G , a vertex u , for a set of anchor edges $E_A = \{(v, u) \mid v \in C_k\}$ such that $u \in C_k(G_A)$, the size of E_A is at least $\varphi(u) = k - |N(u) \cap C_k| - |N(u) \cap \mathcal{F}(u)|$ and there is always a E_A satisfying $|E_A| = \varphi(u)$.

Proof Assume we select u as a pivot. According to the definition of follower, the followers of u will not leave the edge k -core unless u leaves. Initially, u has $|N(u) \cap C_k|$ edges to the k -core. Then u 's followers contribute $|N(u) \cap \mathcal{F}(u, G)|$ degrees to u . If we anchor $k - |N(u) \cap C_k| - |N(u) \cap \mathcal{F}(u, G)|$ more edges between the k -core and u , we have $\deg(u, C_k(G_A)) = k$. We can always construct such an E_A . \square

Theorem 2 shows the cost (the number of required anchor edges) of adding a pivot into the edge k -core. On the other hand, the number of followers measures the gains of adding a pivot into the edge k -core. The anchor score $\phi(u)$ is defined as the number of followers divided by the anchor cost.

$$\phi(u) = \frac{|\mathcal{F}(u)|}{\varphi(u)} = \frac{|\mathcal{F}(u)|}{k - |N(u) \cap C_k| - |N(u) \cap \mathcal{F}(u)|}$$

A pivot with a higher anchor score can bring in more edges at a lower cost. Similar to Theorem 1, Theorem 2 can also be applied to G_A with E_A added, by considering G_A as the original graph.

$$\phi(u, G_A) = \frac{|\mathcal{F}(u, G_A)|}{\varphi(u, G_A)} = \frac{|\mathcal{F}(u, G_A)|}{k - |N(u) \cap C_k(G_A)| - |N(u) \cap \mathcal{F}(u, G_A)|} \tag{1}$$

Compute $\mathcal{F}(u)$: the followers of a vertex We adopt the onion-layer based follower computation in [42], which explores candidate followers layer by layer, with a heap to maintain the traversal order. The heap can be replaced by multiple queues. When generating a candidate follower, instead of inserting it into a heap keyed by the layer number, we push it into the queue of the corresponding layer, so that the maintenance cost decreases from $O(\log n)$ to $O(1)$.

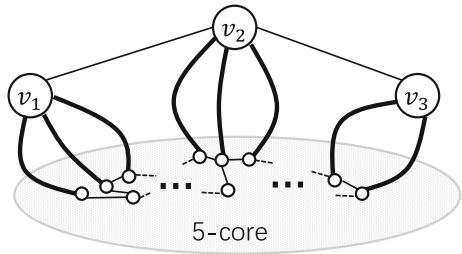
Compute anchor cost $\varphi(u)$ We conduct a core decomposition [16] before each iteration, so that $|N(u) \cap C_k(G_A)|$ for any vertex u can be pre-computed in $O(m)$ time. We also maintain a hash table that answers if a vertex pair is adjacent. After that, $|N(u) \cap \mathcal{F}(u, G_A)|$ for a given u can be answered in $O(|\mathcal{F}(u, G_A)|)$ time, by traversing the followers of u and check if it is a neighbor of u .

3.4 Optimized anchor score

In Section 3.3, we select a pivot u with the highest anchor score in each iteration. The scoring function can be further optimized by looking back. Specifically, we can replace or delete some selected anchor edges, such that we use fewer anchor edges to retain the vertices in the edge k -core. This saves the edge budget for further pivot selections.

Example 2 (Look-Back Ability) Figure 2 can illustrate the need for look-back ability. In this example, we seek to maximize the size of the edge 5-core. The edges between the 5-core and the outside are highlighted in bold. Using the anchor score in Section 3.3, we first select v_1 using 2 anchor edges, then select v_2 with two edge budgets, which results in a follower v_3 . The total edge budget is 4. However, we can anchor (v_1, v_3) and an edge from v_3 to the 5-core, to bring all three vertices into the 5-core. By looking back and removing the redundant edges, we can save 2 edge budgets in this example.

Figure 2 Maximize the Size of the Edge 5-Core. The edges between the 5-core and $\{v_1, v_2, v_3\}$ are bold



In the following, we discuss two strategies that optimize the effectiveness of anchor score. The optimized anchor score uses (i) *Budget Return*: deletes redundant anchor edges and return its budget; (ii) *Edge Reconnection*: reconnects anchor edges to save budget without losing the vertices in the edge k -core.

Budget return (Line 2-2) Assume we bring v into the k -core using anchor edges E_A , and then we want to bring in another pivot u . The followers of u are $\mathcal{F}(u, G_A)$. Let $C = E_A \cap \{(v, v') \mid v' \in C_k\}$ and $D = N(v) \cap \mathcal{F}(u, G_A)$.

Here, C is a set of selected anchor edges between the original k -core and v , while D is the set of u 's followers that are adjacent to v . The followers of u can add $|D|$ degrees to v . On the other hand, the anchor edges in C , which are previously used to increase the degree of v , can be removed due to u 's followers. To sum up, we can safely delete $\min(|C|, |D|)$ edges in C from the anchor edge set.

Example 3 (Budget Return) In Figure 2, assume we first (Step.A) anchor pivot v_1 into the 5-core with 2 anchor edges, then (Step.B) bring pivot v_2 into the 5-core using two anchor edges, and result in a follower v_3 . The degree of v_1 increase by 1 due to the (v_1, v_3) , so we can return one edge budge that is used to increase the degree of v_1 in Step.A, while all vertices still exist in the k -core.

Edge reconnection (Line 2-2) Suppose a pivot v has been inserted into the k -core using anchor edges E_A , and then another pivot u is to be selected. Let $C = E_A \cap \{(v, v') \mid v' \in C_k\}$, which is the anchor edges between the original k -core and v . The new pivot u needs $\varphi(u)$ additional degrees to engage in the edge k -core. Connecting u to the k -core can increase u 's degree by one. However, connecting (u, v) can contribute to the degree of both v and u , so that one anchor edge in C can be removed without affecting the edge k -core.

Example 4 (Edge Reconnection) In Figure 2, assume v_1 has been inserted in the edge 5-core, and v_1 and v_3 is not adjacent. When selecting v_3 as a pivot, we can connect (v_1, v_3) and remove an anchor edge from v_1 to k -core (no edge budget is used). Then, rather than two edges, we only need to add one edge from the 5-core to v_3 to insert v_3 into the edge 5-core.

Algorithm 2 illustrates how to compute the optimized anchor score for a given vertex u . V_P is the set of selected pivot vertices, E_A is the set of anchor edges we select. We use E_{next} to maintain the set of anchor edges, and initialize it with E_A . $Cost$ is the anchor cost of u , i.e., how many edges we need to connect between u and the original k -core, while $Score$ is the anchor score u .

Algorithm 2: *GetAnchorScore**(u, V_P, G_A).**Input** : u : a vertex, V_P : a set of selected pivots, E_A : a set of anchor edges**Output**: *Score*: the optimized anchor score of u , E_{next} : the set of anchor edges after selecting u as a pivot

```

1  $E_{next} \leftarrow E_A$ ;
2  $Cost \leftarrow k - |N(u) \cap C_k(G_A)| - |N(u) \cap \mathcal{F}(u, G_A)|$ ; // Section 3.3
3 foreach  $v \in V_P$  do
4    $C \leftarrow E_{next} \cap \{(v, v') \mid v' \in C_k\}$ ;
5    $D \leftarrow N(v) \cap \mathcal{F}(u, G_A)$ ;
6   if  $|C| > 0$  then
7      $E_{next} \leftarrow E_{next} \setminus \{\min(|C|, |D|) \text{ edges in } C\}$ ;
8 foreach  $v \in V_P \setminus N(u)$  do
9    $C \leftarrow E_{next} \cap \{(v, v') \mid v' \in C_k\}$ ;
10  if  $|C| > 0$  and  $Cost > 0$  then
11     $Cost \leftarrow Cost - 1$ ;
12     $E_{next} \leftarrow E_{next} \cup (u, v) \setminus \{\text{one edge in } C\}$ ;
13  $Cost \leftarrow Cost - (|E_A| - |E_{next}|)$ ;
14  $Score \leftarrow \frac{|\mathcal{F}(u, G_A)|}{Cost}$  if  $Cost > 0$  else  $\infty$ ;
15 return  $Score, E_{next}$ ;

```

The initial anchor cost of u is computed in Line 2. Line 2–2 apply the budget return strategy described above, where we compute C, D and remove $\min(|C|, |D|)$ redundant edges in C from E_{next} . Line 2-2 conduct the edge reconnection strategy. For each selected pivot v that is not adjacent to u , we use (u, v) to replace an anchor edge $e \in C$. Line 2 decreases $Cost$ by $(|E_A| - |E_{next}|)$ the number of anchor edges we removed from E_A . In Line 2, $Cost$ is the number of anchor edges to connect between the original k -core and u , so Line 2 divides the number of followers by the cost to get the optimized anchor score. In case $Cost \leq 0$, Line 2 sets $Score$ to ∞ , because the edge budget would not decrease after selecting u as a pivot, i.e., we should choose u anyway. Finally, Line 2 returns the results.

3.5 VEK Algorithm

Algorithm 3 organizes the techniques in the previous sections into our proposed VEK algorithm, which implements the ideas in Section 3.1. The algorithm keeps picking a pivot vertex from the pruned candidates (Section 3.2) whose optimized anchor score (Section 3.4) is the highest, until the edge budget is used up.

V_P is the set of pivot vertices we have selected, E_A is the set of anchor edges attached to the pivots. The arrays in Line 3 exactly match the return value of Algorithm 2, that is, $Score[u]$ stores the optimized anchor score of u , and $E_{next}[u]$ stores the set of anchor edges after selecting u as a pivot.

Line 3 initializes the pivot vertex set V_P and the anchor edge set E_A . Line 3-3 selects a pivot u^* and updates E_A using the selected pivot, until the number of anchor edges $|E_A|$ reaches the edge budget b . Line 3–3 chooses from the pruned candidate set (Section 3.2) a pivot u^* such that (i) the optimized anchor score $Score[u^*]$ (Section 3.4) is the highest, and (ii) after choosing u^* , the number of anchor edges $|E_{next}[u^*]|$ are within the budget

b . Line 3 terminates the loop if there is no valid pivot to select. Otherwise, Line 3 adds u^* to the pivot set V_P , and Line 3 updates E_A to E_{next} which is the set of anchor edges after selecting u^* as a pivot. Line 3 returns the result anchor edges E_A .

Algorithm 3: VEK(G, k, b).

Input : G : a graph, k : degree constraint, b : edge budget
Output: E_A : a set of anchor edges

```

1  $V_P \leftarrow \emptyset$ ;  $E_A \leftarrow \emptyset$ ;
2 while  $|E_A| < b$  do
3   foreach  $u \in N(H_{k-1}(G_A))$  do // Section 3.2
4      $Score[u], E_{next}[u] \leftarrow GetAnchorScore^*(u, V_P, G_A)$ ; // Algorithm 2
5      $u^*$  has the highest  $Score[u^*]$  (Line 3) where  $|E_{next}[u^*]| \leq b$ ;
6     if  $u^* = \text{NULL}$  then break;
7      $V_P \leftarrow V_P \cup u^*$ ;
8      $E_A \leftarrow E_{next}[u^*]$ ;
9 return  $E_A$ 

```

Reuse intermediate results The intermediate results of Algorithm 3 can be reused. In each round of Line 3–3, only one vertex u^* is selected as a pivot. For a vertex v that is not connected to u^* in the original graph G , the selected pivot u^* will not affect v , so we can reuse the optimized anchor score and the follower set of v in the next round. By pre-computing the connectivity of the graph, we can efficiently skip those vertices and reuse the results.

Algorithm complexity We analyze the time and space complexity of VEK. VEK iteratively introduces a most promising vertex into k -core in each round until the budget is exhausted. Although two strategies Budget Return and Edge Reconnection are adopted to maximize the effectiveness of the budget, the vertex selected by the algorithm has to be linked to at least one addition edge finally. Considering that the budget is b , VEK runs at most $2b$ rounds. In one iteration, VEK evaluates the advanced anchor score for each candidate pivot. VEK takes $O(m)$ time at most for the evaluation of one candidate pivot. The size of candidate pivot set $|N(H_{k-1}(G_A))| \leq n$. Then VEK can update the graph in $O(1)$ time. As a result, the overall time complexity is $O(b \cdot n \cdot m)$. It costs $O(m)$ to store the graph and $O(n)$ space to maintain the candidate pivot. Anchor score and corresponding anchor edge set also takes $O(n)$ space to keep in record. Thus, the space complexity of VEK is $O(m)$.

4 Experiment evaluation

4.1 Experimental setting

Datasets Flickr dataset is from Network Repository¹, and the others are from SNAP². Table 2 shows the statistics of the datasets deployed in the experiments.

¹<http://networkrepository.com/>

²<https://snap.stanford.edu/data/>

Table 2 Statistics of datasets

Dataset	Vertices	Edges	d_{avg}	k_{max}
Facebook	4,039	88,234	43.69	115
Enron	36,692	183,831	10.02	43
Brightkite	58,228	214,078	7.35	52
Gowalla	196,591	950,327	9.67	51
DBLP	317,080	1,049,866	6.62	113
Twitter	81,306	1,768,149	33.02	96
Stanford	281,903	2,312,497	14.14	71
YouTube	1,134,890	2,987,624	5.27	51
Flickr	513,969	3,190,452	12.41	309

Algorithms. Our proposed algorithm VEK is described in Section 3.5. We compare VEK with existing solutions including EKC and other heuristics as introduced in [46]. The onion layers \mathcal{L} in [42] is an auxiliary structure to facilitate the follower computation and candidate pruning for EKC. Table 3 shows the summary of algorithms.

Parameters All programs are implemented in C++. All experiments are performed on Intel Xeon 2.20GHz CPU and Linux system. We vary the parameters k and b . The effectiveness is evaluated by the number of the followers produced. The efficiency is measured by running time.

4.2 Effectiveness

Figure 3 compares the number of followers w.r.t. b anchor edges produced by all the algorithms in Table 3, when $k = 10$ and $b = 20$. In Figure 3, Layer or Degree fail to get any followers in several datasets, because a vertex in higher onion layer or with a larger degree does not necessarily have a follower. For Rand, we report the average number of followers for 200 independent tests. Given the inputs of k and b , the resulting number of followers is fixed for any of the other four approaches. If we randomly anchor (insert) edges in the whole graph, it is common to get no follower after 20 random insertions. It is because the candidate edges with more than 1 follower are quite rare in the graph, while the number of candidate edges is large. Thus, in Rand, we only choose the edges from the set E' where each edge has at least one follower if inserted. Note that the computation of E' costs more

Table 3 Summary of algorithms

Algorithm	Description
Rand	randomly chooses b anchors (each with at least one follower) from $E'_{cand} = \{(u, v) \mid u \in H_{k-1}, v \in C_{k-1}\} \setminus E(C_{k-1})$
Degree	chooses b anchors from E'_{cand} with largest degrees
Layer	chooses b anchors from E'_{cand} at highest onion layers in \mathcal{L}
EKC	the state-of-the-art algorithm in [46] (Section 2.2)
VEK	our proposed vertex-oriented approach (Section 3.5)

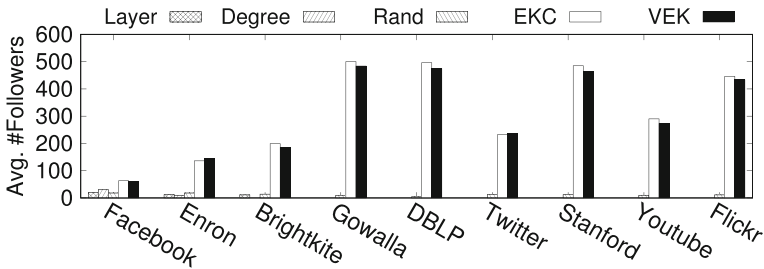


Figure 3 Effectiveness of different heuristics, $k = 10, b = 20$

time than VEK, as it requires to compute the follower set of each candidate vertex pair (candidate anchor edge).

Although the majority of the candidate edges does not have any followers, the state-of-the-art EKC produces many followers in the experiments, which shows the greedy heuristic is effective. For our proposed VEK, with the help of promising pivot vertices, we can easily identify a small part of candidate edges with lots of followers. Thus, though VEK is much faster than EKC as shown in latter experimental results, the number of followers produced by VEK is similar to that of EKC. As reported in Figure 3, the margins between the results of VEK and EKC are very small. In some cases, the followers from VEK can be even more than that from EKC, e.g., for Enron and Twitter. It is because VEK considers the addition of edges in a batch manner based on the pivot vertices. Figure 4 shows the number of followers for other settings by varying the value of k or b , the results are similar to that in Figure 3. As EKC focuses on finding a best anchor edge which is always from $(k-1)$ -core in one iteration, there may be no single anchor edge with any followers. Thus, the performance of EKC may degenerate in some settings, e.g., $k = 25$ and $b = 10$ on Twitter. However, as our

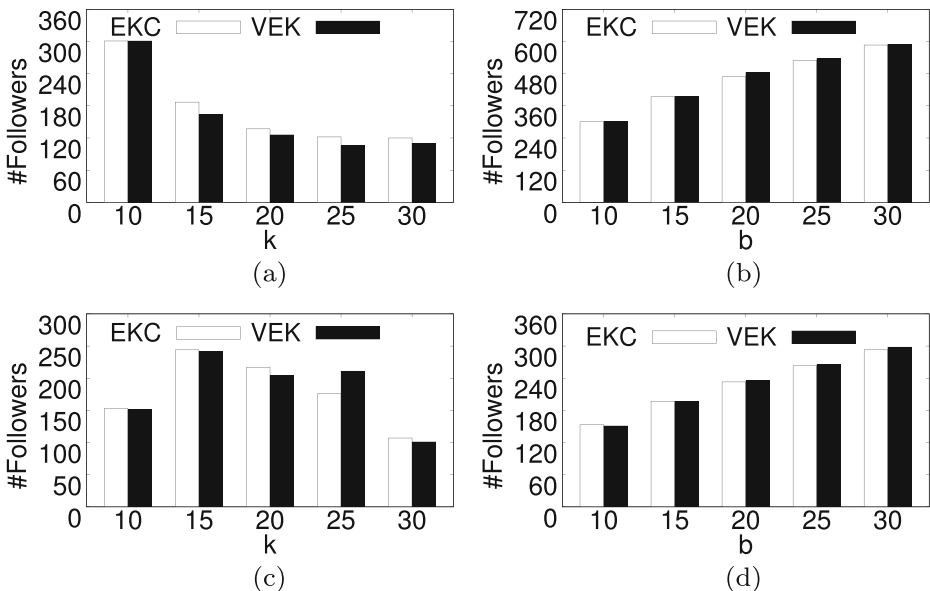


Figure 4 EKC vs VEK on the number of follower

VEK aims to preserve a good pivot vertex which adds a batch of edges at the same time, the performance of VEK is more robust than EKC. Besides, since VEK is always faster than EKC by at least 1 order of magnitude, it is often worthwhile to use VEK instead of EKC in real applications.

4.3 Efficiency

Figure 5 reports the time cost of EKC and VEK on all the datasets with $k = 10$ and $b = 20$. If the running time exceeds 48 hours, we set it to timeout. Figure 5 shows that the time cost of VEK is smaller than EKC by 3 orders of magnitude on most datasets. Our VEK can return in 100 seconds on most settings while the time cost of EKC is larger than 10000 seconds on many settings. We observe that the runtime among different datasets is largely influenced by the size of the dataset, the average degree, and the onion layer structures [42], while none of the characteristics is deterministic on time cost. The outperformance of VEK is more obvious when the dataset is larger, because the number of candidate edges in EKC are much larger than candidate pivots used in VEK on larger real-life datasets.

Figure 6 shows the impact of different k and b on the two algorithms. Figure 6(a) reports that the runtime decreases with a larger input of k . This is because the number of candidate edges becomes fewer as k increases. When k is relatively large, there is no anchor in one iteration with any follower for EKC, then the set of candidate edges for EKC is empty. In this case, the margin between EKC and VEK becomes smaller because EKC is basically a random method. Figure 6(b) reports that the runtime is basically proportional to the input of b for both EKC and VEK. The outperformance of VEK against EKC is clear on every setting. As the result quality (the number of produced followers) of VEK is similar to EKC, our proposed VEK becomes the state-of-the-art solution for the edge k -core problem.

5 Related work

The concept of k -core and the computing algorithm are first introduced in [28, 31]. Core decomposition finds the k -core for any k value, which costs $O(m)$ time by recursively removing each vertex with degree less than k [2]. The computation of k -core is also studied in other environments, including multi-core platform [11, 14], distributed environment [29], and I/O efficient configuration [38] where the space complexity is bounded by $O(n)$. [23] studies the core decomposition in multilayer graphs, and an in-depth discussion of core decomposition is conducted by [25]. The hierarchical core maintenance on dynamic graphs is studied by [21]. The best k in a k -core decomposition can be found by [8].

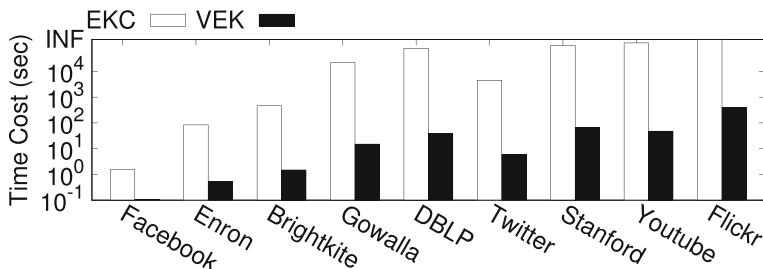


Figure 5 Running time on different datasets

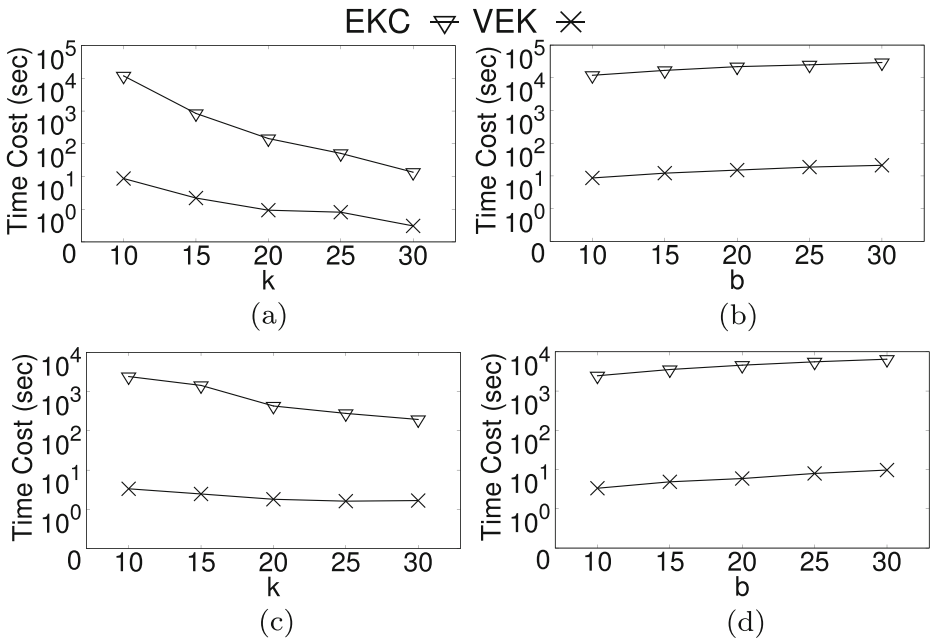


Figure 6 Effect of k and b

The mode of k -core is widely applied in different areas, e.g., community discovery [13, 36, 37], anomaly detection [32], influential spreader identification [12, 26], and user engagement study [3, 9, 43]. The variants of k -core combine the degree constraint with auxiliary informations, such as influential community [20], skyline community [19], and spatial-aware community [13]. Besides k -core, many other models detect cohesive subgraphs using different structural property, e.g., k -truss [10, 35], clique [4, 6], quasi-clique [30], and k -ecc [5].

As the computation of k -core precisely captures the decaying dynamic of a network, the size of k -core is often used to evaluate the structural stability of a network [3, 27, 44]. To improve the stability of a network, some models are proposed to maximize the k -core through different operations, e.g., the anchored k -core by enhancing vertices [3], the collapsed k -core by defending key vertices [43], and the edge k -core by adding critical edges [7]. Edge addition methods are also applied in other topics, e.g., modifying graph diameter [33], enlarging graph bandwidth [18] and anonymizing a set of vertices [15]. As most of the stability problems are NP-hard, different heuristics are evaluated in existing studies and the greedy solution is shown to be effective in many datasets, including the EKC algorithm [46] for the edge k -core problem. However, due to the huge number of candidates, EKC still cannot efficiently handle large graphs. Thus, a faster vertex-oriented approach is proposed in this paper.

6 Conclusion

In this paper, we propose a novel vertex-oriented solution for the edge k -core problem : given a graph G , an integer k , and a budget b , it aims to connect b non-adjacent vertex

pairs s.t. the number of vertices in the k -core is maximized. Due to the huge number of candidate vertex pairs to connect, the state-of-the-art algorithm EKC is not efficient enough for real-life large graphs. Our proposed VEK algorithm iteratively chooses a pivot vertex by a scoring function, and carefully adds the edges to the pivot s.t. it is preserved in the k -core. We design a budget return strategy and an edge re-connection method to optimize the scoring function and ensure the quality of our pivot selection. Effective optimization techniques are proposed to prune unpromising pivots and reuse the intermediate results. Extensive experiments demonstrate that VEK outperforms EKC in efficiency by 1–3 orders of magnitude while the result is still of high quality.

Acknowledgments Fan Zhang is supported by National Natural Science Foundation of China under Grant 62002073, and Guangzhou Basic and Applied Basic Research Foundation under Grant 202102020675.

Funding The grants declared in Acknowledgements were received to assist with the preparation of this manuscript.

Availability of data and material All the datasets used in this paper is public and every one can access to it.

Code Availability All the source codes will be shared online after the manuscript is accepted.

Declarations

Conflict of Interests The authors have no relevant financial or non-financial interests to disclose.

References


- Bader, G.D., Hogue, C.W.V.: An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinforma.* **4**, 2 (2003)
- Batagelj, V., Zaversnik, M.: An $o(m)$ algorithm for cores decomposition of networks. *CoRR* cs.DS/0310049 (2003)
- Bhawalkar, K., Kleinberg, J.M., Lewi, K., Roughgarden, T., Sharma, A.: Preventing unraveling in social networks: The anchored k -core problem. *SIAM J. Discrete Math.* **29**(3), 1452–1475 (2015)
- Bron, C., Kerbosch, J.: Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM* **16**(9), 575–576 (1973)
- Chang, L., Yu, J.X., Qin, L., Lin, X., Liu, C., Liang, W.: Efficiently computing k -edge connected components via graph decomposition. In: *SIGMOD*, pp. 205–216 (2013)
- Cheng, J., Ke, Y., Fu, A.W., Yu, J.X., Zhu, L.: Finding maximal cliques in massive networks by h^* -graph. In: *SIGMOD*, pp. 447–458 (2010)
- Chitnis, R., Talmon, N.: Can we create large k -cores by adding few edges? In: *CSR*, pp. 78–89 (2018)
- Chu, D., Zhang, F., Lin, X., Zhang, W., Zhang, Y., Xia, Y., Zhang, C.: Finding the best k in core decomposition: A time and space optimal solution. In: *ICDE*, pp. 685–696 (2020)
- Chwe, M.S.Y.: Communication and coordination in social networks. *Rev. Econ. Stud.* **67**(1), 1–16 (2000)
- Cohen, J.: Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report* **16**, 3–1 (2008)
- Dasari, N.S., Ranjan, D., Zubair, M.: Park: An efficient algorithm for k -core decomposition on multicore processors. In: *2014 IEEE International Conference on Big Data*, pp. 9–16. *IEEE Computer Society* (2014)
- Elsharkawy, S., Hassan, G., Nabhan, T., Roushdy, M.: Effectiveness of the k -core nodes as seeds for influence maximisation in dynamic cascades. *Int. J. Comput.*, 2 (2017)
- Fang, Y., Cheng, R., Li, X., Luo, S., Hu, J.: Effective community search over large spatial graphs. *PVLDB* **10**(6), 709–720 (2017)
- Kabir, H., Madduri, K.: Parallel k -core decomposition on multicore platforms. In: *IPDPS Workshops*, pp. 1482–1491. *IEEE Computer Society* (2017)

15. Kapron, B.M., Srivastava, G., Venkatesh, S.: Social network anonymization via edge addition. In: ASONAM, pp. 155–162. IEEE Computer Society (2011)
16. Khaouid, W., Barsky, M., Venkatesh, S., Thomo, A.: K-core decomposition of large networks on a single PC. *PVLDB* **9**(1), 13–23 (2015)
17. Kitsak, M., Gallos, L.K., Havlin, S., Liljeros, F., Muchnik, L., Stanley, H.E., Makse, H.A.: Identification of influential spreaders in complex networks. *Nat. Phys.* **6**(11), 888 (2010)
18. Lai, Y., Tian, C., Ko, T.: Edge addition number of cartesian product of paths and cycles. *Electron. Notes Discret. Math.* **22**, 439–444 (2005)
19. Li, R., Qin, L., Ye, F., Yu, J.X., Xiao, X., Xiao, N., Zheng, Z.: Skyline community search in multi-valued networks. In: SIGMOD, pp. 457–472 (2018)
20. Li, R., Qin, L., Yu, J.X., Mao, R.: Influential community search in large networks. *PVLDB* **8**(5), 509–520 (2015)
21. Lin, Z., Zhang, F., Lin, X., Zhang, W., Tian, Z.: Hierarchical core maintenance on large dynamic graphs. In: VLDB, vol. 14, pp. 757–770 (2021)
22. Linghu, Q., Zhang, F., Lin, X., Zhang, W., Zhang, Y.: Global reinforcement of social networks: The anchored coreness problem. In: SIGMOD, pp. 2211–2226 (2020)
23. Liu, B., Zhang, F., Zhang, C., Zhang, W., Lin, X.: Corecube: Core decomposition in multilayer graphs. In: WISE 2019, pp. 694–710 (2019)
24. Liu, B., Zhang, F., Zhang, W., Lin, X., Zhang, Y.: Efficient community search with size constraint. In: ICDE (2021)
25. Malliaros, F.D., Giatsidis, C., Papadopoulos, A.N., Vazirgiannis, M.: The core decomposition of networks: theory, algorithms and applications. *VLDB J.* **29**(1), 61–92 (2020)
26. Malliaros, F.D., Rossi, M.E.G., Vazirgiannis, M.: Locating influential nodes in complex networks. *Scientific Reports* **6**, 19307 (2016)
27. Malliaros, F.D., Vazirgiannis, M.: To stay or not to stay: modeling engagement dynamics in social graphs. In: CIKM, pp. 469–478 (2013)
28. Matula, D.W., Beck, L.L.: Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM* **30**(3), 417–427 (1983)
29. Montresor, A., Pellegrini, F.D., Miorandi, D.: Distributed k-core decomposition. *IEEE Trans. Parallel Distrib. Syst.* **24**(2), 288–300 (2013)
30. Pei, J., Jiang, D., Zhang, A.: On mining cross-graph quasi-cliques. In: SIGKDD, pp. 228–238 (2005)
31. Seidman, S.B.: Network structure and minimum degree. *Social Networks* **5**(3), 269–287 (1983)
32. Shin, K., Eliassi-Rad, T., Faloutsos, C.: Corescope: Graph mining using k-core analysis - patterns, anomalies and algorithms. In: ICDM, pp. 469–478 (2016)
33. Suady, S.A., Najim, A.A.: On edge-addition problem. *Journal of Education for Pure Science* **4**(1), 26–36 (2014)
34. Ugander, J., Backstrom, L., Marlow, C., Kleinberg, J.M.: Structural diversity in social contagion. *Proc. Natl. Acad. Sci. U.S.A.* **109**(16), 5962–5966 (2012)
35. Wang, J., Cheng, J.: Truss decomposition in massive networks. *PVLDB* **5**(9), 812–823 (2012)
36. Wang, K., Cao, X., Lin, X., Zhang, W., Qin, L.: Efficient computing of radius-bounded k-cores. In: 34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16–19, 2018, pp. 233–244 (2018)
37. Wang, K., Wang, S., Cao, X., Qin, L.: Efficient radius-bounded community search in geo-social networks. *IEEE Transactions on Knowledge and Data Engineering* (2020)
38. Wen, D., Qin, L., Zhang, Y., Lin, X., Yu, J.X.: I/O efficient core graph decomposition at web scale. In: ICDE, pp. 133–144 (2016)
39. Yu, W., Lin, X., Zhang, W.: Fast incremental simrank on link-evolving graphs. In: IEEE 30th International Conference on Data Engineering, Chicago, pp. 304–315 (2014)
40. Yuan, L., Qin, L., Zhang, W., Chang, L., Yang, J.: Index-based densest clique percolation community search in networks. *IEEE Trans. Knowl. Data Eng.* **30**(5), 922–935 (2018)
41. Zhang, C., Zhang, F., Zhang, W., Liu, B., Zhang, Y., Qin, L., Lin, X.: Exploring finer granularity within the cores: Efficient (k, p)-core computation. In: ICDE, pp. 181–192 (2020)
42. Zhang, F., Zhang, W., Zhang, Y., Qin, L., Lin, X.: OLAK: an efficient algorithm to prevent unraveling in social networks. *PVLDB* **10**(6), 649–660 (2017)
43. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: Finding critical users for social network engagement: The collapsed k-core problem. In: AAAI, pp. 245–251 (2017)
44. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: Efficiently reinforcing social networks over user engagement and tie strength. In: ICDE, pp. 557–568 (2018)

45. Zhang, H., Wang, X., Fan, H., Cai, T., Li, J., Li, X., Leung, V.C.M.: Anchor vertex selection for enhanced reliability of traffic offloading service in edge-enabled mobile P2P social networks. *J. Commun. Inf. Networks* **5**(2), 217–224 (2020)
46. Zhou, Z., Zhang, F., Lin, X., Zhang, W., Chen, C.: K-core maximization: An edge addition approach. In: *IJCAI*, pp. 4867–4873 (2019)

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Zhongxin Zhou¹  · Wenchao Zhang² · Fan Zhang³ · Deming Chu⁴ · Binghao Li⁵

Zhongxin Zhou
zzxecnu@gmail.com

Wenchao Zhang
zhangwenchao.hl@ccb.com

Deming Chu
deming.chu@unsw.edu.au

Binghao Li
binghao.li@unsw.edu.au

¹ East China Normal University, Shanghai, China

² Construction Bank of China Qiqihar Branch, Qiqihar, China

³ Guangzhou University, Guangzhou, China

⁴ School of Computer Science and Engineering, University of New South Wales, Sydney, Australia

⁵ School of Minerals and Energy Resources Engineering, University of New South Wales, Sydney, Australia