Exploring Finer Granularity within the Cores: Efficient (k,p)-Core Computation

Chen Zhang[‡†], Fan Zhang[‡✉], Wenjie Zhang[†], Boge Liu[†], Ying Zhang[§], Lu Qin[§], Xuemin Lin[†]
[‡]Guangzhou University, [†]University of New South Wales, [§]University of Technology Sydney
fanzhang.cs@gmail.com, {chenz, wenjie.zhang, boge.liu, xuemin.lin}@unsw.edu.au, {ying.zhang, lu.qin}@uts.edu.au

***Abstract*—In this paper, we propose and study a novel cohesive subgraph model, named (k,p)-core, which is a maximal subgraph where each vertex has at least k neighbours and at least p fraction of its neighbours in the subgraph. The model is motivated by the finding that each user in a community should have at least a certain fraction p of neighbors inside the community to ensure user engagement, especially for users with large degrees. Meanwhile, the uniform degree constraint k, as applied in the k-core model, guarantees a minimum level of user engagement in a community, and is especially effective for users with small degrees. We propose an $O(m)$ algorithm to compute a (k,p)-core with given k and p, and an $O(dm)$ algorithm to decompose a graph by (k,p)-core, where m is the number of edges in the graph G and d is the degeneracy of G. A space efficient index is designed for time-optimal (k,p)-core query processing. Novel techniques are proposed for the maintenance of (k,p)-core index against graph dynamic. Extensive experiments on 8 real-life datasets demonstrate that our (k,p)-core model is effective and the algorithms are efficient.**

I. Introduction

Graphs are widely used to model the relationships of entities in a large spectrum of applications including social networks, world wide web, collaboration networks, and biology. Cohesive subgraph mining, as a fundamental graph problem, extracts highly connected structures from large graphs. The cohesive subgraph model of k-core has attracted great attention due to its elegant property and linear-time computation [19]. Given a graph G, k-core is a maximal subgraph of G such that every vertex in the subgraph is connected to at least k other vertices within the same subgraph. It has a wide range of applications such as social contagion [22], community detection [28], influential spreader identification [8], collapse prediction [14], anomalies detection [21], core resilience [9], and user engagement study [12].

Many studies in sociology and economics reveal that the behavior of a user (e.g., leave or remain within the social group, adoption of a new technique/product/idea) is highly influenced by his/her neighborhood. A user tends to adopt a new behavior if there are a considerable number of friends in the group who adopted the same behavior [12]. The k-core model is considered as a powerful tool and a common practice in studying the user engagement dynamics of social networks. The nondiscriminatory degree constraint of k ensures a basic engagement level of a vertex in the k-core subgraph.

*Chen Zhang and Fan Zhang are the joint first authors. Fan Zhang is the corresponding author.

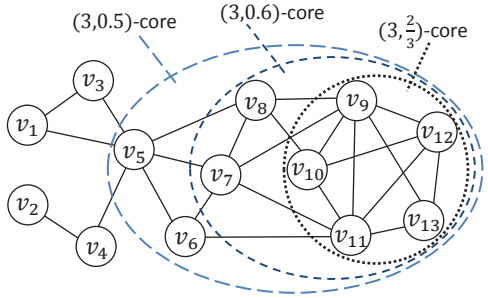


Fig. 1. Three (k,p)-Cores when $k=3$

Nevertheless, in a real-life social network, the same degree constraint k may affect differently to users with different scales of degree (i.e., numbers of neighbors) in the network [15]. An empirical study validates that a large-degree user usually needs more friends (i.e., neighbors) than a small-degree user to motivate him/her to adopt a certain behavior [2]. Thus, in the k-core model, the single degree constraint k may not be sufficient to ensure that every user in the k-core is well-engaged. For instance, consider two users A with 100 friends and B with 20 friends in a social network G. Suppose the k-core of G contains both A and B, where each of them has 20 friends in the k-core. In such a scenario, A has a significantly worse engagement than B, because most of A's friends (80%) are not in the k-core. Therefore, a customized degree constraint should be applied for every vertex in the network to find the groups of well-engaged users.

Similar rational has also been proposed in the long-term community discovery, e.g., a community is identified in [7], [17] where each vertex has more neighbors within the community than outside of the community. In the *contagion* model [15], a user will be influenced by (e.g., adopt) a new behavior in social interactions, if at least a certain fraction p of his/her friends adopted the behavior. Easley and Kleinberg [6] further emphasize the contagion model and show that a community is homogeneous if every vertex in the community has at least a fraction p of its neighbors inside. New innovations are difficult to enter such communities, because people tend to actively interact with their neighbors [6]. Nevertheless, the basic degree constraint k is not considered in above studies. Many small-degree users with a few (but a large fraction of) neighbors may be contained in these communities.

Motivated by the above facts, we propose and study a novel cohesive subgraph model, named (k,p)-core, which is a maximal subgraph where each vertex has at least k neighbours

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00023

and at least p fraction of its neighbours in the subgraph. The fraction constraint p is further applied in the k-core model, to customize and polish the degree constraint for every vertex. Such a fraction requirement helps refine the k-core model to more accurately capture the engagement dynamics of users and explore a finer granularity on top of the k-core model. Figure 1 shows three different (k,p)-cores when $k = 3$. Although the core numbers of vertices v_5 to v_{13} are the same, the 3-core subgraph is further refined into (k,p)-cores with smaller sizes by incorporating the new fraction requirement. Our empirical studies in Section VII demonstrate that such a fine-grained (k,p)-core model better captures the user engagement dynamics in real social networks.

Despite the promising characteristics and performance of (k,p)-core, the extra consideration of neighborhood fraction also brings challenges in both structural analysis and computing technique development. In the paper, we aim to comprehensively study the *computation* of (k,p)-core for given k and p values, the *decomposition* of (k,p)-core (computing the (k,p)-core for every possible pair of k and p), the *index-based* query processing techniques to support frequent (k,p)-core computation requests, as well as the *maintenance* of the index over dynamic graphs. The extra fraction aspect in the (k,p)-core model brings challenges especially in dynamic maintenance of (k,p)-cores due to the non-trivial change of fraction values of vertices with edge insertion or deletion.

In the paper, we address the above challenges and objectives. The principal contributions of this paper are summarized in the following.

- We propose and advocate a novel cohesive subgraph model (k,p)-core. Based on k-core, the new model considers the extra fraction constraint p to customize the degree requirement for every vertex, and better captures the engagement dynamics in real social networks.
- We analyze the (k,p)-core subgraphs from hierarchical structures regarding increasing p values for a given k. We devise an algorithm with $O(m)$ time complexity to compute a (k,p)-core subgraph with given k and p values where m is the number of edges in a graph G. A (k,p)-core decomposition algorithm is developed with $O(dm)$ time to compute the (k,p)-core for all possible inputs of k and p values where d is the degeneracy of graph G, i.e., the maximum k for a non-empty k-core of G.
- We propose an index with $O(m)$ space cost and the index maintenance techniques to answer (k, p)-core query over large dynamic graphs. The index can support optimal (k,p)-core query processing with processing time depending on the size of result. For maintenance of the index, we identify a small part of the index which may be affected, by deriving effective fraction upper and lower bounds.
- Comprehensive experiments are conducted over 8 real-world datasets to demonstrate the effectiveness of our (k,p)-core model and the efficiency of our algorithms. Case studies are also reported on real datasts.

TABLE I
SUMMARY OF NOTATIONS

Notation	Definition
$G = (V, E)$	an unweighted and undirected graph
n, m	the number of vertices and edges in G, respectively (assume $m > n$)
u, v	a vertex in the graph
S	a subgraph of G
$V(S)$; $E(S)$	the vertex set of S; the edge set of S
$E(v, S)$	the edges in S that are incident to v
$deg(v, S)$	the number of adjacent vertices of v in S
$N(v, S)$	the set of adjacent vertices of v in S
k	the degree threshold
p	the fraction threshold
$C_k(G)$	the k-core of G
$cn(v, G)$	the core number of v in G
d, or $d(G)$	the degeneracy of G, i.e., $max\{k\|C_k(G) \neq \emptyset\}$
$frac(v, S, G)$	the fraction of v's neighbors in S over in G, i.e., $deg(v, S)/deg(v, G)$
$C_{k,p}(G)$	the (k,p)-core of G
$pn(v, k, G)$	the p-number of a vertex v in G for a given k, i.e., $max\{p\|v \in C_{k,p}(G)\}$
$\mathcal{I}$	the (k,p)-core index of G, i.e., $\bigcup_{1 \leq k \leq d(G)} A_k$
A_k	an array of $\mathcal{I}$, i.e., (V_k, P_k) where V_k is the vertex set of $C_k(G)$ and P_k contains the unique p-numbers of the vertices in V_k

II. PRELIMINARIES

Let $G = (V, E)$ be a simple unweighted and undirected graph, where V represents the set of vertices and E represents edges in G. We denote $n = |V|$, $m = |E|$ and assume $m > n$. $N(v, G)$ denotes the set of adjacent (neighbor) vertices of v in G. Let S denote a subgraph of G. We use $deg(v, S)$, the *degree* of v in S, to represent the number of adjacent vertices of v in S. We summarize the notations in Table I.

When the context is clear, we omit the input graph in notations, e.g, using $deg(e)$ instead of $deg(e, G)$.

Definition 1. ***k-core***. *Given a graph G and an integer k, a subgraph S is the k-core of G, denoted by $C_k(G)$, if (i) $deg(v, S) \geq k$ for every $v \in S$ (degree constraint); and (ii) S is maximal, i.e., any subgraph S' is not a k-core if S is a subgraph of S' and $S \neq S'$.*

Given a graph G and an integer k, the k-core of G is unique [26]. The ***core number*** of a vertex u in G is defined as the largest k such that u is contained in the k-core of G. The ***degeneracy*** $d(G)$ of a graph G is defined as the maximum k such that the k-core of G is not empty. The ***core decomposition*** problem is to compute the core number of every vertex in G, or equivalently, the k-core for every integer k from 1 to $d(G)$. Given a graph, the k-core can be computed by recursively removing every vertex with degree less than k, with time complexity of $O(m)$ [3].

Before introducing (k,p)-core, we first define the fraction of a vertex regarding a subgraph S and the graph G.

Definition 2. ***fraction***. *Given a graph G, and a subgraph S of G, the fraction of vertex v in S is defined as the degree of v in S divided by the degree of v in G, that is, $frac(v, S, G)$ = $deg(v, S)/deg(v, G)$.*

Based on the definition of k-core and the fraction of a vertex, we define (k,p)-core as follows.

Definition 3. ***(k,p)-core****. Given a graph G, an integer k and a decimal p, a subgraph S is the (k,p)-core of G, denoted by $C_{k,p}(G)$, if for every vertex v in S, (i) $deg(v,S) \geq k$ (degree constraint) and (ii) $frac(v,S,G) \geq p$ (fraction constraint); and (iii) S is maximal, i.e., any subgraph S' is not a (k,p)-core if S is a subgraph of S' and $S \neq S'$.*

Given a graph G with any input of k and p, the (k,p)-core is always a subgraph of k-core. When k and p are specified, the (k,p)-core of G is unique. Suppose there exist two subgraphs G_1 and G_2 which are both the (k,p)-core of G and $G_1 \neq G_2$. Then, G_1+G_2 forms a larger (k,p)-core which contradicts with that G_1 and G_2 are both (k,p)-core (the maximality constraint).

Example 1. *In Figure 1, the 3-core is induced by 9 vertices: v_5 to v_{13}. The core number of vertex v_5 is 3. The fraction of vertex v_5 in the 3-core is 0.5, because 3 out of its 6 neighbors are in the 3-core. The 3-core itself is a (3,0.5)-core which contains two smaller (k,p)-cores: (3,0.6)-core and (3,2/3)-core.*

Problem Definition. Given a graph G, we comprehensively study the (k,p)-core computation: (i) we compute the (k,p)-core of G with specified k and p values; (ii) the (k,p)-core decomposition that retrieves the (k,p)-cores of G for all possible pairs of (k,p) values where $1 \leq k \leq d(G)$ and $0 \leq p \leq 1$; and (iii) the index construction for fast (k,p)-core query, and the maintenance of the index on dynamic graphs.*

III. (k,p)-Core Computation with Given k and p

Algorithm 1 describes the pseudo-code for computing the (k,p)-core of G, with specified k and p values. In Line 1, we first compute the combined degree threshold for each vertex, which is the larger one in k and $\lceil p \times deg(v,G) \rceil$. Note that this threshold will not change during the computation. Then we do the degree check on G. For each vertex with insufficient degree value (Line 3), we delete the vertex and its incident edges (Line 4). We use a queue Q to record the vertices waiting for deletion in Line 3. The deletion of a vertex will decrease the degrees of its neighbors by 1 in Line 4. Once the degree of a vertex v decreases from $t[v]$ to $t[v]-1$, we push v into the queue Q. We maintain a tag set T to distinguish the status of every vertex: (1) in G but not in Q, (2) in G and in Q, and (3) not in G. Note that we do not need to update the vertex degree which is already smaller than its combined threshold.

Example 2. *Consider the graph in Figure 1, suppose $k = 3$ and $p = 2/3$, we firstly push v_1, v_2, v_3 and v_4 to the queue waiting for deletion, as their degrees are all smaller than k. After deleting the 4 vertices one by one, the degree of v_5 is 3, less than its combined threshold $max(k, \lceil p \times deg(v_5) \rceil) = max(3,4) = 4$. The deletion of v_5 will remove v_6, v_7 and v_8 sequentially. Then, every vertex remaining satisfies its combined threshold and the (k,p)-core is produced.*

*While the (k,p)-core with given k and p can be computed by trivially extending the k-core computation (Section III), the decomposition (Section IV), index construction (Section V), and index maintenance (Section VI) for (k,p)-core are non-trivial and challenging due to the extra consideration of fraction constraint (p).

Algorithm 1: **kpCore**

Input : a graph G, the degree threshold k, the fraction threshold p
Output : the (k,p)-core of G

```
1 t[v] ← max(k, ⌈p × deg(v, G)⌉) for every vertex v ∈ G;
2 G' ← G;
3 while exists v ∈ G' with deg(v, G') < t[v] do
4     G' ← G' \ {v ∪ E(v, G')};
5 return G'
```

Complexity. In Algorithm 1, the vertex deletion and edge deletion take $O(n)$ and $O(m)$, respectively. Finding the initial vertices to delete takes $O(n)$. The degree update takes $O(m)$. So the time complexity is $O(m)$.

Correctness. Every vertex in the subgraph S returned by Algorithm 1 satisfies the degree constraint k and the fraction constraint p; otherwise, Algorithm 1 will continue to delete the vertices violating the two constraints. Suppose S is not a maximal (k,p)-core, and there is a supergraph S' of S which is a larger (k,p)-core. It contradicts with deletion condition in Line 3. Thus, Algorithm 1 is correct.

IV. (k,p)-Core Decomposition

According to the definition of (k,p)-core, the containment property is immediate: given a graph, if $k \geq k'$ and $p \geq p'$, the (k,p)-core is a subgraph of (k',p')-core. Although the input k ranges from integer 1 to integer $d(G)$ and the input p ranges from 0 to 1 continuously, many (k,p)-cores are exactly same with different inputs of k and p. For instance, the (3, 0.55)-core is the same as the (3, 0.6)-core in Figure 1.

We define the p-number of a vertex to facilitate the decomposition of (k,p)-core.

Definition 4. ***p-number****. Given a graph G and a degree threshold k, a decimal p is called the p-number of a vertex u, denoted by $pn(u,k,G)$, if (i) the (k,p)-core contains u and (ii) (k,p')-core does not contain u for any $p' > p$.*

Given a graph and an input k, the p-number of a vertex is unique according to the definition. Note that the p-numbers of the vertices can be very different for different k values. The ***(k,p)-core numbers*** of a vertex are multiple pairs of k and p, where the value of p is the p-number of the vertex according to the value of k. In the following, we introduce the (k,p)-core decomposition algorithm. Since the value of p is continuous, we decompose the graph with a fixed value of k, i.e., lifting the value of p based on the definition of p-number.

(k,p)-Core Decomposition. Algorithm 2 shows the process of (k,p)-core decomposition. The decomposition is computed for k from 1 to $d(G)$ (Line 1). Since the (k,p)-core is always a subgraph of k-core ($p = 0$), the largest value of k is $d(G)$, and the decomposition with fixed k can be computed on the k-core of G (Line 2). Here the k-core with every k value is computed by the k-core decomposition [3]. The fraction value of every vertex in the k-core S may be different. At Line 4, we find

Algorithm 2: **kpCoreDecom**

```
Input  : a graph G
Output : p-numbers of vertices for every k
1 for k from 1 to d(G) do
2     S ← the k-core of G;
3     while S is not empty do
4         p_min ← min{deg(v,S)/deg(v,G) | ∀v ∈ S};
5         while exists v ∈ S with deg(v,S) < k or
          deg(v,S)/deg(v,G) ≤ p_min do
6             p_k[v] ← p_min;
7             S ← S \ {v ∪ E(v,S)};
8 return p_k[v] of every v in k-core for every k
```

the minimum fraction value among all the k-core vertices. For every vertex with degree smaller than k or fraction not larger than p_{min} in S (Line 5), we record the p-number of the vertex by p_{min} at Line 6, and remove the vertex from S at Line 7. After deleting all such vertices, we find the next smallest fraction value again and follow the process to delete vertices. We repeat this process until S is empty. Finally, we return the p-numbers for every k.

Example 3. *Consider the graph in Figure 1. When $k = 1$, the p_{min} in Algorithm 2 is 1. Thus G itself is a $(1, 1)$-core. When $k = 2$, the 2-core of G is induced by $V(G)$ minus v_2 and v_4. In 2-core, the p_{min} is $\frac{5}{6}$ from v_5. So the p-number of v_5 is $\frac{5}{6}$ (Line 6), and v_5 is deleted (Line 7). The deletion of v_5 will result in the deletion of all the vertices in 2-core, according to Line 5. Thus, the p-number of every vertex in 2-core is $\frac{5}{6}$ for $k = 2$. The (k,p)-cores with $k = 3$ are shown in Figure 1 which can be computed similarly following Algorithm 2.*

Complexity. The k-core decomposition takes $O(m)$. For (k,p)-core decomposition with a fixed k (Line 2 to 7), finding p_{min} takes $O(n)$, the vertex deletion takes $O(n)$, the edge deletion takes $O(m)$, and the degree update takes $O(m)$, and the update of the queue and the tag set takes $O(n)$. The decomposition is computed for d times, so the time complexity is $O(dm)$.

Correctness. The correctness follows the correct computation of p-number of each vertex for every possible k. Given a k-core at Line 2, the first p-number (denoted by p_1) in Line 4 is correct, because every vertex in the k-core satisfies the constraints of (k, p_1)-core and $p = p_1$ is the largest input for a (k,p)-core to contain all the k-core vertices. After deleting the vertices with p_1 as their p-number, the second correct p-number p_2 is computed similarly. Recursively, we get that all the correct p-numbers for each k.

V. (k,p)-Core Index and Query Processing

In Section III we present an algorithm for computing a (k,p)-core with given k and p, which takes $O(m)$ time by traversing the entire graph. Considering that the real graph can be very large and the requests to compute (k,p)-core can be frequent in real applications, in this section, we present an efficient algorithm based on index. Particularly, we organize the (k,p)-cores into a linear space index structure, through which a (k,p)-core query can be answered in optimal time. We

Algorithm 3: **kpCoreQuery**

```
Input  : the KP-Index I of the graph G, the degree threshold
         k, the fraction threshold p
Output : the vertex set of (k,p)-core C_{k,p}(G)
1 (V_k, P_k) ← access A_k of I;
2 if k > d or p' > the largest p-number in P_k then
3     return ∅;
4 p' ← the first p-number in P_k which is not smaller than p;
5 u ← the vertex in V_k pointed by p';
6 for each vertex v from u to end of V_k in its order do
7     V_{k,p} ← V_{k,p} ∪ v;
8 return V_{k,p}
```

first introduce the index structure, KP-Index, and analyze its space complexity. Based on KP-Index, we propose an optimal query processing algorithm.

A. KP-Index

We build an index structure $\mathcal{I}$, named KP-Index, to maintain all the p-numbers and the corresponding vertices, for every integer k. The index $\mathcal{I}$ consists of $d(G)$ number of arrays, i.e., $\mathcal{I} = \bigcup_{1 \le k \le d(G)} A_k$, where each array A_k contains V_k - the vertex set of k-core, and P_k - the unique p-numbers of the vertices in V_k. In every A_k, the vertices in V_k are ordered by their deletion sequence in Line 5 of Algorithm 2; and every p-number in P_k points to the first vertex v in V_k with such p-number. Then, the vertices behind v in V_k have the same p-number unless they are pointed by another p-number from P_k. In each P_k of A_k, the p-numbers are ordered by ascending values.

Example 4. *Considering the graph G in Figure 2, the KP-Index $\mathcal{I}$ of G is given in Figure 3. When k = 3, there are four p-numbers in P_k where each p-number points to the first vertex in V_k with such p-number, i.e., in the $(3, p)$-core.*

Space Complexity of KP-Index. Next we prove that the KP-Index structure is space efficient.

Lemma 1. *Given a graph G, the space cost of its KP-Index $\mathcal{I}$ is bounded by $O(m)$.*

Proof. Every vertex $u \in G$ appears at most $cn(u)$ times in the KP-Index $\mathcal{I}$. Thus, let $t(u)$ denote the times that u appears in $\mathcal{I}$, the size of $\mathcal{I}$ is $\sum_{u \in V(G)} t(u) \le \sum_{u \in V(G)} cn(u) \le \sum_{u \in V(G)} deg(u) = 2|E(G)|$. Consequently, the vertex part of $\mathcal{I}$, i.e., $\bigcup_{1 \le k \le d(G)} V_k$, is bounded by $O(m)$. The p-number part of $\mathcal{I}$, i.e., $\bigcup_{1 \le k \le d(G)} P_k$, is not larger than the vertex part. Thus, the space cost of $\mathcal{I}$ is bounded by $O(m)$. □

B. Optimal Query Processing

With KP-Index, we can optimally retrieve the vertex set of (k,p)-core with given k and p by visiting only the vertices in the (k,p)-core. If required, the edge set of the (k,p)-core can be immediately retrieved by visiting the neighbor set of every vertex of (k,p)-core. Algorithm 3 shows the pseudo-code for the (k,p)-core query. For a given query with parameter k, the corresponding array in $\mathcal{I}$ is located in Line 1. Then, p' is found in P_k by the first value of p-numbers which is not smaller than

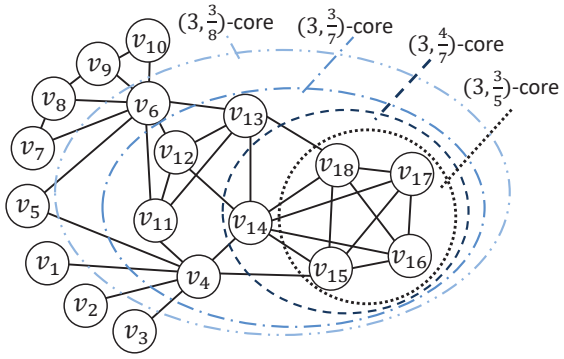


Fig. 2. An Example Graph

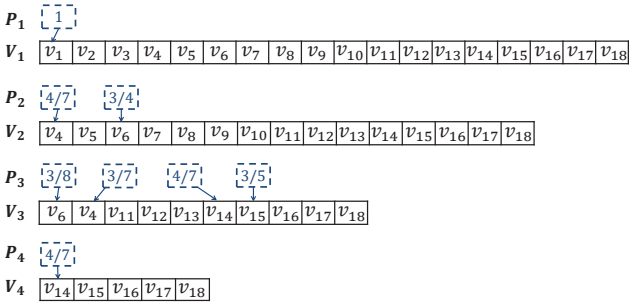


Fig. 3. KP-Index $\mathcal{I}$

the given parameter p in Line 4. The result $V_{k,p}$ consists of all the vertices pointed by p' to the end of the vertex set of V_k (Line 7). In case that the input k is larger than the degeneracy of G, or the input p is greater than the largest p value in P_k, the result is empty (Line 3).

Example 5. *For a (k,p)-core query with $k = 3$ and $p = 0.5$, by checking the p-numbers in P_k, we find $\frac{4}{7}$ is the first value not smaller than 0.5. Then, the vertex pointed by p-number $\frac{4}{7}$ and all the vertices behind form the $(3, 0.5)$-core, i.e., v_{14}, v_{15}, v_{16}, v_{17}, and v_{18}.*

The time complexity of the (k,p)-core query is linear regarding the result size. As it requires at least linear time to output the result, the algorithm is optimal. The following theorem is immediate as only the vertices in $C_{k,p}(G)$ and the corresponding p-numbers are visited.

Theorem 1. *Given a (k,p)-core query with specified k and p on the graph G, Algorithm 3 computes the vertex set of $C_{k,p}(G)$ in $O(|V(C_{k,p}(G))|)$ time.*

Discussion of KP-Index. As introduced above, KP-Index $\mathcal{I}$ is space-efficient taking $O(m)$ space and supports time-optimal (k,p)-core query. Revisit the example in Fig. 3, one may wonder if the space could be further reduced since k-core is a subgraph of k'-core when $k > k'$. Observe that the orders of vertices in V_k across different k in $\mathcal{I}$ are inconsistent. For example, when $k = 2$, vertex v_4 is pointed by a p-number smaller than that points to v_6, while this order is reversed when $k = 3$. Thus, we remark that it is not feasible to trivially record only the vertex set of G (e.g., the vertices in 1-core) and still support time-optimal query of (k,p)-core.

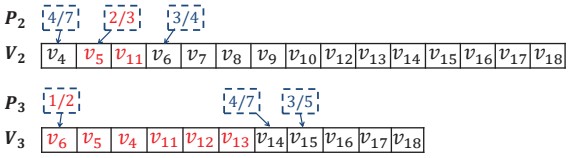


Fig. 4. The Change of KP-Index after Inserting (v_5, v_{11})

VI. Maintenance of (k,p)-Core Index

A real-world graph can be highly dynamic where new vertices/edges are inserted to the graph and some existing vertices/edges are deleted. In this section, we propose efficient algorithm to update the KP-Index $\mathcal{I}$ against edge insertion and deletion in the graph.

The dynamic of vertices on the graph G can be processed as follows: (1) if a non-isolated vertex v is inserted to G, the index $\mathcal{I}$ is updated by inserting each of v's incident edges. Both $cn(v, G)$ and $pn(v, k, G)$ are set to 0 in the proposed algorithm; and (2) if an existing vertex v is deleted from G, the index $\mathcal{I}$ is updated by deleting each of v's incident edges.

Due to the non-trivial change of p-numbers, it is challenging to efficiently maintain the KP-Index $\mathcal{I}$. We firstly introduce the case of edge insertion and then for edge deletion.

A. Edge Insertion

Let (u, v) denote an edge which will be inserted to the graph G. The following theorem implies that the some arrays in the (k,p)-core index, i.e., some A_k, remain the same, with the insertion of (u, v).

Theorem 2. *Given a graph G, and an edge $(u, v) \notin G$, let $G_+ = G + \{(u, v)\}$, if $k > max(cn(u, G_+), cn(v, G_+))$, the insertion of (u, v) to G does not change the p-number of any vertex w in the A_k, i.e., $pn(w, k, G) = pn(w, k, G_+)$.*

Proof. When $k > max(cn(u, G_+), cn(v, G_+))$, the k-core of G is the same as the k-core of G_+, as proved in [18]. Because both u and v are not in the k-core $C_k(G)$, the neighbor set of every vertex in $C_k(G)$ keeps same after the insertion of (u, v). Thus, the p-number of each vertex in $C_k(G)$ remains unchanged. □

According to Theorem 2, we do not need to update the (k,p)-core index for $k > max(cn(u, G_+), cn(v, G_+))$. Nevertheless, the p-numbers in the index may change for every $k \leq max(cn(u, G_+), cn(v, G_+))$. For different k values, the changed part of p-number index by edge insertion may also be different, as shown in Example 6.

Example 6. *After the insertion of edge (v_5, v_{11}), A_2 and A_3 of the KP-index in Figure 3 are updated as shown in Figure 4, where A_1 and A_4 keep unchanged. For A_2, a new p-number $pn = \frac{2}{3}$ appears, and v_{11} moves from p-number $3/4$ to $2/3$. For A_3, the change of p-numbers and the vertex order are quite different to the change in A_2.*

In the following, we explore the change of p-numbers after inserting (u, v) for different cases.

Insertion Case 1: core number keeps same. In this case, the insertion of (u, v) does not change the core number of any

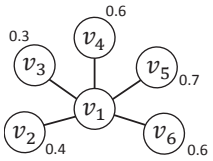


Fig. 5. Example for Computing the Upper Bound of p-Number

vertex, which is usually the major case for edge insertion [30]. Then, for each A_k in the index, we find the p-number range $[p_-, p_+]$ such that a vertex in A_k with p-number p_0 will not change its p-number if $p_0 < p_-$ or $p_0 > p_+$. Therefore, we aim to find a large p_- and a small p_+.

Case 1.1: $k \leq min\{cn(u), cn(v)\}$. For conciseness, we firstly present the techniques for $k \leq min\{cn(u), cn(v)\}$ if (u, v) will be inserted to G, and deal with the other case afterwards. For p_-, the following theorem finds a proper value.

Theorem 3. *Given a graph G, a degree threshold k and an edge $(u, v) \notin G$, let $p_- = min\{pn(u, k, G), pn(v, k, G)\}$, for any vertex w with $pn(w, k, G) < p_-$, the insertion of (u, v) will not change the p-number of w in A_k, i.e., $pn(w, k, G) = pn(w, k, G + \{(u, v)\})$.*

Proof. After the insertion of (u, v), for every vertex w with p-number less than p_-, the neighbor set of w in the k-core keeps same. The same topology leads to the same p-numbers of these vertices. □

Given the insertion of $(u, v) \notin G$, the p-number of a vertex w with $pn(w, k, G) \geq min\{pn(u, k, G), pn(v, k, G)\}$ may increase. To find a proper p_+, we propose the upper bound of p-number.

Definition 5. ***upper bound of p-number, ($\hat{p}$).*** *Given a graph G, a degree threshold k, the k-core $C = C_k(G)$, and a vertex w in C, the $\hat{p}$ upper bound of $pn(w, k, G)$ is defined by*

$$\begin{aligned} &\hat{p}(w, k, G) = max(\tfrac{i}{deg(w,G)}) \ s.t. \\ &|\{v \mid v \in N(w, C),\ \tfrac{deg(v,C)}{deg(v,G)} \geq \tfrac{i}{deg(w,G)}\}| \geq i \end{aligned} \tag{1}$$

Lemma 2. *Given the input graph G, for every valid input of k and every vertex $w \in C_k(G)$, the upper bound $\hat{p}(w, k, G)$ is correct, i.e., $\hat{p}(w, k, G) \geq pn(w, k, G)$.*

Proof. Suppose $pn(w, k, G) = \frac{t}{deg(w,G)}$, let C_1 denote the $(k, \frac{t}{deg(w,G)})$-core, there are at least t neighbors of w in C_1 with fraction values of at least $\frac{t}{deg(w,G)}$. We have $\frac{deg(v,C_k(G))}{deg(v,G)} \geq \frac{deg(v,C_1)}{deg(v,G)} \geq \frac{t}{deg(w,G)}$ for each v of the t neighbors of w. According to Equation 1, $\hat{p}(w, k, G) \geq \frac{t}{deg(w,G)} = pn(w, k, G)$. □

Example 7. *In Figure 5, given an input k, we show vertex v_1 and its 5 neighbours in the k-core. The p-number of each neighbor is shown in the figure. Suppose $deg(v_1, G) = 6$, we find that v_1 has 3 neighbours with p-number not smaller than $3/6 = 0.5$, and only one neighbor of v_1 has a p-number of at least $4/6$. Thus, the p-number upper bound of v_1 is $\hat{p}(v_1, k, G) = 3/6 = 0.5$.*

A tighter upper bound can be derived from $\hat{p}$.

Definition 6. ***upper bound of p-number, ($\widetilde{p}$).*** *Given a graph G, a degree threshold k, the k-core $C = C_k(G)$, and a vertex w in C, the $\widetilde{p}$ upper bound of $pn(w, k, G)$ is defined by*

$$\begin{aligned} &\widetilde{p}(w, k, G) = max(\tfrac{i}{deg(w,G)}) \ s.t. \\ &|\{v \mid v \in N(w, C),\ \hat{p}(v, k) \geq \tfrac{i}{deg(w,G)}\}| \geq i \end{aligned} \tag{2}$$

When the context is clear, we omit the input graph G in notations, e.g, using $pn(v, k)$ instead of $pn(v, k, G)$.

Lemma 3. *Given the input graph G, for every valid input of k and every vertex $w \in C_k(G)$, the upper bound $\widetilde{p}(w, k)$ is correct and tight, i.e., $\hat{p}(w, k) \geq \widetilde{p}(w, k) \geq pn(w, k)$.*

Proof. (i) Suppose $pn(w, k, G) = \frac{t}{deg(w,G)}$, let C_1 denote the $(k, \frac{t}{deg(w,G)})$-core, for each vertex $v \in C_1$, there are at least t neighbors of v in C_1 with fraction values of at least $\frac{t}{deg(w,G)}$. For a neighbor v of w in C_1, we have $\hat{p}(v, k, G) \geq pn(v, k, G) \geq \frac{t}{deg(w,G)}$. According to Equation 2, $\widetilde{p}(w, k, G) \geq \frac{t}{deg(w,G)} = pn(w, k, G)$.

(ii) Suppose $\widetilde{p}(w, k, G) = \frac{t}{deg(w,G)}$, let C_2 denote the $(k, \frac{t}{deg(w,G)})$-core, there are at least t neighbors of w in C_2 with $\hat{p}$ upper bound of at least $\frac{t}{deg(w,G)}$. For a neighbor v of w in C_2, we have $\frac{deg(v,C)}{deg(v,G)} \geq \hat{p}(v, k, G) \geq \frac{t}{deg(w,G)}$. According to Equation 1, we have $\hat{p}(w, k, G) \geq \frac{t}{deg(w,G)} = \widetilde{p}(w, k, G)$. □

The above upper bounds enable us to find a small p_+ to locate the candidate range of vertices for p-number update.

Theorem 4. *Given a graph G, a degree threshold k, and an edge (u, v) to be inserted, let $G_+ = G + \{(u, v)\}$ and $p_+ = max\big\{min\{\widetilde{p}(u, k, G_+), \widetilde{p}(v, k, G_+)\}, pn(u, k), pn(v, k)\big\}$, a vertex w with p-number $pn(w, k) > p_+$ will not change its p-number if (u, v) is inserted.*

Proof. Suppose $pn(u, k) \leq pn(v, k)$ without loss of generality, let $p_m = min\{\widetilde{p}(u, k, G_+), \widetilde{p}(v, k, G_+)\}$. For any p-number $p_0 > p_+$, the following holds.

(i) If $p_m \geq pn(v, k)$, suppose $p_m = \widetilde{p}(u, k, G_+) \leq \widetilde{p}(v, k, G_+)$, the (k, p_0)-core of G_+ does not contain u since $\widetilde{p}(u, k, G_+) \leq p_+ < p_0$. Then the insertion of (u, v) may reduce the p-number of v in G because (u, v) is not in the $(k, pn(v, k))$-core of G_+ and $deg(v, G_+) = deg(v, G) + 1$. Thus, the (k, p_0)-core of G_+ does not contain v as $p_0 > p_+ \geq pn(v, k)$. The p-number of every vertex w with $pn(w, k) > p_+$ keeps same.

(ii) If $p_m < pn(v, k)$, then $p_+ = pn(v, k)$. Since $p_0 > p_+ > p_m$, the (k, p_0)-core of G_+ does not contain both u and v. The p-number of every vertex w with $pn(w, k) > p_+$ keeps same. □

Note that for the insertion of (u, v), let $p_m = min\{\widetilde{p}(u, k, G_+), \widetilde{p}(v, k, G_+)\}$, a vertex w with $pn(w, k) > p_m$ may change its p-number.

Case 1.2: $cn(u) < k \leq cn(v)$ suppose $cn(u) < cn(v)$. In this case, the p-number of v may decrease. Thus, we have $p_+ = pn(v, k, G)$. The value of p_- can be found by the following theorem.

Theorem 5. *Given a graph G, an edge $(u,v) \notin G$ with $cn(u) < cn(v)$, and a degree threshold k with $cn(u) < k \leq cn(v)$, let $p_1 = pn(v,k,G)$ and $C = C_{k,p_1}(G)$, we have*

$$p_- = max\{\tfrac{i}{deg(v,G)+1}\}\ s.t.\ |\{w \mid w \in N(v,C), \tfrac{deg(w,C)}{deg(w,G)} \geq \tfrac{i}{deg(v,G)+1}\}| \geq i \quad (3)$$

for any vertex w with $pn(w,k,G) < p_-$, the insertion of (u,v) will not change the p-number of w, i.e., $pn(w,k,G) = pn(w,k,G+\{(u,v)\})$.

Proof. Suppose $p_- = \frac{t}{deg(v,G)+1}$, there are at least t vertices in $N(v,C)$ with fraction value of at least p_-. For every vertex $w \in C$, $\frac{deg(w,C)}{deg(w,G)} \geq pn(v,k,G) \geq p_-$. Thus, $\frac{t}{deg(v,G)+1}$ is a lower bound of $pn(v,k,G+\{(u,v)\})$. For every vertex w with $pn(w,k,G) < p_-$, the neighbor set of w keeps same after the insertion of (u,v). Thus, $pn(w,k,G) = pn(w,k,G+\{(u,v)\})$. □

Insertion Case 2: core number changes. It is proved in [18] that the insertion of an edge (u,v) can only increase the core number of the incident vertex u or v by at most 1. Besides, if $cn(u) < cn(v)$, the core number $cn(v)$ will not change given the insertion of (u,v).

Compared to Case 1, the difference in Case 2 is that some new vertices are added to the A_k of the index, where $k = min\{cn(u), cn(v)\} + 1$. Note that this is the minor case for core maintenance as shown in [30]. We apply the state-of-the-art core maintenance algorithm [30] to find these vertices with increased core numbers.

For a degree threshold k and an edge $(u,v) \notin G$, let $G_+ = G + \{(u,v)\}$, $C_+ = C_k(G_+)$, and $V_+ = V(C_+) \setminus V(C_k(G))$ as the new vertices added to A_k. We set p_- by 0 for A_k. If $cn(u,G) < cn(v,G)$, we have $p_+ = pn(v,k,G)$. If $cn(u,G) = cn(v,G)$, we have $p_+ = min\{\widetilde{p}(u,k,G_+), \widetilde{p}(v,k,G_+)\}$.

The following theorem can fast determine whether a A_k in the index keeps same, for both Case 1 and 2.

Theorem 6. *Given a graph G, an edge $(u,v) \notin G$ to be inserted, and a degree threshold k, let $p_1 = pn(v,k,G)$, $C = C_{k,p_1}(G)$ and $G_+ = G + \{(u,v)\}$, if $cn(u,G_+) < k \leq cn(v,G_+)$ and $p_*(v,k,G_+) \geq pn(v,k,G)$, the A_k of the index does not change, where*

$$p_*(v,k,G_+) = max\{\tfrac{i}{deg(v,G)+1}\} s.t.\ |\{w \mid w \in N(v,C),\ \tfrac{deg(w,C)}{deg(w,G)} \geq \tfrac{i}{deg(v,G)+1}\}| \geq i \quad (4)$$

Proof. Suppose $p_*(v,k,G_+) = \frac{t}{deg(v,G)+1}$, there are at least t neighbors of v in $C_{k,p_1}(G)$ with fraction value of at least $\frac{t}{deg(v,G)+1}$. After the insertion of (u,v), if $p_*(v,k,G_+) \geq pn(v,k,G) = p_1$, every vertex w in $C_k(G)$ has $pn(w,k,G_+) = pn(w,k,G)$ due to the same topology of $C_k(G)$ and $C_k(G_+)$. Thus, A_k does not change. □

Algorithm for Edge Insertion. The pseudo-code of the edge insertion is shown in Algorithm 4. First, we assume $cn(u) \leq cn(v)$, and let $G_+ = G + \{(u,v)\}$. For each A_k with k from 2 to $max\{cn(v,G_+), cn(u,G_+)\}$, we consider two cases to compute p_- and p_+, i.e., the major case (Line 4 to 11) and the minor case (Line 12 to Line 16). In each subcase, we compute p_- and p_+ according to the proposed theorems. When A_k of the index will not change according to Theorem 6, we can jump to the next loop (Line 17 to Line 18). At Line 19, we update the p-numbers of the vertices in A_k with p-number from p_- to p_+.

Algorithm 4: **kpIndexInsert**

Input : a graph G, the (k,p)-core index $\mathcal{I}$ of G, an edge (u,v) not in G
Output : $\mathcal{I}$: the (k,p)-core index of $G + \{(u,v)\}$

1 $G_+ \leftarrow G + \{(u,v)\}$; Suppose $cn(u,G) \leq cn(v,G)$;
2 **for** each integer $k \in [2, max\{cn(v,G_+), cn(u,G_+)\}]$ **do**
3 $C_+ \leftarrow C_k(G_+)$;
4 **if** $C_k(G) = C_+$ **then**
5 `/* major case */`
6 **if** $k \leq cn(u)$ (Case 1.1) **then**
7 $p_- \leftarrow min\{pn(u,k,G), pn(v,k,G)\}$ `(Theorem 3)`;
8 $p_+ \leftarrow max\{min\{\widetilde{p}(u,k,G_+), \widetilde{p}(v,k,G_+)\}, pn(u,k), pn(v,k)\}$ `(Theorem 4)`;
9 **else if** $cn(u) < k \leq cn(v)$ (Case 1.2) **then**
10 $p_- \leftarrow$ `Theorem 5`;
11 $p_+ = pn(v,k,G)$;
12 **else**
13 `/* minor case */`
14 $p_- \leftarrow 0$;
15 $p_+ \leftarrow pn(v,k,G)$ **if** $cn(u,G) < cn(v,G)$;
16 $p_+ \leftarrow min\{\widetilde{p}(u,k,G_+), \widetilde{p}(v,k,G_+)\}$ **if** $cn(u,G) = cn(v,G)$;
17 **if** $cn(u,G_+) < k \leq cn(v,G_+)$ and $p_*(v,k,G_+) \geq pn(v,k,G)$ **then**
18 **continue** (`Theorem 6`);
19 Update A_k of $\mathcal{I}$ from p-number p_- to p_+;
20 **return** $\mathcal{I}$

Example 8. *We give an example by inserting an edge (v_5, v_{11}) to the graph in Figure 2. We show the new index in Figure 4 after the insertion. By Theorem 2, the larger core number between v_5 and v_{11} is 3, hence A_4 of the index will not change. For A_3, the corresponding k-core changes (the minor case). Since $p_- = 0$, and $p_+ = pn(v_{11}, 3, G) = \frac{3}{7}$ according to Algorithm 4, we update the vertices in A_3 with p-number from 0 to $\frac{3}{7}$. We can update A_2 similarly.*

B. Edge Deletion

Let (u,v) denote an edge in the graph G which will be deleted. The following theorem implies that the some A_k in the (k,p)-core index remain same, with the deletion of (u,v).

Theorem 7. *Given a graph G, and an edge $(u,v) \in G$, let $G_- = G - \{(u,v)\}$, if $k > max(cn(u,G), cn(v,G))$, the deletion of (u,v) does not change the p-number of a vertex w in A_k, i.e., $pn(w,k,G) = pn(w,k,G_-)$.*

Proof. When $k > max(cn(u,G), cn(v,G))$, the k-core of G is the same as the k-core of G_-, as proved in [18]. Because both u and v are not in the k-core $C_k(G)$, the neighbor set of every vertex in $C_k(G)$ keeps same after the deletion of

(u, v). Thus, the p-number of each vertex in the k-core keeps same. □

According to Theorem 7, we do not need to update the (k,p)-core index for every $k > max(cn(u, G), cn(v, G))$. Note that the p-numbers in the index may change for every $k \leq max(cn(u, G), cn(v, G))$. For different k values, the changed part of p-number index by edge deletion may be different.

Deletion Case 1: core number keeps same. In this case, the deletion of (u, v) does not change the core number of any vertex, which is usually the major case for edge deletion [30]. Then, for each A_k in the index, we find the p-number range $[p_-, p_+]$ such that a vertex with p-number p_0 will not change its p-number if $p_0 < p_-$ or $p_0 > p_+$. Therefore, we aim to find a large p_- and a small p_+.

When $k \leq min\{cn(u), cn(v)\}$, a proper lower bound for p_- is as follows.

Definition 7. ***lower bound of p-number, ($\underline{p}$).*** *Given a graph G, an edge $(u, v) \in G$ to be deleted, and a degree threshold $k \leq min\{cn(u), cn(v)\}$, let $p_1 = pn(u, k, G)$, $C = C_{k,p_1}(G)$, and $G_- = G - \{(u, v)\}$, the $p_\wedge$ value for u is defined by*

$$\begin{aligned} &p_\wedge(u, k, G_-) = min\{p_1, p_0\}; \\ &p_0 = max\{\tfrac{i}{deg(u,G)-1}\}\ s.t. \\ &|\{w \mid w \in N(u, C) \setminus v, \tfrac{deg(w,C)}{deg(w,G)} \geq \tfrac{i}{deg(u,G)-1}\}| \geq i \end{aligned} \tag{5}$$

If $pn(u, k, G) < pn(v, k, G)$, the $\underline{p}$ lower bound of u is defined by $\underline{p}(u, k, G_-) = p_\wedge(u, k, G_-)$. If $pn(u, k, G) = pn(v, k, G)$ the $\underline{p}$ lower bound of u or v is defined by $\underline{p}(u, k, G_-) = \underline{p}(v, k, G_-) = min\{p_\wedge(u, k, G_-), p_\wedge(v, k, G_-)\}$.

Lemma 4. *Given a graph G, an edge $(u, v) \in G$ to be deleted, and a degree threshold $k \leq min\{cn(u), cn(v)\}$, suppose $pn(u, k, G) \leq pn(v, k, G)$, let $G_- = G - \{(u, v)\}$, the lower bound $\underline{p}(u, k, G_-)$ is correct, i.e., $pn(u, k, G_-) \geq \underline{p}(u, k, G_-)$.*

Proof. Let $p_1 = pn(u, k, G)$ and $p_2 = pn(v, k, G)$. Suppose $\underline{p}(u, k, G_-) = \frac{t}{deg(u,G_-)}$, there are at least t neighbors of u in $C_{k,p_1}(G) \setminus v$ with fraction value of at least $\frac{t}{deg(u,G_-)}$.

(i) If $p_1 < p_2$, we have $pn(v, k, G_-) \geq pn(v, k, G)$ because every vertex $w \in C_{k,p_2}(G)$ have $\frac{deg(w,C_{k,p_2}(G))}{deg(w,G_-)} \geq p_2 > p_1 \geq \underline{p}(u, k, G_-)$. For every vertex $r \in C_{k,p_1}(G)$, we have $\frac{deg(r,C_{k,p_1}(G))}{deg(r,G_-)} \geq \frac{t}{deg(u,G_-)}$. Thus, $pn(u, k, G_-) \geq \underline{p}(u, k, G_-)$.

(ii) If $p_1 = p_2$, For every vertex $r \in C_{k,p_1}(G)$, we have $\frac{deg(r,C_{k,p_1}(G))}{deg(r,G_-)} \geq \frac{t}{deg(u,G_-)}$. Thus, $pn(u, k, G_-) \geq \underline{p}(u, k, G_-)$. □

When $k \leq max\{cn(u), cn(v)\}$, a proper p_- for edge deletion is as follows.

Theorem 8. *Given a graph G, a degree threshold k, an edge $(u, v) \in G$, and $G_- = G - \{(u, v)\}$, suppose $cn(u) \leq cn(v)$, let $p_- = pn(v, k, G)$ if $cn(u) < k \leq cn(v)$, and $p_- = \underline{p}(u, k, G_-)$ if $k \leq cn(u)$. For any vertex w with $pn(w, k, G) < p_-$, the deletion of (u, v) will not change the p-number of w, i.e., $pn(w, k, G) = pn(w, k, G_-)$.*

Algorithm 5: **kpIndexDelete**

Input : a graph G, the (k,p)-core index $\mathcal{I}$ of G, an edge (u, v) in G
Output : $\mathcal{I}$: the (k,p)-core index of $G - \{(u, v)\}$

```
1  G_- ← G − {(u,v)}; Suppose cn(u) ≤ cn(v);
2  for each integer k ∈ [2, cn(v,G)] do
3      C_- ← C_k(G_-);
4      if C_k(G) = C_- then
5          /* major case */
6          if k ≤ cn(u) then
7              p_- ← p(u,k,G_-) (Theorem 8);
8              p_+ ← max{p̃(u,k,G_-), p̃(v,k,G_-)}
                (Theorem 9);
9          else if cn(u) < k ≤ cn(v) then
10             p_- ← pn(v,k,G) (Theorem 8);
11             p_+ ← p̃(v,k,G_-) (Theorem 9);
12     else
13         /* minor case */
14         p_- ← 0;
15         p_+ ← p̃(u,k,G_-) if u ∈ C_- and v ∉ C_-;
16         p_+ ← p̃(v,k,G_-) if u ∉ C_- and v ∈ C_-;
17         p_+ ← pn(u,k,G) = pn(v,k,G) if u ∉ C_- and
           v ∉ C_-;
18     Update A_k of I from p-number p_- to p_+;
19 return I
```

Proof. If $cn(u) < k \leq cn(v)$, we have $deg(v, G_-) = deg(v, G) - 1$ and $C_k(G_-) = C_k(G)$. Thus, we have $p_- = pn(v, k, G) \geq pn(v, k, G_-)$. For every vertex w with $pn(w, k, G) < p_-$, the same topology in $C_k(G_-)$ leads to the same p-number. If $k \leq cn(u)$, the fraction values of u will not increase after the deletion of (u, v). Since $pn(u, k, G_-) \geq \underline{p}(u, k, G_-)$ by Lemma 4, every vertex w with $pn(w, k, G) < p_- = \underline{p}(u, k, G_-)$ has the same p-number in G_-. □

Upper Bound of p-Number for Edge Deletion. The deletion of (u, v) may increase the p-number of a vertex, the upper bound of p-number of a vertex w is the same as the case of edge insertion. The following theorem finds a proper p_+.

Theorem 9. *Given a graph G, an edge $(u, v) \in G$ with $cn(u) \leq cn(v)$, and a degree threshold k, let $G_- = G - \{(u, v)\}$, if $cn(u) < k \leq cn(v)$, we have $p_+ = \widetilde{p}(v, k, G_-)$; if $k \leq cn(u)$, we have $p_+ = max\{\widetilde{p}(u, k, G_-), \widetilde{p}(v, k, G_-)\}$.*

Proof. It is proved by the correctness of Lemma 3. □

Deletion Case 2: core number changes. It is proved in [18] that the deletion of an edge (u, v) can only decrease the core number of the incident vertex u or v by at most 1. Besides, if $cn(u) < cn(v)$, the core number $cn(v)$ will not change given the deletion of (u, v).

Compared to Case 1, the difference in Case 2 is that some vertices are deleted from A_k of the index, where $k = min\{cn(u), cn(v)\}$. Note that this is the minor case for core maintenance as shown in [30]. We apply the state-of-the-art core maintenance algorithm [30] to find these vertices with decreased core numbers.

TABLE II
STATISTICS OF DATASETS

Dataset	Vertices	Edges	d_{avg}	d_{max}
Facebook	4,039	88,234	43.69	1,045
Brightkite	58,228	214,078	7.35	1,134
Gowalla	196,591	950,327	9.66	14,730
YouTube	1,134,890	2,987,624	5.26	28,754
Pokec	1,212,349	8,320,600	13.72	7,266
DBLP	1,431,977	8,221,193	11.48	2,268
LiveJournal	3,997,962	34,681,189	17.34	14,815
Orkut	3,072,441	117,185,083	76.28	33,313

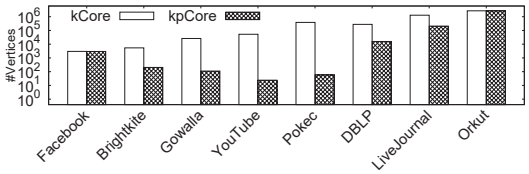


Fig. 6. Core Size in All Datasets, $k = 10, p = 0.6$

For a degree threshold k and an edge $(u, v) \in G$, let $G_- = G - \{(u, v)\}$, $C_- = C_k(G_-)$, and $V_- = V(C_k(G)) \setminus V(C_k(G_-))$ as the deleted vertices in A_k. We set p_- by 0 for A_k in Theorem 3. If $u \in C_-$ and $v \notin C_-$, we have $p_+ = \widetilde{p}(u, k, G_-)$. If $u \notin C_-$ and $v \in C_-$, we have $p_+ = \widetilde{p}(v, k, G_-)$. If $u \notin C_-$ and $v \notin C_-$, the deletion of u or v will remove all the vertices in V_- by the degree constraint. Thus, every vertex w in V_- has the same p-number, and we have $p_+ = pn(u, k, G) = pn(v, k, G)$.

Algorithm for Edge Deletion. The pseudo-code for the edge deletion is shown in Algorithm 5. we assume $cn(u) \leq cn(v)$, and let $G_- \leftarrow G - \{(u, v)\}$. We have two cases to compute p_- and p_+ in edge deletion, i.e., the major case (Line 4 to 11) and the minor case (Line 12 to Line 17). In each subcase, we compute p_- and p_+ according to the proposed theorems. For every k, we update the p-numbers of the vertices in A_k with p-number from p_- to p_+.

VII. EXPERIMENTAL EVALUATION

We conduct extensive performance studies to evaluate the effectiveness of our proposed algorithms.

A. Experimental Setting

Datasets. We use 8 real-life graphs in our experiments. The DBLP data is extracted from https://dblp.uni-trier.de/xml/ and the other datasets are downloaded from http://snap.stanford.edu/. In DBLP, each author corresponds to a vertex and there is an edge for a pair of authors iff they have at least 1 co-authored paper. In other datasets, there are existing edge data, i.e., the connections between vertex pairs. Table II shows the statistics of the datasets, ordered by the number of edges.

Algorithms. To the best of our knowledge, no existing work investigates the (k,p)-core model and the corresponding algorithms. We compare the (k,p)-core with k-core since it is the base of our model and also the most related model. Specifically, we evaluate the following algorithms:

- kCore(Comp): the state-of-the-art k-core computation algorithm in [3].

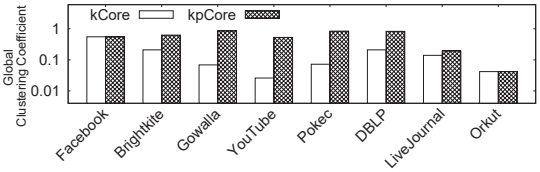


Fig. 7. Global Clustering Coefficient

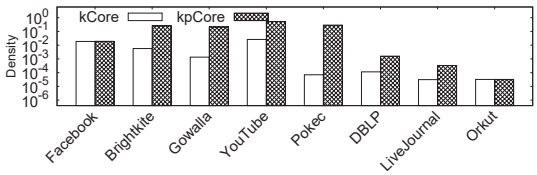


Fig. 8. Graph Density

- kcoreDecomp: the state-of-the-art core decomposition algorithm in [3].
- kpCore(Comp): our (k,p)-core computation with given k and p (Algorithm 1).
- kpCoreDecomp: our (k,p)-core decomposition algorithm (Algorithm 2).
- kpCoreQuery: our query algorithm on the KP-Index (Algorithm 3).
- kpIndexInsert: our KP-Index update algorithm for edge insertion (Algorithm 4).
- kpIndexDelete: our KP-Index update algorithm for edge deletion (Algorithm 5).

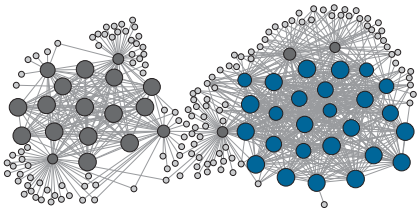
All algorithms are implemented in C++ and compiled by GNU GCC 4.8.2 with O3 optimization. All experiments are conducted on a machine with an Intel Xeon 2.2GHz CPU and 64GB main memory.

Parameters. We conducted experiments under different settings by varying the degree threshold k and the fraction threshold p. The default values of k and p are 10 and 0.6, respectively.

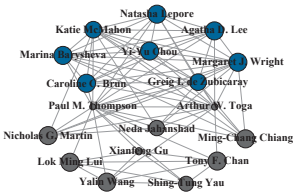
B. Evaluation of Effectiveness

Statistics. In Figure 6, we report the size of (k,p)-core on 8 different graphs using default settings. We can clearly see that, except for Facebook and Orkut, the size of our $(10, 0.6)$-cores is much smaller than the size of 10-cores in terms of vertex number. The similar size of $(10, 0.6)$-core and 10-core on both Facebook and Orkut is due to their high density such that most vertices have larger fraction values than in the other datasets.

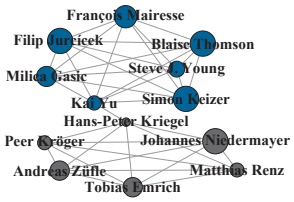
Comparing (k, p)-core with k-core For each graph, we report the global clustering coefficient ($\frac{3\times|\triangle|}{|triplet|}$) [11] and graph density ($\frac{2m}{n\times(n-1)}$) [5] of k-core and (k,p)-core using the default setting ($k = 10$ and $p = 0.6$). Figure 7 shows that the (k,p)-core has significantly higher global clustering coefficient than that of k-core on all the datasets, with the consideration of vertex fraction. Figure 8 reports that the graph density of (k,p)-core is higher than that of k-core on most datasets. This experiment indicates that (k,p)-core may be more promising than k-core, considering the similar computation costs.



(a) Case study of k-core and (k,p)-core on DBLP-3



(b) Case study 1 on DBLP-10



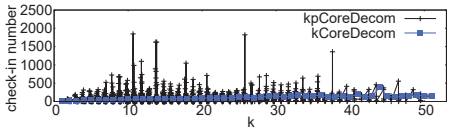
(c) Case study 2 on DBLP-10

Fig. 9. Real-life k-Cores and (k,p)-Cores

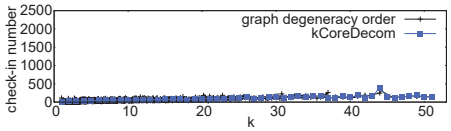
Case study on DBLP. Figure 9(a) depicts one connected component of the k-core and (k,p)-core with $k = 15$ and $p = 0.5$ on DBLP-3 where an edge exists between two vertices iff the corresponding two authors have co-authored at least 3 papers. As the whole k-core and (k,p)-core are too large to present, we show one connected component on DBLP-3 for a clear visualization. The vertices of k-core are marked by dark grey and blue. The vertices of (k,p)-core are marked by blue color. The light grey vertices are the 1-hop neighbors of the k-core vertices, which help to illustrate the connection of the k-core vertices to the rest of the graph. The size of each vertex in k-core reflects the fraction value of the vertex i.e., its degree in k-core divided by its degree in the whole graph. We can see that, with fraction constraint, the vertices in (k,p)-core are less connected to rest of the graph compared to the vertices in k-core. Figures 9(b)(c) show two connected components of k-core on DBLP-10 (10 co-authored papers required for each edge), when $k = 5$ and $p = 0.4$. Similarly, the (k,p)-core vertices are marked by blue and the k-core vertices are marked by dark grey and blue. In Figure 9(b), the author "Xiangfang Gu" has the smallest faction value, i.e., the highest connection to the outside vertices of k-core. In the (k,p)-core computation, the leave of "Xiangfang" leads to the departure of the other 7 authors. Similarly in Figure 9(c), the leave of "Hans-Peter Kriegel" results in the leave of a group of authors and then produces the (k,p)-core.

Case study on Gowalla. The effectiveness of k-core and (k,p)-core models in analyzing user engagement over dataset Gowalla is demonstrated in Figure 10. The Gowalla dataset is a location-based social network which records both friendship network and the spatial checking-ins of each user, launched in Feb. 2009 to Oct. 2010. We measure the engagement of users using their check-in times (i.e., numbers). We compare the check-in times based on k-core decomposition, (k,p)-core decomposition and onion layers (graph degeneracy order) [26].

In Figure 10(a), for every core number k in core decomposition, we show the average check-in times of the vertices



(a) (k,p)-core vs k-core



(b) k-core vs graph degeneracy order

Fig. 10. User Check-in Number in Gowalla

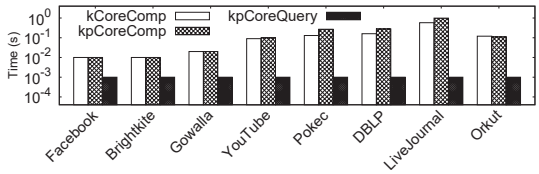


Fig. 11. Computation Time, $k = 10$, $p = 0.6$

with same core number, and the average check-in times of the vertices with same p-number given the core number k. The x-axis is the k value for k-core decomposition, and $(k+p-0.5)$ for (k,p)-core decomposition. We can see the average check-in time basically increases with a larger core number in k-core, while some users with small core numbers can have many check-ins, as found by the (k,p)-core decomposition. The users who are more active (with more check-ins) basically have larger p-numbers. The (k,p)-core decomposition reveals finer granularity in k-core decomposition.

In Figure 10(b), we further compare (k,p)-core decomposition with the onion layers in k-core decomposition. The onion layers cannot effectively distinguish the users with different activeness (check-in times) on a same core number k.

C. Evaluation of Efficiency

Computation time. Figure 11 reports the time of computing the (k,p)-core with default k and p values, by kpCoreComp and kpCoreQuery, respectively. We also report the time of kCoreComp for the same input of k. We can see that kpCoreComp is slightly slower than kCoreComp because some k-core vertices are deleted in (k,p)-core computation. The efficiency of the two algorithms are quite close, because we can use one combined vertex threshold set to conduct the (k,p)-core computation which is as efficient as using the degree threshold k in k-core computation. kpCoreQuery outperforms the other two algorithms in runtime by more than 1 order of magnitude, because its time cost is linear to the size of output.

In Figure 12, we show the effect of different k and p to the algorithms on Orkut. The time costs of kCoreComp and kpCoreComp both increase with a larger input of k, as they need to remove more vertices and edges. The runtime of kpCoreComp becomes slightly higher with a larger input of p due to more deletion steps. The runtime of kpCoreQuery is much faster and almost same for different k and p values.

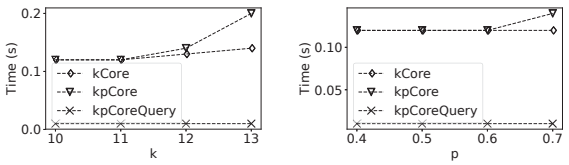


Fig. 12. Effect of Different Inputs

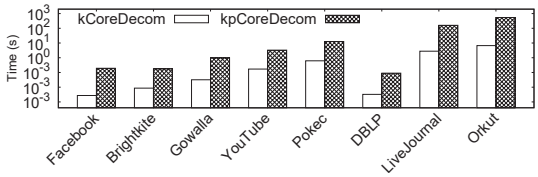


Fig. 13. Decomposition Time

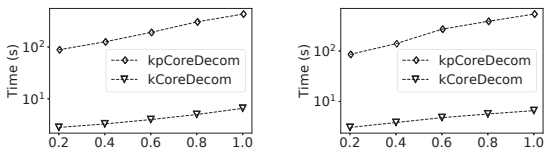
Decomposition time. Figure 13 compares the decomposition time between `kcoreDecomp` and `kpCoreDecomp` on all the datasets. We can see that `kpCoreDecomp` costs more time than `kcoreDecomp` because it conducts $d(G)$ times the decomposition for p-numbers. In general, our (k,p)-core decomposition can finish in 10s on most datasets and can finish in 1000s on a dataset with more than 100 million edges (`Orkut`).

Scalability of (k,p)-core decomposition. To test the scalability of k-core decomposition (`kcoreDecomp`) and (k,p)-core decomposition (`kpCoreDecomp`), we vary the number of nodes (resp. edges) by randomly sampling nodes (resp. edges) from 20% to 100% on Orkut dataset. We use the induced subgraphs of the sampled vertex/edge set as the input graphs. As shown in Figure 14, the time cost of both k-core and (k,p)-core decomposition increases when the input data becomes larger. We can see both algorithms are scalable given the larger datasets. The trends are similar on other datasets.

D. Index Maintenance

Update time of index on different datasets. In this experiment, we randomly remove 500 distinct existing edges from the graph and then insert them back, to test the time cost of index update. We report the average processing time for one edge insertion and removal, respectively. `kpCoreDecomp` is used as the baseline algorithm which computes the (k,p)-core index from scratch for each edge insertion or deletion. The results are shown in Figure 15. Our proposed (k,p)-core maintenance algorithms (`kpIndexInsert` and `kpIndexDelete`) can achieve up to two orders of magnitude speed-up, compared with the baseline algorithm. For example, on LiveJournal, our proposed algorithm can handle one edge insertion and removal in 1.6s and 5.3s, respectively, while `kpCoreDecomp` requires 159s. It is benefit from the well-designed fraction upper bounds and lower bounds in Section VI. Our (k,p)-core maintenance algorithms only need to update a small portion of the index for both edge insertion and deletion.

Scalability of (k,p)-core index maintenance. We sample the graph in the same way as in the scalability test of (k,p)-core decomposition. Figure 16 shows the results on Orkut and the trends are similar on other datasets. The result shows that our



(a) Varying edges (b) Varying vertices

Fig. 14. Scalability of Decomposition Algorithms

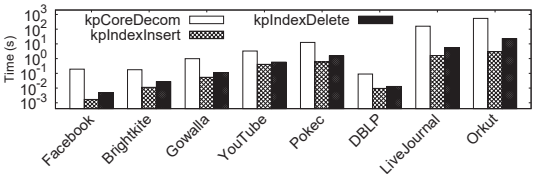


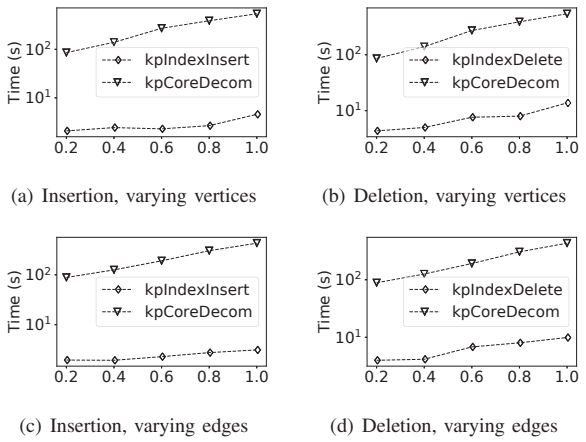
Fig. 15. Update Time of KP-Index

proposed index maintenance algorithms are as scalable as the decomposition algorithm `kpCoreDecomp`. This is because our algorithms always visit a small part of the index for update which is proportional to the size of input graph.

VIII. Related Work

Cohesive subgraph mining is a fundamental problem in graph structure analysis. The clique [11] is defined as a subgraph in which every vertex is adjacent to every other vertex in the subgraph. As the definition of clique is often too rigid, many clique-relaxed cohesive subgraphs are proposed, e.g., k-plex [20] and quasi-clique [1]. However, the above models suffer from computation intractability due to their exponential results or NP-hard computations. Recently, the model of k-core receives a lot of interests for its elegant structures as well as efficient solutions [19]. The k-core is shown to be the maximal equilibrium in the game that each vertex incurs a cost k to remain engaged while obtains a benefit of 1 from each engaged neighbor [4]. A linear-time solution for k-core computation and decomposition is proposed in [3]. A fast order-based core number maintenance algorithm is proposed in [30]. There are many k-core related studies, such as [10], [23], [24], [25], [26], [27], [29], [31]. Nevertheless, none of the above models consider the fraction constraint [2] of vertex neighbors in modeling cohesive subgraphs.

On a time-evolving social network, an empirical study [2] of user-to-user content transfer demonstrates that (1) user adoption rates increase as more friends are adopting (reasoning degree constraint k in (k,p)-core) and (2) high-degree users need more friends in adopting to reach a certain adoption rate, compared with low-degree users (reasoning fraction constraint p in (k,p)-core). Besides, the contagion model [15] applies the fraction constraint to model user interactions, where a user adopts a behaviour if at least a certain fraction of his/her network neighbors adopted the behaviour. In their textbook [6], Easley and Kleinberg emphasize the contagion model and show that the fraction constraint may be used in cohesive subgraphs. It is promising to apply the fraction constraint p in modeling cohesive subgraphs.



(a) Insertion, varying vertices (b) Deletion, varying vertices

(c) Insertion, varying edges (d) Deletion, varying edges

Fig. 16. Scalability of KP-Index Maintenance

The (α, β)-cluster model proposed in [13] requires that each vertex in the cluster is adjacent to at least β-faction of vertices in the cluster and each vertex outside of the cluster is adjacent to at most α-fraction of vertices in the cluster. When β gets closer to 1, the cluster approaches a clique; when $\beta < 1$ and $\alpha = (1 - \epsilon)\beta$, the cluster becomes a quasi-clique [16]. Our (k,p)-core model is inherently different from (α, β)-cluster as (k,p)-core imposes the requirement on the *degree* of a vertex while (α, β)-cluster imposes requirement for a vertex regarding the adjacency to the *whole cluster*. A vertex u in an (α, β)-cluster may not be well-engaged in the cluster, because its outside neighbors can be much larger than the neighbors inside of the cluster, while this is avoided in the (k,p)-core model as the fraction constraint p is applied to every vertex.

IX. Conclusion

In this paper, we propose a novel cohesive subgraph, named (k,p)-core, which requires that each vertex has at least k neighbors and at least p fraction of its network neighbors in the subgraph. We propose an efficient algorithm to compute the (k,p)-core with given k and p. Besides, we present an algorithm for the decomposition of (k, p)-core, which can also be applied to build the KP-Index for time-optimal (k, p)-core query. Novel index maintenance algorithms are also introduced to handle the dynamic of real-world graphs. Extensive experiments validate that our proposed algorithms are efficient and the (k,p)-core model is effective on real-life networks.

Acknowledgments

Xuemin Lin is supported by 2018YFB1003504, NSFC61232006, ARC DP180103096 and DP170101628. Ying Zhang is supported by ARC DP180103096 and FT170100128. Lu Qin is supported by ARC DP160101513. Wenjie Zhang is supported by ARC DP180103096.

References

[1] J. Abello, M. G. C. Resende, and S. Sudarsky. Massive quasi-clique detection. In *LATIN*, pages 598–612, 2002.
[2] E. Bakshy, B. Karrer, and L. A. Adamic. Social influence and the diffusion of user-created content. In *ACM Conference on Electronic Commerce (EC)*, pages 325–334, 2009.
[3] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.
[4] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015.
[5] T. F. Coleman and J. J. Moré. Estimation of sparse jacobian matrices and graph coloring blems. *SIAM journal on Numerical Analysis*, 20(1):187–209, 1983.
[6] D. A. Easley and J. M. Kleinberg. *Networks, Crowds, and Markets - Reasoning About a Highly Connected World*. Cambridge University Press, 2010.
[7] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *KDD*, pages 150–160, 2000.
[8] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. Identification of influential spreaders in complex networks. *Nature physics*, 6(11):888–893, 2010.
[9] R. Laishram, A. E. Sariyüce, T. Eliassi-Rad, A. Pinar, and S. Soundarajan. Measuring and improving the core resilience of networks. In *WWW*, pages 609–618, 2018.
[10] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou. Efficient (α, β)-core computation: an index-based approach. In *WWW*, pages 1130–1141, 2019.
[11] R. D. Luce and A. D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.
[12] F. D. Malliaros and M. Vazirgiannis. To stay or not to stay: modeling engagement dynamics in social graphs. In *CIKM*, pages 469–478, 2013.
[13] N. Mishra, R. Schreiber, I. Stanton, and R. E. Tarjan. Finding strongly-knit clusters in social networks. *Internet Math.*, 5(1-2):153–172, 2009.
[14] F. Morone, G. Del Ferraro, and H. A. Makse. The k-core as a predictor of structural collapse in mutualistic ecosystems. *Nature Physics*, 15(1):95, 2019.
[15] S. Morris. Contagion. *The Review of Economic Studies*, 67(1):57–78, 2000.
[16] J. Pattillo, N. Youssef, and S. Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013.
[17] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658–2663, 2004.
[18] A. E. Sariyüce, B. Gedik, G. Jacques-Silva, K. Wu, and Ü. V. Çatalyürek. Streaming algorithms for k-core decomposition. *PVLDB*, 6(6):433–444, 2013.
[19] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.
[20] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.
[21] K. Shin, T. Eliassi-Rad, and C. Faloutsos. Corescope: Graph mining using k-core analysis - patterns, anomalies and algorithms. In *ICDM*, pages 469–478, 2016.
[22] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *PNAS*, 109(16):5962–5966, 2012.
[23] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin. Efficient computing of radius-bounded k-cores. In *ICDE*, pages 233–244, 2018.
[24] F. Zhang, C. Li, Y. Zhang, L. Qin, and W. Zhang. Finding critical users in social communities: The collapsed core and truss problems. *TKDE*, 2018.
[25] F. Zhang, L. Yuan, Y. Zhang, L. Qin, X. Lin, and A. Zhou. Discovering strong communities with user engagement and tie strength. In *DASFAA*, pages 425–441, 2018.
[26] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin. OLAK: an efficient algorithm to prevent unraveling in social networks. *PVLDB*, 10(6):649–660, 2017.
[27] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Finding critical users for social network engagement: The collapsed k-core problem. In *AAAI*, pages 245–251, 2017.
[28] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. When engagement meets similarity: Efficient (k, r)-core computation on social networks. *PVLDB*, 10(10):998–1009, 2017.
[29] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Efficiently reinforcing social networks over user engagement and tie strength. In *ICDE*, pages 557–568, 2018.
[30] Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin. A fast order-based approach for core maintenance. In *ICDE*, pages 337–348, 2017.
[31] Z. Zhou, F. Zhang, X. Lin, W. Zhang, and C. Chen. K-core maximization: An edge addition approach. In *IJCAI*, pages 4867–4873, 2019.