# Finding the Best $k$ in Core Decomposition: A Time and Space Optimal Solution

Deming Chu[⋆†], Fan Zhang[⋆], Xuemin Lin[§], Wenjie Zhang[§], Ying Zhang[♯], Yinglong Xia[‡], Chenyi Zhang[♮]

[⋆]*Guangzhou University,* [†]*East China Normal University,* [§]*University of New South Wales*
[♯]*University of Technology Sydney,* [‡]*Facebook AI,* [♮]*Huawei Technologies*
{ned.deming.chu, fanzhang.cs}@gmail.com, {lxue, zhangw}@cse.unsw.edu.au
ying.zhang@uts.edu.au, yxia@fb.com, zhangchenyi2@huawei.com

***Abstract*—The mode of $k$-core and its hierarchical decomposition have been applied in many areas, such as sociology, the world wide web, and biology. Algorithms on related studies often need an input value of parameter $k$, while there is no existing solution other than manual selection. In this paper, given a graph and a scoring metric, we aim to efficiently find the best value of $k$ such that the score of the $k$-core (or $k$-core set) is the highest. The problem is challenging because there are various community scoring metrics and the computation is costly on large datasets. With the well-designed vertex ordering techniques, we propose time and space optimal algorithms to compute the best $k$, which are applicable to most community metrics. The proposed algorithms can compute the score of every $k$-core (set) and can benefit the solutions to other $k$-core related problems. Extensive experiments are conducted on 10 real-world networks with size up to billion-scale, which validates both the efficiency of our algorithms and the effectiveness of the resulting $k$-cores.**

## I. Introduction

Graphs are widely adopted to model the structure of complex networks with a large spectrum of applications. As a fundamental graph problem, cohesive subgraph mining is to extract groups of densely connected vertices. The well-studied model of $k$-core is defined as a maximal *connected* subgraph in which every vertex is connected to at least $k$ other vertices in the same subgraph [41], [49]. We use $k$-core set to denote the subgraph formed by all the (connected) $k$-cores in the graph, for a fixed parameter $k$. The hierarchical decomposition by $k$-core (or core decomposition) computes the $k$-core set for every possible $k$. The hierarchy of core decomposition builds on the containment property of $k$-core sets: the $(k+1)$-core set is always a subgraph of the $k$-core set.

The $k$-core and its hierarchical structure have a wide range of applications, such as finding communities in the web [22] and social networks [25], [55], [63], discovering molecular complexes in protein interaction networks [6], recognizing hub-nodes in brain functional networks [8], analyzing underlying structure of the Internet and its functional consequences [10], understanding software systems using software networks [64], and predicting structural collapse in mutualistic ecosystems [43].

Despite the elegant structure of $k$-core hierarchy and the various applications, a common open problem is to find the best $k$-core set or the best single $k$-core. To the best of our knowledge, there is no previous work that studies the best $k$ value for $k$-core and for other models of cohesive subgraph, e.g. $k$-truss, $k$-plex, $k$-vcc, $k$-ecc. The existing solution is to simply adopt the user-defined $k$, or to determine $k$ by trivial statistics, e.g., setting $k$ to average vertex degree. The underlying problem is essentially to ask the quality of the $k$-cores, since different input values lead to different $k$-cores. In this paper, given a graph and a community scoring metric, we aim to efficiently compute a value of $k$ such that the $k$-core set has the highest score among all the $k$-core sets for every integer $k$. Besides, we also aim to efficiently find a best single $k$-core among all the $k$-cores for every integer $k$.

There is no uniform metric to evaluate the quality of a subgraph [61]. Different community scoring metrics are designed for different objectives considering particular network properties [11], [61]. For example, clustering coefficient is a goodness metric for clustering tendency [57], and modularity measures the quality of a graph partition [44], [45]. To apply our proposed algorithms to various community metrics and even new metrics, we extract representative primary values which can be utilized to calculate most community scores [11].

A basic solution of finding the best $k$ is to conduct quality computation on the $k$-core set for every possible input of $k$. In our preliminary experiments, for a graph that can be loaded into memory within 100 seconds, the basic solution spent more than one day for some scoring metrics. To efficiently handle billion-scale graphs, we design a light-weight vertex ordering procedure to facilitate incremental score computation. We compute the score of $k$-core sets from the center core of the graph to the margins, with optimal time and space cost.

Our paradigm can also be applied to find a best single $k$-core when the connectivity of different $k$-cores is considered in the computation. The score of every $k$-core (set) can be retrieved by our algorithms. Moreover, our algorithm can help solve other $k$-core related problems, e.g., finding densest subgraph, maximum clique, and size-constrained $k$-core. The experiments in the paper show that our algorithm can serve as better approximate solutions compared to the state-of-the-art or produce useful intermediate results.

**Example 1.** *Figure 1 illustrates the hierarchy of a network decomposed by the $k$-core model, where $k$ varies from 1 to 3. When $k = 1$, the 1-core (set) is whole graph, i.e., $G_1$.*

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

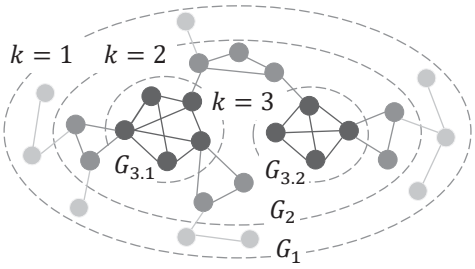



Fig. 1. Hierarchical Core Decomposition

*When $k = 2$, the $k$-core decomposition recursively deletes every vertex with degree less than 2, resulting in the 2-core (set) $G_2$. When $k = 3$, the 3-core set consists of two 3-cores: $G_{3.1}$ and $G_{3.2}$.*

*Suppose the scoring metric is average degree (twice the number of edges divided by the number of vertices), (i) the best $k$-core set is the 3-core set, because it has the highest score (average degree) among all the $k$-core sets, and the best $k$ is 3; and (ii) the best single $k$-core is $G_{3,1}$ as it has the highest score among all the $k$-cores with different $k$.*

**Contributions.** The principal contributions in this paper are summarized as follows.

- To the best of our knowledge, this is the first work to study the following problems: given a graph $G$ and a community scoring metric $Q$, (i) find the best $k$-core set; and (ii) find a best single $k$-core, according to $Q$.
- We develop efficient and extensible algorithms to solve our proposed problems. We prove the algorithms are worst-case optimal in both time and space complexities. For many scoring metrics, our score computation can return in $O(n)$ time where $n$ is the number of vertices in the graph. The algorithms can apply to various community metrics and can shed light on finding the best $k$ for other decomposition models.
- Extensive experiments are conducted on 10 real graphs with up to billions of edges. The results show that the quality of $k$-core (set) with the resulting $k$ is much better than that with user-defined $k$ or other trivial values. Towards efficiency, our algorithms outperform the baselines by 1-4 orders of magnitude in runtime. For some $k$-core related problems, our algorithms can serve as good approximate solutions or benefit the algorithm design.

## II. Preliminaries

We consider an undirected and unweighted simple graph $G = (V, E)$, with $n = |V|$ vertices and $m = |E|$ edges (assume $m > n$). Given a vertex $v$ in a subgraph $S$, $N(v, S)$ denotes the neighbor set of $v$ in $S$, i.e., $N(v, S) = \{u \mid (u, v) \in E(S)\}$. The degree of $v$ in subgraph $S$, i.e., $|N(v, S)|$, is denoted by $d(v, S)$. For the input graph $G$, we omit $G$ in the notations when the context is clear, e.g., we abbreviate $N(v, G)$ to $N(v)$. Table I summarizes the notations.

### A. Core Decomposition

The model of $k$-core [41], [49] is defined as follows.

TABLE I
Summary of Notations

| Notation | Definition |
|---|---|
| $G = (V, E)$ | an undirected, unweighted simple graph |
| $n; m$ | number of vertices/edges in $G$ ($m > n$) |
| $S$ | a subgraph of $G$ |
| $V(S)$ | the set of vertices in $S$ |
| $E(S)$ | the set of edges in $S$ |
| $id(v)$ | unique identification of $v$ in $G$ |
| $N(v); N(v, S)$ | the set of neighbors of $v$ in $G$ / in $S$ |
| $d(v); d(v, S)$ | degree of $v$ in $G$ / in $S$ |
| $C_k$ | the $k$-core set of $G$ |
| $c(v)$ | coreness of $v$ in $G$, $\max\{k \mid v \in C_k\}$ |
| $k_{max}$ | largest $k$ s.t. $C_k$ is not empty |
| $H_k$ | the $k$-shell of $G$, $\{v \mid c(v) = k\}$ |
| $rank(v)$ | the ranking of vertex $v$ in $V$ |
| $n(S)$ | number of vertices in $S$ |
| $m(S)$ | number of edges in $S$ |
| $b(S)$ | number of boundary edges of $S$ |
| $\Delta(S)$ | number of triangles in $S$ |
| $t(S)$ | number of triplets in $S$ |

**Definition 1** ($k$-Core)**.** *Given a graph $G$ and an integer $k$, a subgraph $S$ is a $k$-core of $G$, if (i) each vertex $v \in S$ has at least $k$ neighbors in $S$, i.e., $d(v, S) \geq k$; (ii) $S$ is connected; and (iii) $S$ is maximal, i.e., any supergraph of $S$ is not a $k$-core except $S$ itself.*

We use the $k$-core set to denote the subgraph containing every (connected) $k$-core with a fixed parameter $k$.

**Definition 2** ($k$-Core Set)**.** *Given a graph $G$ and an integer $k$, the $k$-core set of $G$, denoted by $C_k$, is the subgraph formed by all the (connected) $k$-cores in $G$.*

If an integer $k' \geq k$, the $k'$-core set is always a subgraph of the $k$-core set, i.e., $C_{k'} \subseteq C_k$. This implies that each vertex in a graph has a unique number of coreness [7].

**Definition 3** (Coreness and $k$-Shell)**.** *Given a graph $G$, the coreness of a vertex $v \in G$ is $c(v) = \max\{k \mid v \in C_k\}$, i.e., the largest $k$ such that the vertex $v$ is in the $k$-core. The $k$-shell of $G$ is $H_k = \{v \mid c(v) = k\}$, i.e., the set of vertices with coreness $k$.*

We use $k_{max}$ (graph degeneracy) to denote the largest coreness in $G$, i.e., the largest $k$ such that the $k$-core of $G$ is not empty.

**Definition 4** (Core Decomposition)**.** *Given a graph $G$, core decomposition is to compute the coreness $c(v)$ for every vertex $v \in V(G)$.*

The algorithm for core decomposition is to recursively remove the vertex with the smallest degree in the graph, with time complexity of $O(m)$ [7].

**Example 2.** *Figure 2 shows a graph of 12 vertices. The graph itself is a 2-core. When $k = 3$, the $k$-core computation recursively deletes every vertex with degree less than 3, i.e., $v_5, v_7, v_6$ and $v_8$. The 3-core set is the induced subgraph by the rest vertices. The coreness of $v_5, v_7, v_6$ and $v_8$ is 2, and*

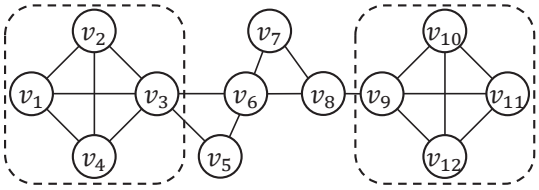



Fig. 2. A 2-core graph in which the 3-core is circled

*the coreness of other vertices is* 3*. There are two connected components (*3*-cores) in the* 3*-core set, as circled in the figure.*

### B. Problem Definition

We are the first to propose and study the following problems of finding the best $k$-core (set).

- find the $k^*$-core set $C_{k^*}$ such that it has the highest score among all the $k$-core sets for every integer $k$ with $0 \leq k \leq k_{max}$;
- find the $k^*$-core $S$ such that it has the highest score among all the $k$-cores for every integer $k$ with $0 \leq k \leq k_{max}$.

### C. Community Scoring Metrics

Although there are various community scoring metrics considering different community properties, most of them are constructed on the following primary values [61].

**Primary Values.** Let $S$ be a subgraph to evaluate, we study the following common primary values.

- $n(S)$: the number of vertices in $S$, i.e., $n(S) = |V(S)|$;
- $m(S)$: the number of edges in $S$, i.e., $m(S) = |E(S)|$;
- $b(S)$: the number of boundary edges in $S$, $b(S) = |\{(u, v) \mid (u, v) \in E, u \in S, v \notin S\}|$;
- $\Delta(S)$, the number of triangles in $S$;
- $t(S)$, the number of triplets (three vertices connected by two edges) in $S$, i.e., $t(S) = \sum_{v \in S} \binom{d(v,S)}{2}$;

**Metrics.** Based on the above primary values, some community scoring metrics are defined as follows.

- Average Degree: $f(S) = \frac{2 \times m(S)}{n(S)}$ is the average degree of vertices in $S$;
- Internal Density: $f(S) = \frac{2 \times m(S)}{n(S) \times (n(S)-1)}$ is the internal density of vertices in $S$;
- Cut Ratio: $f(S) = 1 - \frac{b(S)}{n(S) \times (n - n(S))}$ is the difference of 1 and the fraction of existing boundary edges over all the possible ones;
- Conductance: $f(S) = 1 - \frac{b(S)}{2 \times m(S) + b(S)}$ is the difference of 1 and the proportion of degrees where the vertex points to the outside;
- Modularity: $f(P) = \sum_{i=1}^{k} \left( \frac{m(P_i)}{m} - \left( \frac{2 \times m(P_i) + b(P_i)}{2 \times m} \right)^2 \right)$ is the modularity for a partition $P$ on $G$, where $P_i$ is the $i^{th}$ community in the partition [45];
- Clustering Coefficient: $f(S) = \frac{3 \times \Delta(S)}{t(S)}$ measures the tendency of vertices to cluster together;

Given a scoring metric, for an integer $k$, the target subgraphs for computing the scores are all the $k$-cores with the given $k$, i.e., the subgraphs in $C_k$.

## III. Finding the Best K-Core Set

In this section, we propose the solution to find the best $k$ for the $k$-core set, including the baseline in Section III-A, and the improved algorithm in the rest sections.

### A. Baseline Algorithm

Given a graph $G$ and a community scoring metric $Q$, a baseline solution is to first conduct core decomposition, order the vertices by coreness to fast retrieve a $k$-core set, and then compute the score of every $k$-core set according to $Q$, for every integer $k$ with $0 \leq k \leq k_{max}$.

We use a bin sort to order the vertices in $G$ by coreness, and record every start position of the vertices with the same coreness, which takes $O(n)$ time. Then, retrieving the vertex set of a $k$-core set $C_k$ takes $O(|V(C_k)|)$ time. Let $O(q_k)$ be the time cost to compute the score of $C_k$ given $V(C_k)$, $Q$ and $G$. The time complexity of the baseline algorithm is $O\left(\sum_{k=0}^{k_{max}} (q_k + |V(C_k)|)\right)$. Although the baseline runs in polynomial time, it is costly to handle large graphs.

### B. Vertex Ordering for Optimal Neighbor Query

In order to compute the best $k$ in optimal time and space cost, we first introduce the vertex ordering techniques.

We divide the vertices in $G$ into $k_{max} + 1$ arrays where each is associated with a unique coreness value. The arrays are ordered by the associated coreness to fast locate the vertices with different coreness.

For every vertex $v$ in $G$, the neighbor set of $v$ is ordered by increasing values of vertex rank which is defined as follows.

**Definition 5** (Vertex Rank)**.** *Given two vertices $u$ and $v$, the rank of $v$ is higher than $u$, denoted by $rank(v) > rank(u)$, if (i) $c(v) > c(u)$; OR (ii) $c(v) = c(u)$ and $id(v) > id(u)$, where $id(v)$ is the unique identification of $v$.*

For every vertex $v$, to efficiently retrieve specific parts of its neighbour set, we also record some position tags where $same$ is the position of the first $u$ in $N(v)$ such that $c(u) \geq c(v)$, $plus$ is the position of the first $u$ in $N(v)$ such that $c(u) > c(v)$, and $high$ is the position of the first $u$ in $N(v)$ such that $rank(u) > rank(v)$. The above tag is set to $|N(v)|$ when there is no such $u \in N(v)$ satisfying the condition. The ordering is summarized in Table II.

Algorithm 1 shows the pseudo-code for vertex ordering. At Line 1-4, we use $k_{max} + 1$ bins to sort vertices by rank, where each bin represents a unique value of coreness. Inspired by [33], we flatten $k_{max} + 1$ bins to sort the edge set: (i) At Line 5-8, for every vertex $v$ with ascending order of vertex id, we push the pair $(v, u)$ from every neighbor $u$ of $v$ into the bin represented by the coreness of $v$; (ii) At Line 9-11, we retrieve the sorting from the $k_{max} + 1$ bins by one scan from bin 0 to bin $k_{max}$, where $N'(u)$ is empty initially for every $u \in V$. For every element $(v, u)$ in the bins, we push $v$ into the new neighbor set of $u$ s.t. the neighbors of $u$ are ordered by vertex rank. Besides, we record the subscripts in Table II for each vertex via one scan of the new edge set (Line 13).

**Complexity of Vertex Ordering and Query.** In Algorithm 1, the sorting of vertices takes $O(n)$ since every vertex is visited

TABLE II
THE ORDERED NEIGHBOR SET OF $v$

| Subscript of $u \in N(v)$ | Constraint of $u$ | Vertex Set |
|---|---|---|
| $[0, same-1]$ | $c(u) < c(v)$ | $N(v,<)$ |
| $[same, plus-1]$ | $c(u) = c(v)$ | $N(v,=)$ |
| $[plus, \lvert N(v)\rvert -1]$ | $c(u) > c(v)$ | $N(v,>)$ |
| $[same, \lvert N(v)\rvert -1]$ | $c(u) \geq c(v)$ | $N(v,\geq)$ |
| $[high, \lvert N(v)\rvert -1]$ | $rank(u) > rank(v)$ | $N(v,>_r)$ |

Every $u$ in $N(v)$ is ordered by *ascending* order of rank;
$\underline{same}$: position of the first $u \in N(v)$ s.t. $c(u) \geq c(v)$;
$\underline{plus}$: position of the first $u \in N(v)$ s.t. $c(u) > c(v)$;
$\underline{high}$: position of the first $u \in N(v)$ s.t. $rank(u) > rank(v)$.

**Algorithm 1**: vertex ordering

**Input** : a graph $G = (V, E)$, the coreness $c(v)$ of every vertex $v \in V$
**Output** : $G$ with ordered $V$ and $E$ by vertex rank, and position tags

/* Order vertex set $V$ */
1 bin1 $\leftarrow$ a list of $(k_{max}+1)$ empty arrays;
2 **for** each $v$ in $V$ with *ascending* order of vertex id **do**
3 bin1[$c(v)$].push_back($v$);
4 $V \leftarrow$ concatenation of bin1[0], bin1[1], ..., bin1[$k_{max}$];
/* Order edge set $E$ */
5 bin2 $\leftarrow$ a list of $(k_{max}+1)$ empty arrays;
6 **for** each $v$ in $V$ with *ascending* order of vertex id **do**
7 **for** each $u$ in $N(v)$ **do**
8 bin2[$c(v)$].push_back(pair($v, u$));
9 **for** each $i$ from 0 to $k_{max}$ **do**
10 **for** each element $(v, u)$ in bin2[$i$] from head to tail **do**
11 $N'(u)$.push_back($v$);
12 $E \leftarrow$ each $v \in V$ has neighbors $N'(v)$;
13 $same$, $plus$ and $high$ of $v \leftarrow$ scan $N'(v)$ for every $v \in V$;
14 **return** $G = (V, E)$ with ordered $V$ and $E$, and position tags

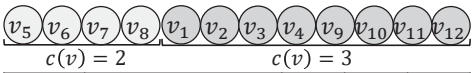
twice, and the sorting of the edge set takes $O(m)$ since every edge is visited at most 5 times. The overall time complexity is $O(m)$. The space complexity is dominated by the size of the graph which is $O(m)$.

After the vertex ordering, according to Table II, we can answer $|N(v,\cdot)|$ in $O(1)$ time, e.g., $|N(v,=)| = plus - same$, and return $N(v,\cdot)$ in $O(|N(v,\cdot)|)$ time.

**Example 3.** *Figure 3 illustrates the vertex ordering for the graph in Figure 2. Each vertex in Figure 3 is colored according to different coreness values. On the top, all the vertices are ordered by coreness with ties broken by vertex id. The vertices in every neighbor set are also sorted by the above order. In the figure, we show the ordered neighbor sets of four vertices and their position tags. For vertex $v_1$ and $v_9$, the position $plus$ equals to the length of the neighbor set because the coreness of any neighbor is not larger than the vertex. By checking the ordering results and Table II, all types of $|N(v,\cdot)|$ queries can be answered in $O(1)$, e.g., $|N(v_6,>)| = |N(v_6)| - plus = 1$.*

### C. The Improved Algorithm

An important property of hierarchical core decomposition is its containment nature: $C_{k+1} \subseteq C_k$ for every coreness $k$. This suggests that the primary values of the $k$-core set can be




| Vertex | Ordered Neighbors | *same* | *plus* | *high* |
|---|---|---|---|---|
| $v_1$ | $v_2$ $v_3$ $v_4$ | 0 | 3 | 0 |
| $v_6$ | $v_5$ $v_7$ $v_8$ $v_3$ | 0 | 3 | 1 |
| $v_8$ | $v_6$ $v_7$ $v_9$ | 0 | 2 | 2 |
| $v_9$ | $v_8$ $v_{10}$ $v_{11}$ $v_{12}$ | 1 | 4 | 1 |

Fig. 3. Example of Vertex Ordering

**Algorithm 2**: computing the best $k$

**Input** : a graph $G$, a community scoring metric $Q$
**Output** : the best $k$ value for a $k$-core set
1 compute each $k$-core set $C_k$ by core decomposition;
2 order $G$ by Algorithm 1;
3 metric $\leftarrow [0, \ldots, 0]$;
4 in $\leftarrow 0$, out $\leftarrow 0$, num $\leftarrow 0$;
5 **for** each integer $k$ from $k_{max}$ to 0 **do**
6 **for** $v \in H_k$ **do**
7 in $\leftarrow$ in $+ |N(v,>)| + \frac{1}{2}|N(v,=)|$;
8 out $\leftarrow$ out $+ |N(v,<)| - |N(v,>)|$;
9 num $\leftarrow$ num $+ 1$;
10 metric[$k$] $\leftarrow$ compute_metric($Q$, in, out, num);
11 **return** metric

incrementally derived from the primary values of the $(k+1)$-core set, with minor modification according to their difference. Based on the above observation, we compute the scores of the $k$-core sets in a top-down manner, i.e., for $k$ from $k_{max}$ to 0. Note that it is costly to count some primary values in a bottom-up manner, i.e., for $k$ from 0 to $k_{max}$.

Algorithm 2 shows the pseudo-code for computing the best $k$. The algorithm incrementally computes and updates the primary values of $k$-core sets. Note that we present the computation for clustering coefficient in Section III-D. In Algorithm 2, array metric records the score of every $k$-core set. Let $S$ be the subgraph induced by the visited vertices at Line 6. Variable in records the number of internal edges of $S$. Variable out records the number of boundary edges of $S$ where each edge has exactly one endpoint in $S$. Variable num records the number of visited vertices.

Line 3-4 initializes array metric and the primary values. Line 5-10 incrementally compute the score of the $k$-core set from $k = k_{max}$ to $k = 0$. For each value of $k$, we only visit the vertices with coreness $k$ at Line 6, and update the primary values at Line 7-9. Based on the primary values for the $(k+1)$-core set, we update the values by considering every neighbor $u$ of the visited vertex $v$ as follows.

- If $c(u) = c(v)$, then $(u, v)$ should be counted in the value in. Because $(u, v)$ will be counted in in when we visit both $u$ and $v$ at Line 6, we add $\frac{1}{2}|N(v,=)|$ to in for visiting $v$.
- If $c(u) < c(v)$, then $(u, v)$ should be counted in the value out. Because $(u, v)$ will only be counted in out when $v$

is visited at Line 6, we add $|N(v,<)|$ to out for visiting $v$.

- If $c(u) > c(v)$, then $u \in C_{k+1}$ and $(u,v) \notin C_{k+1}$, i.e., $(u,v)$ is a boundary edge of the $(k+1)$-core set. For the $k$-core, $(u,v)$ becomes an internal edge since $(u,v) \in C_k(G)$. So we add $|N(v,>)|$ to in and subtract $|N(v,>)|$ from out.

In Line 10, the community score is calculated from the primary values, e.g., the average degree of a $k$-core set is $2 \times \texttt{in}/\texttt{num}$, the cut ratio is $1 - \texttt{out}/(\texttt{num} \times (n - \texttt{num}))$, etc. Line 11 returns the result.

**Example 4.** *Given a community scoring metric such as average degree, the execution of Algorithm 2 on the graph in Figure 2 is as follows. First, we compute the primary values of the 3-core set by visiting every vertex $v$ in the 3-core set. Because there are 8 vertices in $C_3$, and every $v \in C_3$ has $|N(v,>)| = 0$ and $|N(v,=)| = 3$, the number of internal edges in the 3-core set is* $\texttt{in} = 8 \times (0+3/2) = 12$. *The average degree of 3-core set is $2 \times 12/8 = 3$. Then we visit each vertex in 2-shell and incrementally count the internal edges in 2-core. When $v_5, v_6, v_7$ and $v_8$ are sequentially visited, we have* $\texttt{in} = 12+(1+1/2)+(1+3/2)+(0+2/2)+(1+2/2) = 19$. *The average degree of 2-core set is $2 \times 19/12 \approx 3.17$. So 2-core set is preferred when the metric is average degree.*

**Correctness.** The correctness of Algorithm 2 is based on the correct computation of primary values for every $k$-core set, which is immediate according to the definitions of primary values, the containment property of $k$-cores, and Line 7-9.

**Complexity.** (Space) The space complexity is dominated by the size of the graph which takes $O(m)$. (Time) Benefitting from the ordering in Section III-B, $|N(v,\cdot)|$ can be retrieved in $O(1)$ time for every vertex $v$. The calculation of a community score takes $O(1)$. Each vertex is visited exactly once in Algorithm 2. The score computation in Algorithm 2 takes $O(n)$. Both core decomposition and vertex ordering take $O(m)$. The time complexity of Algorithm 2 is $O(m)$.

**Optimality.** Since the time or space complexity of computing an existing community metric on $G$ is at least $O(m)$ [11], the worst-case complexity of Algorithm 2 is optimal.

### *D. Computation of Triangles and Triplets*

Algorithm 3 shows the variant of Algorithm 2 for computing the best $k$ when the community metric is based on triangles and triplets, e.g., clustering coefficient. Similar to Algorithm 2, we incrementally compute the primary values in a top-down manner: from $k = k_{max}$ to $k = 0$. For every $k$-core set, array metric records its score, variable triangle records the number of triangles, and variable triplet records the number of triplets. They are initialized in Line 1-2.

**Triangle Counting.** At Line 7-12, the algorithm incrementally updates the number of triangles in the $k$-core set based on that in the $(k+1)$-core set. For each edge $(u,v)$ with $rank(u) > rank(v)$ (Line 7-8), we count the triangles containing $(u,v)$ by visiting the neighbors of $x$ where $x$ is the one in $\{u,v\}$ with smaller degree (Line 9-12). To count triangles, we enumerate each vertex $w$ in $N(x,>_r)$ and check if $w$ is also in $N(y,>_r)$. By visiting $N(\cdot,>_r)$ of each vertex, we ensure $rank(w) > rank(u) > rank(v)$ for every counted triangle, i.e., every triangle is counted for exactly once since we count the three vertices by increasing order of vertex rank.

**Algorithm 3**: computing $k$ by triangles and triplets

```
   Input  : a graph G, a community scoring metric Q
   Output : the best k value for a k-core set
1  metric ← [0, ..., 0];
2  triangle ← 0, triplet ← 0;
3  f_>[v] ← f_≥[v] ← 0 for each v ∈ V;
4  compute each k-core set C_k by core decomposition;
5  order G by Algorithm 1;
6  for each k from k_max to 0 do
7      for v ∈ H_k do
8          for u ∈ N(v, >_r) do
9              x ← u, y ← v;
10             swap(x, y) if deg(x) > deg(y) ;
11             for w ∈ N(x, >_r) do
12                 triangle ← triangle + 1 if
                   w ∈ N(y, >_r);
13         triplet ← triplet + C(|N(v, ≥)|, 2);
14     kshell_nbr ← ⋃_{u∈H_k} N(u, >);
15     for v ∈ kshell_nbr do
16         f_>[v] ← f_≥[v];
17     for v ∈ H_k do
18         for u ∈ N(v) do
19             f_≥[u] ← f_≥[u] + 1;
20     for v ∈ kshell_nbr do
21         gt_k ← f_>[v], eq_k ← f_≥[v] − f_>[v];
22         triplet ← triplet + C(eq_k, 2) + gt_k × eq_k;
23     metric[k] ←
       compute_metric(Q, triangle, triplet);
24 return metric
```

**Triplet Counting.** Recall that a triplet $\langle u,v,w \rangle$ is formed by three vertices connected by two edges $(v,u)$ and $(v,w)$, centered on $v$. (i) For each center vertex $v$ in $H_k$, any pair of the neighbors of $v$ in the $k$-core can form a triplet centered on $v$. Line 13 adds the number of such pairs to triplet. (ii) Line 14-22 counts the new triplets centered on the vertices of the $(k+1)$-core set.

We use $f_>[v]$ and $f_\geq[v]$ to count the number of $u \in N(v)$ such that $c(u) > k$ and $c(u) \geq k$, respectively. Here we do not use $N(v,\cdot)$, because a neighbor query in Section III-B only works for vertices in the $k$-shell, i.e., $f_>[v] = |N(v,>)|$ and $f_\geq[v] = |N(v,\geq)|$ iff $k = c(v)$. Line 14 collects the vertices in $N(u,>)$ of every $u$ with coreness $k$, because they are the center vertices in the $(k+1)$-core set. Line 15-19 computes $f_>[v]$ and $f_\geq[v]$ for every vertex $v$ in the $(k+1)$-core set, by updating the values from the previous iteration.

Line 20 iterates over each center vertex in the $(k+1)$-core set, and Line 21 assigns to $gt\_k$ and $eq\_k$ the number of $v$'s neighbor $u$ such that $c(u) > k$ and $c(u) = k$, respectively. To form a triplet (uncounted) with a center vertex in the $(k+1)$-core set, (a) the other two vertices are from $H_k$; or (b) one vertex is from $C_{k+1}$ and the other is from $H_k$. In Line 22,

$\binom{eq\_k}{2}$ is the number of the triplets in case (a), and $gt\_k \times eq\_k$ is the number of the triplets in case (b).

The community score is calculated in Line 23 using the primary values, e.g., the clustering coefficient of a $k$-core is $3 \times \texttt{triangle/triplet}$. Line 24 returns the result.

**Example 5.** *Given a community scoring metric such as clustering coefficient, the execution of Algorithm 3 on the graph in Figure 2 is as follows. We first compute the number of triangles and triplets in 3-core set by Line 7-13:* `triangle` $= 8$ *and* `triplet` $= 24$. *The clustering coefficient of 3-core is* $3 \times 8/24 = 1$. *Then, we incrementally compute the values for 2-core set. Line 7-12 detect two new triangles* $(v_5, v_6, v_3)$ *and* $(v_6, v_7, v_8)$, *so* `triangle` $= 8 + 2 = 10$. *Line 13 detects* $\binom{2}{2}$ *new triplets for* $v_5$, $\binom{4}{2}$ *for* $v_6$, $\binom{2}{2}$ *for* $v_7$, *and* $\binom{3}{2}$ *for* $v_8$. *Line 22 iterates over* `kshell_nbr`, *that is* $\{v_3, v_9\}$, *and counts* $\binom{2}{2} + 3 \times 2$ *new triplets for* $v_3$ *and* $3 \times 1$ *new triplets for* $v_9$. *To sum up,* `triplet` $= 24+21 = 45$. *The clustering coefficient of 2-core is* $3 \times 10/45 \approx 0.67$. *So, the 3-core set is preferred when the metric is clustering coefficient.*

**Correctness.** For every triangle $(v, u, w)$ in a $k$-core but not in a $(k+1)$-core, suppose $rank(v) < rank(u) < rank(w)$, $v$ is certainly in $k$-shell; otherwise, $(v, u, w)$ is in a $(k+1)$-core according to vertex rank, a contradiction. Every such triangle is counted exactly once in Line 7-12, since the visiting of $v$, $u$, and $w$ is unique according to their increasing vertex rank.

For every triplet $\langle u, v, w \rangle$ in a $k$-core but not in a $(k+1)$-core, when the center vertex $v$ is in the $k$-shell, Line 13 correctly counts such triplets. When $k = k_{max}$, $f_{\geq}[v]$ of every $v \in C_{k_{max}}$ is correctly generated by Line 17-19. For the next iteration $k = k_{max} - 1$, $f_{>}[v]$ of every $v \in C_{k_{max}}$ is correctly assigned by $f_{\geq}[v]$ from last iteration (Line 15-16). Then $f_{\geq}[u]$ of every $u \in C_{k_{max}-1}$ is correctly updated by visiting the neighbors of every $v \in H_k$. For every $k$, $f_{\geq}[v]$ and $f_{>}[v]$ for every $v \in C_{k+1}$ are correct by induction. For every $v \in$ `kshell_nbr`, the correctness of $gt\_k$, $eq\_k$ and `triplet` immediately follows. The community score is correct based on the correct primary values.

**Complexity.** (Space) The space complexity is dominated by the size of graph which takes $O(m)$. (Time.A) The time cost for triangle counting is $O(m^{1.5})$. We prove $N(z, >_r) \leq 2\sqrt{m}$, for every vertex $z$ visited in Line 7-12. If $deg(z) \leq \sqrt{m}$, then $|N(z, >_r)| \leq deg(z) \leq 2\sqrt{m}$. If $deg(z) > \sqrt{m}$, we prove by contradiction. Suppose $|N(z, >_r)| > 2\sqrt{m}$, for every vertex $v \in N(z, >_r)$, $|N(v, >_r)| \geq |N(z, >_r)| > 2\sqrt{m}$ since the vertex rank obeys degeneracy ordering. Then $\sum_{v \in N(z,>_r)} deg(v) \geq deg(z) \cdot |N(z, >_r)| > \sqrt{m} \cdot 2\sqrt{m} = 2m$, contradict with $\sum_{v \in N(z,>_r)} deg(v) \leq 2m$. So the time cost is $\sum_{v \in V}(|N(v, >_r)| \cdot deg(v)) \leq \sum_{v \in V}(2\sqrt{m} \cdot deg(v)) = 2\sqrt{m} \cdot m = O(m^{1.5})$. (Time.B) For triplet counting, Line 13 costs $O(n)$ time, as each vertex is visited once. Maintaining $f_{>}[\cdot]$ and $f_{\geq}[\cdot]$ costs $O(m)$, because each edge is visited at most three times (Line 14 and 18). So triplet counting costs $O(m)$ time. The time complexity of Algorithm 3 is $O(m^{1.5})$.

**Optimality.** The time complexity and space complexity of triangle counting are $O(m^{1.5})$ and $O(m)$, respectively [35]. For any input metric $Q$ requiring triangle counting, the worst-case complexity of Algorithm 3 is optimal.

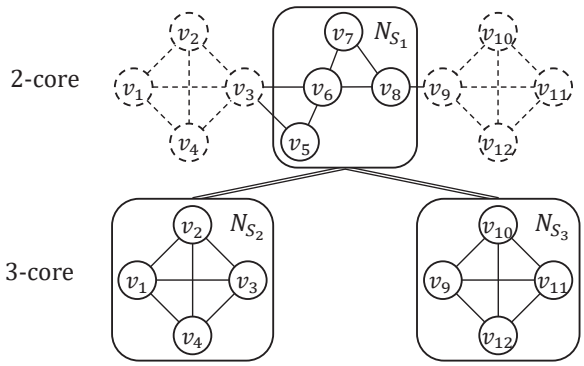



Fig. 4. A Core Forest (Tree)

## IV. Finding the Best Single k-Core

In this section, we firstly review the forest hierarchy of $k$-cores in Section IV-A. To find the best single $k$-core, we propose the baseline algorithm in Section IV-B and the improved algorithm in Section IV-C.

### A. Hierarchy of k-Cores

According to the definition of $k$-core, the following properties hold for every integer $k$:

- (Disjointness) every $k$-core is disjoint from each other in the same $k$-core set.
- (Containment) a $k$-core is contained by exactly one $(k-1)$-core.

The hierarchy of $k$-cores can be represented by a set of trees where each node is associated with a $k$-core. To illustrate the structure, we first define the $k$-core tree node.

**Definition 6** ($k$-Core Tree Node)**.** *Given a graph $G$, for each $k$-core $S$, there is a uniquely associated $k$-core tree node $N_S$ that contains the vertices in $S$ with coreness $k$, i.e., $S \cap H_k$, if $S \cap H_k$ is not empty.*

Note that the vertices in a $k$-core tree node are not certainly connected, because the $k$-shell vertices in a $k$-core may be separated by other vertices with higher coreness.

**Definition 7** (Parent Tree Node)**.** *Given a graph $G$, a $k_1$-core $S_1$ associated with $N_{S_1}$, and a $k_2$-core $S_2$ associated with $N_{S_2}$, the $k_1$-core tree node $N_{S_1}$ is the parent of the $k_2$-core tree node $N_{S_2}$, if (i) $k_1 < k_2$; (ii) $S_2 \subset S_1$; and (iii) any $k'$-core tree node with $k_1 < k' < k_2$ is not the parent of $N_{S_2}$.*

Given a $k$-core $S$ associated with $N_S$, the tree node contains vertices in the $k$-shell while leaving the rest in its descendants. Suppose $N_S$ has children $N_{C_1}, \cdots N_{C_c}$, the original $S$ can be reconstructed recursively by the vertices in $N_S$ and $\bigcup_{i=1}^{c} C_i$, if every $C_i$ has been reconstructed before.

**Definition 8** (Core Forest)**.** *Given a graph $G$, the core forest is constituted by all $k$-core tree nodes for every integer $k$ from 0 to $k_{max}$, where every tree node is connected to its parent if it exists.*

**Algorithm 4**: the LCPS algorithm

**Input** : a graph $G = (V, E)$, the coreness $c(v)$ of every vertex $v \in V$
**Output** : a forest of $k$-cores of $G$

```
1  X ← ∅;
2  bin ← a list of (k_max + 1) empty arrays;
3  for any u ∈ V \ X do
4      cur_p ← the root of a new tree, k ← 0;
5      bin[0] ← bin[0] ∪ {u};
6      while there exists non-empty array in bin do
7          r ← the largest r such that bin[r] is non-empty;
8          pop v from bin[r];
9          if v ∉ X then
10             if k > r then adjust cur_p so that k ← r;
11             if c(v) > r then adjust cur_p so that k ← c(v);
12             insert v into the tree node pointed by cur_p;
13             X ← X ∪ {v};
14             for each w ∈ N(v) \ X do
15                 p ← min{c(w), c(v)};
16                 bin[p] ← bin[p] ∪ {w};
17 return the forest
```

The above forest organizes all $k$-cores into a hierarchy, where each tree corresponds to a connected component of the original graph. The core forest can be stored in $O(n)$ space, because each vertex exists in exactly one tree node and each tree node has exactly one parent.

**Example 6.** *Figure 4 shows the core forest of Figure 2 which has only one tree. There are three tree nodes $N_{S_1}$, $N_{S_2}$, and $N_{S_3}$, associated with $S_1, S_2$, and $S_3$, respectively. $N_{S_1}$ is associated with the whole graph (a 2-core) while only the vertices in the 2-shell are contained in $N_{S_1}$. Although $N_{S_1}$ is also a 1-core, no 1-core tree node would be built according to our definition. We can reconstruct $S_1$ from the vertices in $N_{S_1}$, $S_2$, and $S_3$. Also, we have $|S_1| = |N_{S_1}|+|S_2|+|S_3|$, and $m(S_1) = m(N_{S_1}) + m(S_2) + m(S_3) + 3$ (boundary edges).*

**Forest Construction.** The state-of-the-art algorithm for constructing the forest of $k$-cores is proposed in [41], named LCPS (Level Component Priority Search). The time complexity of LCPS is $O(m)$, if a bucket data structure is applied in the implementation [48]. Since the LCPS is introduced at a high-level, we provide the pseudo-code in Algorithm 4 for LCPS to show more details.

To adapt LCPS for our need, three steps are required: (i) run Algorithm 4; (ii) compress tree structure, i.e., if a tree node stores no elements then we eliminate this node; and (iii) store all remaining tree nodes into an array $\mathcal{T}$ and sort them in descending order of the coreness of elements.

### B. Baseline Algorithm

Given a graph $G$ and a community scoring metric $Q$, a baseline solution is to firstly conduct core decomposition, and forest construction for fast retrieve of a $k$-core. Then, it computes the score of every $k$-core according to $Q$, for every integer $k$ from 0 to $k_{max}$.

By visiting the forest of $k$-cores, it takes $O(|V(S_i)|)$ time to retrieve the vertex set of every $k$-core $S_i$. Let $O(q_i)$ be the time cost to compute the score of $S_i$ given $V(S_i)$, $Q$ and $G$. The time complexity of the baseline algorithm is $O\left(\sum_{k=0}^{k_{max}} \sum_{S_i \in C_k} (q_i + |V(S_i)|)\right)$. Although the baseline runs in polynomial time, it is still costly to handle large graphs.

**Algorithm 5**: computing the best single $k$-core

**Input** : a graph $G$, a community scoring metric $Q$
**Output** : the best single $k$-core according to $Q$

```
1  metric ← [0, ..., 0];
2  pri_val ← [0, ..., 0];
3  compute the coreness of every vertex by core decomposition;
4  order G by Algorithm 1;
5  construct the forest T by Algorithm 4;
6  for each i from 0 to T.size() − 1 do
7      for each tv ∈ T.child do
8          pri_val[i] ← pri_val[i] + pri_val[tv];
9      for each v ∈ T.delta do
10         pri_val[i] ← pri_val[i] + impact of v;
11     metric[i] ←compute_metric(Q, pri_val[i]);
12 return the k-core with highest score in metric
```

### C. The Improved Algorithm

We apply the vertex ordering in Section III-B and the forest structure in Section IV-A to design the improved algorithm. The pseudo-code is shown in Algorithm 5. Array `metric` stores the score of each $k$-core. Array `pri_val` stores the primary values of each $k$-core. Here we use one variable `pri_val`$[i]$ to illustrate the computation of all the primary values at tree node $i$. Array $\mathcal{T}$ stores the tree nodes. $\mathcal{T}[i]$ is the $i^{th}$ element (a tree node) in $\mathcal{T}$, where $i$ is the id of $\mathcal{T}[i]$.

At Line 6, the algorithm processes each tree node in $\mathcal{T}$ sequentially, as the nodes in $\mathcal{T}$ represent all the $k$-cores for every integer $k$ from $k_{max}$ to 0. At Line 7-8, for each $k$-core, its primary values are incrementally computed, based on the values from the child nodes (Line 9-10) and the additional values caused by current tree node (Line 9-11). Line 11 computes the score according to the metric $Q$ and the primary values. Line 12 returns the best single $k$-core.

**Primary Values.** In Line 8, the basic primary values `in`, `out`, and `num` can be updated in three arrays, respectively, like `pri_val`. To compute these primary values, Line 10 can be replaced by Line 7-9 of Algorithm 2 where `in`, `out`, `num` are replaced by `in`$[i]$, `out`$[i]$, and `num`$[i]$, respectively. To compute the primary values `triangle` and `triplet`, we replace Line 10 by Line 7-22 of Algorithm 3 where `triangle`, `triplet` become `triangle`$[i]$ and `triplet`$[i]$, respectively.

**Correctness.** The correctness of Algorithm 5 is based on the correct computation of primary values for every $k$-core (as proved for Algorithm 2 and 3), and the correctness of forest construction [41].

**Complexity.** (Space) The space complexity is dominated by the size of graph, which is $O(m)$. (Time.A) For the primary values `in`, `out`, and `num`, the time complexity is the same to Algorithm 2, which is $O(n)$, because every vertex is visited for exactly one time and each visit costs $O(1)$ time. (Time.B)

TABLE III
STATISTICS OF DATASETS

| Dataset | $n$ | $m$ | $d_{avg}$ | $k_{max}$ |
|---|---|---|---|---|
| **A**stro-**P**h | 18,772 | 198,110 | 21.1 | 56 |
| **G**owalla | 196,591 | 950,327 | 9.7 | 51 |
| **D**BLP | 317,080 | 1,049,866 | 6.6 | 113 |
| **Y**outube | 1,134,890 | 2,987,624 | 5.3 | 51 |
| **A**s-**S**kitter | 1,696,415 | 11,095,298 | 13.1 | 111 |
| **L**ive**J**ournal | 3,997,962 | 34,681,189 | 17.4 | 360 |
| **H**ollywood | 1,069,126 | 56,306,653 | 105.3 | 2208 |
| **O**rkut | 3,072,441 | 117,185,083 | 76.3 | 253 |
| **H**uman-**J**ung | 784,262 | 267,844,669 | 683.1 | 1200 |
| **F**riend**S**ter | 65,608,366 | 1,806,067,135 | 55.1 | 304 |

TABLE IV
BEST $k$ FOR THE K-CORE (SET)

| Algo | AP | G | D | Y | AS | LJ | H | O | HJ | FS |
|---|---|---|---|---|---|---|---|---|---|---|
| CS-ad | 36 | 45 | 113 | 47 | 97 | 320 | 2208 | 229 | 1059 | 274 |
| CS-den | 56 | 51 | 113 | 51 | 111 | 360 | 2208 | 253 | 1200 | 304 |
| CS-cr | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 1 | 44 |
| CS-con | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CS-mod | 26 | 13 | 7 | 6 | 14 | 27 | 303 | 62 | 675 | 112 |
| CS-cc | 56 | 51 | 113 | 51 | 111 | 360 | 2208 | 253 | 1200 | 304 |
| C-ad | 36 | 45 | 64 | 47 | 90 | 194 | 2208 | 229 | 1058 | 274 |
| C-den | 56 | 8 | 113 | 5 | 3 | 89 | 2208 | 253 | 11 | 9 |
| C-cr | 17 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 5 | 1 |
| C-con | 17 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 5 | 61 |
| C-mod | 26 | 13 | 7 | 6 | 14 | 27 | 303 | 62 | 675 | 66 |
| C-cc | 56 | 8 | 113 | 5 | 3 | 89 | 2208 | 253 | 11 | 9 |

For the primary values `triangle` and `triplet`, the time complexity is the same to Algorithm 3 which is $O(m^{1.5})$.

**Optimality.** (i) For the primary values `in`, `out`, and `num`, since the time or space complexity of computing an existing community metric on $G$ is at least $O(m)$ [11], the complexity of Algorithm 5 is optimal. (ii) For the primary values `triangle` and `triplet`, the time complexity and space complexity of triangle counting are $O(m^{1.5})$ and $O(m)$, respectively [35]. For any input metric $Q$ requiring triangle counting, the worst-case complexity of Algorithm 5 is optimal.

## V. EXPERIMENTAL EVALUATION

In this section, extensive experiments are conducted to verify the effectiveness and the efficiency of the algorithms.

**Datasets.** In the experiments, we use 10 public real-world networks from different areas including collaboration networks, Internet topology, brain networks, and social networks. Hollywood and Human-Jung are from http://networkrepository.com, and the others are from http://snap.stanford.edu. The details of the data are shown in Table III, ordered by the number of edges, where $d_{avg}$ is the average degree and $k_{max}$ is the largest vertex coreness.

**Metrics.** Our experiments adopt six representative community scoring metrics introduced in Section II-C, i.e., average degree, density, cut ratio, conductance, modularity, and clustering coefficient. For conciseness, we abbreviate them to `ad`, `den`, `cr`, `con`, `mod`, and `cc`, respectively.

**Algorithms.** For each of the 6 community metrics, we implement both the baselines (Section III-A and IV-B) and the improved algorithms (Algorithm 2, 3 and 5), for both finding the best $k$-core set and finding the best single $k$-core. The vertex ordering procedure (Algorithm 1) and the forest construction (Algorithm 4) are included in the improved algorithms.

**Environment.** We perform our experiments on a CentOS Linux server (Release 7.5.1804) with Quad-Core Intel Xeon CPU (E5-2630 v4 @ 2.20GHz), and 128G memory. All the algorithms are implemented in C++. The source code is compiled by GCC (7.3.0) under O3 optimization.

### A. Community Quality by Different $k$

**Scores of $k$-Core Sets.** Figure 5 shows the scores of the $k$-core set for different $k$, regarding different community metrics. According to the trends of score on different $k$, using trivial values for $k$ will often find low-quality $k$-core sets. Choosing a user-defined $k$ value (e.g., 100) is not promising. Besides, using a value based on dataset statistics (e.g., average degree or $k_{max}$) does not necessarily produce a high-quality $k$-core set. Very small $k$ values are often preferred for the metric of cut ratio or conductance, because they only measure the interconnection among different $k$-cores. Thus, they may not be used solely for finding the best $k$-core set.

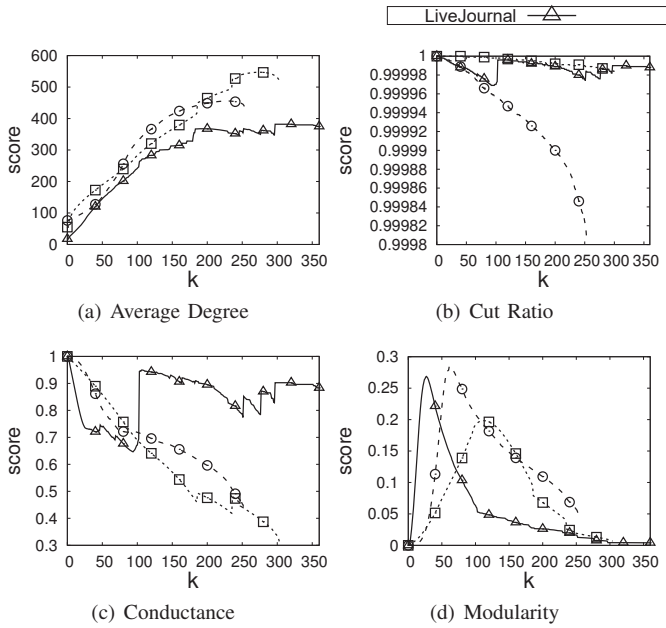
**Scores of $k$-Cores.** Figure 6 shows the scores of all the $k$-cores, where $c$ in the $x$ axis is the sequence id of a $k$-core in the order that ranks all the $k$-cores by ascending values of $k$ with ties broken by ascending scores of the $k$-cores. The $y$-axis is the community score of a $k$-core. The figure presents a finer granularity of score distribution in core decomposition on different metrics. To clearly show the result, the trends are depicted by the average score of every 20 (resp. 5) consecutive $k$-cores in LiveJournal (resp. Orkut and Friendster). The trends of score imply that many high-score $k$-cores come from relatively low-score $k$-core sets. Similar to finding the best $k$-core set, considering only the cross-connections of $k$-cores (e.g., cut ratio and conductance) may not be effective since extremely small $k$ values are preferred by these metrics. We may consider to use a combination of different metrics to find the high-quality $k$-cores.

**Best $k$ Values.** Table IV shows the best $k$ identified by finding the best $k$-core set and the best single $k$-core, respectively. We use `CS-` to represent the result of the best $k$-core set, e.g., `CS-ad` is the result by average degree. We use `C-` to represent the result for the best $k$-core, e.g., `C-cc` is the result by clustering coefficient. The largest $k$ is recorded if multiple values of $k$ are the best on a metric.

The result shows different community metrics have different preferences on the best value of $k$. For $k$-core set, `ad`, `den` and `cc` prefer large values of $k$ for high cohesiveness, while `cr` and `con` give high scores when $k$ is small for sparse cross-connection. Some metrics choose an extreme value of $k$ in core decomposition, which may imply to use a combination of these metrics. The `mod` metric chooses moderate values of $k$ considering both inner and outer connection of the $k$-cores.

### B. Case Study on High Score $k$-Cores

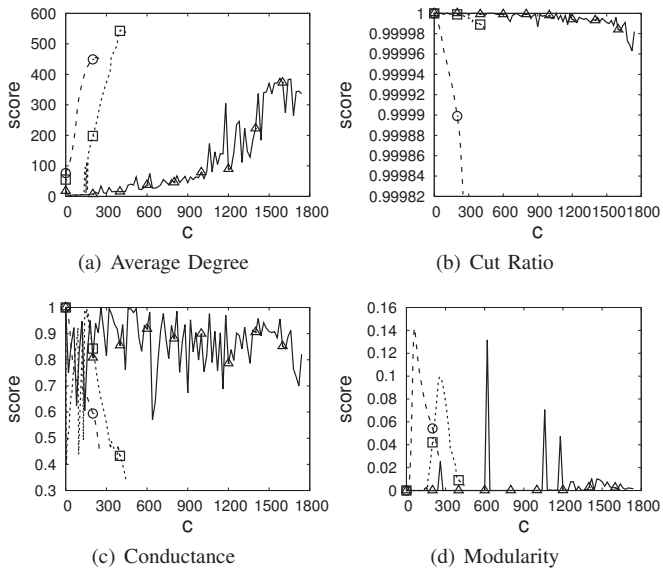
By adopting different community scoring metrics, our algorithms find different high-quality $k$-cores in DBLP dataset.




(a) Average Degree (b) Cut Ratio

(c) Conductance (d) Modularity

Fig. 5. Scores of Every $k$-Core Set




(a) Average Degree (b) Cut Ratio

(c) Conductance (d) Modularity

Fig. 6. Score of Every $k$-Core

TABLE V
COMMUNITY A ($k = 17$)

| | | |
|---|---|---|
| Vijay Gadepally | William Arcand | Andrew Prout |
| Charles Yee | Michael Jones | Albert Reuther |
| Anna Klein | Matthew Hubbell | Jeremy Kepner |
| Bill Bergeron | David Bestor | Julie Mullen |
| Antonio Rosa | Chansup Byun | Peter Michaleas |
| Lauren Milechin | Siddharth Samsi | Michael Houle |

TABLE VI
COMMUNITY B ($k = 9$)

| | | |
|---|---|---|
| Danyang Zhao | Xianyi Wang | Congliang Liu |
| Cheng Liu | Yueqiang Sun | Qifei Du |
| Dongwei Wang | Yuerong Cai | Chunjun Wu |
| Weihua Bai | Junming Xia | Xiangguang Meng |

TABLE VII
SCORES OF DETECTED COMMUNITIES

| ID | ad | den | cc | cr | con |
|---|---|---|---|---|---|
| A | 17.0 | 1.0 | 1.0 | 0.999998 | 0.9871 |
| B | 10.67 | 0.9697 | 0.971 | 1.0 | 1.0 |

Table V shows community A (a 17-core) in which the members are mainly from Lincoln Laboratory Supercomputing Center, MIT. This community is identified as the best by the score of average degree, internal density, and clustering coefficient, respectively, as shown in Table VII. The members of community A are highly collaborated in research, as reflected by the co-authored papers: they have 6 co-authored papers by all the members, 15 co-authored papers by at least 17 members, and 20 co-authored papers by at least 16 members.

Table VI shows community B (a 9-core) in which the members are from National Space Science Center, Chinese Academy of Sciences. Community B is relatively isolated from the other authors, according to its highest cut ratio and conductance in Table VII. The members in Community B are also tightly connected: there are 8 co-authored papers by all the members, and 13 co-authored papers by at least 11 members.

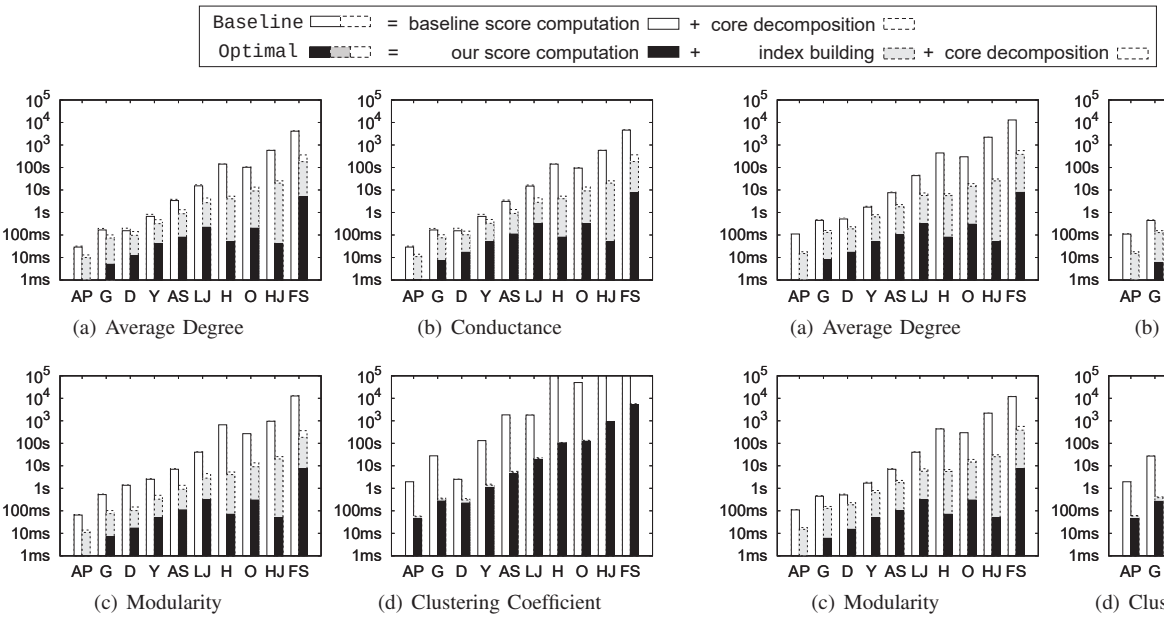
### C. Runtime of Algorithms

**Finding the Best $k$ Value.** In Figure 7, `Baseline` is the baseline algorithm proposed in Section III-A which contains score computation and core decomposition [7]. `Optimal` is the improved algorithm (Algorithm 2 or 3) which contains score computation, core decomposition, and index building, i.e., the vertex ordering proposed in Section III-B.

Figure 7 shows that `Optimal` is significantly faster than `Baseline` on all the datasets. For each dataset, index building and core decomposition only need to be executed by one time, while score computation can be run for many times given the different community metrics. The margins become larger (1-4 orders of magnitude) if we do not count index building time in `Optimal`. On Hollywood, Human-Jung, and FriendSter, the baseline cannot finish within $10^5$ seconds for clustering coefficient. As analyzed in Section III-C, the time complexity of score computation is determined by the number of vertices for 5 metrics, which exactly matches the trends in Figure 7. The runtimes on density and cur ratio are almost same to that on average degree.

**Finding the Best Single $k$-Core.** In Figure 8, `Baseline` is the baseline algorithm proposed in Section IV-B, `Optimal` is the improved algorithm (Algorithm IV-C) where index building contains the vertex ordering proposed in Section III-B and the forest construction introduced in Section IV-A.

Figure 7 shows that `Optimal` still largely outperforms `Baseline`. The trends of runtime are very similar to Figure 7, except that the runtime of the algorithms becomes larger due to the computation of $k$-core connectivity. On Hollywood, Human-Jung, and FriendSter, the baseline cannot finish in $10^5$ seconds for clustering coefficient. The runtime results on density and cur ratio are almost same to the results on average degree. Overall, `Optimal` outperforms `Baseline` by 1-4 orders of magnitude in runtime on different datasets.




(a) Average Degree (b) Conductance

(c) Modularity (d) Clustering Coefficient

Fig. 7. Performance of Finding the Best $k$-Core Set




(a) Average Degree (b) Conductance

(c) Modularity (d) Clustering Coefficient

Fig. 8. Performance of Finding the Best Single $k$-Core

### *D. Application on Other Problems*

In this experiment, we investigate the application of our algorithm on 3 problems related to $k$-core, i.e., finding densest subgraph, maximum clique and size-constrained $k$-core. Note that the above problems are all NP-hard. In the following, we show that our algorithm can serve as better approximate solutions compared with the state-of-the-art, or produce useful intermediate results. Let `Opt-D` denote our algorithm which returns the best $k$-core regarding average degree (Algorithm 5). Let $S^*$ denote the output of `Opt-D`.

**Densest Subgraph.** The densest subgraph (DS) problem is to find the subgraph with the largest average vertex degree [26]. Our `Opt-D` produces a $\frac{1}{2}$-approximate solution for DS problem, because the $k_{max}$-core is one of our candidate results, which is a $\frac{1}{2}$-approximate solution [26]. We compare `Opt-D` with the recent approximate solution named `CoreApp` [26] which is the fastest algorithm for DS problem. As shown in Table VIII, `Opt-D` outperforms `CoreApp` in both output quality (average degree) and runtime on most datasets. Note that the better values in Table VIII are marked in bold.

**Maximum Clique.** The maximum clique (MC) problem is to find the largest subset of vertices such that every pair of vertices in the subset is adjacent [12]. As shown in Table VIII, it is likely that $S^*$ contains the maximum clique (6 out of 10 datasets), although $S^*$ is not large (the proportion of $S^*$ in the whole vertex set is often within 1%). This finding can benefit the algorithm design for MC problem.

**Size-Constrained $k$-Core.** Given an integer $k$, an integer $h$, and a query vertex $v$, the query of size-constrained $k$-core (SCK) is to find a $k$-core of size $h$ which contains $v$. Based on the average degree of every $k$-core computed by `Opt-D`, the algorithm `Opt-SC` first selects the $k'$-core $S'$ that has the highest average degree, where (i) $k' \geq k$ (ii) $S'$ contains $v$; and (iii) $|V(S')| \geq h$. Then, `Opt-SC` peels $S'$ to find more results ($k$-cores) until $|V(S')| \leq h$: in each step of the peeling, it removes the vertex with the lowest degree (skip $v$) and the vertices with degree less than $k$ in $S'$.

TABLE VIII
PERFORMANCE OF `Opt-D` ON DENSEST SUBGRAPH & MAXIMUM CLIQUE

| Dataset | `CoreApp` | | `Opt-D` | | `Opt-D` (output $S^*$) | |
|---|---|---|---|---|---|---|
| | $d_{avg}$ | time (s) | $d_{avg}$ | time (s) | $MC \subseteq S^*$ | $\lvert S^*\rvert/n$ |
| AP | 56.035 | 0.11 | **58.923** | **0.02** | ✓ | 7.87% |
| G | 76 | 0.195 | **87.593** | **0.093** | | 0.28% |
| D | 113 | 0.233 | **113.13** | **0.138** | ✓ | 0.04% |
| Y | 86.066 | 0.594 | **91.1** | **0.498** | ✓ | 0.18% |
| AS | 150.018 | **1.145** | **178.801** | 1.374 | | 0.03% |
| LJ | 374.71 | 4.943 | **387.027** | **4.832** | ✓ | 0.01% |
| H | **2208** | **3.002** | **2208** | 3.635 | ✓ | 0.21% |
| O | 438.64 | 20.14 | **455.732** | **11.72** | | 0.85% |
| HJ | 2013.879 | 15.272 | **2114.915** | **14.457** | ✓ | 1.15% |
| FS | 513.852 | 1041.528 | **547.035** | **836.279** | | 0.08% |

**$S^*$ is the output of `Opt-D`. $d_{avg}$ is the average degree of the output. $MC \subseteq S^*$ means the maximum clique is contained in $S^*$. $|S^*|/n$ is the vertex proportion of $S^*$ in the whole graph.**

TABLE IX
PERFORMANCE OF `Opt-SC` ON SIZE-CONSTRAINED $k$-CORE (DBLP)

| $c(v)$ | $k=10$ | $k=15$ | $k=20$ | $k=30$ | $k=40$ |
|---|---|---|---|---|---|
| 30 | 96.45% | 88.31% | 76.21% | 20.97% | / |
| 43 | 97.46% | 91.41% | 82.10% | 37.87% | 6.69% |
| 51 | 99.12% | 96.75% | 92.64% | 49.81% | 30.77% |
| 64 | 98.61% | 95.15% | 89.62% | 61.88% | 57.86% |
| 113 | 98.61% | 95.15% | 89.62% | 61.88% | 57.86% |

**Given a random query point $v$ with coreness $c(v)$, the percentage that `Opt-SC` returns a $k$-core with at most 5% size deviation to $h$.**

Table IX shows the performance of `Opt-SC` on DBLP. We say `Opt-SC` hits a query if the returned $k$-core contains $v$ and has at most 5% size deviation to $h$. `Opt-SC` is very likely to hit the query when $c(v)$ is larger than $k$. Note that some coreness values do not exist, i.e., no vertex has such coreness. As the time complexity of `Opt-SC` is linear to graph size, it can benefit the algorithm design for SCK problem.

## VI. Discussions and Future Work

In this section, we explore several future extensions of our proposed models and algorithms.

### A. Other Community Metrics

There are various existing community scoring metrics and many potential new metrics by combining the existing ones or adopting new structure properties. Our algorithms can handle most community metrics based on the studied 5 primary values in Section II-C. A survey of community metrics [11] organizes and reviews many existing metrics where the majority (suitable for core decomposition) are based on the above 5 primary values. The exceptions include some metrics based on higher-order motifs [5], some metrics for overlapped communities, etc. They are interesting for further study.

### B. Other Hierarchical Decompositions

If a cohesive subgraph model with input $k$ has the containment property, i.e., the $(k+1)$-subgraph is always a subgraph of the $k$-subgraph, our algorithm for finding the best $k$ may be applied to find the best $k$. When the forest structure of a hierarchical decomposition is similar to that of core decomposition in Section IV-A, our algorithm for finding the best single $k$-core is also applicable.

For instance, our algorithms can shed light on the solutions on the decomposition of $k$-truss. To find the best $k$-truss set, we may rank the incident edges of every vertex by their truss numbers and record some position tags, to facilitate the incremental score computation. Given the score of $(k+1)$-truss set $S$, to compute the score for $k$-truss set, we can visit the vertices in $k$-truss set but not in $S$, and additional vertices incident to edges with truss number $k$. The solution for computing the best single $k$-truss can be derived similarly, while designing an optimal solution is still challenging.

## VII. Related Work

Diverse models of cohesive subgraph are proposed to accommodate different scenarios, for example, clique [17], quasi-clique [46], $k$-core [7], [33], [49], $k$-truss [19], [31], [54], $k$-plex [56], and $k$-ecc [13], [66]. Some cohesive subgraph models decompose a graph into hierarchical structure, e.g., core decomposition [42], [59], truss decomposition [50], [54], [65], and ecc decomposition [13], [62]. Core decomposition is one of the most well-studied models, due to its effectiveness in various applications including community discovery [15], [16], [28], [37], [38], influential spreader identification [24], [34], [39], [40], network analysis [4], [21], [30], [51], anomaly detection [51], evaluating contagion power of vertices [34], [53], and graph visualization [3], [20], [65].

An $O(m)$ time in-memory algorithm for core decomposition is proposed in [7]. The construction of core forest can also be computed in $O(m)$ time [41]. The core decomposition under distributed configuration is introduced in [42]. An I/O efficient algorithm for core decomposition is proposed in [59]. Various variants of $k$-core are explored, including $(k,r)$-core [63], diversified coherent $k$-core [67], and skyline $k$-core [37].

The model of $k$-core is extended to weighted graphs where each edge has its weight and each vertex has its weighted degree, as introduced in [23], [27], [58]. The weighted core decomposition can be applied to evaluate the cooperation in $k$-core communities [29] and identify influential spreaders [1]. The techniques for $k$-core computation and decomposition cannot be applied to solve our problem, because the prior works only aim to find all the $k$-core structures and do not compute the score of any $k$-core. Our algorithm may shed light on finding the best $k$-core on weighted graphs if we apply the weighted community scores. For the problem of extracting a dense subgraph, there is a solution that aims to find the subgraph $S$ such that $f_\alpha(S) = m(S) - \alpha\binom{n(S)}{2}$ is maximized, while the subgraph $S$ is not guaranteed to be a $k$-core, and the techniques cannot be used to solve our problems [52].

Metrics for community evaluation are surveyed in [11], [61] such as modularity [9], [44] and clustering coefficient [32], [47]. Community scoring metrics can be used to effectively compare the communities produced by different algorithms, and formalize the notion of network communities [36]. Different communities can be found by the optimization guided by different community metrics, e.g., [2], [14], [18], [60]. To the best of our knowledge, this paper is the first attempt to apply community scoring metrics to compute the quality of $k$-cores and other cohesive subgraph models such as $k$-truss.

## VIII. Conclusion

In this paper, we study the problems of computing the best $k$-core set and the best single $k$-core with respect to a community scoring metric. Since there are various metrics considering different standards, we focus on 5 common primary values to handle different metrics. Benefitting from a light-weight vertex ordering procedure with $O(m)$ time complexity, we propose time and space optimal algorithms for the studied problems. We conduct extensive experiments on 10 real-world graphs with size up to billion-scale, where our algorithms not only significantly outperform the baselines in runtime but also provide profound insights for the related problems.


## Acknowledgments

Xuemin Lin is supported by 2018YFB1003504, NSFC6123 2006, ARC DP180103096 and DP170101628. Wenjie Zhang is supported by ARC DP180103096. Ying Zhang is supported by ARC DP180103096 and FT170100128.


## References


[1] M. A. Al-garadi, K. D. Varathan, and S. D. Ravana. Identification of influential spreaders in online social networks using interaction weighted k-core decomposition method. *Physica A*, 468:278–288, 2017.

[2] D. Aloise, G. Caporossi, P. Hansen, L. Liberti, S. Perron, and M. Ruiz. Modularity maximization in networks by variable neighborhood search. In *Graph Partitioning and Graph Clustering*, pages 113–128, 2012.

[3] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *NIPS*, pages 41–50, 2005.

[4] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *NHM*, 3(2):371–393, 2008.

[5] A. Arenas, A. Fernandez, S. Fortunato, and S. Gomez. Motif-based communities in complex networks. *Journal of Physics A: Mathematical and Theoretical*, 41(22):224001, 2008.

[6] G. D. Bader and C. W. V. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4:2, 2003.

[7] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[8] M. Bola and B. A. Sabel. Dynamic reorganization of brain functional networks during cognition. *Neuroimage*, 114:398–413, 2015.

[9] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity clustering. *TKDE*, 20(2):172–188, 2008.

[10] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of internet topology using k-shell decomposition. *PNAS*, 104(27):11150–11154, 2007.

[11] T. Chakraborty, A. Dalmia, A. Mukherjee, and N. Ganguly. Metrics for community analysis: A survey. *ACM Comput. Surv.*, 50(4):54:1–54:37, 2017.

[12] L. Chang. Efficient maximum clique computation over large sparse graphs. In *KDD*, pages 529–538, 2019.

[13] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang. Efficiently computing k-edge connected components via graph decomposition. In *SIGMOD*, pages 205–216, 2013.

[14] C. Chen, R. Peng, L. Ying, and H. Tong. Network connectivity optimization: Fundamental limits and effective algorithms. In *KDD*, pages 1167–1176, 2018.

[15] L. Chen, C. Liu, K. Liao, J. Li, and R. Zhou. Contextual community search over large social networks. In *ICDE*, pages 88–99, 2019.

[16] L. Chen, C. Liu, R. Zhou, J. Li, X. Yang, and B. Wang. Maximum co-located community search in large scale social networks. *PVLDB*, 11(10):1233–1246, 2018.

[17] J. Cheng, Y. Ke, A. W. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks. *ACM Trans. Database Syst.*, 36(4):21:1–21:34, 2011.

[18] A. Clauset. Finding local community structure in networks. *Physical review E*, 72(2):026132, 2005.

[19] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 16:3–1, 2008.

[20] P. Colomer-de-Simon, M. Á. Serrano, M. G. Beiró, J. I. Alvarez-Hamelin, and M. Boguñá. Deciphering the global organization of clustering in real complex networks. *CoRR*, abs/1306.0112, 2013.

[21] M. Daianu, N. Jahanshad, T. M. Nir, A. W. Toga, C. R. J. Jr., M. W. Weiner, and P. M. Thompson. Breakdown of brain connectivity between normal aging and alzheimer's disease: A structural *k*-core network analysis. *Brain Connectivity*, 3(4):407–422, 2013.

[22] Y. Dourisboure, F. Geraci, and M. Pellegrini. Extraction and classification of dense implicit communities in the web graph. *TWEB*, 3(2):7:1–7:36, 2009.

[23] M. Eidsaa and E. Almaas. S-core network decomposition: A generalization of k-core analysis to weighted networks. *Physical Review E*, 88(6):062819, 2013.

[24] S. Elsharkawy, G. Hassan, T. Nabhan, and M. Roushdy. Effectiveness of the k-core nodes as seeds for influence maximisation in dynamic cascades. *International Journal of Computers*, 2, 2017.

[25] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *PVLDB*, 10(6):709–720, 2017.

[26] Y. Fang, K. Yu, R. Cheng, L. V. S. Lakshmanan, and X. Lin. Efficient algorithms for densest subgraph discovery. *PVLDB*, 12(11):1719–1732, 2019.

[27] A. Garas, F. Schweitzer, and S. Havlin. A k-shell decomposition method for weighted networks. *New Journal of Physics*, 14(8):083030, 2012.

[28] C. Giatsidis, F. D. Malliaros, D. M. Thilikos, and M. Vazirgiannis. Corecluster: A degeneracy based graph clustering framework. In *AAAI*, pages 44–50, 2014.

[29] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. Evaluating cooperation in communities with the k-core structure. In *ASONAM*, pages 87–93, 2011.

[30] L. Hébert-Dufresne, A. Allard, J.-G. Young, and L. J. Dubé. Percolation on random networks with arbitrary k-core structure. *Physical Review E*, 88(6):062820, 2013.

[31] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[32] L. Katzir and S. J. Hardiman. Estimating clustering coefficients and size of social networks via random walk. *TWEB*, 9(4):19:1–19:20, 2015.

[33] W. Khaouid, M. Barsky, S. Venkatesh, and A. Thomo. K-core decomposition of large networks on a single PC. *PVLDB*, 9(1):13–23, 2015.

[34] 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, 2010.

[35] M. Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.*, 407(1-3):458–473, 2008.

[36] J. Leskovec, K. J. Lang, and M. W. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640, 2010.

[37] R. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng. Skyline community search in multi-valued networks. In *SIGMOD*, pages 457–472, 2018.

[38] R. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai. Persistent community search in temporal networks. In *ICDE*, pages 797–808, 2018.

[39] J.-H. Lin, Q. Guo, W.-Z. Dong, L.-Y. Tang, and J.-G. Liu. Identifying the node spreading influence with largest k-core values. *Physics Letters A*, 378(45):3279–3284, 2014.

[40] F. D. Malliaros, M.-E. G. Rossi, and M. Vazirgiannis. Locating influential nodes in complex networks. *Scientific reports*, 6:19307, 2016.

[41] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983.

[42] A. Montresor, F. D. Pellegrini, and D. Miorandi. Distributed k-core decomposition. *IEEE TPDS*, 24(2):288–300, 2013.

[43] 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.

[44] M. E. Newman. Modularity and community structure in networks. *PNAS*, 103(23):8577–8582, 2006.

[45] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[46] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *KDD*, pages 228–238, 2005.

[47] J. Saramäki, M. Kivelä, J.-P. Onnela, K. Kaski, and J. Kertesz. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75(2):027105, 2007.

[48] A. E. Sariyüce and A. Pinar. Fast hierarchy construction for dense subgraphs. *PVLDB*, 10(3):97–108, 2016.

[49] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[50] Y. Shao, L. Chen, and B. Cui. Efficient cohesive subgraphs detection in parallel. In *SIGMOD*, pages 613–624, 2014.

[51] 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.

[52] C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A. Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *KDD*, pages 104–112, 2013.

[53] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *PNAS*, 109(16):5962–5966, 2012.

[54] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[55] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin. Efficient computing of radius-bounded k-cores. In *ICDE*, pages 233–244, 2018.

[56] Y. Wang, X. Jian, Z. Yang, and J. Li. Query optimal k-plex based community in graphs. *DSE*, 2(4):257–273, 2017.

[57] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world'networks. *Nature*, 393(6684):440, 1998.

[58] B. Wei, J. Liu, D. Wei, C. Gao, and Y. Deng. Weighted k-shell decomposition for complex networks based on potential edge weights. *Physica A*, 420:277–283, 2015.

[59] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. X. Yu. I/O efficient core graph decomposition at web scale. In *ICDE*, pages 133–144, 2016.

[60] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: On free rider effect and its elimination. *PVLDB*, 8(7):798–809, 2015.

[61] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowl. Inf. Syst.*, 42(1):181–213, 2015.

[62] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang. I/O efficient ECC graph decomposition via graph reduction. *VLDB J.*, 26(2):275–300, 2017.

[63] 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.

[64] H. Zhang, H. Zhao, W. Cai, J. Liu, and W. Zhou. Using the k-core decomposition to analyze the static structure of large-scale software systems. *The Journal of Supercomputing*, 53(2):352–369, 2010.

[65] F. Zhao and A. K. H. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *PVLDB*, 6(2):85–96, 2012.

[66] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li. Finding maximal k-edge-connected subgraphs from a large graph. In *EDBT*, pages 480–491, 2012.

[67] R. Zhu, Z. Zou, and J. Li. Diversified coherent core search on multi-layer graphs. In *ICDE*, pages 701–712, 2018.