OLAK: An Efficient Algorithm to Prevent Unraveling in Social Networks

Fan Zhang[†], Wenjie Zhang[§], Ying Zhang[†], Lu Qin[†], Xuemin Lin[§]

[†]*CAI, University of Technology Sydney*, [§]*University of New South Wales*

fanzhang.cs@gmail.com, {ying.zhang, lu.qin}@uts.edu.au, {zhangw, lxue}@cse.unsw.edu.au

ABSTRACT

In this paper, we study the problem of the anchored k-core. Given a graph G, an integer k and a budget b, we aim to identify b vertices in G so that we can determine the largest induced subgraph J in which every vertex, except the b vertices, has at least k neighbors in J. This problem was introduced by Bhawalkar and Kleinberg et al. in the context of user engagement in social networks, where a user may leave a community if he/she has less than k friends engaged. The problem has been shown to be NP-hard and inapproximable. A polynomial-time algorithm for graphs with bounded tree-width has been proposed. However, this assumption usually does not hold in real-life graphs, and their techniques cannot be extended to handle general graphs.

Motivated by this, we propose an efficient algorithm, namely onion-layer based anchored k-core (OLAK), for the anchored k-core problem on large scale graphs. To facilitate computation of the anchored k-core, we design an *onion layer* structure, which is generated by a simple onion-peeling-like algorithm against a small set of vertices in the graph. We show that computation of the best anchor can simply be conducted upon the vertices on the *onion layers*, which significantly reduces the search space. Based on the well-organized layer structure, we develop efficient candidates exploration, early termination and pruning techniques to further speed up computation. Comprehensive experiments on 10 real-life graphs demonstrate the effectiveness and efficiency of our proposed methods.

1. INTRODUCTION

In social networks, where vertices represent individuals and edges represent friendships, the behavior of an individual may be influenced by that of his/her friends. In recent years, user engagement on social networks has been studied in the literature (e.g., [7, 10, 11, 12, 13, 19, 27]) to model the behavior of users where each user may choose to remain engaged in, or leave, a group or a community. In a basic model, a user will remain engaged if, and only if, at least k of his/her friends are engaged. A user with less than k friends engaged will leave. His/her departure may be contagious and form a cascade of departures in the network. This procedure is called *network unraveling* in which active individuals may leave by the negative influence of his/her friends. As shown in [7], the unraveling process stops when the remaining engaged individuals correspond to the k-core of the network, a well-known concept in graph theory, which is the maximal induced subgraph in which *every* vertex has at least k neighbors.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.

Proceedings of the VLDB Endowment, Vol. 10, No. 6

Copyright 2017 VLDB Endowment 2150-8097/17/02.

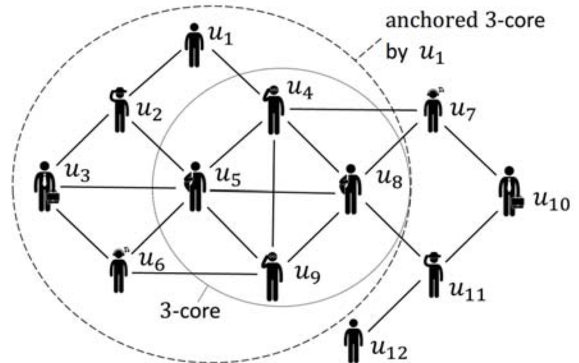


Figure 1: Motivating Example

To prevent unraveling in social networks, Bhawalkar and Kleinberg et al. [7] formally introduce the problem of anchored k-core. The aim is to retain (anchor) some users with incentives to ensure they will not leave regardless of the behavior of others, so that the largest number of users will remain engaged when the unraveling stops. Formally speaking, it is to anchor a set of b vertices such that the induced k-core is the largest one. This problem has a wide range of applications, and can help users identify critical vertices (e.g., people) whose participation is critical to overall engagement of the networks. Below is a motivating example.

Example 1. *Suppose there is a computer science study group, and the number of friends in the group represents the willingness of a member to engage in the collaborative learning. If one leaves, he/she will weaken the willingness of his/her friends to remain engaged, which may incur the unraveling of the study group. As illustrated in Figure 1, we model 12 members in a study group and their relationship as a network. According to the above engagement model with k=3, i.e., a person will leave if there are less than three friends, four members will remain engaged eventually; that is, 3-core of the network includes u_4, u_5, u_8 and u_9. To prevent unraveling, we may persuade the member u_1 not to leave through additional incentives, such as a regular personal tutoring or priority booking of the study room . As a result, members u_2, u_3 and u_6 will also remain engaged since each of them now has three friends in the study group. This motivates us to find the most cost effective way to "anchor" a set of b members so that the size of the resulting k-core*

is maximized. It turns out that the optimal solution is $\{u_1\}$ *and* $\{u_1, u_{10}\}$ *for* $b = 1$ *and* $b = 2$*, respectively.*

Challenges. It is shown in [7] that the anchored k-core problem is NP-hard and inapproximable when $k \geq 3$. Two following works show the problem is also NP-hard even on a planar graph [10, 11]. A polynomial-time algorithm in graphs with bounded tree-width was proposed in [7], but to the best of our knowledge there is no practical algorithm for general large-scale graphs.

To avoid enumerating all possible anchor sets with size b, we resort to greedy heuristics, where the best anchor is calculated in each iteration by computing the k-core for each possible anchor vertex. As demonstrated in our empirical study, a straightforward implementation of the greedy algorithm is very time consuming. For instance, it would take more than a month to find one best anchor on the medium size network `Yelp` with $552,339$ vertices and $1,781,908$ edges. The reasons are two-fold. (1) The large number of candidate anchors. It is clear that we do not need to anchor the vertices in the k-core of the graph. However, the number of remaining vertices is still large. (2) Although the computing time of k-core is linear in the number of edges, the cost is expensive given the large number of candidate anchors.

Our Solution. To address the above issues, we design an auxiliary structure $\mathcal{L}$, called *onion layers*, to maintain a small set of vertices and develop corresponding efficient techniques to significantly reduce the search space.

Due to the existence of anchor vertices, some new vertices will join k-core, which are termed **followers** in this paper. The number of followers is the *gain* of the anchoring activity. As we also adopt the greedy heuristics, our research focuses on finding the best anchor in the graph, i.e., the vertex with the largest number of followers. We observe that when we only consider one anchor, all followers must reside on the (k-1)-shell, i.e., the vertices in (k-1)-core but not in k-core. We put these vertices and their neighbors into *onion layers* $\mathcal{L}$ and show that we only need to consider the vertices in $\mathcal{L}$ as the candidate anchors to find the best anchor vertex. By doing so, the number of candidate anchors is significantly reduced. More importantly, all of the follower computations for finding the best anchor are restricted to the vertices in $\mathcal{L}$, which significantly reduces the search space.

We enhance the computation of k-core for a given anchor by imposing the layer structure on $\mathcal{L}$ and develop an efficient algorithm to quickly find its followers. The key idea is that, considering we will try a large number of candidate anchors, it is worthwhile to partition $\mathcal{L}$ into several layers in each iteration of the greedy algorithm so that the unraveling procedure can be conducted following a layer-by-layer paradigm. Thanks to the flexibility of the deletion order in k-core computation, i.e., the leave order can be arbitrary as long as each vertex has less than k neighbors when it quits. we formally prove that our layer-by-layer computation can always produce the correct results. By using the well-organized layer structure $\mathcal{L}$, we can effectively identify the candidate followers, and develop early termination and candidate anchor pruning techniques to eliminate non-promising followers and anchors at an early stage.

Contributions. Our principal contributions are summarized as follows.

- We develop the first efficient algorithm, `OLAK`, to solve the anchored k-core problem on general large graphs.

Table 1: Summary of Notations

Notation	Definition
G	an unweighted and undirected graph
u, v, x	vertex in the graph
n, m	the number of vertices and edges in G
A	a set of anchor vertices
$NB(u, G)$	the set of adjacent vertices of u in G
$deg(u, G)$	$\lvert NB(u, G)\rvert$ if $u \notin A$; $+\infty$ if $u \in A$
G_A (G_x)	graph G anchored by A (x)
$\mathtt{cn}(u)$	core number of the vertex u
k	the degree constraint
b	the budget for the number of anchors
$C_k(G)$, $S_k(G)$	k-core and k-shell of G
$\mathcal{L}$ (i.e., L_0^s)	*onion layers* of G (with $s+1$ layers)
L_i	vertices on i-th layer of $\mathcal{L}$
L_i^j	$\bigcup_{i \leq k \leq j} L_k$
$l(u)$	layer index of the vertex u in $\mathcal{L}$
$\mathcal{F}(x)$ ($\mathcal{F}(A)$)	followers of an anchor x (a set A of anchors)
$CF(x)$	the candidate followers of an anchor x
$d^+(u)$	degree upper bound of u in $C_k(G_x)$

- We introduce a novel *onion layer* structure $\mathcal{L}$, which contains a small set of vertices, so that we can efficiently find the best anchor in each iteration of the greedy algorithm. We show that only the vertices in $\mathcal{L}$ need to be considered during computation, which significantly reduces the search space.

- By using the well-organized *onion layer* structure $\mathcal{L}$, we develop an efficient algorithm to compute the followers for the candidate anchor in a layer-by-layer paradigm. With the concept of support path, we only need to explore a very small portion of the vertices in $\mathcal{L}$. Together with early termination and pruning techniques, we further reduce the number of anchor and follower candidates and hence significantly enhance performance.

- Our comprehensive experiments on 10 real-life networks demonstrate the effectiveness and efficiency of our proposed techniques. For instance, our algorithm can prevent the unraveling of 630 vertices by anchoring *one single* vertex in the `Pokec` network. Regarding the running time, our `OLAK` algorithm outperforms the straightforward implementation of the greedy algorithm by *at least three orders* of magnitude.

Road Map. Section 2 introduces k-core and the anchored k-core problem. Section 3 presents our solution. Section 4 shows the experimental results. Section 5 reviews related work and Section 6 concludes the paper.

2. PRELIMINARIES

In this section, we first give some necessary notations and introduce the concept of k-core and its corresponding algorithm. Then, we formally define the anchored k-core problem and show its hardness. Table 1 summarizes the mathematical notations used throughout this paper.

2.1 Problem Definition

We consider an unweighted and undirected graph $G = (V, E)$, where V (resp. E) represents the set of vertices (resp. edges) in G. We denote $n = |V|$, $m = |E|$ and assume $m > n$. $NB(u, G)$ is the set of adjacent vertices of u in G, which is also called the neighbor set of u in G. We use $deg(u, G)$, the degree of u in G, to represent the number of adjacent vertices of u in G if $u \notin A$. $NB(u, G)$ (resp. $deg(u, G)$) is also written as $NB(u)$ (resp. $deg(u)$) when the context is

clear. We also use G to represent the vertices in G. Given a subgraph $J \subseteq G$, $NB(J)$ denotes the neighbor set of the vertices in J, i.e., $NB(J) = \{u \mid NB(u, J) \neq \emptyset \ \& \ u \in G\}$.

The concept of k-core has been widely used to describe cohesive subgraphs, which is formally defined as follows.

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

Algorithm 1: **ComputeCore(G, k)**

```
  Input  : G : a social network, k : degree constraint
  Output : C_k(G)
1 while exists u ∈ G with deg(u, G) < k do
2   G := G \ {u};
3 return G
```

Note that we have $C_{k+1}(G) \subseteq C_k(G)$ [6]. As shown in Algorithm 1, the k-core of a graph G can be obtained by recursively removing the vertices whose degrees are less than k, with a time complexity of $\mathcal{O}(m)$. The *core number* of a vertex $u \in G$ is the highest core where u appears, denoted by $\texttt{cn}(u)$. In this paper, we use k-shell to denote the vertices with the core number k, which is defined as follows.

Definition 2. ***k-shell***. *Given a graph G, the k-shell of G, denoted by $S_k(G)$, is the set of vertices with core number k; that is, $S_k(G) = C_k(G) \setminus C_{k+1}(G)$.*

Example 2. *In Figure 1, we have 3-core $C_3(G) = \{u_4, u_5, u_8, u_9\}$, 2-core $C_2(G) = \{u_1, u_2, ..., u_{11}\}$, and 2-shell $S_2(G) = \{u_1, u_2, u_3, u_6, u_7, u_{10}, u_{11}\}$.*

In this paper, once a vertex u in G is **anchored**, it is always retained in k-core regardless of the number of neighbors, i.e., $deg(u, G) = +\infty$ if $u \in A$.

Definition 3. ***anchored k-core***. *Given a graph G and a vertex set $A \subseteq G$, the anchored k-core, denoted by $C_k(G_A)$, is the corresponding k-core of G with vertices in A anchored.*

According to the definition of vertex degree, the computation of k-core with anchors is exactly the same as the k-core computation without anchors.

In addition to the anchored vertices in A and vertices in $C_k(G)$, more vertices might be retained in the $C_k(G_A)$ due to the contagious nature of the k-core computation. These vertices are called **followers** of the anchor vertices A, denoted by $\mathcal{F}(A, G)$, because they will not appear in k-core without the underpinning of A. The size of the followers reflects the effectiveness of the anchor vertices, where $\mathcal{F}(A, G) = C_k(G_A) \setminus \{C_k(G) \cup A\}$. In the following, we may use *anchor* to represent the *anchor vertex*, and we use $\mathcal{F}(A)$ to denote $\mathcal{F}(A, G)$ when the context is clear.

Problem Statement. Given a graph G, a degree constraint k and a budget b, the **anchored k-core problem** aims to find a set A of b vertices in G such that the size of the resulting anchored k-core, $C_k(G_A)$, is maximized; that is, $\mathcal{F}(A, G)$ is maximized.

Example 3. *In Figure 1, we have 3-core $C_3(G) = \{u_4, u_5, u_8, u_9\}$. If we set $A = \{u_1\}$, we have $C_3(G_A) = \{u_1, u_2, ..., u_9\}$ and $\mathcal{F}(A) = \{u_2, u_3, u_6\}$. We can find that u_1 is the best anchor if $b = 1$, and $\{u_1, u_{10}\}$ is the set of best anchors if $b = 2$.*

Problem Complexity. Given a set A of anchor vertices, we can immediately use a linear algorithm to compute $C_k(G_A)$ by not considering any vertex in A at Line 1 of Algorithm 1. However, it is very challenging to find the optimal A. As shown in [7], when $k \geq 3$ the problem of anchored k-core is NP-hard and W[2]-hard w.r.t the budget b. This implies that there is no non-trivial polynomial-time approximation algorithm even for $k > 2$, not mentioning the exact solution. In this paper, we adopt the greedy heuristic. Not surprisingly, the greedy algorithm may fail in some particular cases. For example, a graph G consists of two separate subgraphs G_1 and G_2. Specifically, G_1 is a chain of $\lceil n/2 \rceil + 1$ vertices with with $V(G_1) = \{v_1, v_2, ..., v_{\lceil n/2 \rceil + 1}\}$ and $E(G_1) = \{(v_i, v_{i+1}) \mid v_i, v_{i+1} \in V(G_1) \ \& \ 1 \leq i \leq \lceil n/2 \rceil\}$. When $k = 2$ and $b = 2$, the greedy algorithm will never choose an anchor x from $V(G_1)$ because $\mathcal{F}(x, G) = 0$. However, anchoring v_1 and $v_{\lceil n/2 \rceil + 1}$ can immediately get $\lceil n/2 \rceil - 1$ followers. We can infer that if there is a large subgraph in which most vertices can only becomes followers by anchoring a set U of vertices simultaneously, and anchoring a single vertex in U can not get enough followers, then the greedy algorithm will fail.

The inapproximability of the problem motivated the authors in [7] to develop a polynomial-time algorithm in graphs with bounded tree-width. However, this assumption does not hold in many real-life graphs (e.g., social networks). This motivated us to develop efficient heuristic algorithms to tackle the problem of anchored k-core on general graphs, and significantly improve performance by imposing an *onion layer* structure.

3. OUR APPROACH

This section presents our onion-layer based anchored k-core (OLAK) algorithm to effectively and efficiently find a set of anchors for the anchored k-core problem. In Section 3.1, we briefly introduce the motivation behind our *onion layer* based techniques. Section 3.2 shows how to limit the number of candidate anchors, and Section 3.3 presents efficient algorithms to compute the number of followers for a given anchor. Section 3.4 further develops new pruning techniques to reduce the number of candidate anchors, and present our OLAK algorithm by integrating these new techniques. Section 3.5 extends our approach to a general setting that each vertex has a different cost to be anchored.

3.1 Motivation

A straightforward solution for the anchored k-core problem is to exhaustively enumerate all possible set A with size b, and compute the resulting anchored k-core for each possible A. The time complexity of $\mathcal{O}(\binom{n}{b}m))$ is cost-prohibitive. Considering the hardness of the problem, we resort to a greedy heuristic which iteratively finds the best anchor vertex, i.e., the vertex with the largest number of followers. A straightforward implementation of the greedy algorithm is shown in Algorithm 2. The time complexity is $\mathcal{O}(bnm)$, where n and m correspond to the number of candidate anchors in each iteration (Line 3) and the cost of follower computation (Line 4). Note that we exclude the vertices in $C_k(G)$ at Line 3 because they are already in k-core.

Although the greedy algorithm does not have the submodular property due to the inapproximability of the problem, our empirical study shows its resulting anchor vertices have similar numbers of followers compared to that of the exact solution.

However, a simple implementation of the greedy algorithm is still unscalable on large-scale networks. In this paper,

Algorithm 2: **GreedyAK**(G, k, b)

```
  Input  : G : a social network, k : degree constraint,
           b : number of anchor vertices
  Output : A : the set of anchor vertices
1 A := ∅; i := 0;
2 while i < b do
3 |  for each u ∈ G \ {A ∪ C_k(G)} do
4 |  |_ Compute F(A ∪ u,G);
5 |  u* ← the best anchor vertex in this iteration;
6 |_ A := A ∪ u*; i := i + 1;
7 return A
```

we aim to significantly improve the two components of the greedy algorithm: (i) the number of candidate anchors in each iteration (Line 3); and (ii) the computation cost of finding followers (Line 4), which is determined by the number of candidate followers to explore.

Motivated by this, we propose an auxiliary structure $\mathcal{L}$, namely *onion layers*, to facilitate the computation such that we can significantly reduce the number of candidate anchors and candidate followers. At a high level, $\mathcal{L}$ consists of a subset of vertices such that we only need to consider the vertices within $\mathcal{L}$ as the candidate anchors. Moreover, the vertices within $\mathcal{L}$ are organized by different layers. Another nice property of the layer structure is that, by exploiting the layer structure, we can effectively bound the region (i.e., candidate followers) influenced by an anchor. We also develop early termination techniques based on the layer structure to recursively eliminate non-promising candidate followers.

In this section, we will focus on the problem of anchored k-core with $b = 1$, i.e., finding the *best* anchor which has the largest number of followers. In Section 3.4.3 we show the proposed algorithm can be immediately used in each iteration of Algorithm 2 by considering the previously anchored vertices. Note that the enhanced greedy algorithm produces the same result as Algorithm 2 because they follow the same greedy heuristic.

3.2 Reducing # Candidate Anchors

The *onion layers* of G, denoted by $\mathcal{L}$, consists of the vertices in (k-1)-shell and their neighbors that are not in k-core; that is, $\mathcal{L} := S_{k-1}(G) \cup \{NB(S_{k-1}(G), G) \setminus C_k(G)\}$. Below, we show that only the vertices in $\mathcal{L}$ need to be considered, and that $F(u)$ is empty for every $u \notin \mathcal{L}$.

The following theorem indicates that only the vertices from (k-1)-shell can be the followers for a given anchor.

Theorem 1. *Given a graph G and its (k-1)-shell $S_{k-1}(G)$, if a vertex x is anchored, all of its followers come from (k-1)-shell; that is, $u \in F(x, G)$ implies $u \in S_{k-1}(G)$.*

Proof. We prove correctness by contradiction. The intuition is that if a follower comes from a k'-shell with $k' < k-1$, we show that it belongs to (k-1)-shell instead.

Let M and N be the k-core and (k-1)-core of G respectively before anchoring the vertex x. As a follower u cannot come from $C_k(G)$, u has a core number k' with $k' < k - 1$ if $u \notin S_{k-1}(G)$. Let M' be the k-core after x is anchored, we have $u \in M'$, and $deg(v, M') \geq k$ for every vertex $v \in M'$. If we delete x and its corresponding edges from M, we have $deg(v, M' \setminus \{x\}) \geq k - 1$ for every vertex $v \in NB(x, M')$ because $deg(v, M') \geq k$ and only one edge is removed from v. This means the deletion of x will not be cascaded since all of its neighbors in M' stay in the computation of $C_{k-1}(M' \setminus \{x\})$. Consequently, all vertices in $M' \setminus \{x\}$ satisfy the $k-1$ degree constraint and hence $M' \setminus \{x\} \subseteq C_{k-1}(G)$. As $u \in M'$ and $u \neq x$, we have u which belongs to $C_{k-1}(G)$ and this contradicts with the fact that the core number of u is smaller than $k - 1$. □

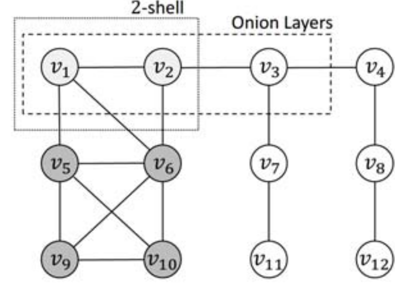


Figure 2: Bounded Anchors and Followers

Then the following theorem can significantly reduce the size of the candidate anchor vertices.

Theorem 2. *Given a graph G, if an anchored vertex x has at least one follower, x is from $\mathcal{L}$; that is, $|F(x, G)| > 0$ implies that $x \in S_{k-1}(G) \cup \{NB(S_{k-1}(G), G) \setminus C_k(G)\}$.*

Proof. $F(x)$ is the follower set of x and suppose $F(x) \neq \emptyset$. Additionally, $NB(x) \cap F(x) \neq \emptyset$, otherwise, for every x's neighbor u and $u \notin C_k(G)$, we have $u \notin C_k(G_x)$, which leads to $C_k(G) = C_k(G_x)$ and thus $F(x) = \emptyset$. Since $F(x) \subseteq S_{k-1}(G)$ by Theorem 1, x is in $S_{k-1}(G)$ or at least one of its neighbors in $S_{k-1}(G)$, i.e., $x \in S_{k-1}(G) \cup \{NB(S_{k-1}(G), G) \setminus C_k(G)\}$. □

Example 4. *In Figure 2 with $k = 3$, we have 3-core $C_3(G) = \{v_5, v_6, v_9, v_{10}\}$, 2-shell $S_2(G) = \{v_1, v_2\}$ and onion layers $\mathcal{L} = \{v_1, v_2, v_3\}$. By Theorem 1, followers of any vertex are inside $S_2(G)$. Promising anchors are inside $\mathcal{L}$ by Theorem 2. Consequently, to find the best anchor, we only consider the vertices in $\mathcal{L}$ as candidate anchors, which are $\{v_1, v_2, v_3\}$ in this example. Once a vertex u is anchored, only vertices $\{v_1, v_2\}$ may become followers.*

3.3 Efficiently Finding Followers

In this subsection, we develop efficient algorithms to compute followers for a chosen anchor vertex. A straightforward implementation is to directly apply the k-core computation algorithm (Algorithm 1) with the existence of the anchor. As an alternative, one may extend the continuous k-core maintenance algorithms [18, 22, 30] by setting the core number of the anchor vertex as infinite and then update core numbers for other vertices. A vertex with core number increased to k is a follower. This greatly improves the computational cost. Nevertheless, we show the performance can be significantly enhanced by using the well-organized structure of *onion layers*.

In the experiments, we observe that the size of candidate anchors in Theorem 2 is still considerably large and there are many unavoidable attempts to find the best anchor vertex. This implies that it is worthwhile to carefully build an auxiliary data structure to facilitate the computation of followers for all candidate vertices. Specifically, we revisit k-core computation and design the *onion layer* structure of $\mathcal{L}$ such that the computation of the followers in each attempt can be greatly enhanced.

3.3.1 The Onion Layer Structure

We notice that the k-core computation (Algorithm 1) does not explicitly consider the deletion (i.e., leave) order of the non k-core vertices. We say the deletion order of an instance of k-core computation is *valid* if (1) the vertex violates the

degree constraint at the time it is deleted; and (2) all remaining vertices satisfy the degree constraint when the deletion stops. Theorem 3 below shows that any *valid order* will come up with the k-core.

Theorem 3. *Algorithm 1 always return $C_k(G)$ w.r.t any valid deletion order of non anchored k-core vertices.*

PROOF. Suppose there are two different valid deletion orders, O_1 and O_2, leading to two different k-cores C_1 and C_2, respectively. Let $M = C_1 \setminus C_2$ and $M \neq \emptyset$. This implies that all vertices in M are discarded in the access order O_1. Suppose u_1 is the first removed vertex in M, this implies that $deg(u_1, C_1 \cup C_2) \geq k$ because none of the vertices in C_1 or M are removed when u_1 is accessed. This implies O_1 is not a valid order. Consequently, the Theorem holds. □

Theorem 3 motivates us to impose an *onion layer* structure on $\mathcal{L}$ to facilitate computation of the followers. $\mathcal{L}$ consists of $s+1$ layers, $\{L_0, L_1, \ldots, L_s\}$ ($L_0^s = \mathcal{L}$), produced by an *onion-peeling-like* algorithm. The pseudo-code is shown in Algorithm 3. We first compute $C_{k-1}(G)$ at Line 1, then start to peel the (k-1)-shell by removing *all* vertices not satisfying the degree constraint at the same time (Lines 2 and 6), which are kept in the same layer (Line 4). When the peeling process terminates, we have $i = s$, $\mathcal{L} = L_0^s = S_{k-1}(G)$ and $N = C_k(G)$. Then we put the neighbors of $S_{k-1}(G)$ (excluding the ones in $C_k(G)$ and L_1^s) to L_0 as the highest layer (Line 7). In this paper, we use L_i^j ($i < j$) to denote the vertices between layer i and layer j (inclusive), and $l(u)$ to denote the layer index of a vertex u in $\mathcal{L}$.

Algorithm 3: **OnionPeeling**(G, k)

```
  Input  : G : a social network, k : degree constraint
  Output : onion layers ℒ (i.e., L_0^s)
1 N := C_{k-1}(G); i := 0;
2 P := {u | deg(u, N) < k & u ∈ N};
3 while P ≠ ∅ do
4     i := i + 1; L_i := P;
5     N := N \ P;
6     P := {u | deg(u, N) < k & u ∈ N};
7 L_0 := {u | u ∈ NB(L_1^i, G) \ {N ∪ L_1^i}};
8 return L_0^i
```

Algorithm Correctness. We show that $S_{i-1}(G)$, obtained in Algorithm 3, is correct; that is, N is $C_k(G)$ when the peeling process (Lines 3-6) terminates. We can choose an arbitrary deletion order for the vertices in the same layer since they do not satisfy the degree constraint. On the other hand, when the peeling process terminates (i.e., $V = \emptyset$ at Line 6), we have $deg(u, N) \geq k$ for every vertex $u \in N$. Therefore, we can get a valid deletion order and hence N is $C_k(G)$ when the peeling process terminates.

The time complexity of Algorithm 3 is $\mathcal{O}(m)$ in the worst case. Although we need to update the *onion layer* structure in each iteration of the greedy algorithm, this cost is greatly amortized by a considerably large number of follower computations.

Example 5. *In Figure 3 with k=3, we have 3-core $C_3 = \{v_5, v_6, v_8, v_9\}$ and 2-core $C_2 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_8, v_9\}$. By checking degree constraint against the vertices in C_2, we have $deg(v_1, C_2) < 3$ and $deg(v_4, C_2) < 3$. So the layer $L_1 = \{v_1, v_4\}$. After deleting v_1 and v_4 from C_2, we have $deg(v_2, C_2) < 3$ and $L_2 = \{v_2\}$. Iteratively, after deleting v_2 from C_2, we have $deg(v_3, C_2) < 3$ and $L_3 = \{v_3\}$. After deleting v_3 from C_2, 3-core computation is finalised. Note that for original 2-core C_2 we have $v_7 \in NB(C_2) \setminus C_2$. Thus, $L_0 = \{v_7\}$ and all layers are generated.*

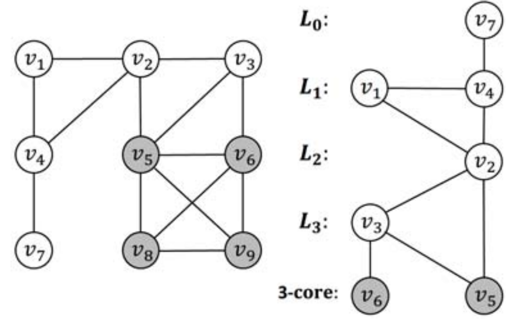


Figure 3: Onion Layer Structure (L_0^3)

3.3.2 Onion Layer based Follower Computation

Now, we present an algorithm to efficiently compute the followers for a given anchor x based on the *onion layer* structure. The algorithm has two key techniques: (1) *finding candidate followers*, which explores the candidate followers for the anchor x; and (2) *early termination*, which recursively discards the non-promising candidates during the computation.

(1) Finding candidate followers. We first introduce the concept of a *support path*, and show how to find the candidate followers of an anchor x, denoted by $CF(x)$.

Definition 4. ***Support Path***. *We say there is a support path for a vertex $u \in L_1^s$ w.r.t a given anchor vertex x if there is a path $x \rightsquigarrow u$ such that all vertices are from L_1^s and we have $l(y) < l(z)$ for every two consecutive vertices y and z along this path. Note that $l(u)$ is the layer index of the vertex $u \in \mathcal{L}$.*

The following theorem indicates that we do not need to consider the vertices without any support path as the candidate followers.

Theorem 4. *A vertex $u \in S_{k-1}(G)$ is a follower of the anchor x (i.e., $u \in CF(x)$) implies that there is a support path $x \rightsquigarrow u$.*

PROOF. According to the definition of vertex degree (note that $deg(u, G) = +\infty$ if $u \in A$), we can immediately employ the onion-peeling algorithm (Lines 1-6 in Algorithm 3) to compute $C_k(G_x)$ in which the vertex x is anchored. In the computation of $C_k(G)$ (without any anchors), all vertices in L_1^s are removed in Algorithm 3, while some of them (i.e., followers) may survive the computation of $C_k(G_x)$ (with anchoring x). Let i denote the layer index of x ($i = l(x)$). In the computation of $C_k(G_x)$, for a vertex $u \in L_1^{i-1}$, when u is accessed, the degree of u is less than k because the degree at current time is exactly the same as u is accessed in the computation of $C_k(G)$. So all vertices in L_1^{i-1} are deleted. Then, when vertices in L_1^{i-1} have been deleted and no vertex in L_i has been deleted, for every vertex $v \in L_i$, the degree of v is less than k because the degree at current time is the same as in the computation of $C_k(G)$. Consequently, all vertices in $L_1^i \setminus \{x\}$ cannot follow x and are deleted. At this point, only neighbors of x in L_{i+1}^s become candidate followers and clearly they have support paths. For a candidate follower y, only its neighbors in L_{j+1}^s ($j = l(y)$) become candidate followers because y cannot save other vertices in the computation of $C_k(G_x)$, i.e., the degrees of other vertices are still less than k with the existence of y. Consequently, the candidate spread from x is strictly a top-down search through x's edges and candidates' edges, which constitute support paths. For a vertex z without any support paths, when non-candidate vertices in L_1^{p-1} ($p = l(z)$) have been

deleted and no vertex in L_p has been deleted, the degree of z is less than k because it is same as z is accessed in the computation of $C_k(G)$. Consequently, every candidate follower of x has at least one support path which implies there is always a support path for a follower of x. $\square$

According to the above theorem, we may generate candidate followers by iteratively activating the neighbors at lower level of $\mathcal{L}$. In this paper, we use $CF(x)$ to denote all candidate followers of an anchor x obtained based on Theorem 4.

Example 6. *In Figure 3 with k=3, we have 3-core* $C_3 = \{v_5, v_6, v_8, v_9\}$ *and* L_0^3 *shown on the right side. If* v_1 *is anchored, only* v_2 *will be activated as a candidate follower, because* v_2 *is a neighbor of* v_1 *and is on a lower layer. Although* v_4 *is also a neighbor of* v_1, v_4 *will not be a candidate follower because they are on the same layer, i.e.,* $l(v_1) = l(v_4)$. *Similarly, after* v_2 *becomes a candidate follower,* v_3 *will be a candidate follower as well, while* v_4 *will not. Thus, we only need to consider the vertices on* $\{v_1, v_2, v_3\}$ *for follower computation.*

(2) Early termination. We remark that existing a *support path* is a necessary condition for a follower, and hence we need to conduct k-core computation on $CF(x) \cup C_k(G) \cup \{x\}$ to identify the true followers. To avoid this, we introduce an early termination technique to prune the search space. In this technique, we find the candidate followers of an anchor x in a layer-by-layer fashion, i.e., for all the vertices which have been found to be inside of $CF(x)$ and are waiting to be explored, we explore the vertex with the smallest layer number first (ties are broken by the vertices' IDs).

In the layer-by-layer seach, each vertex in $\mathcal{L}$ has **three statuses**. We say a vertex u is **unexplored** if it has not been checked with the degree constraint in our layer-by-layer traversal. A vertex is **survived** if it has survived the degree check, otherwise it becomes **discarded**. For a given anchor, a *discarded* vertex will never be involved in the following computation, and a *survived* vertex may become *discarded* later due to the deletion cascade. Note that some vertices are *implicitly* marked as *discarded* since they are never accessed due to the candidate followers pruning technique.

We use $d^+(u)$ to denote the degree upper bound of a vertex u in $C_k(G_x)$. Specifically, $d^+(u) = d_s^+(u) + d_u^+(u) + d_c(u)$ where $d_s^+(u)$ (resp. $d_u^+(u)$) is the number of *survived* (resp. *unexplored*) neighbors in $\mathcal{L}$ and $d_c(u)$ is the number of neighbors in $C_k(G)$. The following theorem indicates that we can safely exclude a candidate follower u if $d^+(u) < k$. The removal of a vertex may invoke the deletion of other vertices, where details are described in Algorithm 4. When the shrink function terminates, all of the vertices affected by the removal of u will be correctly updated.

Theorem 5. *A vertex* $u \in L_1^s$ *cannot be a follower if* $d^+(u) < k$.

Proof. Neighbors of a vertex u can be classified into four disjoint sets N_0, N_1, N_2 and N_3. N_0 denotes the set of neighbors *not in* $C_{k-1}(G)$. N_1 (resp. N_2) denotes the explored (resp. *unexplored*) neighbors in $\mathcal{L}$, and N_3 denotes the neighbors from $C_k(G)$. Clearly, none of the neighbors in N_0 contributes degree support to u because they have been discarded during the computation of $C_{k-1}(G)$. Once a vertex in $\mathcal{L}$ is *explored*, it will be marked as either *survived* or *discarded*, and a *discarded* vertex cannot provide degree support to u w.r.t $C_k(G_x)$. Therefore, d_s^+(u) is correct. Moreover, we have $d_u^+(u) = |N_2|$ and $d_c(u) = |N_3|$. Consequently, the degree of u in $C_k(G_x)$ is bounded by $d^+(u)$. $\square$

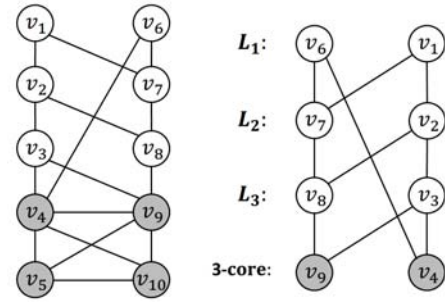


Figure 4: Early Termination

Example 7. *In Figure 4 with k=3, we have 3-core* $C_3 = \{v_4, v_5, v_9, v_{10}\}$ *and* L_1^3 *shown on the right side. If* v_1 *is anchored, the candidate follower set without early termination technique is* $\{v_2, v_3, v_7, v_8\}$. *The candidates starts from* v_1 *which puts* v_7 *and* v_2 *in the waiting list for future exploration. Then* v_7 *is explored and we have* $d_s^+(v_7) = 1$, $d_u^+(v_7) = 1$ *and* $d_c(v_7) = 0$. *Since* $d^+(v_7) < 3$, v_7 *is discarded. Then* v_2 *is explored and we have* $d^+(v_2) = 3$ *which means* v_8 *and* v_3 *can be put in the waiting list. Similarly,* v_8 *is explored and discarded which leads to the deletion of* v_2 *and* v_3. *Thus,* v_1 *does not have any followers.*

Algorithm 4: **Shrink**(u)

Input : u : the vertex for degree check
1 **for** each *survived* neighbor v with $v \neq x$ **do**
2 $\quad d^+(v) := d^+(v) - 1$;
3 $\quad T \leftarrow v$ **If** $d^+(v) < k$;
4 **for** each $v \in T$ **do**
5 $\quad u$ is set *discarded*;
6 $\quad$ **Shrink**(v);

(3) Finding Followers. Algorithm 5 lists the pseudo-code of the follower computation for a chosen anchor x. A min heap H is used to keep the candidate followers, and the key of a vertex u is $l(u)$ with ties broken by the vertices' IDs. In this way, we explore the candidates in a layer-by-layer fashion and it is easy to check whether a vertex u has been *explored* based on its ID and layer index $l(u)$. For each popped vertex u, Line 4 computes its degree upper bound $d^+(u)$. If u survives the degree check or u is the anchor x, u will be set to *survived* (Line 6) and its neighbors in lower layers (i.e., unexplored candidate followers) will be pushed into H if they are not already in H (Lines 7-9). Otherwise, u is set to *discarded* and the early termination process is invoked (Lines 11-12). The deletion may be cascaded and some *survived* vertices may be set to *discarded* during the process. When the algorithm terminates, all *survived* vertices in $\mathcal{L} \setminus \{x\}$ are the followers of x. The time complexity of the algorithm is $\mathcal{O}(m)$ in the worst case because each edge is at most accessed three times: to push neighbors into H, compute upper bound and compute the cascade of the deletion.

Algorithm Correctness. We show the deletion of the vertices in Algorithm 5 has a *valid order* O for the computation of $C_k(G_x)$. A vertex u may be *implicitly* deleted if (1) $u \notin CF(x)$; or (2) $u \in CF(x)$, but u is not pushed into H because some of vertices on its support path has been set to *discarded*. We assume all these vertices on the layer i are deleted in O right before the first vertex on this layer is popped from H. The correctness of case (1) is immediate since $u \notin CF(x)$. In case (2), we conclude that there does not exist a support path for u in which all vertices are

Algorithm 5: **FindFollowers**(x, $\mathcal{L}$)

Input : x : the anchor; $\mathcal{L}$: *onion layers*
Output : F : the followers of x
1 $H := \emptyset$; $H.push(x)$;
2 **while** $H \neq \emptyset$ **do**
3 $u \leftarrow H.pop()$;
4 Compute $d^+(u)$;
5 **if** $d^+(u) \geq k$ **then**
6 u is set *survived*;
7 **for** each $v \in NB(u) \cup \mathcal{L}$ and $l(v) > l(u)$
8 and $v \notin H$ **do**
9 $H.push(v)$;
10 **else**
11 u is set *discarded* ;
12 **Shrink**(u);
13 **return** *survived* vertices in $\mathcal{L} \setminus \{x\}$

followers. Using similar rationale to Theorem 4, u cannot be supported by x and hence can be safely discarded. A vertex u may also be *explicitly* deleted in O if (3) u is set *discarded* at Line 11 because it fails the degree check when it is popped; or (4) $d^+(u)$ deceases below k due to the deletions of the other vertices (Line 5 of Algorithm 4). Because $d^+(u)$ is correctly computed (Line 4) and maintained (Algorithm 4), u does not satisfy the degree constraint when u is deleted in cases (3) and (4). Let M denote $C_k(G)$ and L' denote the remaining *survived* vertices when Algorithm 5 terminates. Now, we show that none of the vertices in L' can be discarded. As all of the vertices in $\mathcal{L}$ have been *explored* explicitly or implicitly, we have $d^+(u) = deg(u, L' \cup M)$ for every vertex $u \in L'$ since $d^+_s(u) = 0$, $d^+_u(u) = deg(u, L')$ and $d_c(u) = deg(u, M)$. As $d^+(u) \geq k$ for every vertex $u \in L'$, we have $deg(u, L' \cup M) \geq k$ and none of the vertices in $L' \cup M$ can be discarded. As such, O is a *valid order* and $L' \cup M = C_k(G_x)$.

REMARK 1. *Note that we can also apply the onion layer structure to facilitate continuous core maintenance [30]. In this problem, when a new edge is inserted, we can set the corresponding vertex, whose core number increases to k, as an anchor. With the similar rationale, the layer structure can be used to reduce the search region of the candidate followers, whose core values may be updated. However, it is not cost effective because the onion layer structure must be updated after every insertion of an edge, and the new edges may arrive in a streaming fashion.*

3.4 The OLAK Algorithm

In this section, we introduce two pruning techniques to further reduce the number of candidate anchors. Then we present our `OLAK` algorithm to find the best anchor in graph G and show how to handle the case with multiple anchors.

3.4.1 Follower based Pruning

The following theorem indicates that, to find the best anchor, we do not need to consider an anchor if it is a follower of another anchor; that is, an anchor u is shadowed by x if $u \in \mathcal{F}(x)$.

Theorem 6. *Given two vertices x and u in $\mathcal{L}$, we have $|\mathcal{F}(x)| > |\mathcal{F}(u)|$ if $u \in \mathcal{F}(x)$.*

PROOF. $u \in \mathcal{F}(x)$ implies that there is a support path $x \rightsquigarrow u$, and hence we have $CF(u) \subset CF(x)$ where $CF(u)$ and $CF(x)$ are candidate followers of u and x obtained by Theorem 4, respectively. $u \in \mathcal{F}(x)$ also implies that u has enough degree support when u is not anchored and x is anchored, i.e., $deg(u, \mathcal{F}(x) \cup \{x\} \cup C_k(G)) \geq k$. Consequently, every vertex v in $\mathcal{F}(u)$ will not be discarded in the computation of $\mathcal{F}(x)$ since $deg(v, \mathcal{F}(u) \cup \{u\} \cup C_k(G)) \geq k$. So $\mathcal{F}(u) \subset \mathcal{F}(x)$ and then $|\mathcal{F}(x)| > |\mathcal{F}(u)|$. □

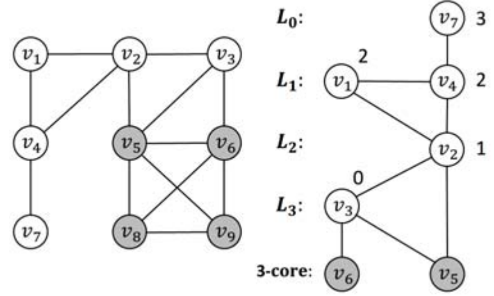


Figure 5: Followers Pruning and Upper Bound Pruning

Example 8. *In Figure 5 with k=3, we have 3-core $C_3 = \{v_5, v_6, v_8, v_9\}$ and L^3_0 shown on the right side. In the procedure of finding a best anchor, if v_1 has been tried as an anchor, we have $\mathcal{F}(v_1) = \{v_2, v_3\}$. So neither v_2 nor v_3 can be a best anchor.*

3.4.2 Upper Bound based Pruning

Let $W(x)$ denote the neighbors of a vertex x in lower layers, i.e., $W(x) = \{u \mid u \in NB(x) \cap \mathcal{L}$ and $l(u) > l(x)\}$. We use $UB(x)$ to denote the upper bound of $|\mathcal{F}(x)|$, where

$$UB(x) = \begin{cases} \sum_{u \in W(x)} (UB(u) + 1) & \text{if } |W(x)| > 0; \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

The following theorem shows that we can accumulatively compute the upper bound of the number of followers for vertices in L^s_1. The correctness is evident since $|CF(x)| \leq UB(x)$ and $|\mathcal{F}(x)| \leq |CF(x)|$ for every vertex x.

Theorem 7. *Let λ denote the number of followers of the best anchor seen so far. We can exclude any candidate anchor x if $UB(x) < \lambda$.*

Example 9. *In Figure 5 with k=3, we have the 3-core $C_3 = \{v_5, v_6, v_8, v_9\}$ and L^3_0 shown on the right side. By Equation (1), we have $UB(v_3) = 0$, $UB(v_2) = 1$, $UB(v_1) = 2$, $UB(v_4) = 2$ and $UB(v_7) = 3$. In the procedure to find a best anchor, if v_1 is tried for anchoring and it becomes current best anchor with $|\mathcal{F}(v_1)| = 2$, we don't need to consider anchoring any vertices except for v_7 because their upper bounds do not exceed 2.*

Based on Equation (1), our implementation computes the upper bound of the follower size for each vertex in $\mathcal{L}$ in a bottom-up fashion with a time complexity of $\mathcal{O}(m)$. To get tighter upper bounds, we can replace the $UB(u)$ in Equation (1) with $CF(u)$ for each u whose $CF(u)$ has been computed. This does not pay off because the time complexity becomes $\mathcal{O}(nm)$ which reaches the complexity of the greedy algorithm for the anchored k-core problem with $b = 1$. It is also confirmed by initial experiments.

3.4.3 Combining the Elements

Algorithm 6 illustrates the details of `OLAK` which finds the best anchor vertex for a given graph G (i.e., $b = 1$). Particularly, we first apply Algorithm 3 to compute the *onion layers* of G (Line 1) and the upper bound of each vertex in $\mathcal{L}$ (Line 2). Initially, the candidate anchor set T is set

Algorithm 6: **OLAK**(G, k)

Input : G : a social network, k : degree constraint
Output : the best anchor vertex

```
1  Compute onion layers L (Algorithm 3);
2  Compute UB(x) for each vertex x ∈ L;
3  T ← L (Theorem 2); λ = 0;
4  for each x ∈ T with decreasing order of UB(x) do
5      if UB(x) < λ then
6          Break (Theorem 7);
7      F(x) ← FindFollowers(x, L) (Algorithm 5);
8      if F(x) ≠ ∅ then
9          T := T \ F(x) (Theorem 6);
10     if |F(x)| > λ then
11         λ := |F(x)|;
12 return the best anchor
```

to L_1^s according to Theorem 2. Then we sequentially access vertices in T based on their upper bounds of the number of followers in decreasing order, and compute their followers by Algorithm 5. According to the follower-based pruning, Line 9 excludes the followers of current accessed vertex from T. We continuously maintain the largest number of followers seen so far for one vertex, denoted by λ, which may eliminate some non-promising candidate anchors in $\mathcal{T}$ by upper bound based pruning (Line 6). We have the best anchor when the algorithm terminates.

To handle general cases where $b > 1$, our `OLAK` algorithm can easily fit within the greedy algorithm (Replacing Lines 3-4 of Algorithm 2) to find the best vertex in each iteration. The only difference is that we need to enforce that the anchored vertices in previous iterations remain in the k-core. Note that in order to avoid computing $C_{k-1}(G_A)$ (Line 1 of Algorithm 3) from scratch in each iteration, we adopt an existing core maintenance technique [30] to continuously maintain the (k-1)-core and the k-core after inserting a best anchor. Moreover, if the (k-1)-core consists of a set of disconnected subgraphs, we can avoid the re-computation of the followers of a subgraph in the next iteration unless there is a new anchor in this subgraph. The time complexity of the algorithm remains $\mathcal{O}(bnm)$ in the worst case. Nevertheless, our empirical study shows we can significantly improve performance of the straightforward implementation (Algorithm 2) by at least 3 orders of magnitude, due to a much smaller number of candidate anchors and a more efficient follower computation algorithm.

Algorithm Correctness. (1) For the anchored k-core problem with $b = 1$ on graph G, we get the correct result immediately based on the correctness of proposed techniques. (2) Assume the algorithm is correct when $b = i$, $i \in N^+$ and returns the anchor set A. (3) Consider the problem with $b = i + 1$, now the k-core of G is $C_k(G_A)$ since we have $deg(u, G) \geq k$ for any $u \in C_k(G_A)$ (note that $deg(v, G) = +\infty$ for any $v \in A$) and $C_k(G_A)$ is maximal (according to Definition 3). Then the (k-1)-core is updated correctly by the core maintenance algorithm. Thus, we get the updated *onion layers* $\mathcal{L}$ correctly by Algorithm 3. Since all the techniques are based on $\mathcal{L}$, after running `OLAK` on G with $b = 1$ again, we get the correct result $A \cup \{x\}$ for the case of $b = i + 1$ on G. Note that in the (k-1)-core N of G, for every disconnected subgraph S with $S \in N$ and $x \notin S$, S keeps same after anchoring x, thus, the previous result of anchoring any vertex in S can be reused.

3.5 The Setting for Different Vertex Costs

A more practical and general setting is that each vertex has a different cost to be anchored. Towards this setting, we show that all proposed techniques also work well.

Given a degree constraint k, a budget b and a graph $G = (V, E, w)$ where $w(u)$ is the cost of anchoring u for every $u \in V(G)$. The anchored k-core problem becomes finding a set A of vertices in $V(G)$ such that the size of the resulting anchored k-core is maximized and $\Sigma_{v \in A} w(v) \leq b$.

Since the computation of the k-core, onion-layers and followers are not relevant to the anchor cost of the vertex, our updates mainly focus on the suitability evaluation of the anchor vertex. Instead of using the number of followers (i.e., $|\mathcal{F}(u)|$), we use a utility function $f(u)$ to evaluate the suitability of a candidate anchor u with $f(u) = \frac{|\mathcal{F}(u)|}{w(u)}$ where $w(u)$ is the the cost of the vertex u to be anchored. We need to do the following updates:

- The consumption of budget is the summation of the anchoring costs of the anchored vertices. We cannot exceed the budget when introducing a new anchor vertex (Line 5 of Algorithm 2, Line 4 of Algorithm 6).
- Replace the $|\mathcal{F}(u)|$ with $f(u)$ at Lines 10-11 of Algorithm 6. The score threshold λ in Algorithm 6 is the best $f(u)$ value seen so far.
- At Line 9 of Algorithm 6, we only remove a vertex $u \in \mathcal{F}(x)$ when $w(u) \geq w(x)$.

Note that we do not need to re-consider the candidate vertex (i.e., onion-layers) under the new settings because we have $|\mathcal{F}(u)| = 0$ (i.e., $f(u) = 0$) for every vertex u not on the onion-layers.

4. EXPERIMENTAL EVALUATION

This section evaluates the effectiveness and efficiency of all techniques through comprehensive experiments.

4.1 Experimental Setting

Algorithms. To the best of our knowledge, no existing work investigates efficient algorithms for the anchored k-core problem on general graphs. Towards the effectiveness, we tested five algorithms (`Rand`, `Rand1`, `Rand2`, `Degree` and `Exact`) to choose different anchors to see the number of followers, compared with our greedy result (`OLAK`). Case studies were made on the greedy result. We also implemented and evaluated the algorithms to assess our techniques incrementally, from a naive algorithm (`Naive`) through to the final advanced algorithm (`OLAK`). One baseline algorithm (`Baseline2`) based on core maintenance is also evaluated. Table 2 shows the summary for algorithms.

Datasets. Ten real-life networks were deployed in our experiments and we assume all vertices in each network are initially engaged. The original data of `Yelp` was downloaded from `https://www.yelp.com.au/dataset_challenge`, `DBLP` came from `http://dblp.uni-trier.de/` and the others were from `http://snap.stanford.edu/`. Table 3 shows the statistics of the 10 datasets, listed in increasing order of their edge numbers.

Parameters. We conducted experiments under different settings by varying the degree constraint k and the budget for the anchors b. The default values of k and b were both 20. In the experiments, the range of k varied from 5 to 50 and the range of b varied from 1 to 100.

All programs were implemented in standard C++ and compiled with G++ in Linux. All experiments were performed on a machine with Intel Xeon 2.3GHz CPU and Redhat Linux System. We evaluate the effectiveness of the algorithms by reporting the number of the followers for the

Table 2: Summary of Algorithms

Algorithm	Description
Rand	randomly chooses b anchors from $G \setminus C_k(G)$
Rand1	randomly chooses b anchors from N_1 where $N_1 = \mathcal{L} \cap NB(C_k(G))$
Rand2	randomly chooses b anchors from N_2 where $N_2 = \mathcal{L} \cap NB(N_1)$
Degree	chooses the b anchors from N_2 with highest degrees in $\mathcal{L} \setminus L_0$
Exact	identifies the optimal solution by exhaustively searching all possible combinations of b anchors by Algorithm 5
Naive	computes a k-core on G for each candidate anchor $u \in G \setminus C_k(G)$ to find the best anchor in each iteration of Algorithm 2
Baseline1	computes a k-core on G for each candidate anchor $u \in \mathcal{L}$ (Theorem 2) to find the best anchor in each iteration of Algorithm 2
Baseline2	applies the state-of-art core maintenance algorithm [30] to compute followers for each candidate anchor in Baseline1
BL1+C	computes a k-core on $\{x\} \cup C_{k-1}(G)$ (Theorem 1) for each candidate anchor x in Baseline1
BL1+CF	computes a k-core on $CF(x) \cup \{x\} \cup C_k(G)$ (Theorem 4) for each candidate anchor x in Baseline1
BL1+CFE	finds followers from $CF(x)$ by early termination technique (Theorem 5) for each candidate anchor x in Baseline1, i.e., applies Algorithm 5
BL1+CFEP	equips the follower based pruning (Theorem 6) in BL1+CFE
OLAK	equips the upper bound based pruning (Theorem 7) in BL1+CFEP and arrives at Algorithm 6

Table 3: Statistics of Datasets

Dataset	Nodes	Edges	d_{avg}	d_{max}
Facebook	4,039	88,234	43.7	1045
Brightkite	58,228	194,090	6.7	1098
Gowalla	196,591	456,830	4.7	9967
Yelp	552,339	1,781,908	6.5	3812
Flickr	105,938	2,316,948	43.7	5465
YouTube	1,134,890	2,987,624	5.3	28754
DBLP	1,566,919	6,461,300	8.3	2023
Pokec	1,632,803	8,320,605	10.2	7266
LiveJournal	3,997,962	34,681,189	17.4	14815
Orkut	3,072,441	117,185,083	76.3	33313

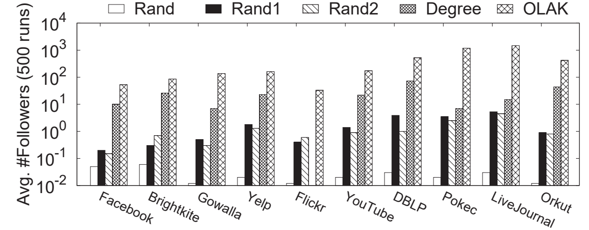
resulting anchors. The efficiency of the algorithms is measured by its running time and the number of the candidate anchors and followers accessed.

4.2 Effectiveness

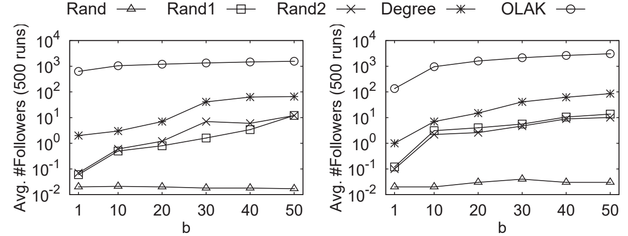
We used the result of OLAK to evaluate the effectiveness for all greedy algorithms, since they follow the same heuristic and produce the same results. We also conducted case studies to show real-world examples for the anchored k-core.

4.2.1 *Effectiveness of the Greedy Algorithm*

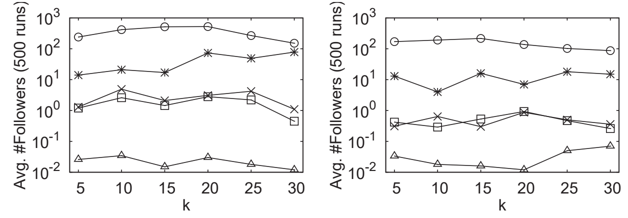
Figure 6 compares the number of followers w.r.t b anchors identified by OLAK with three random approach (Rand, Rand1 and Rand2) and one degree based approaches (Degree).We report the average number of followers for 500 independent tests in three random methods. The resulting numbers for other two methods are always unique because we choose the highest degree anchors in Degree and find the best anchors in OLAK. Note that $\mathcal{L} \setminus L_0$ is the set which contains all followers for any anchor, and $\mathcal{L}$ contains all promising anchors. Degree basically improve performance by choosing high degree anchors, but is still significantly outperformed by OLAK. In Figure 6(a), Degree fails to get any follower in Flickr because a high degree vertex u in N_2 does not necessarily



(a) 10 Datasets, k=20, b=20



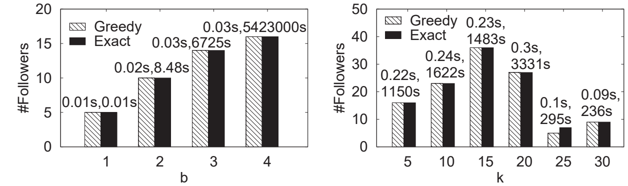
(b) Pokec, k=20 (c) LiveJournal, k=20



(d) DBLP, b=20 (e) Gowalla, b=20

Figure 6: Number of Followers

have a follower. It is common to find no followers after 20 random anchors are chosen. This is because we observe that, the majority of the vertices do not have any followers in all 10 real-life networks. For OLAK, we notice that there are 630 followers in Pokec dataset with a single anchored vertex. Figures 6(a) shows that OLAK underpins more than 1000 followers with 20 anchors on the Livejournal and Pokec. Figures 6(b)-(e) show the margin between OLAK and the other algorithms does not change much when k and b vary.



(a) Facebook, k=30 (b) Brightkite, b=2

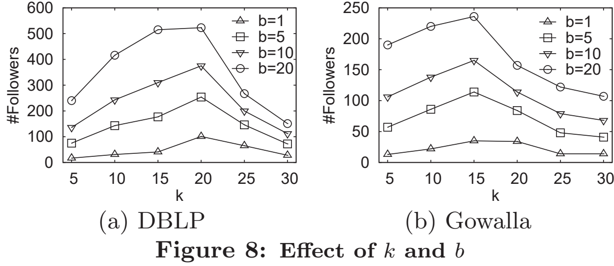
Figure 7: Greedy vs Exact

To further justify the effectiveness of OLAK, we also compare its performance with Exact, which identifies the optimal solution by exhaustively searching two relatively small networks, where b varies from 1 to 4 on Facebook and k varies from 5 to 30 on Brightkite. Figure 7 shows that OLAK finds the optimal solution in all but one setting. Note that we only test Exact on small datasets with small b values because we cannot afford its running time for other settings.

Figure 8 reports the impact of b and k on the size of the followers for OLAK. The number of the followers clearly grows with the increase of the budget b. The size becomes relatively small when k is small or large.

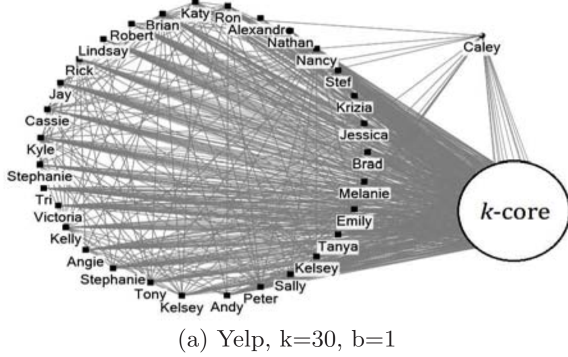
4.2.2 *Case Studies*

We show the anchors identified by OLAK and their corresponding followers in Figure 9(a)-(b). Figure 9(a) shows

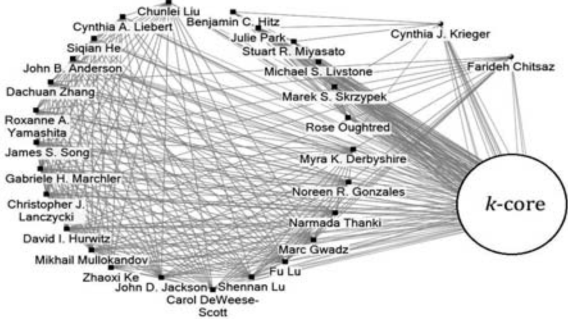


(a) DBLP (b) Gowalla

Figure 8: Effect of k and b



(a) Yelp, k=30, b=1



(b) DBLP, k=20, b=2

Figure 9: Case Studies

that when the user "Caley" alone is anchored, there are 31 followers in `Yelp` with $k = 30$. It is interesting that only 10 of them are neighbors of "Caley", and the others are supported indirectly.

In Figure 9(b), `DBLP` is deployed and k is set to 20. In the case study, there is an edge between two authors if they co-author at least three papers. When $b = 2$, two authors are identified by `OLAK` and there are 26 followers. We find that although the two anchored authors have not co-authored any papers, they belong to the same community and 8 out of their 9 papers in `DBLP` have been published in Nucleic Acids Research. Not surprisingly, all their followers are also from the same community, and there are already considerably large number of co-authored papers among them.

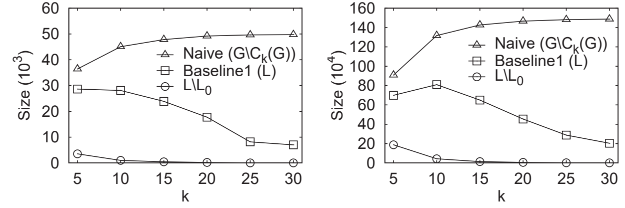
We also investigated the characteristics of the anchors identified by `OLAK` in different settings and datasets. It is non-trivial to understand the potential of a vertex based on its local structure information (e.g., degree or neighbor's degrees), due to the complicated cascade behavior of unraveling in the networks.

4.3 Efficiency

We first investigate the efficiency of the techniques proposed in this paper, then compare our `OLAK` algorithm with `Baseline1` and `Baseline2` under different settings.

4.3.1 Evaluation of Individual Techniques

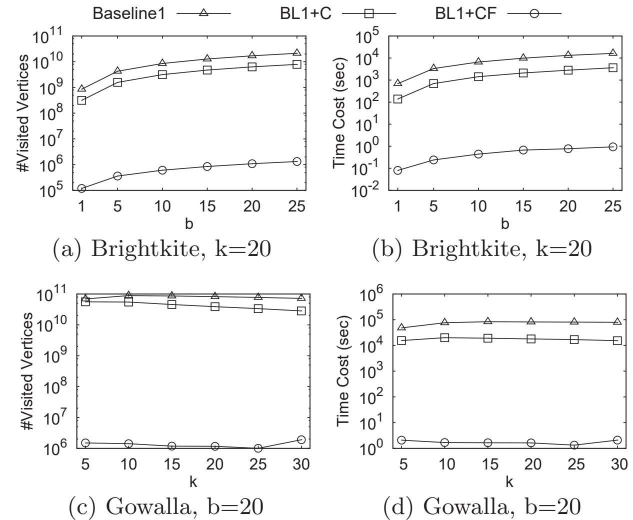
The essence of our proposed techniques is to use the *onion layer* structure to speed up the anchored k-core computation by significantly reducing the number of candidate anchors and candidate followers. Below, we evaluate the proposed techniques against these two criteria.



(a) Brightkite, b=1 (b) DBLP, b=1

Figure 10: Reducing Candidate Anchors

Reducing Candidate Anchors. Figure 10 reports the sizes of $G \setminus C_k(G)$, *onion layers* $\mathcal{L}$ and (k-1)-shell (i.e.,$\mathcal{L} \setminus L_0$) on two networks `Brightkite` and `DBLP` with $b = 1$ and k varied from 5 to 30. Recall that `Naive` checks all vertices in $G \setminus C_k$, and the other three algorithms (`Baseline1`, `Baseline2` and `OLAK`) only consider the vertices from $\mathcal{L}$ as candidate anchors (Theorem 2). We also report the size of (k-1)-shell ($\mathcal{L} \setminus L_0$), which bounds the size of the candidate followers by Theorem 1. As expected, the size of $G \setminus C_k(G)$ grows with k because the size of k-core decreases with k. Conversely, the size of (k-1)-shell is much smaller, and drops with the growth of k. The size of *onion layers* $\mathcal{L}$ also decreases quickly with k. It also shows that the majority of the vertices in $\mathcal{L}$ are neighbors of (k-1)-shell vertices, especially for small k, which is not considered in the computation of the followers.



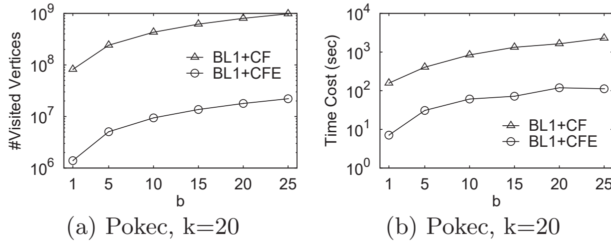
(a) Brightkite, k=20 (b) Brightkite, k=20

(c) Gowalla, b=20 (d) Gowalla, b=20

Figure 11: Pruning Candidate Followers

Pruning Candidate Followers. Figure 11 demonstrates the effectiveness of the pruning techniques which help us to eliminate non-promising candidate followers. Three algorithms were evaluated using the number of visited followers and the running time on two networks `Brightkite` and `Gowalla` by varying b and k, respectively. In `Baseline1`, all vertices in G are regarded as candidate followers during the k-core computation. `BL1+C` represents `Baseline1` equipped with *onion layers*, where candidate followers are obtained from the (k-1)-shell. (Theorem 1), and `BL1+CF` is `Baseline1` equipped with the candidate exploration technique, which only explores the vertices in $CF(x)$ for each candidate an-

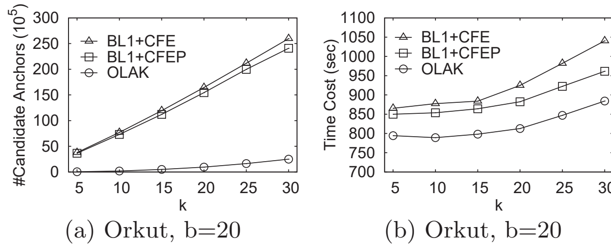
chor x (Theorem 4). We report that both pruning techniques significantly reduce the number of the candidate followers explored, especially the support path based candidate follower exploration (Theorem 4).



(a) Pokec, k=20 (b) Pokec, k=20

Figure 12: Effectiveness of Early Termination

Early Termination. Figure 12 shows that our early termination technique (Theorem 5) , applied in `BL1+CFE`, can further significantly reduce the number of explored vertices during computation, and hence improves performance by at least one order of magnitude.



(a) Orkut, b=20 (b) Orkut, b=20

Figure 13: Further Pruning Candidate Anchors

Pruning Candidate Anchors. Figure 13 evaluates the effectiveness of the follower based and upper-bound based candidate anchor pruning techniques on the `Orkut` dataset with k ranging from 5 to 30. In particular, `BL1+CFE` is the `OLAK` algorithm without these two pruning techniques. `BL1+CFEP` includes the follower-based pruning (Theorem 6), and `OLAK` further equips the algorithm with upper-bound based pruning (Theorem 7). We report that both pruning techniques contribute to the performance of `OLAK`. One interesting observation is that although upper-bound based pruning eliminates many more candidate anchors than follower-based pruning, but their contributions in terms of running time do not have a big difference. This is because the majority of the candidate anchors pruned by their upper-bound are immediately excluded by our follower computation algorithm (Algorithm 5).

We observe that the most powerful optimization is pruning candidate followers by onion layers, followed by early termination, reducing candidate anchors and two pruning rules for candidate anchors.

4.3.2 *Performance Evaluation*

We evaluate the performance of `Baseline1`, `Baseline2` and `OLAK` in different settings.

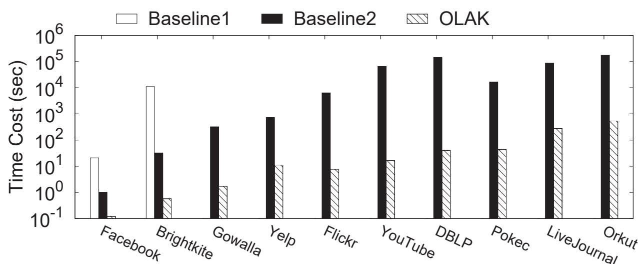
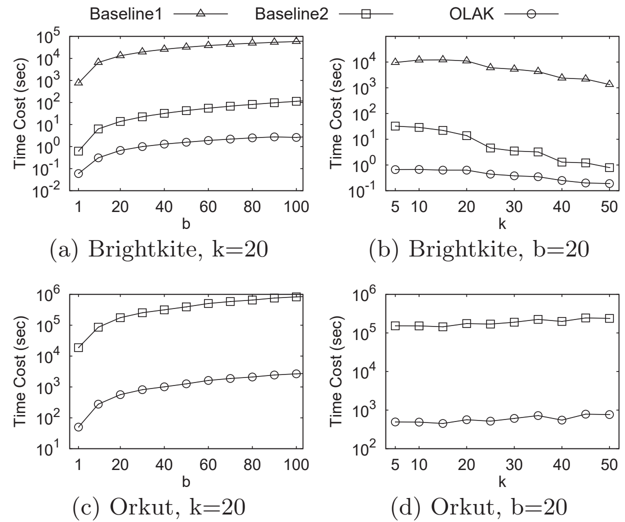


Figure 14: Running time on Different Datasets

Different Datasets (Networks). Figure 14 reports the performance of the three algorithms on 10 networks with $k = 20$ and $b = 20$. The datasets are ordered by their network sizes (i.e., the number of edges). Not surprisingly, the performance of `Baseline1` is very poor and cannot finish computation on 8 networks within one week. Its performance is significantly enhanced by applying the state-of-the-art core maintenance algorithm for the follower computation on *onion layers*. `OLAK` outperforms `Baseline2` by a large margin with up to 3 orders of magnitude.

We observed that the size of the onion layers has great impact on the running time of `OLAK`, which is closely related to the network size. This is because the main computation is conducted on the vertices in the onion layers. Other factors such as avg. degree, max. degree and the number of onion layers in datasets do not make noticeable differences.



(a) Brightkite, k=20 (b) Brightkite, b=20

(c) Orkut, k=20 (d) Orkut, b=20

Figure 15: Effect of k and b

Effect of k and b. Figure 15 studies the impact of k and b on the three algorithms against two datasets `Brightkite` and `Orkut`, with b varied from 1 to 100 and k ranged from 5 to 50. `OLAK` significantly outperforms the two baseline algorithms under all settings. We omit `Baseline1` for `Orkut` as it cannot finish the computation within one month.

5. RELATED WORK

k-core computation, first introduced by Seidman [23], is a fundamental graph problem with a wide spectrum of applications such as social contagion [24], event detection [20], network analysis [3], network visualization [29, 31], internet topology [4, 8], dense subgraph problems [5], influence study [17, 25], graph clustering [15], graph model validation [16], structure analysis of software system [28], and protein function prediction [2]. Batagelj and Zaversnik [6] present a linear-time in-memory algorithm to compute the core numbers of all vertices in a graph. Wen *et* al. [26] and Cheng *et* al. [9] propose I/O efficient algorithms for core number computation on graphs that cannot fit in the main memory of a machine. Locally computing and estimating core numbers are studied in [14] and [21] respectively. Algorithms for core number maintenance on dynamic graphs are proposed by [1, 18, 22, 30]. As we do not need to decompose or estimate core numbers and we aim to propose an in-memory solution, the only applicable existing technique is core maintenance, which can be used by setting the core number of a candidate anchor as infinite and inserting its edges (`Baseline2` in the experiments).

There has been significant focus on the engagement dynamic in social networks (e.g., [7, 10, 11, 12, 13, 19, 27]). In this research direction, preventing network unraveling, recently studied by Bhawalkar and Kleinberg *et* al., is modelled as the anchored k-core problem [7]. This is because the degeneration property of k-core can be used to quantify engagement dynamics in real social networks [19]. As shown in [7], the unraveling process stops when the remaining engaged individuals correspond to the k-core of the network. Nevertheless, the anchored k-core problem has been proven to be an NP-hard problem even on a planar graph [7, 10]. In [7], an algorithm was proposed to solve the anchored k-core problem on graphs with bounded tree-width, which is inapplicable to real-life social networks. To the best of our knowledge, the algorithm proposed in this paper is the first practical algorithm to solve the anchored k-core problem on general large networks.

6. CONCLUSION

In this paper, we study the problem of anchored k-core, which aims to *anchor* a set of vertices in a network such that the size of the resulting k-core is maximized. The hardness of this problem motivate us to develop greedy algorithms. We design the *onion layer* structure to maintain a small set of vertices in the graph, such that (1) we only need to consider the vertices in the *onion layers* to find the best anchor; and (2) the layer structure enables us to develop an efficient follower computation algorithm using a layer-by-layer paradigm. The layer structure also helps us to develop early termination and pruning techniques to further prune follower and anchor candidates. Then we present our `OLAK` algorithm by combining all proposed techniques. Empirical study shows that we can find critical vertices in the network whose participation may lead to a large number of followers. Extensive experiments on 10 real-life networks show that `OLAK` improves the performance of the naive solution by at least 3 orders of magnitude.

7. ACKNOWLEDGMENTS

Wenjie Zhang is supported by ARC DP150103071 and DP150102728. Ying Zhang is supported by ARC DE140100679 and DP170103710. Lu Qin is supported by ARC DE140100999 and DP160101513. Xuemin Lin is supported by NSFC61232006, ARC DP150102728, DP140103578 and DP170101628.

8. REFERENCES

[1] H. Aksu, M. Canim, Y.-C. Chang, I. Korpeoglu, and Ö. Ulusoy. Distributed-core view materialization and maintenance for large dynamic graphs. *TKDE*, 26(10):2439–2452, 2014.

[2] M. Altaf-Ul-Amine, K. Nishikata, T. Korna, T. Miyasato, Y. Shinbo, M. Arifuzzaman, C. Wada, M. Maeda, T. Oshima, H. Mori, et al. Prediction of protein functions based on k-cores of protein-protein interaction networks and amino acid sequences. *Gen. Inf.*, 14:498–499, 2003.

[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] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *WAW*, pages 25–37, 2009.

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

[7] 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.

[8] 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.

[9] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011.

[10] R. Chitnis, F. V. Fomin, and P. A. Golovach. Parameterized complexity of the anchored k-core problem for directed graphs. *Inf. Comput.*, 247:11–22, 2016.

[11] R. H. Chitnis, F. V. Fomin, and P. A. Golovach. Preventing unraveling in social networks gets harder. In *AAAI*, 2013.

[12] M. S.-Y. Chwe. Structure and strategy in collective action 1. *American Journal of Sociology*, 105(1):128–156, 1999.

[13] M. S.-Y. Chwe. Communication and coordination in social networks. *The Review of Economic Studies*, 67(1):1–16, 2000.

[14] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

[15] C. Giatsidis, F. Malliaros, D. M. Thilikos, and M. Vazirgiannis. Corecluster: A degeneracy based graph clustering framework. In *IAAI*, 2014.

[16] J. Healy, J. Janssen, E. E. Milios, and W. Aiello. Characterization of graphs using degree cores. In *WAW*, pages 137–148, 2006.

[17] 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.

[18] R. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *TKDE*, 26(10):2453–2465, 2014.

[19] F. D. Malliaros and M. Vazirgiannis. To stay or not to stay: modeling engagement dynamics in social graphs. In *CIKM*, pages 469–478, 2013.

[20] P. Meladianos, G. Nikolentzos, F. Rousseau, Y. Stavrakas, and M. Vazirgiannis. Degeneracy-based real-time sub-event detection in twitter stream. In *ICWSM*, pages 248–257, 2015.

[21] M. P. O'Brien and B. D. Sullivan. Locally estimating core numbers. In *ICDM*, pages 460–469, 2014.

[22] 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.

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

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

[25] D. Vogiatzis. Influence study on hyper-graphs. In *AAAI*, 2013.

[26] 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.

[27] S. Wu, A. D. Sarma, A. Fabrikant, S. Lattanzi, and A. Tomkins. Arrival and departure dynamics in social networks. In *WSDM*, pages 233–242, 2013.

[28] 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.

[29] Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *ICDE*, pages 1049–1060. IEEE, 2012.

[30] Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin. A fast order-based approach for core maintenance. *CoRR*, abs/1606.00200, 2016.

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