Incremental Graph Computation: Anchored Vertex Tracking in Dynamic Social Networks

Taotao Cai, Shuiqiao Yang, Jianxin Li*, Quan Z. Sheng, Jian Yang, Xin Wang, Wei Emma Zhang, and Longxiang Gao

Abstract—User engagement has recently received significant attention in understanding the decay and expansion of communities in many online social networking platforms. When a user chooses to leave a social networking platform, it may cause a cascading dropping out among his friends. In many scenarios, it would be a good idea to persuade critical users to stay active in the network and prevent such a cascade because critical users can have significant influence on user engagement of a whole network. Many user engagement studies have done to find a set of critical *(anchored)* users in the static social network. However, social network is highly dynamic and its structure is continuously evolving. In order to fully utilize the power of anchored users in evolving networks, existing studies have to mine multiple sets of anchored users at different times, which incurs the expensive computational cost. In this paper, we target a new research problem called *Anchored Vertex Tracking* (AVT) that aims to track the anchored users at each timestamp of evolving networks. Nonetheless, it is nontrivial to handle the AVT problem which we have proved to be NP-hard. To address the challenge, we develop a greedy algorithm inspired by the previous anchored k-core study in the static network. Furthermore, we design an incremental algorithm to efficiently solve the AVT problem by utilizing the smoothness of the network structure's evolution. The extensive experiments conducted on real and synthetic datasets demonstrate the performance of our proposed algorithms and the effectiveness in solving the AVT problem.

Index Terms—Anchored vertex tracking, user engagement, dynamic social networks, k-core computation

1 Introduction

In recent years, user engagement has become a hot research topic in network science, arising from a plethora of online social networking and social media applications, such as *Web of Science Core Collection*, *Facebook*, and *Instagram*. Newman [26] studied the collaboration of users in a collaboration network, and found that the probability of collaboration between two users highly related to the number of common neighbors of the selected users. Kossinets and Watts [18], [19] verified that two users who have numerous common friends are more likely to be friends by investigating a series of social networks. Cannistraci et al. [8] presented that two social network users are more likely to become friends if their common neighbors are members of a local community, and the strength of their relationship relies on the number of their common neighbors in the community. Moreover, Laishram et al. [20] mentioned that the incentives for keeping users' engagement on a social network platform partially depend on how many friends they can keep in touch with. Once the users' incentives are low, they may leave the platform. The decreased engagement of one user may affect others' engagement incentives, further causing them to leave. Considering a model of user engagement in a social network platform, where the participation of each user is motivated by the number of engaged neighbors. The user engagement model is a natural equilibrium corresponding to the k-core of the social network, where k-core is a popular model to identifies the maximal subgraph in which every vertex has at least k neighbors. The leaving of some critical users may cause a cascading departure from the social network platform. Therefore, the efforts of user engagement studies [5], [6], [25], [27], [33] have been devoted to finding the crucial (anchored) users who significantly impact the formation of social communities and the operations of social networking platforms.

The previous studies in user engagement can reveal the evolution of the community's decay and expansion in social networks. The study of user engagement benefits to many applications in real-life business markets. However, most of the previous researches dedicated to user engagement depend on a strong assumption - social networks are modelled as static graphs. This simple premise rarely reflects the evolving nature of social networks, of which the topology often evolves over time in real world [10], [21]. Therefore, for a given dynamic social network, the anchored users selected at an earlier time may not be appropriate to be used for user engagement in the following time due to the evolution of the network.

To better understand user engagement in evolving networks, one possible way is to re-calculate the anchored users after the network structure is dynamically changed. A natural question is that, given a limited budget, how to find l users (denoted as *anchored vertices*) at each timestamp of an evolving social network, so that a k-core community size can be maximized by l number of anchored users at each timestamp. We refer this problem as *Anchored Vertex Tracking* (AVT) in this paper, which aims to find a series of anchored vertex sets with each set size limited to l. By solving the proposed problem, we can efficiently track the anchored users to improve the effectiveness of user engagement in evolving networks.

Tracking the anchored vertices could be very useful for many practical applications, such as sustainable analysis of social networks, impact analysis of advertising placement, and social recommendation. Taking the impact analysis of advertising placement as an example. Given a social network, the users' connection often evolves, which leads to the dynamic change of user influences and

- *Email: taotao.cai@mq.edu.au; shuiqiao.yang@uts.edu.au.*
- *Taotao Cai and Shuiqiao Yang are the joint first authors.*

Manuscript is under review.

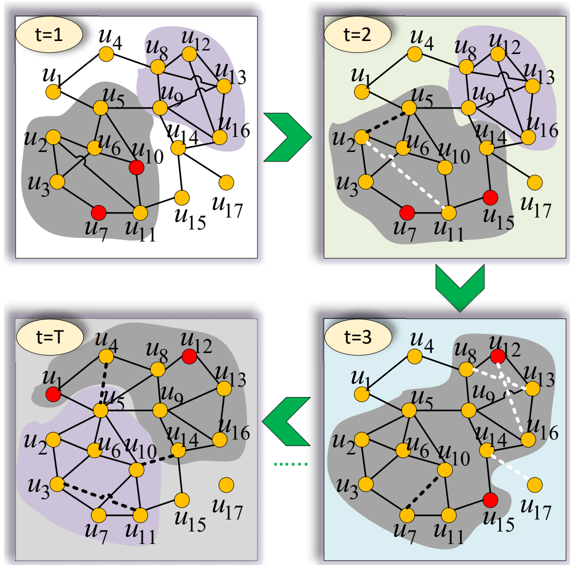


Fig. 1. An example of Anchored Vertex Tracking (AVT).

roles. The AVT study can continuously track the critical users to locate a set of users who favor propagating the advertisements at different times. In contrast, traditional user engagement methods like OLAK [33] and RCM [20] only work well in static networks. Therefore, AVT can deliver timely support of services in many applications. Here, we utilize an example in Figure 1 to explain the AVT problem in details.

Example 1. *Figure 1 presents an evolving social network with 17 members and their relationships. The amount of a member's neighbor in the network reflects his willingness to engage in the network. If one member has many friends, the member would be the core members and willing to remain engaged in the network. According to the above engagement model, if we motivate (anchored) users $\{u_7, u_{10}\}$ to keep engaged in the network at the timestamp $t = 1$, then the number of k-core members would increase from 5 (lavender) to 12 (gray) where k is 3. With the evolution of the network, at the timestamp $t = 2$, a new relationship between users u_2 and u_5 is established (black dotted line) while the relationship of users u_2 and u_{11} is broken (white dotted line). Under this situation, the number of 3-core members would increase from 5 to 14 if we anchor users $\{u_7, u_{15}\}$. However, if we keep users $\{u_7, u_{10}\}$ engaged in the network, the 3-core members would only increase to 11. Similarly, if users $\{u_1, u_{12}\}$ are anchored at timestamp $t = T$, there are 8 users that will become the 3-core members, but no new members will add into 3-core while anchoring $\{u_7, u_{10}\}$. From the above procedure, it can be seen that the optimal anchored vertices selection may vary in each timestamp of an evolving network. Thus, it is critical to track the optimal anchored vertices at each timestamp.*

Challenges. Considering the dynamic change of social networks and the scale of network data, it is infeasible to directly use the existing methods [6], [12], [20], [33] and compute the anchored user set for every timestamp. We prove that the AVT problem is NP-hard. To the best of our knowledge, there is no existing work to solve the AVT problem, particularly when the number of timestamps is large.

To conquer the above challenges, we first develop a Greedy algorithm by extending the previous anchored k-core study in the static graph [5], [33]. However, the Greedy algorithm is expensive for large-scale social network data. Therefore, we optimize the Greedy algorithm in two aspects: (1) reducing the number of potential anchored vertices; and (2) accelerating computation of followers. To further improve the efficiency, we also design an incremental algorithm by utilizing the smoothness of the network structure's evolution.

Contributions. We state our major contributions as follows:

- We formally define the problem of AVT and explain the motivation of solving the problem with real applications.
- We propose a Greedy algorithm by extending the core maintenance method in [36] to answer the AVT problem. Besides, we built several pruning strategies to accelerate the Greedy algorithm.
- We develop an efficient incremental algorithm by utilizing the smoothness of the network structure's evolution and the well-designed fast-updating core maintenance methods in evolving networks.
- We conduct extensive experiments to demonstrate the efficiency and effectiveness of proposed approaches using real and synthetic datasets.

Organization. We present the preliminaries in Section 2. Section 3 formally defines the AVT problem. We propose the Greedy algorithm in Section 4, and further develop an incremental algorithm to solve the AVT problem more efficiently in Section 5. The experimental results are reported in Section 6. Finally, we review the related works in Section 7, and conclude the paper in Section 8.

2 Preliminaries

We define an evolving network as a sequence of graph snapshots $\mathcal{G} = \{G_t\}_1^T$, and $\{1, 2, .., T\}$ is a finite set of time points. We assume that the network snapshots in $\mathcal{G}$ share the same vertex set. Let G_t represent the network snapshot at timestamp $t \in [1, T]$, where V and E_t are the vertex set and edge set of G_t, respectively. Besides, we set $nbr(u, G_t)$ as the set of vertices adjacent to vertex $u \in V$ in G_t, and the degree $d(u, G_t)$ represents the number of neighbors for u in G_t, $i.e.$, $|nbr(u, G_t)|$. Table 1 summarizes the mathematical notations frequently used throughout this paper.

2.1 Anchored k-core

We first introduce a notion of k-**core**, which has been widely used to describe the cohesiveness of subgraph.

Definition 1 (k-core)**.** *Given a graph G_t, the k-core of G_t is the maximal subgraph in G_t, denoted by C_k, in which the degree of each vertex in C_k is at least k.*

The k-core of a graph G_t, can be computed by repeatedly deleting all vertices (and their adjacent edges) with the degree less than k. The process of the above k-core computation is called **core decomposition** [4], which is described in Algorithm 1.

For a vertex u in graph G_t, the **core number** of u, denoted as $core(u)$, is the maximum value of k such that u is contained in the k-core of G_t. Formally,

Definition 2 (Core Number)**.** *Given a graph $G_t = (V, E_t)$, for a vertex $u \in V$, its core number, denoted as $core(u)$, is defined as $core(u, G_t) = max\{k : u \in C_k\}$.*

TABLE 1
Notations Frequently Used in This Paper

Notation	Definition
$\mathcal{G}$	an evolving graph
G_t	the snapshot graph of $\mathcal{G}$ at time instant t
V; E_t	the vertex set and edge set of G_t
$nbr(u, G_t)$	the set of adjacent vertices of u in G_t
$d(u, G_t)$	the degree of u in G_t
$deg^+(u)$	the remaining degree of u
$deg^-(u)$	the candidate degree of u
C_k	the k-core subgraph
$O(G_t)$	the K-order of G_t where $O(G_t) = \{\mathcal{O}_1, \mathcal{O}_2, ...\}$
$C_k(\mathcal{S}_t)$	the anchored k-core that anchored by $\mathcal{S}_t$
$\mathcal{S}_t$	the anchored vertex set of G_t
$F_k(u, G_t)$	followers of an anchored vertex u in G_t
$F_k(\mathcal{S}_t, G_t)$	followers of an anchored vertex set $\mathcal{S}_t$ in G_t
E^+; E^-	the edges insertion and edges deletion from graph snapshots G_{t-1} to G_t
$mcd(u)$	the max core degree of u

When the context is clear, we use $core(u)$ instead of $core(u, G_t)$ for the sake of concise presentation.

Example 2. *Consider the graph snapshot G_1 in Figure 1. The subgraph C_3 induced by vertices $\{u_8, u_9, u_{12}, u_{13}, u_{16}\}$ is the 3-core of G_1. This is because every vertex in the induced subgraph has a degree at least 3. Besides, there does not exist a 4-core in G_1. Therefore, we have $core(v) = 3$ for each vertex $v \in C_3$.*

If a vertex u is **anchored**, in this work, it supposes that such vertex meets the requirement of k-core regardless of the degree constraint. The anchored vertex u may lead to add more vertices into C_k due to the contagious nature of *k-core* computation. These vertices are called as **followers** of u.

Definition 3 (Followers)**.** *Given a graph G_t and an anchored vertex set S_t, the followers of S_t in G_t, denoted as $F_k(S_t, G_t)$, are the vertices whose degrees become at least k due to the selection of the anchored vertex set S_t.*

Definition 4 (Anchored k-core)**.** *Given a graph G_t and an anchored vertex set S_t, the anchored k-core $C_k(S_t)$ consists of the k-core of G_t, S_t, and the followers of S_t.*

Example 3. *Consider the graph G_1 in Figure 1, the 3-core is $C_3 = \{u_8, u_9, u_{12}, u_{13}, u_{16}\}$. If we give users u_7 and u_{10} a special budget to join in C_3, the users $\{u_2, u_3, u_5, u_6, u_{11}\}$ could be brought into C_3 because they have no less than 3 neighbors in C_3 now. Hence, the size of C_3 is enlarged from 16 to 23 with the consideration of u_7 and u_{10} being the "anchored" vertices where the users $\{u_2, u_3, u_5, u_6, u_{11}\}$ are the "followers" of anchored vertex set $S = \{u_7, u_{10}\}$. Also, the anchored 3-core of S would be $C_3(S) = \{u_2, u_3, u_5, .., u_{14}, u_{16}\}$.*

2.2 Problem Statement

The traditional anchored k-core problem aims to explore anchored vertex set for static social networks. However, in real-world social networks, the network topology is almost always evolving over time. Therefore, the anchored vertex set, which maximizes the k-core size, should be constantly updated according to the dynamic changes of the social networks. In this paper, we model the evolving social network as a series of snapshot graphs $\mathcal{G} = \{G_t\}_1^T$. Our goal is to track a series of anchored vertex set $S = \{S_1, S_2, .., S_T\}$ that maximizes the k-core size at each snapshot graph G_t where $t = 1, 2, .., T$. More formally, we formulate the above task as the *Anchored Vertex Tracking* problem.

Algorithm 1: Core decomposition(G_t, k)

```
k ← 1;
while V is not empty do
    while exists u ∈ V with nbr(u, G_t) < k do
        V ← V \ {u};
        core(u) ← k − 1;
        for w ∈ nbr(u, G_t) do
            nbr(w, G_t) ← nbr(w, G_t) − 1;
    k ← k + 1;
return core;
```

Problem formulation: Given an evolving graph $\mathcal{G} = \{G_t\}_1^T$, the parameter k, and an integer l, the problem of *anchored vertex tracking (AVT)* in $\mathcal{G}$ aims to discover a series of anchored vertex set $\mathcal{S} = \{S_t\}_1^T$, satisfying

$$S_t = \arg\max_{|S_t| \le l} |\mathcal{C}_k(S_t)| \quad (1)$$

where $t \in [1, T]$, and $S_t \subseteq V$.

Example 4. *In Figure 1, if we set $k = 3$ and $l = 2$, the result of the anchored vertex tracking problem can be $\mathcal{S} = \{S_1, S_2, S_3, .., S_T\}$ with $S_1 = \{u_7, u_{10}\}$, $S_2 = \{u_7, u_{15}\}$, $S_3 = \{u_{12}, u_{15}\}$, ..., $S_T = \{u_1, u_{12}\}$. Besides, the related anchored k-core would be $\mathcal{C}_k(S_1) = \{u_2, u_3, u_5, u_6, .., u_{13}, u_{16}\}$, $\mathcal{C}_k(S_2) = \{u_2, u_3, u_5, u_6, .., u_{16}\}$, $\mathcal{C}_k(S_3) = \{u_2, u_3, u_5, u_6, .., u_{16}\}$, ..., $\mathcal{C}_k(S_T) = \{u_1, u_2, .., u_{14}, u_{16}\}$.*

3 Problem Analysis

In this section, we discuss the problem complexity of AVT. In particular, we will verify that the AVT problem can be solved exactly while $k = 1$ and $k = 2$ but become intractable for $k \ge 3$.

Theorem 1. *Given an evolving general graph $\mathcal{G} = \{G_t\}_1^T$, the problem of AVT is NP-hard when $k \ge 3$.*

Proof. (1) When $k = 1$ and $t \in [1, T]$, the followers of any selected anchored vertex would be empty. Therefore, we can randomly select l vertices from $\{G_t \setminus C_1\}$ as the anchored vertex set of G_t where G_t is the snapshot graph of $\mathcal{G}$ and C_1 is the 1-core of G_t. Besides, the time complexity of computing the set of $\{G_t \setminus C_1\}$ from snapshot graph G_t is $\mathcal{O}(|V| + |E_t|)$. Thus, the AVT problem is solvable in polynomial time with the time complexity of $\mathcal{O}(\sum_{t=1}^T(|V| + |E_t|))$ while $k = 1$.

(2) When $k = 2$ and $t \in [1, T]$, we note that the AVT problem can be solved by repeatedly answering the anchored 2-core at each snapshot graph $G_t \in \mathcal{G}$. Besides, Bhawalkar et al. [5] proposed an exactly *Linear-Time Implementation* algorithm to solve the anchored 2-core problem in the snapshot graph G_t with time complexity $\mathcal{O}(|E_t| + |V|log|V|)$. From the above, we can conclude that there is an implementation of the algorithm to answer the AVT problem by running in time complexity $\mathcal{O}(\sum_{t=1}^T(|E_t| + |V|log|V|))$. Therefore, the AVT problem is solvable in polynomial time while $k = 2$.

(3) When $k \ge 3$ and $t \in [1, T]$, we first note that the anchored vertex tracking problem is equivalent to a set of anchored k-core

problem at each snapshot graphs in evolving graph $\mathcal{G}$. Thus, we can conclude that the anchored vertex tracking problem is NP-hard once the anchored k-core problem is NP-hard.

Next, we prove the problem of anchored k-core at each snapshot graph $G_t \in \mathcal{G}$ is NP-hard, by reducing the anchored k-core problem to the *Set Cover* problem [16]. Given a fix instance l of set cover with s sets $S_1, .., S_s$ and n elements $\{e_1, .., e_n\} = \bigcup_{i=1}^{s} S_i$, we first give the construction only for instance of set cover such that for all i, $|S_i| \leq k-1$. Then, we construct a corresponding instance of the anchored k-core problem in G_t as follow, by lifting the above restriction while still obtaining the same results.

Considering G_t contains a set of nodes $V = \{u_1, ..., u_n\}$ which is associated with a collection of subsets $\mathcal{S} = \{S_1, ..., S_s\}$, $S_i \subseteq V$. We construct an arbitrarily large graph G', where each vertex in G' has degree k except for a single vertex $v(G')$ that has degree $k-1$. Then, we set $H = \{G'_1, ..., G'_m\}$ as the set of n connected components G'_j of G', where G'_j is associated with an element e_j. When $e_j \in S_i$, there is an edge between u_i and $v(G'_j)$. Based on the definition of k-core in Definition 1, once there exists i such that u_i is the neighbor of $v(G'_j)$, then all vertices in G'_j will remain in k-core. Therefore, if there exists a set cover C with size l, we can set l anchors from u_i while $S_i \in C$ for each i, and then all vertices in H will be the member of k-core. Since we are assuming that $|S_i| < k$ for all sets, each vertex u_i will not in the subgraph of k-core unless u_i is anchored. Thus, we must anchor some vertex adjacent to $v(G'_j)$ for each $G'_j \in G'$, which corresponds precisely to a set cover of size l. From the above, we can conclude that for instances of set cover with maximum set size at most $k-1$, there is a set cover of size l if and only if there exists an assignment in the corresponding anchored k-core instance using only l anchored vertices such that all vertices in H keep in k-core. Hence, the remaining question of reducing the anchored k-core problem to the *Set Cover* problem is to lift the restriction on the maximum set size, i.e. $|S_i| \leq k-1$. Bhawalkar et al. [5] proposed a d-ary tree (defined as $tree(d, y)$) method to lift this restriction. Specifically, to lift the restriction on the maximum set size, they use $tree(k-1, |S_i|)$ to replace each instance of u_i. Besides, if $y_1, ..., y_{|S_i|}$ are the leaves of the d-ary tree, then the pairs of vertices (y_j, u_j) will be constructed for each $u_j \in S_i$.

Since the *Set Cover* problem is NP-hard, we prove that the anchored k-core problem is NP-hard for $k \geq 3$, and so is the anchored vertex tracking problem. □

We then consider the inapproximability of the anchored vertex tracking problem.

Theorem 2. *For $k \geq 3$ and any positive constant $\epsilon > 0$, there does not exist a polynomial time algorithm to find an approximate solution of AVT problem within an $\mathcal{O}(n^{1-\epsilon})$ multiplicative factor of the optimal solution in general graph, unless P = NP.*

Proof. We have reduced the anchored vertex tracking (AVT) problem from the *Set Cover* problem in the proof of Theorem 1. Here, we show that this reduction can also prove the inapproximability of AVT problem. For any $\epsilon > 0$, the Set Cover problem cannot be approximated in polynomial time within $(n^{1-\epsilon})-$ ratio, unless $P = NP$ [13]. Based on the previous reduction in Theorem 1, every solution of the AVT problem in the instance graph G corresponds to a solution of the *Set Cover* problem. Therefore, it is NP-hard to approximate anchored vertex tracking problem on general graphs within a ratio of $(n^{1-\epsilon})$ when $k \geq 3$. □

Algorithm 2: *The Greedy Algorithm*

Input: $\mathcal{G} = \{G_t\}_1^T$: an evolving graph, l: the allocated size of anchored vertex set, and k: degree constraint
Output: $\mathcal{S} = \{S_t\}_1^T$: the series of anchored vertex sets

```
1  S ← ∅;
2  for each t ∈ [1, T] do
3      i ← 0; S_t ← ∅
4      while i < l do
5          /*  Candidate Anchored Vertex      */
6          for each u ∈ V do
7              /*  Computing Followers         */
8              Compute F_k(u, G_t);
9          u' ← the best anchored vertex in this iteration;
10         S_t ← S_t ∪ u'; i ← i + 1;
11     S ← S ∪ S_t;
12 return S
```

4 The Greedy Algorithm

Considering the NP-hardness and inapproximability of the AVT problem, we first resort to developing a Greedy algorithm to solve the AVT problem. Algorithm 2 summzrizes the major steps of the Greedy algorithm. The core idea of our Greedy algorithm is to iteratively find the l number of best anchored vertices which have the largest number of followers in each snapshot graph $G_t \in \mathcal{G}$ (Lines 2-11). For each $G_t \in \mathcal{G}$ where t is in the range of $[1, T]$ (Line 2), in order to find the best anchored vertex in each of the l iterations (Lines 4), we compute the followers of every candidate anchored vertex by using the *core decomposition* process mentioned in Algorithm 1 (Lines 6-8). Specifically, considering the k-core C_k of G_t, if a vertex u is anchored, then the core decomposition process repeatedly deletes all vertices (except u) of G_t with the degree less than k. Thus, the remaining vertices that do not belong to C_k will be the followers of u with regard to the k-core. In other words, these followers will become the new k-core members due to the anchored vertex selection. From the above process of the Greedy algorithm, we can see that every vertex will be the candidate anchored vertex in each snapshot graph $G_t = (V, E_t)$, and every edge will be accessed in the graph during the process of core decomposition. Hence, the time complexity of the Greedy algorithm is $\mathcal{O}(\sum_{t=1}^{T} l \cdot |V| \cdot |E_t|)$.

Since the Greedy algorithm's time complexity is cost-prohibitive, we need to accelerate this algorithm from two aspects: (i) reducing the number of potential anchored vertices; and (ii) accelerating the followers' computation with a given anchored vertex.

4.1 Reducing Potential Anchored Vertices

In order to reduce the potential anchored vertices, we present the below definition and theorem to identify the quality anchored vertex candidates.

Definition 5 (K-order [36])**.** *Given two vertices $u, v \in V$, the relationship $\preceq$ in K-order index holds $u \preceq v$ in either $core(u) < core(v)$; or $core(u) = core(v)$ and u is removed before v in the process of core decomposition.*

Figure 2 shows a K-order index $O = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3\}$ of graph snapshot G_1 in Figure 1. The vertex sequence $\mathcal{O}_k \in O$ records

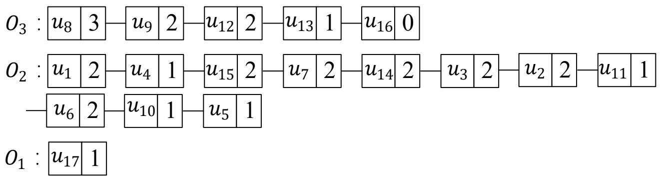


Fig. 2. The K-order O of graph G_1 in Figure 1

all vertices in k-core by following the removing order of core decomposition, *i.e.,* $\mathcal{O}_2$ *records all vertices in 2-core and vertex* u_1 *is removed early than vertex* u_4 *during the process of core decomposition in* G_1.

Theorem 3. *Given a graph snapshot* G_t*, a vertex* x *can become an anchored vertex candidate if* x *has at least one neighbor vertex* v *in* G_t *that satisfies: the neighbor vertex's core number must be k-1 (i.e.,* $core(v) = k-1$*), and* x *is positioned before the neighbor node* v *in K-order (i.e.,* $x \preceq v$*).*

Proof. We prove the correctness of this theorem by contradiction. If $v \preceq x$ in the K-order of G_t, then v will be deleted prior to x in the process of core decomposition in Algorithm 1. In other words, anchoring x will not influence the core number of v. Therefore, v is not the follower of x when $v \preceq x$. On the other hand, it is already proved in [33] that only vertices with core number $k-1$ may be the follower of an anchored vertex. If no neighbor of vertex x has core number $k-1$, then anchoring x will not bring any followers, which is contradicted with the definition of the anchored vertex. From above analysis, we can conclude that the candidate anchored vertex only comes from the vertex x which has at least one neighbor v with core number $k-1$ and behind x in K-order, i.e., $\{x \in V | \exists v \in nbr(x, G_t) \wedge core(v) = k-1 \wedge x \preceq v\}$. Hence, the theorem is proved. □

According to Theorem 3, the anchored vertex candidates will be probed only from the vertices that can bring some followers into the k-core. This also meets the requirement of anchored k-core in Definition 4. Thus, the size of potential anchored vertices at each snapshot graph G_t can be significantly reduced from $|V|$ to $|\{x \in V | \exists v \in nbr(x, G_t) \wedge core(v) = k-1 \wedge x \preceq v\}|$.

Example 5. *Given the graph* G_1 *in Figure 1 and* $k = 3$, u_{15} *can be selected as an anchored vertex candidate because anchoring* u_{15} *would bring the set of followers,* $\{u_{14}\}$*, into the anchored 3-core.*

4.2 Accelerate Followers Computation

To accelerate the computation of followers, a feasible way is to transform the followers' computation into the **core maintenance** problem [23], [36], which aims to maintain the core number of vertices in a graph when the graph is changed. The above problem transformation is based on an observation: given an anchored vertex u, its followers' core number can be increased to k value if $core(u)$ is treated as infinite according to the concept of anchored node.

Therefore, we modify the state-of-the-art core maintenance algorithm, *OrderInsert* [36], to compute the followers of an anchored vertex u in snapshot graph G_t. Explicitly, we first build the K-order of G_t using *core decomposition* method described in Algorithm 1. For each anchored vertex candidate u, we set the core number of u as infinite and denote the set of its followers as V^* initialized to be empty. After that, we iteratively update the core number of u's neighbours and other affected vertices by using the *OrderInsert* algorithm, and record the vertices with core number increasing to k in V^*. Finally, we output V^* as the follower set of u.

Algorithm 3: *ComputeFollower*(G_t, u, $\mathcal{O}(G_t)$)

```
1  K-order O(G_t) = {O_1, O_2, ..., O_max}
2  F_k(u, G_t) ← ∅;
3  for v ∈ nbr(u, G_t) do
4      deg⁻(.) ← 0; V* ← ∅;
5      /* Core phase of the OrderInsert algorithm [36] */
6      if core(v) = k − 1 & u ⪯ v then
7          deg⁺(v) ← deg⁺(v) + 1
8          if deg⁺(v) + deg⁻(v) > k − 1 then
9              remove v from O_{k−1} and append it to V*;
10             for w ∈ nbr(v) ∧ w ∈ O_{k−1} ∧ v ⪯ w do
11                 deg⁻(w) ← deg⁻(w) + 1;
12             Visit the vertex next to v in O_{k−1};
13         else
14             if deg⁻(v) = 0 then
15                 Visit the vertex next to v in O_{k−1};
16             else
17                 for each w ∈ nbr(v) ∧ w ∈ V* do
18                     if deg⁺(w) + deg⁻(w) < k then
19                         remove w from V*;
20                         update deg⁺(w) and deg⁻(w);
21                         nbr(v) ← nbr(v) ∪ nbr(w);
22                         Insert w next to v in O_{k−1};
23                 Visit the vertex with deg⁻(.) = 0 and next
                    to v in O_{k−1};
24     else
25         Continue;
26     Insert vertices in V* to the beginning of O_k in O(G_t);
27     F_k(u, G_t) ← F_k(u, G_t) ∪ V*;
28 return F_k(u, G_t)
```

Besides, we introduce two notations, **remaining degree** (denoted as $deg^+()$) and **candidate degree** (denoted as $deg^-()$), to depict more details of the above followers' computation method. Specifically, for a vertex u in snapshot graph G_t where $core(u) = k-1$, $deg^+(u)$ is the number of remaining neighbors when u is removing during the process of core decomposition, *i.e.,* $deg^+(u) = |v \in nbr(u, G_t) : u \preceq v|$. And $deg^-(u)$ records the number of u's neighbors v included in $\mathcal{O}_{k-1}$ but appearing before u in $\mathcal{O}_{k-1}$, and v is in followers set V^*, *i.e.,* $deg^-(u) = |\{v \in nbr(u, G_t) : v \preceq u \wedge core(v) = k-1 \wedge v \in V^*\}|$. Since, $deg^+(u)$ records the number of u's neighbors after u in the K-order having core numbers larger than or equal to $k-1$, $deg^+(u) + deg^-(u)$ is the upper bound of u's neighbors in the new k-core. Therefore, all vertices s in follower set V^* must have $deg^+(s) + deg^-(s) \geq k$.

The pseudocode of the above process is shown in Algorithm 3. Initially, the K-order of G_t is represented as $O(G_t) = \{\mathcal{O}_1, \mathcal{O}_2, ..., \mathcal{O}_{max}\}$ where max represents the maximum core number of vertices in G_t (Line 1). We then set the followers set of anchored vertex u, $F_k(u, G_t)$ as empty (Line 2). For each

u's neighbours v (Line 3), we iteratively using the *OrderInsert* algorithm [36] to update the core number of v and the other affected vertices due to the core number changes of v, and record the vertices with core number increasing to k in a set V^* (Lines 6-26). After that, we add V^* related to each u's neighbors v into u's follower set $F_k(u, G_t)$ (Line 27). Finally, we output $F_k(u, G_t)$ as the followers set of u (Line 28).

Example 6. *Using Figure 2 and Figure 1, we would like to show the process of followers' computation. Assume $k = 3$, $V^* = \emptyset$, and the K-order, $O = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3\}$, in graph G_1. Initially, the $deg^+(u)$ value of each vertex u is recorded in $O(G_1)$, i.e., $deg^+(u_{14}) = 2$, $deg^-() = 0$ for all vertices in G_1 as V^* is empty. If we anchor the vertex u_{15}, i.e., $core(u_{15}) = \infty$, then we need to update the candidate degree value of u_{15}'s neighbours in $\mathcal{O}_2$, i.e., $deg^-(u_{11}) = 0 + 1$ and $deg^-(u_{14}) = 0 + 1$. We then start to visit the foremost neighbours of u_{15} in $\mathcal{O}_2$, i.e., u_{14}. Since $deg^+(u_{14}) + deg^-(u_{14}) = 2 + 1 \geq 3$ and $deg^+(u_{11}) + deg^-(u_{11}) = 1 + 1 < 3$, we can add u_{14} in V^* and then update the $deg^-()$ of its impacted neighbours. After that, we sequentially explore the vertices s after u_{14} in $\mathcal{O}_2$, and operate the above steps once $deg^+(s) + deg^-(s) \geq 3$. The follower computation terminates when the last vertex in $\mathcal{O}_2$ is processed, i.e., u_{11}. Therefore, the V^* related to u_{14} is $\{u_{14}\}$, and the follower set of u_{15} is $F_k(u_{15}, G_1) = \emptyset \cup V^* = \{u_{14}\}$. Finally, we output the follower set of u_{15}, i.e., $F_k(u_{15}, G_1) = \{u_{14}\}$.*

The time complexity of Algorithm 3 is calculated as follows. The followers' computation of an anchored vertex u can be transformed as the core maintenance problem under inserting edges (u, v) where v is the neighbor of u. Meanwhile, Zhang et al. [36] reported that the core maintenance process while inserting an edge takes $\mathcal{O}(\sum_{v \in V^+} deg(v) \cdot logmax\{|C_{k-1}|, |C_k|\})$ (Lines 6-26), and V^+ is a small set with average size less than 3. Therefore, we conclude that the time complexity of Algorithm 3 is $\mathcal{O}(\sum_{v \in nbr(u)} \sum_{v \in V^+} deg(v) \cdot logmax\{|\mathcal{O}_{k-1}|, |\mathcal{O}_k|\})$. The time complexity of the above followers' computation method is far less than directly using *core decomposition* to compute the followers of a given anchored vertex.

5 Incremental Computation Algorithm

For an evolving graph $\mathcal{G}$, the Greedy approach individually constructs the K-order and iteratively searches the anchored vertex set at each snapshot graph G_t of $\mathcal{G}$. But it does not fully exploit the connection of two neighboring snapshots to advance the performance of solving AVT problem. To address the limitation, in this section, we propose a bounded K-order maintenance approach that can avoid the reconstruction of the K-order at each snapshot graph. With the support of our designed K-order maintenance, we develop an incremental algorithm, called *IncAVT*, to find the best anchored vertex set at each graph snapshot more efficiently.

5.1 The Incremental Algorithm Overview

Let $\mathcal{G} = \{G_1, G_2, .., G_T\}$ be an evolving graph, $\mathcal{S}_t$ be the anchored vertex result set of AVT in G_t where $t \in [1, T]$. E^+ and E^- represent the number of edges to be inserted and deleted at the time when G_{t-1} evolves to G_t. To find out the anchored vertex sets $\mathcal{S} = \{S_t\}_1^T$ of $\mathcal{G}$ using the *IncAVT* algorithm, we first build the K-order of G_1, and then compute the anchored vertex set S_1 of G_1. Next, we develop a bounded K-order maintenance approach to maintain the K-order by considering the change of edges from G_{t-1} to G_t. The benefit of this approach is to avoid the K-order reconstruction at each snapshot G_t. Meanwhile, during the process of K-order maintenance, we use vertex sets V_I and V_R to record the vertices that are impacted by the edge insertions and edge deletions, respectively. After that, we iteratively find the l number of best anchored vertices in each snapshot graph G_t, while the potential anchored vertices are selected to probe from V_I, V_R, and S_{t-1}. The l anchored vertices are recorded in $\mathcal{S}_t$. Finally, we output $\mathcal{S} = \{S_t\}_1^T$ as the result of the AVT problem.

5.2 Bounded K-order Maintenance Approach

In this subsection, we devise a bounded K-order maintenance approach to maintain the K-order while the graph evolving from G_{t-1} to G_t, i.e., $t \in [2, T]$. Our bounded K-order maintenance approach consists of two components: (1) **EdgeInsert**, handling the K-order maintenance while inserting the edges E^+; and (2) **EdgeRemove**, handling the K-order maintenance while deleting the edges E^-.

5.2.1 Handling Edge Insertion

If we insert the edges in E^+ into G_{t-1}, then the core number of each vertex in G_{t-1} either remains unchanged or increases. Therefore, the k-core of snapshot graph G_{t-1} is part of the k-core of snapshot graph G_t where $G_t = G_{t-1} \oplus E^+$. The following lemmas show the update strategies of core numbers of vertices when the edges are added.

Lemma 1. *Given a new edge (u, v) that is added into G_{t-1}, the remaining degree of u increases by 1, i.e., $deg^+(u) = deg^+(u) + 1$, if $u \preceq v$ holds.*

Proof. From Section 4.2 of the remaining degree of a vertex, we get $deg^+(u) = |\{v \in nbr(u) \mid u \preceq v\}|$. Inserting an edge (u, v) into graph snapshot G_{t-1} brings one new neighbour v to u where $u \preceq v$ in the K-order of G_{t-1}, i.e., $O(G_{t-1})$. Therefore, $deg^+(u)$ needs to increase by 1 after inserting (u, v) into G_{t-1}. □

Example 7. *Consider the snapshot graph G_1 in Figure 1, if we add a new edge (u_2, u_5) into G_1 where $u_2 \preceq u_5$ (mentioned in Figure 2), then the remaining degree of u_2, $deg^+(u_2) = deg^+(u_2) + 1 = 3$.*

Lemma 2. *Let $deg^+(u)$ and $core(u)$ be the remaining degree and core number of vertex u in snapshot graph G_t respectively. Suppose we insert a new edge (u, v) into G_t and update $deg^+(u)$. Thus, the core number $core(u)$ of u may increase by 1 if $core(u) < deg^+(u)$. Otherwise, $core(u)$ remains unchanged.*

Proof. We prove the correctness of this lemma by contradiction. From Definition 2 and the definition of remaining degree in Section 4.2, we know that if u's core number do not need to update after inserting edge (u, v) into G_{t-1}, then the number of u's neighbours v with $u \preceq v$ must be no more than $core(u)$. Therefore, the value of updated $deg^+(u)$ should be no more than $core(u)$, which is contradicted with the fact that $core(u) < deg^+(u)$. □

Example 8. *Considering a vertex u_2 in graph G_1, we can see $deg^+(u_2) = 2$, and $core(u_2) = 2$ as shown in Figure 1 and Figure 2. If an edge (u_2, u_5) is inserted into G_1, we can get $deg^+(u_2) = deg^+(u_2) + 1 = 3$ (refer Lemma 1). Since $core(u_2) = 2 < deg^+(u_2) = 3$, the $core(u_2)$ may increase by 1 according to Lemma 2.*

Algorithm 4: *EdgeInsert*(G'_t, O, E^+, k)

```
1  i ← 1, V_I ← ∅, m ← 0, O' ← ∅;
2  for each e = (u, v) & e ∈ E^+ do
3      m ← max{m, min(core(u), core(v))};
4      u ⪯ v ? deg^+(u)+ = 1 : deg^+(v)+ = 1;
5  while i ≤ m do
6      V_C ← ∅, deg^-(.) ← 0;
7      u* ← the first vertex of O_i ∈ O;
8      while u* ≠ nil do
9          if deg^+(u*) + deg^-(u*) > i then
10             remove u* from O_i; append u* into V_C;
11             if i = k − 1 then
12                 add u* into V_I
13             for each
                 v ∈ nbr(u*, G'_t) ∧ core(v) = i ∧ u* ⪯ v do
14                 deg^-(v) ← deg^-(v) + 1;
15         else
16             if deg^-(u*) = 0 then
17                 remove u* from O_i; append u* to O_{i'};
18             else
19                 deg^+(u*) ← deg^+(u*) + deg^-(u*);
20                 deg^-(u*) ← 0;
21                 remove u* from O_i; append u* to O_{i'};
22                 update the deg^+(.) of u*'s neighbors;
23         u* ← the vertex next to u* in O_i;
24     for v ∈ V_C do
25         deg^-(v) ← 0; core(v) ← core(v) + 1;
26         if i = k − 1 then
27             remove v from V_I;
28     insert vertex set V_C into the beginning of O_{i+1};
29     if i = k − 2 then
30         V_I ← V_I ∪ V_C;
31     add O_{i'} to new K-order O' in G'_t;
32     i ← i + 1;
33 return the K-order O' in G'_t, and V_I
```

We present the *EdgeInsert* algorithm for K-order maintenance. It consists of three main steps. Firstly, for each vertex u relating to the inserting edges $(u, v) \in E^+$, we need to update its remaining degree, *i.e.,* $deg^+(u)$ (refer Lemma 1). Then, we identify the vertices impacted by the insertion of E^+ and update its remaining degree value, core number, and positions in K-order (refer Lemma 2). This step is the core phase of our algorithm. Finally, we add the vertex u into the vertex set V_I if u has the updated core number $core(u) = k - 1$ after inserting E^+. This is because the followers only come from vertices with core number $k - 1$ (refer Theorem 3).

The detailed description of our *EdgeInsert* algorithm is outlined in Algorithm 4. The inputs of the algorithm are snapshot graph G_{t-1} where $t \in [2, T]$, the K-order $O = \{\mathcal{O}_1, \mathcal{O}_2, .., \mathcal{O}_k, ..\}$ of G_{t-1}, the edge insertion E^+, and a positive integer k. Initially, for each inserted edge $(u, v) \in E^+$, we increase the remaining degree of u by 1 where vertex $u \preceq v$ (refer Lemma 1), use m to record the maximum core number of all vertices related to E^+ (Lines 2-4). Next, for $i \in [0, m]$, we iteratively identify the vertices in $\mathcal{O}_i \in O$ whose core number increases after the insertion of E^+, and we also update $\mathcal{O}_i$ of K-order (Lines 5-32). Here, a new set V_C is initialized as empty and it will be used to maintain the new vertices whose core number increases from $i - 1$ to i. And then, we start to select the first vertex u^* from $\mathcal{O}_i$ (Line 7). In the inner *while* loop, we visit the vertices in $\mathcal{O}_i$ in order (Lines 8-22). The visited vertex u^* must satisfy one of the three conditions: (1) $deg^+(u^*) + deg^-(u^*) > i$; (2) $deg^+(u^*) + deg^-(u^*) \leq i \wedge deg^-(u^*) = 0$; (3) $deg^+(u^*) + deg^-(u^*) \leq i \ \wedge \ deg^-(u^*) > 0$. For condition (1), the core number of the visited vertex u^* may increase. Then, we remove u^* from $\mathcal{O}_i$ and add it into V_C. Besides, the candidate degree of each neighbour v of u^* should increase by 1 if $u^* \preceq v$ (Lines 9-14). For condition (2), the core number of u^* will not change. So we remove u^* from the previous $\mathcal{O}_i$ and append it into $\mathcal{O}_{i'}$ of the new K-order O' of graph $G'_t = G_{t-1} \oplus E^+$ (Lines 16-17). For condition (3), we can identify that u^*'s core number will not increase. So we need to update the remaining degree and candidate degree of u^*, and remove u^* from $\mathcal{O}_i$ and append it to $\mathcal{O}_{i'}$. We also need to update the remaining degree of the neighbours of u^* (Lines 19-22). After that, V_I maintains the vertices that are affected by the edge insertion, and these vertices have core number $k - 1$ in new K-order O' of graph G'_t (Lines 24-30). Finally, when the outer *while* loop terminates, we can output the maintained K-order and the affected vertices set V_I (Line 33).

5.2.2 Handling Edge Deletion

Here, we present the procedure of K-order maintenance for edge deletions. The following definitions and lemmas show the update strategies of core numbers of vertices when the edges are deleted.

Lemma 3. *Suppose an edge (u, v) is deleted while graph evolves from G_{t-1} to G_t, then the remaining degree of u from G_{t-1} to G_t decreases by 1, i.e., $deg^+(u) = deg^+(u) - 1$, if $u \preceq v$ holds.*

Proof. From Section 4.2 of the remaining degree of a vertex, we get $deg^+(u) = |\{v \in nbr(u) \mid u \preceq v\}|$. Deleting an edge (u, v) from graph snapshot G_{t-1} evolving to G_t removes one neighbour v of u where $u \preceq v$ in the K-order of G_t. Therefore, the $deg^+(u)$ needs to decrease 1 after deleting (u, v) from G_{t-1}. □

Example 9. *Consider the snapshot graph G_1 and G_2 in Figure 1, if we remove edge (u_2, u_{11}) from G_1 to G_2 where $u_2 \preceq u_{11}$ (mentioned in Figure 2), then the remaining degree of u_2 will decrease from 2 to 1.*

We then introduce an important notion, called *max core degree*, and the related lemma.

Definition 6 (Max core degree [28])**.** *Given a graph G_t, the max-core degree of a vertex u in G_t, denoted as $mcd(u)$, is the number of u's neighbours whose core number no less than $core(u)$.*

Example 10. *Consider the snapshot graph G_1 in Figure 1, we have $core(u_9) = 3$, $core(u_{14}) = 2$, $core(u_{15}) = 2$, $core(u_{16}) = 3$, and $core(u_{17}) = 1$. Therefore, the max core degree of vertex u_{14} is 3 due to 3 of u_{14}'s neighbors $\{u_9, u_{15}, u_{16}\}$ has core number no less than $core(u_{14})$.*

Based on k-core definition (refer Definition 1), $mcd(u) < core(u)$ means that u does not have enough neighbors who meet the requirement of k-core. Thus, u itself cannot stay in k-core as well. Therefore, it can conclude that for a vertex, its max core degree is always larger than or equal to its core number, *i.e,* $mcd(u) \geq core(u)$.

Lemma 4. *Let $mcd(u)$ and $core(u)$ be the Max-core degree and core number of vertex u in snapshot graph G_t. Suppose we delete an edge (u, v) from G_t and the updated $mcd(u)$. Thus, the core*

Algorithm 5: *EdgeRemove*(G'_t, O', E^-, k)

1 /* $mcd(u)$ is the number of u's neighbour v with $core(u) \leq core(v)$ */
2 $G_t := G'_t \oplus E^-$, $O' = \{\mathcal{O}_1, \mathcal{O}_2, ...\}$;
3 $V_R \leftarrow \emptyset$, and $m \leftarrow 0$;
4 let Q be an empty queue and $V^* = \{V_1, V_2, ..\}$, $V_i \in V^*$ be the empty list;
5 /* identify the vertex need to remove from O' */
6 **for** *each* $e = (u, v)$ & $e \in E^-$ **do**
7     $u' \leftarrow u$ if $u \preceq v$, otherwise v;
8     $deg^+(u') \leftarrow deg^+(u') - 1$
9   compute $mcd(u', G_t)$ of u'; $i \leftarrow core(u')$;
10   **if** $mcd(u', G_t) < core(u')$ **then**
11   remove u' from O'_i, enqueue u' to Q;
12         $core(u') \leftarrow core(u') - 1$;
13 **while** *Q is not empty* **do**
14   dequeue u from Q, $i \leftarrow core(u)$;
15   append u to V_i, $m \leftarrow max\{m, i\}$;
16   **for** $u' \in nbr(u, G'_t) \wedge core(u') > i$ **do**
17   repeat lines 9-12;
18 /* update the k-order O' */
19 **for** $i \leftarrow m$ *to* 1 **do**
20   **for** *each* $u \in V_i$ *in order* **do**
21         $deg^+(u) \leftarrow 0$;
22   **for** $u' \in nbr(u, G_t)$ **do**
23   **if** $core(u') > core(u) \vee u' \in V_i$ **then**
24                 $deg^+(u) \leftarrow deg^+(u) + 1$;
25   recompute $deg^+(u')$;
26   append u to the end of $\mathcal{O}_i$;
27 $V_R \leftarrow V_{k-1}$, $O(G_t) \leftarrow O'$;
28 **return** *the K-order $O(G_t)$ of G_t, and V_R*

number $core(u)$ of u may decrease by 1 if $mcd(u) < core(u)$. Otherwise, $core(u)$ remain unchanged.

Proof. Based on Definition 1 and Definition 2, the core number of vertex u is identified by the number of it's neighbours with core number no less than u. Moreover, a vertex u must have at least $core(u)$ number of neighbours with core number no less than $core(u)$. From Definition 6, the max core degree of a vertex u is the number of u's neighbour with core number no less than u, i.e, $mcd(u) = |\{v \mid v = nbr(u) \wedge core(v) \geq core(u)\}|$. Therefore, we can conclude that $mcd(u) \geq core(u)$ always holds. Hence, if $mcd(u) < core(u)$ after delete an edge from G_t and update $mcd(u)$, then $core(u)$ also needs to be decreased by 1 to ensure $mcd(u) > core(u)$ in the changed graph. □

Example 11. *Consider the vertex u_{17} of evolving graph $\mathcal{G} = \{G_1, G_2, .., G_T\}$ in Figure 1. Both of the core number and the max degree of u_{17} are 1 in G_2, i.e., $core(u_{17}) = 1$ and $mcd(u_{17}) = 1$. From G_2 to G_3, the edge (u_{14}, u_{17}) is deleted, and u_{17}'s max degree, $mcd(u_{17})$ is decreased to 0 due to no neighbor of u_{17} having core number no less than u_{17} in G_3. Then we have $mcd(u_{17}) = 0 < core(u_{17})$. At this situation, the core number of u_{17} also decreases from 1 to 0 after the deletion of edge (u_{14}, u_{17}).*

The *EdgeRemove* algorithm is presented in Algorithm 5. The inputs of the algorithm are the graph G'_t constructed by G_{t-1} with the insertion edges of E^+, i.e., $G'_t = G_{t-1} \oplus E^+$, and O' is the K-order of G'_t. The main body of Algorithm 5 consists of three steps. In the first step (Lines 6-17), we identify the vertices that needs to be removed from their previous position of K-order O' after the edges deletion. Specifically, we first update the remaining degree of vertices related with E^- to reflect the deletion of edges in E^- based on Lemma 3 (Lines 6-8), and then compute the max core degree of these vertices (Line 9). Meanwhile, we add the influenced vertex u related to the deleting edges, i.e., $mcd(u) < core(u)$, into a queue Q. All vertices in Q need to update their core numbers based on Lemma 4 (Lines 10-12). After that, the algorithm recursively probes each neighboring vertex v of vertices in Q, and adds v into the vertex set V^* if $mcd(v) < core(v)$ (Lines 13-17). In the second step, we maintain the K-order O' by adjusting the position of vertices in V^*, which is identified in Step 1, to reflect the edges deletion of E^- (Lines 19-26). In details, for each $u \in V_i$, we update the $deg^+(.)$ of u and its neighbours, remove u from $\mathcal{O}'_t$, and insert u to the end of $\mathcal{O}'_{t-1}$. In the final step, we use V_R to record the vertices that may become the potential followers once the anchored vertices are selected, i.e., these vertices' core number becomes $k-1$ in the new K-order O' (Line 27).

Algorithm 6: *IncAVT*

Input: $\mathcal{G} = \{G_t\}_1^T$: an evolving graph, l: the allocated size of anchored vertex set, and k: degree constraint
Output: $\mathcal{S} = \{S_t\}_1^T$: the series of anchored vertex sets

1 Build the K-order $O(G_1)$ of G_1; /* using Algorithm 1 */
2 Compute the anchored vertex set S_1 of G_1 with size l using Algorithm 2;
3 $\mathcal{S} := \{S_1\}$; $t := 2$;
4 **while** $t < T$ **do**
5     $G'_t := G_{t-1} \oplus E^+$, $S_t \leftarrow S_{t-1}$;
6   /* maintain K-order by using Algorithm 4, 5 */
7     $(O', V_I) \leftarrow$ *EdgeInsert*$(G'_t, O(G_{t-1}), E^+, k)$;
8     $(O(G_t), V_R) \leftarrow$ *EdgeRemove*(G'_t, O', E^-, k);
9   **for** *each* $u \in S_{t-1}$ **do**
10   compute $F_k(S_t, G_t)$, $F \leftarrow |F_k(S_t, G_t)|$;
11         $F_{max} \leftarrow 0$, $u' \leftarrow u$;
12   **for** *each* $\{v | v \in \{V_I \cup V_R \cup nbr(V_I \cup V_R) \setminus C_k(G_t)\} \wedge \{\exists u \in nbr(v) \wedge core(u) = k-1 \wedge v \preceq u\}\}$ **do**   */* Theorem 3 */*
13   **if** $F_{max} < F_k(S_t \setminus u \cup v, G_t)$ **then**
14                 $F_{max} \leftarrow F_k(S_t \setminus u \cup v, G_t)$, $u' \leftarrow v$;
15   **if** $F_{max} > F$ **then**
16   remove u from S_t, add u' to S_t;
17     $\mathcal{S} := \mathcal{S} \cup S_t$; $t \leftarrow t + 1$;
18 **return** $\mathcal{S}$

5.3 The Incremental Algorithm

Base on the above K-order maintenance strategies and the impacted vertex sets V_I and V_R, we propose an efficient incremental algorithm, *IncAVT*, for processing the *AVT* query. Algorithm 6 summarizes the major steps of *IncAVT*. Given an evolving graph $\mathcal{G} = \{G_t\}_1^T$, the allocated size of selected anchored vertex set l, and a positive integer k, the *IncAVT* algorithm returns a series of anchored vertex set $S = \{\mathcal{S}_t\}_1^T$ of $\mathcal{G}$ where each S_t has size l. Initially, we build the K-order $O(G_1)$ of G_1 by using Algorithm 1, and then compute the anchored vertex set S_1 of G_1 by using Algorithm 2 where T is set as 1 (Lines 1-3). The

TABLE 2
Statistics of the selected datasets

Dataset	Nodes	Edges	d_{avg}	T	Type
email-Enron	36,692	183,831	10.02	30	Communication
Gnutella	62,586	147,878	4.73	30	P2P Network
Deezer	41,773	125,826	6.02	30	Social Network

TABLE 3
Parameters and their values

Parameter	Values	Default
l	$[5, 10, 15, 20]$	10
k	$[2, 3, 4, 5]$ or $[5, 10, 15, 20]$	3 or 10
T	$[0 - 30]$	30

while loop at lines 4-17, computes the anchored vertex set of each snapshot graph $G_t \in \mathcal{G}$. E^+ and E^- represent the edges insertion and edges deletion between G_{t-1} to G_t respectively, and we initialize the anchored vertex set S_t in G_t as S_{t-1} (Line 5). The K-order is maintained by using Algorithm 4 while considering the edge insertion E^+ to G_{t-1} and consequently, the vertex set V_I is returned to record the vertices, which is impacted by inserting E^+ and has core number $k-1$ in the updated K-order (Line 7). Similarly, we use Algorithm 5 to update the K-order while considering the edges deletion of E^- and use V_R to record the vertices which has core number k-1 and impacted by the edge deletion (Line 8). Next, an inner *for* loop is to track the anchored vertex set of G_t (Lines 9-16). More specifically, we first compute S_t's followers set size F (Line 10). Then, for each vertex u in S_{t-1}, we only probe the vertices v in vertex set $\{V_I \cup V_R \cup nbr(V_I \cup V_R) \setminus C_k(G_t)\}$ based on Theorem 3 (Lines 9-14). If the number of followers of anchored vertex set $\{S_t \setminus u \cup v\}$ is large than F, we then update S_t by using v to replacement u (Lines 15-16). After the inner *for* loop finished, we add the anchored vertex set S_t of G_t into $\mathcal{S}$ (Line 17). The *IncAVT* algorithm finally returns the series of anchored vertex set $\mathcal{S}$ as the final result (Line 18).

6 Experimental Evaluation

In this section, we present the experimental evaluation of our proposed approaches for the AVT problem: the Greedy algorithm (**Greedy**); and the incremental algorithm (**IncAVT**).

6.1 Experimental Setting

Algorithms. To the best of our knowledge, no existing work investigates the *Anchored Vertex Tracking* (AVT) problem. To further validate, we compared with two baselines adapted from the existing works: (i) **OLAK**, which is proposed in [33] to find out the best anchored vertices at each snapshot graphs. (ii) **RCM**, which is the state-of-the-art anchored k-core algorithm that proposed in [20], for tracking the best anchored vertices selection at each snapshot graphs.

Datasets. We conduct the experiments using three publicly available datasets from the *Stanford Large Network Dataset Collection*[1]: *email-Enron*, *Gnutella*, and *Deezer*. The statistics of the datasets are shown in Table 2. As the orginal datasets do not contain temporal information, we thus generate 30 synthetic time evolving snapshots for each dataset by randomly inserting new edges and removing old edges. More specifically, we use it as the first snapshot T_1. Then, we randomly remove $100 - 250$ edges from T_1, denoted as T_1' and randomly add $100 - 250$ new edges into T_1', denoted as T_2. By repeating the similar operation, we generate 30 snapshots for each dataset.

1. http://snap.stanford.edu/

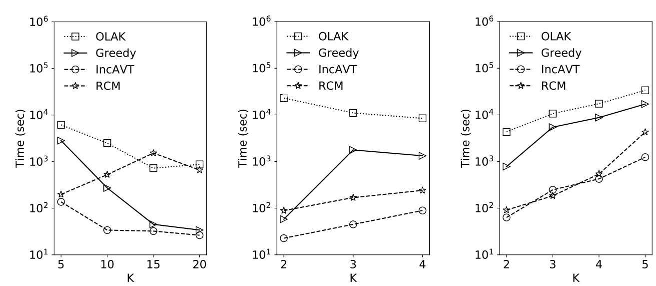
Parameter Configuration. Table 3 presents the parameter settings. We consider three parameters in our experiments: core number k, anchored vertex size l, and the number of snapshots T. In each experiment, if one parameter varies, we use the default values for the other parameters. Besides, we use the sequential version of the RCM algorithm in the following discussion and results. All the programs are implemented in C++ and compiled with GCC on Linux. The experiments are executed on the same computing server with 2.60GHz Intel Xeon CPU and 96GB RAM.

6.2 Efficiency Evaluation

In this section, we study the efficiency of the approaches for the AVT problem regarding running time under different parameter settings.

6.2.1 Varying Core Number k

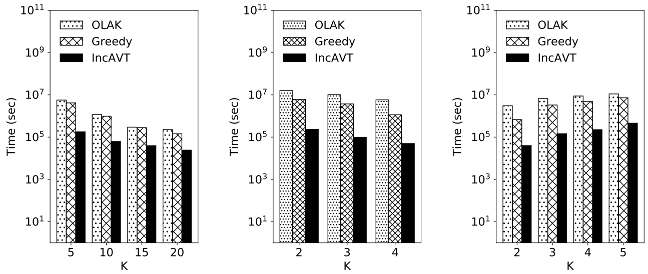
We compare the performance of different approaches by varying k. Due to the various average degree of three datasets, we set different k for them. Figure 3(a) - 3(c) show the running time of *OLAK*, *Greedy*, *IncAVT*, and *RCM*, on the three datasets. From the results, we can see that *Greedy* and *RCM* perform faster than *OLAK*, and *IncAVT* performs one to two orders of magnitude faster than the other three approaches in all datasets. As expected, we do not observe any noticeable trend from all three approaches when k is varied. This is because, in some networks, the increase of the core number may not induce the increasing of the size of k-core subgraph and the number of candidate anchored vertices needing to probe.



(a) email-Enron (b) Gnutella (c) Deezer

Fig. 3. Time cost of algorithms with varying k

Since the performance of *Greedy*, *OLAK*, and *IncAVT* are highly influenced by the number of visited candidate anchored vertices in algorithm execution, we also investigate the number of candidate anchored vertices that need to be probed for these approaches in different datasets. Figure 4(a) - 4(c) show the number of visited candidate anchored vertices for the three approaches when k is varied. We notice that *OLAK* visits much more number of candidate anchored vertices than the other two approaches, and *IncAVT* shows the minimum number of visited candidate anchored vertices.

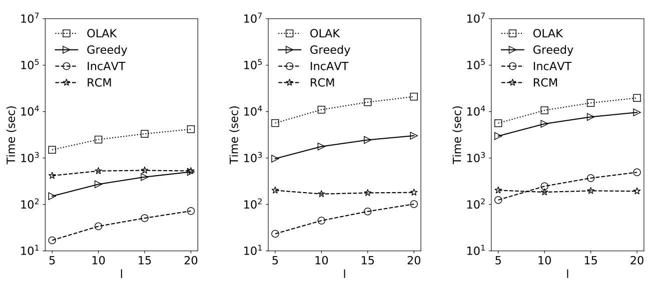


(a) email-Enron (b) Gnutella (c) Deezer

Fig. 4. Number of candidate anchored vertices with varying k

6.2.2 Varying Anchored Vertex Set Size l

Figure 5(a) - 5(c) show the average running time of the approaches by varying l from 5 to 20. As we can see, *IncAVT* is significantly efficient than *Greedy* and *OLAK*. Specifically, *IncAVT* can reduce the running time by around 36 times and 230 times compared with *Greedy* and *OLAK* respectively under different l settings on the *Gnutella* dataset. The improvements are built on the facts that *IncAVT* visits less number of candidate anchored vertices than *Greedy* and *OLAK*. Besides, *IncAVT* performs far well than *RCM* in *Enron* and *Gnutella*. Meanwhile, the running time of *IncAVT* is slightly higher than *RCM* in *Deezer*. From the result, we notice that the performance of above approaches are also influenced by the type of networks.



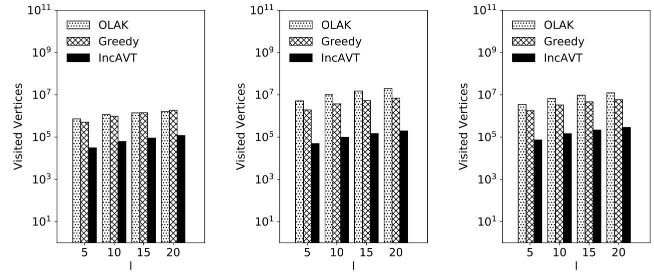
(a) email-Enron (b) Gnutella (c) Deezer

Fig. 5. Time cost of algorithms with varying l

Figure 6(a) - 6(c) show the total number of visited anchored vertices. We can see that *IncAVT* visits much less anchored vertices than the other two methods even though it shows a slightly increased number of visited vertices as l increases. The visited candidate anchored vertices in *OLAK* is around 2.8 times more than *Greedy*, and 102 times more than *IncAVT* on the Gnutella dataset. The total number of visited candidate anchored vertex set in *IncAVT* is minimum during the anchored vertex tracking process across all the datasets.

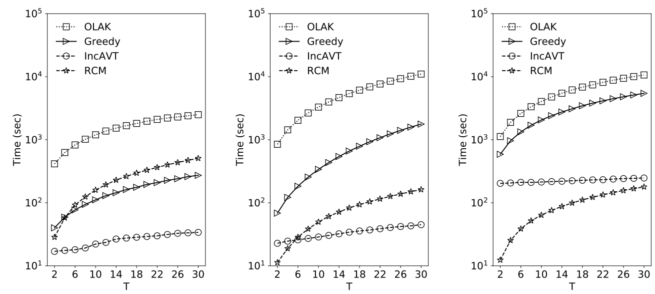
6.2.3 Varying Snapshot Size T

We also test our proposed algorithms by varying T from 2 to 30. Figure 7(a) - 7(c) present the running time with varied values of T. The results show similar findings that *IncAVT* outperforms *OLAK*, *Greedy*, and *RCM* significantly in efficiency as it utilizes the smoothness of the network structure in evolving network to reduce the visited candidate anchored vertices. Meanwhile, the speed of running time increasing in *IncAVT* is much slower than the other three algorithms in each snapshot when T increases. In other words, the performance advantage of *IncAVT* will enhance with the increase of the network snapshot size.



(a) email-Enron (b) Gnutella (c) Deezer

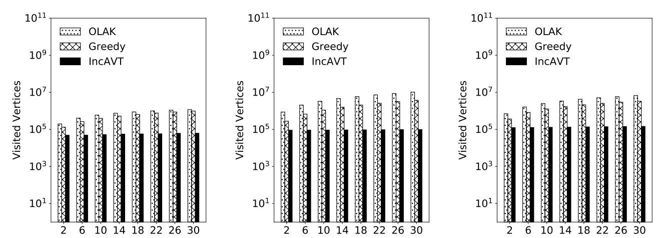
Fig. 6. Number of candidate anchored vertices with varying l



(a) email-Enron (b) Gnutella (c) Deezer

Fig. 7. Time cost of algorithms with varying T

Figure 8(a) - 8(c) report our further evaluation on the number of visited candidate anchored vertices when T is varied. As expected, *IncAVT* has the minimum number of visited candidate anchored vertices than the other two approaches. What is more, the number of visited candidate anchored vertices by *IncAVT* in each snapshot is steady than *Greedy* and *OLAK*.

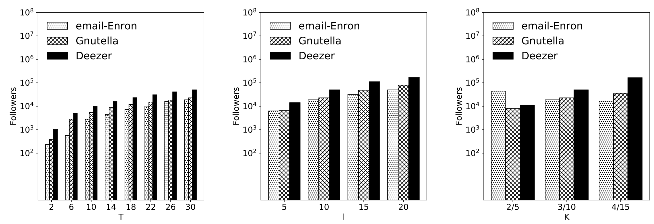


(a) email-Enron (b) Gnutella (c) Deezer

Fig. 8. Number of candidate anchored vertices with varying T

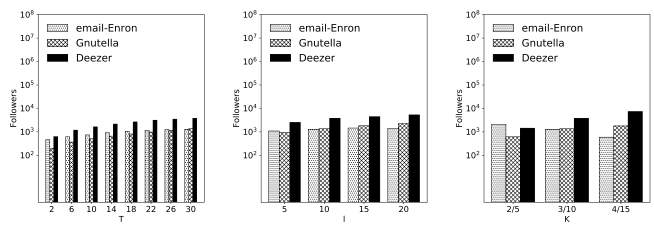
6.3 Effectiveness Evaluation

In this experiment, we evaluate the number of followers produced by the AVT problem with different datasets and approaches in Figure 9 - Figure 12 by varying one parameter and setting the other two as defaults. As we can see, the number of followers in each snapshot discovered by all four approaches increases rapidly in all datasets with the evolving of the network. For example, in Figure 9(a), the follower size in the *Deezer* dataset is about one thousand when $T = 2$ and goes up to 50,000 when $T =$



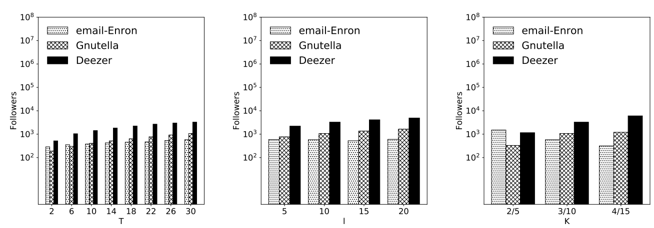
(a) Vary (T) (b) Vary (l) (c) Vary (k)

Fig. 9. Number of followers discovered by *IncAVT*



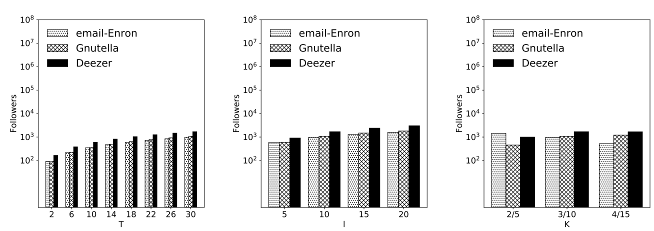
(a) Vary (T) (b) Vary (l) (c) Vary (k)

Fig. 10. Number of followers discovered by *OLAK*



(a) Vary (T) (b) Vary (l) (c) Vary (K)

Fig. 11. Number of followers discovered by *Greedy*



(a) Vary (T) (b) Vary (l) (c) Vary (K)

Fig. 12. Number of followers discovered by *RCM*

30. Similar pattern can also be found in Figure 9(b) as more followers can be found when we increase l with the other two parameters fixed. As expected, we do not observe a noticeable followers trend from all four approaches when varying k. This is because the anchored k-core size is highly related to the network structure. From the above experimental results, we can conclude that tracking the anchored vertices in an evolving network is necessary to maximize the benefits of expanding the communities.

7 Related work

k-core Decomposition: The model of k-core was first introduced by Seidman et al. [29], and has been widely used as a metric for measuring the structure cohesiveness of a specific community in the topic of social contagion [30], user engagement [5], [25], Internet topology [2], [9], influence studies [17], [22], and graph clustering [14], [23]. The k-core can be computed by using core decomposition algorithm, while the core decomposition is to efficiently compute for each vertex its core number [4]. Besides, with the dynamic change of the graph, incrementally computing the new core number of each affected vertices is known as core maintenance, which has been studied in [1], [3], [15], [23], [28], [35], [36].

Anchored k-core Problem: User engagement in social networks has attracted much attention while quantifying user engagement dynamics in social networks is usually measured by using k-core [5], [7], [11], [20], [24], [31], [32], [34], [37]. Bhawalker et al. [5] first introduced the problem of anchored k-core, which was inspired by the observation that the user of a social network remains active only if her neighborhood meets some minimal engagement level: in k-core terms. Specifically, the anchored k-core problem aims to find a set of anchored vertices that can further induce maximal anchored k-core. Then, Chitnis et al. [11] proved that the anchored k-core problem on general graphs is solvable in polynomial time for $k \leq 2$, but is NP-hard for $k > 2$. Later, Zhang et al. in 2017 [36] proposed an efficient greedy algorithm by using the vertex deletion order in k-core decomposition, named OLAK. In the same year, another research [34] studied the collapsed k-core problem, which aims to identify critical users that may lead a large number of other users to drop out from a social network once they leave. Zhou et al. [37] introduced a notion of resilience in terms of the stability of k-cores while the vertex or edges are randomly deleting, which is close to the anchored k-core problem. Cai et al. [7] focused on a new research problem of anchored vertex exploration that considers the users' specific interests, structural cohesiveness, and structure cohesiveness, making it significantly complementary to the anchored k-core problem in which only the structure cohesiveness of users is considered. Very recently, Ricky et al. in 2020 [20] proposed a novel algorithm by selecting anchors based on the measure of anchor score and residual degree, called Residual Core Maximization (RCM). The RCM algorithm is the state-of-the-art algorithm to solve the anchored k-core problem. However, all of the works mentioned above on anchored k-core only consider the static social networks. To the best of our knowledge, our work is the first to study the anchored vertex tracking problem to find the anchored vertices at each timestamp of evolving networks.

8 Conclusions

In this paper, we focus on a novel problem, namely the anchored vertex tracking (AVT) problem, which is the extension of the *anchored k-core* problem towards dynamic networks. The AVT problem aims at tracking the anchored vertex set dynamically such that the selected anchored vertex set can induce the maximum anchored k-core at any moment. We develop a Greedy algorithm to solve this problem. We further accelerate the above algorithm from two aspects, including (1) reduce the potential anchored vertices that need probing; and (2) propose an algorithm to improve the followers' computation efficiency with a given anchored vertex. Moreover, an incremental computation method is designed by utilizing the smoothness of the evolution of the network structure and the well-designed Bounded K-order maintenance methods in an evolving graph. Finally, the extensive performance evaluations also reveal the practical efficiency and effectiveness of our proposed methods in this paper.

REFERENCES

[1] H. Aksu, M. Canim, Y. Chang, I. Korpeoglu, and Ö. Ulusoy. Distributed k -core view materializationand maintenance for large dynamic graphs. *IEEE Trans. Knowl. Data Eng.*, 26(10):2439–2452, 2014.

[2] 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.

[3] W. Bai, Y. Chen, and D. Wu. Efficient temporal core maintenance of massive graphs. *Inf. Sci.*, 513:324–340, 2020.

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

[5] K. Bhawalkar, J. M. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: The anchored k-core problem. In *ICALP*, pages 440–451, 2012.

[6] K. Bhawalkar, J. M. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: The anchored k-core problem. *SIAM J. Discrete Math.*, 29(3):1452–1475, 2015.

[7] T. Cai, J. Li, N. A. H. Haldar, A. Mian, J. Yearwood, and T. Sellis. Anchored vertex exploration for community engagement in social networks. In *ICDE*, pages 409–420, 2020.

[8] C. V. Cannistraci, G. Alanis-Lobato, and T. Ravasi. From link-prediction in brain connectomes and protein interactomes to the local-community-paradigm in complex networks. *Scientific Reports*, 3(1):1613, 2013.

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

[10] X. Chen, G. Song, X. He, and K. Xie. On influential nodes tracking in dynamic social networks. In *SDM*, pages 613–621, 2015.

[11] 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.

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

[13] U. Feige. A threshold of ln *n* for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

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

[15] H. Jin, N. Wang, D. Yu, Q. Hua, X. Shi, and X. Xie. Core maintenance in dynamic graphs: A parallel approach based on matching. *IEEE Trans. Parallel Distrib. Syst.*, 29(11):2416–2428, 2018.

[16] R. M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103, 1972.

[27] G. Rossetti, L. Pappalardo, R. Kikas, D. Pedreschi, F. Giannotti, and M. Dumas. Community-centric analysis of user engagement in skype social network. In *ASONAM*, pages 547–552, 2015.

[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*, 6:888–893, 2010.

[18] G. Kossinets and D. Watts. Origins of homophily in an evolving social network. *American Journal of Sociology*, 115(2):405–450, 2009.

[19] G. Kossinets and D. J. Watts. Empirical analysis of an evolving social network. *Science*, 311(5757):88–90, 2006.

[20] R. Laishram, A. E. Sariyüce, T. Eliassi-Rad, A. Pinar, and S. Soundarajan. Residual core maximization: An efficient algorithm for maximizing the size of the *k*-core. In *SDM*, pages 325–333, 2020.

[21] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In *SIGKDD*, pages 462–470, 2008.

[22] C. Li, L. Wang, S. Sun, and C. Xia. Identification of influential spreaders based on classified neighbors in real-world complex networks. *Appl. Math. Comput.*, 320:512–523, 2018.

[23] R. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *IEEE Trans. Knowl. Data Eng.*, 26(10):2453–2465, 2014.

[24] Q. Linghu, F. Zhang, X. Lin, W. Zhang, and Y. Zhang. Global reinforcement of social networks: The anchored coreness problem. In *SIGMOD*, pages 2211–2226, 2020.

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

[26] M. Newman. Clustering and preferential attachment in growing networks. *Physical Review E*, 64(2):025102, 2001.

[28] 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.

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

[30] J. Ugander, L. Backstrom, C. Marlow, and J. M. Kleinberg. Structural diversity in social contagion. *Proc. Natl. Acad. Sci. U.S.A.*, 109(16):5962–5966, 2012.

[31] 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.

[32] F. Zhang, C. Li, Y. Zhang, L. Qin, and W. Zhang. Finding critical users in social communities: The collapsed core and truss problems. *IEEE Trans. Knowl. Data Eng.*, 32(1):78–91, 2020.

[33] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin. OLAK: an efficient algorithm to prevent unraveling in social networks. *PVLDB*, 10(6):649–660, 2017.

[34] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Finding critical users for social network engagement: The collapsed k-core problem. In *AAAI*, pages 245–251, 2017.

[35] Y. Zhang and J. X. Yu. Unboundedness and efficiency of truss maintenance in evolving graphs. In *SIGMOD*, pages 1024–1041, 2019.

[36] Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin. A fast order-based approach for core maintenance. In *ICDE*, pages 337–348, 2017.

[37] Z. Zhou, F. Zhang, X. Lin, W. Zhang, and C. Chen. K-core maximization: An edge addition approach. In *IJCAI*, pages 4867–4873, 2019.