Incremental Graph Computation: Anchored Vertex Tracking in Dynamic Social Networks

Taotao Cai, Shuiqiao Yang, Jianxin Li, Quan Z. Sheng, Jian Yang, Xin Wang, Member, IEEE, 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 her 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 the whole network. Many user engagement studies have been conducted to find a set of critical (anchored) users in the static social network. However, social networks are highly dynamic and their structures are 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 an expensive computational cost. To better understand user engagement in evolving network, we target a new research problem called Anchored Vertex Tracking (AVT) in this article, aiming 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 networks. 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 [29] studied the collaboration of users in a collaboration network, and found that the probability of collaboration between two users is highly related to the number of common neighbors of the selected users. Kossinets and Watts [21], [22] 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. Centola et al. [10] stated that in the presence of high clustering (i.e., \( k \)-core), any additional adoption of messages is likely to produce more multiple exposures than in the case of low clustering. Each additional exposure significantly increases the chance of message adoption. Weng et al. [34] pointed out that people are more susceptible to the information from peers in the same community. This is because the people in the same community sharing similar characteristics naturally establish more edges among them. Moreover, Laishram et al. [23] mentioned that the incentives for keeping users’ engagement on a social network platform partially depends 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 identify 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], [28], [30], [36] have been devoted to finding the crucial (anchored) users who significantly impact the formation of social communities and the operations of social networking platforms. In particular,
We state our major contributions as follows:

**Contributions.** We state our major contributions as follows:

1. **We formally define the problem of AVT and explain the motivation of solving the problem with real applications.**
2. **We propose a Greedy algorithm by extending the core maintenance method in [39] to tackle the AVT problem.** Besides, we build several pruning strategies to accelerate the Greedy algorithm.
3. **We develop an efficient incremental pruning algorithm by utilizing the smoothness of the network structure’s evolution and the well-designed fast-updating core maintenance methods in evolving networks.**
We present the preliminaries in Section 2.

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 undirected evolving network as a sequence of graph snapshots \( \mathcal{G} = \{G_t\}_{t=1}^T \) and \( \{1, 2, \ldots, T\} \) 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. Similar to [14], [18], we can create “dummy” vertices at each time step \( t \) to represent the case of vertices joining or leaving the network at time \( t \) (e.g., \( V = \bigcup_{t=1}^T V_t \) where \( V_t \) is the set of vertices truly exist at \( t \)).

Besides, we set \( \text{nbr}(u, G_t) \) 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., \( |\text{nbr}(u, G_t)| \). Table 1 summarizes the mathematical notations frequently used throughout this paper.

### Table 1: Notations Frequently Used in This Paper

| Notation | Definition |
|----------|------------|
| \( G \) | an undirected evolving graph |
| \( G_t \) | the snapshot graph of \( G \) at time instant \( t \) |
| \( V; E_t \) | the vertex set and edge set of \( G_t \) |
| \( \text{nbr}(u, G_t) \) | the set of adjacent vertices of \( u \) in \( G_t \) |
| \( d(u, G_t) \) | the degree of \( u \) in \( G_t \) |
| \( \text{deg}^-(u) \) | the remaining degree of \( u \) |
| \( \text{deg}^+(u) \) | the candidate degree of \( u \) |
| \( C_k \) | the \( k \)-core subgraph |
| \( O(G_t) \) | the \( k \)-order of \( G_t \) where \( O(G_t) = \{O_1, O_2, \ldots\} \) |
| \( C(G_t) \) | the anchored \( k \)-core that anchored by \( S_t \) |
| \( S_t \) | the vertex set of \( G_t \) |
| \( F_{O}(t, G_t) \) | followers of an anchored vertex \( u \) in \( G_t \) |
| \( F_{S}(S_t, G_t) \) | followers of an anchored vertex set \( S_t \) in \( G_t \) |
| \( E^*; E^- \) | the edges insertion and edges deletion from graph snapshots \( G_{t-1} \) to \( G_t \) |
| \( \text{mcd}(u) \) | the max core degree of \( u \) |

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

**Definition 1 (k-core [4]).** Given an undirected 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 \( \text{core}(u) \), is the maximum value of \( k \) such that \( u \) is contained in the \( k \)-core of \( G_t \). Formally,

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

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

**Example 2.** Consider the graph snapshot \( G_t \) in Fig. 1. The subgraph \( C_3 \) induced by vertices \( \{u_5, u_8, u_{12}, u_{13}, u_{16}\} \) is the 3-core of \( G_t \). This is because every vertex in the induced subgraph has a degree at least 3. Besides, there does not exist a 4-core in \( G_t \). Therefore, we have \( \text{core}(v) = 3 \) 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 an undirected graph \( G_t \) and an anchored vertex set \( S_t \), the followers of \( S_t \) in \( G_t \), denoted as \( F_{S}(S_t, G_t) \), 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 [5]).** Given an undirected 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 \( G_t \) in Fig. 1, the 3-core is \( C_3 = C_3 \). 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 \( C_3 \). 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 \( C_3(S) = \{u_2, u_3, u_5, u_6, u_{11}\} \).

**Algorithm 1.** Core Decomposition(\( G_t, k \))

1: \( k \leftarrow 1 \);
2: while \( V \) is not empty do
3: \( \text{while exists } u \in V \text{ with } \text{nbr}(u, G_t) < k \) do
4: \( V \leftarrow V \setminus \{u\} \);
5: \( \text{core}(u) \leftarrow k - 1 \);
6: for \( w \in \text{nbr}(u, G_t) \) do
7: \( \text{nbr}(w, G_t) \leftarrow \text{nbr}(w, G_t) - 1 \);
8: \( k \leftarrow k + 1 \);
9: return \( \text{core} \);

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 $G = \{G_t\}_{t=1}^T$. Our goal is to track a series of anchored vertex set $S = \{S_t, S_2, ..., S_T\}$ 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.

**Problem Formulation.** Given an undirected evolving graph $G = \{G_t\}_{t=1}^T$, the parameter $k$, and an integer $l$, the problem of anchored vertex tracking (AVT) in $G$ aims to discover a series of anchored vertex set $S = \{S_t\}_{t=1}^T$, satisfying

$$S_t = \arg \max_{S_t \subseteq V} |C_k(S_t)|,$$  

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

**Example 4.** In Fig. 1, if we set $k = 3$ and $l = 2$, the result of the anchored vertex tracking problem can be $S = \{S_1, S_2, \ldots\}$ with $S_1 = \{u_7, u_{10}\}$, $S_2 = \{u_7, u_{13}\}$. Besides, the related anchored k-core of snapshot graph $G_1$ and $G_2$ would be $C_k(S_1) = \{u_2, u_3, u_5, u_6, \ldots, u_{13}, u_{16}\}$ and $C_k(S_2) = \{u_2, u_3, u_5, u_6, \ldots, u_{16}\}$, respectively.

## 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 $k \geq 3$.

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

**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 $G_l$ where $G_l$ is the snapshot graph of $G$ and $C_1$ is the 1-core of $G_l$. Besides, the time complexity of computing the set of $\{G_l \setminus C_1\}$ from snapshot graph $G_l$ is $O(|V| + |E_l|)$. Thus, the AVT problem is solvable in polynomial time with the time complexity of $O(\sum_{i=1}^{T} (|V| + |E_i|))$ 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 $G_l \in G$. Besides, Bhawalkar et al. [5] proposed a linear-time algorithm to solve the anchored 2-core problem in a polynomial time with the time complexity $O(|E_l| + |V| \log |V|)$. From the above, we conclude that there is an implementation of the algorithm to answer the AVT problem by running in time complexity $O(\sum_{i=1}^{T} (|E_i| + |V| \log |V|))$. Therefore, the AVT problem is solvable in polynomial time while $k = 2$.

(3) When $k \geq 3$ and $t \in [1, T]$, we first note that the anchored vertex tracking problem is equivalent to a set of anchored k-core problems at snapshot graphs $G_l \in G$. 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 $G_l \in G$ is NP-hard, by reducing the anchored k-core problem to the Set Cover problem [19]. Given a fixed instance $I$ 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 $i$, $|S_i| \leq k - 1$. In the following, we construct a corresponding instance of the anchored k-core problem in $G_l$ by lifting the above restriction while still obtaining the same results.

Considering $G_l$ contains a set of nodes $V = \{u_1, \ldots, u_n\}$ which is associated with a collection of subsets $S = \{S_1, \ldots, S_s\} \subseteq V$. We construct an arbitrary large graph $G_l'$ where each vertex in $G_l'$ has degree $k$ except for a single vertex in $G_l'$ that has degree $k - 1$. Then, we set $H = \{G_l', \ldots, G_m'\}$ as the set of $n$ connected components $G_l'$ of $G_l$, where $G_l'$ is associated with an element $e_i$. When $e_j \in S_t$, there is an edge between $u_i$ and $v(G_l')$. Based on the definition of $k$-core in Definition 1, once there exists $i$ such that $u_i$ is the neighbor of $v(G_l')$, then all vertices in $G_l'$ will remain in $k$-core. Therefore, if there exists a set cover $C$ with size $l$, we can set $l$ anchors from $u_i$, where $S_i \subseteq C$ for each $i$, and then all vertices in $H$ will be the member of the $k$-core. Since we are assuming that $|S_i| < k$ for all sets, each vertex $u_i$ will not be in the subgraph of $k$-core unless $u_i$ is anchored. Thus, we must anchor some vertex adjacent to $v(G_l')$ for each $G_l' \in G_l'$, 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 split 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 $\text{tree}(d,y)$) method to lift this restriction. Specifically, to lift the restriction on the maximum set size, they use $\text{tree}(k-1, |S_i|)$ to replace each instance of $u_i$. Besides, if $y_1, \ldots, y_{|S_i|}$ 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 $\text{OPT}^{(1-\epsilon)}$ multiplicative factor of the optimal solution in general graph, unless $P = \text{NP}$.

**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 $(1-\epsilon)$-ratio, unless $P = \text{NP}$ [15]. 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 $(1-\epsilon)$ when $k \geq 3$. □
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 summarizes 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 $G_t \in G$ (Lines 2-11). For each $G_t \in G$ 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 $u$. 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 $O(\sum_{t=1}^{T} |V_t| \cdot |E_t|)$.

Algorithm 2. The Greedy Algorithm

Input: $G = \{G_t\}_t^T$ : an evolving graph, $l$: the allocated size of anchored vertex set, and $k$: degree constraint

Output: $S = \{S_t\}_t^T$ : the series of anchored vertex sets

1: $S \leftarrow \emptyset$;
2: for each $t \in [1, T]$ do
3: $i \leftarrow 0$; $S_t \leftarrow \emptyset$
4: while $i < l$ do
5: /* Candidate Anchored Vertex */
6: for each $u \in V$ do
7: /* Computing Followers */
8: Compute $F_t(u, G_t)$;
9: $u' \leftarrow$ the best anchored vertex in this iteration;
10: $S_t \leftarrow S_t \cup u'$; $i \leftarrow i + 1$;
11: $S \leftarrow S \cup S_t$;
12: return $S$

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

Fig. 2 shows a K-order index $O = \{O_1, O_2, O_3\}$ of graph snapshot $G_1$ in Fig. 1. The vertex sequence $O_k \in O$ records all vertices in k-core by following the removing order of core decomposition, i.e., $O_2$ records all vertices in 2-core and vertex $u_1$ is removed early than vertex $u_0$ 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., $x \leq v$).

Proof. We prove the correctness of this theorem by contradiction. If $v \leq x$ 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 $v \leq x$. On the other hand, it is already proved in [36] 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., $\{x \in V|\exists v \in nbr(x, G_t) \cap core(v) = k-1 \land x \leq v\}$. 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 $|\{x \in V|\exists v \in nbr(x, G_t) \cap core(v) = k-1 \land x \leq v\}|$.

Example 5. Given the graph $G_1$ in Fig. 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 Accelerating Followers Computation

To accelerate the computation of followers, a feasible way is to transform the followers’ computation into the core maintenance problem [26], [39], which aims to maintain the core number of vertices in a graph when the graph changes. 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 [39], 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.

Fig. 2. The K-order $O$ of graph $G_1$ in Fig. 1.
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 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 Figs. 2 and 1, we would like to show the process of followers’ computation. Assume \( k = 3 \), \( V^* = \emptyset \), and the \( K \)-order, \( O = \{O_1, O_2, O_3\} \), in graph \( G_1 \). Initially, the \( \text{deg}^+ (u) \) value of each vertex \( u \) is recorded in \( O(G_1) \), i.e., \( \text{deg}^+(u_{11}) = 2 \) for all vertices in \( G_1 \) as \( V^* \) is empty. If we anchor the vertex \( u_{15} \), i.e., \( \text{core}(u_{15}) = \infty \), then we need to update the candidate degree value of \( u_{15} \)'s neighbours in \( O_2 \), i.e., \( \text{deg}^-(u_{11}) = 0 + 1 \) and \( \text{deg}^+(u_{14}) = 0 + 1 \). We then start to visit the foremost neighbours of \( u_{15} \) in \( O_2 \), i.e., \( u_{14} \). Since \( \text{deg}^+(u_{14}) + \text{deg}^-(u_{11}) = 2 + 1 \geq 3 \) and \( \text{deg}^+(u_{11}) + \text{deg}^+(u_{11}) = 1 + 1 < 3 \), we can add \( u_{14} \) to \( V^* \) and then update the \( \text{deg}^+(u) \) of its impacted neighbours. After that, we sequentially explore the vertices \( s \) after \( u_{14} \) in \( O_3 \) and operate the above steps once \( \text{deg}^+(u) + \text{deg}^-(s) \geq 3 \). The follower computation terminates when the last vertex in \( O_3 \) 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. [39] reported that the core maintenance process while inserting an edge takes \( \mathcal{O}(\sum_{v \in V^*} \text{deg}(v) \cdot \log_{\max\{\max(1, |C_{k-1}|), |C_k|\}}) \) (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 \( \mathcal{O}(\sum_{v \in \text{nbr}(v) \wedge v \in V^* \text{ and } v \geq w} \text{deg}(w) \cdot \log_{\max\{\max(1, |C_{k-1}|), |C_k|\}}) \). 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 \( G \), the Greedy approach individually constructs the \( K \)-order and iteratively searches the anchored vertex set at each snapshot graph \( G_t \) of \( G \). However, 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 \( G = \{G_1, G_2, ..., G_T\} \) be an evolving graph, \( S_t \) 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 \( S = \{S_t\}_t^T \) of \( G \) using the IncAVT algorithm, we first build the \( K \)-order of \( G_t \), and then compute
the anchored vertex set $S_i$ of $G_1$. Next, we develop a bounded $K$-order maintenance approach to maintain the $K$-order by considering the change of edges from $G_{i-1}$ to $G_i$. 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 $V_{IR}$ 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 $G_{ir}$, while the potential anchored vertices are selected to probe from $V_I$, $V_{IR}$, and $S_{i-1}$. The $l$ anchored vertices are recorded in $S_i$. Finally, we output $S = \{S_i\}_1^T$ 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 $G_{i-1}$ to $G_i$, 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 $G_{i-1}$, then the core number of each vertex in $G_{i-1}$ either remains unchanged or increases. Therefore, the $k$-core of snapshot graph $G_{i-1}$ is part of the $k$-core of snapshot graph $G_i$ where $G_i = G_{i-1} \oplus E^+$. 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 $G_{i-1}$, the remaining degree of $u$ increases by 1, i.e., $\deg^+(u) = \deg^+(u) + 1$, if $u \leq v$ holds.

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

Example 7. Consider the snapshot graph $G_1$ in Fig. 1, if we add a new edge $(u_2, u_3)$ into $G_1$ where $u_2 \leq u_3$ (mentioned in Fig. 2), then the remaining degree of $u_2$, $\deg^+(u_2) = \deg^+(u_2) + 1 = 3$.

Lemma 2. Let $\deg^+(u)$ and $\text{core}(u)$ 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 $\text{core}(u)$ of $u$ may increase by 1 if $\text{core}(u) < \deg^+(u)$. Otherwise, $\text{core}(u)$ 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 does not need to be updated after inserting edge $(u, v)$ into $G_{i-1}$, then the number of $u$’s neighbours $v$ with $u \leq v$ must be no more than $\text{core}(u)$. Therefore, the value of updated $\deg^+(u)$ should be no more than $\text{core}(u)$, which is contradicted with the fact that $\text{core}(u) < \deg^+(u)$.

Example 8. Considering a vertex $u_2$ in graph $G_1$, we can see $\deg^+(u_2) = 2$, and $\text{core}(u_2) = 2$ as shown in Figs. 1 and 2. If an edge $(u_2, u_3)$ is inserted into $G_1$, we can get $\deg^+(u_2) = \deg^+(u_2) + 1 = 3$ (refer Lemma 1). Since $\text{core}(u_2) = 2 < \deg^+(u_2) = 3$, the $\text{core}(u_2)$ may increase by 1 according to Lemma 2.

We present the EdgeInsert algorithm for $K$-order maintenance. It consists of three main steps. First, 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 $\text{core}(u) = k - 1$ after inserting $E^+$. This is because the followers only come from vertices with core number $k - 1$ (refer Theorem 3).

Algorithm 4. EdgeInsert ($G'_t, O, E^+, k$)

1: $i \leftarrow 0$, $V_I \leftarrow \emptyset$, $m \leftarrow 0$, $O' \leftarrow \emptyset$
2: for each $e = (u, v) \in E^+$ do
3: $m \leftarrow \max \{m, \min (\text{core}(u), \text{core}(v))\}$
4: $u \leq v$ ? $\deg^+(u) + 1 = 1 : \deg^+(v) + 1 = 1$
5: while $i \leq m$ do
6: $V_C \leftarrow \emptyset$, $\deg^-(v) \leftarrow 0$
7: $u^* \leftarrow$ the first vertex of $O_i \in O$
8: while $u^* \neq nil$ do
9: if $\deg^+(u^*) + \deg^-(v) > i$ 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 $v \in \text{nbr}(u^*, G'_t) \land \text{core}(v) = i \land u^* \leq v$ do
14: $\deg^+(v) \leftarrow \deg^+(v) + 1$
15: else
16: if $\deg^-(u^*) = 0$ then
17: remove $u^*$ from $O_i$; append $u^*$ to $O'_i$
18: else
19: $\deg^+(u^*) \leftarrow \deg^+(u^*) + \deg^-(u^*)$
20: $\deg^-(u^*) \leftarrow 0$
21: remove $u^*$ from $O_i$; append $u^*$ to $O'_i$
22: update the $\deg^-(.)$ of $u^*$'s neighbors;
23: $u^* \leftarrow$ the vertex next to $u^*$ in $O_i$
24: for $v \in V_C$ do
25: $\deg^-(v) \leftarrow 0$; $\text{core}(v) \leftarrow \text{core}(v) + 1$
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: $V_I \leftarrow V_I \cup V_C$
31: add $O_i$ to new $K$-order $O'$ in $G'_t$
32: $i \leftarrow i + 1$
33: return the $K$-order $O'$ in $G'_t$, and $V_I$

The detailed description of our EdgeInsert algorithm is outlined in Algorithm 4. The inputs of the algorithm are snapshot graph $G_{i-1}$ where $t \in [2, T]$, the $K$-order $O = \{O_1, O_2, \ldots, O_k, \ldots\}$ of $G_{i-1}$, 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 $u \leq v$ (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 $O_i \in O$ whose core number increases after the insertion of $E^+$, and we also update $O_i$ 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 $O_i$ (Line 7). In the inner while loop, we visit the vertices in $O_i$, in order (Lines 8-22). The visited vertex $u^*$ must satisfy one of the three conditions: (1) $deg^+(u^*) + deg^-(u^*) > i$; (2) $deg^+(u^*) + deg^-(u^*) \leq i \land deg^-(u^*) = 0$; (3) $deg^+(u) + deg^-(u) \leq i \land deg^-(u) > 0$. For condition (1), the core number of the visited vertex $u^*$ may increase. Then, we remove $u^*$ from $O_i$ and add it into $V_C$. Besides, the candidate degree of each neighbour $v$ of $u^*$ should increase by 1 if $u^* \not \in v$ (Lines 9-14). For condition (2), the core number of $u^*$ will not change. So we remove $u^*$ from the previous $O_i$ and append it into $O_{i+1}$ of the new $K$-order $O'$ of graph $G_i' = G_{i-1} \oplus E^+$ (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 $O_i$ and append it to $O_{i+1}$. 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 $G_i'$ (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 $G_{i-1}$ to $G_i$, then the remaining degree of $u$ from $G_{i-1}$ to $G_i$ decreases by 1, i.e., $deg^-(u) = deg^-(u) - 1$, if $u \not \in v$ holds.

**Proof.** From Section 4.2 of the remaining degree of a vertex, we get $deg^+(u) = |\{v \in nbr(u) \mid u \not \in v\}|$. Deleting an edge $(u, v)$ from graph snapshot $G_{i-1}$ evolving to $G_i$ removes one neighbour $v$ of $u$ where $u \not \in v$ in the $K$-order of $G_i$. Therefore, $deg^-(u)$ needs to decrease by 1 after deleting $(u, v)$ from $G_{i-1}$.

**Example 9.** Consider the snapshot graph $G_1$ and $G_2$ in Fig. 1, if we remove edge $(u_2, u_{11})$ from $G_1$ to $G_2$ where $u_2 \not \in u_{11}$ (mentioned in Fig. 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 [31]).** Given an undirected graph $G_i$, the max-core degree of a vertex $u$ in $G_i$, 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 Fig. 1, we have core($u_9$) = 3, core($u_{11}$) = 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 $u_{11}$'s neighbours $\{u_9, u_{15}, u_{16}\}$ 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 $G_i$. Suppose we delete an edge $(u, v)$ from $G_i$ and the updated $mcd(u)$. Thus, the core number core($u$) of $u$ may decrease by 1 if $mcd(u) < core(u)$. Otherwise, core($u$) remain unchanged.

**Proof.** Based on Definitions 1 and 2, the core number of vertex $u$ is identified by the number of its 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., $mcd(u) = |\{v \mid v \in nbr(u) \land core(v) \geq core(u)\}|$. Therefore, we can conclude that $mcd(u) \geq core(u)$ always holds. Hence, if $mcd(u) < core(u)$ after deleting an edge from $G_i$ and updating $mcd(u)$, then core($u$) also needs to be decreased by 1 to ensure $mcd(u) > core(u)$ in the changed graph.

**Algorithm 5. EdgeRemove($G_i', O', E^-, k$)**

1: /* $mcd(u)$ is the number of $u$'s neighbour $v$ with core($v$) $\leq$ core($u$) */
2: $O' = \{O_1, O_2, \ldots\}$; Initialize array $F[|V|]$
3: $V_f \leftarrow \emptyset$, and $m = 0$;
4: let $Q$ be an empty queue and $V^* = \{V_1, V_2, \ldots\}$, $V_i \in V^*$ be the empty list;
5: /* Identify the vertex need to remove from $O'$ */
6: for each $e = (u, v) \in E^-$ do
7: $u^* \leftarrow u$ if $u \not \in v$, otherwise $v$;
8: $G_i := G_i \oplus e; j \leftarrow core(u, G_i')$;
9: compute $mcd(u, G_i')$ of $u'$;
10: if $mcd(u', G_i') < j$ then
11: remove $u'$ from $O_j'$, enqueue $u'$ to $Q$;
12: core($u') \leftarrow core(u') - 1$;
13: if $F[u'] = 1$ then
14: remove $u'$ from $V_i$;
15: else
16: $F[u'] = 1$;
17: while $Q$ is not empty do
18: dequeue $u$ from $Q$, $i \leftarrow core(u, G_i)$;
19: append $u$ to $V_i$, $m \leftarrow max\{m, i\}$;
20: for $u' \in nbr(u, G_i) \land core(u') = 0$ do
21: repeat lines 9-16;
22: $G_i' := G_i$;
23: /* update the k-order $O'$ */
24: for $i \leftarrow m$ to 1 do
25: for each $u \in V_i$ in order do
26: $deg^+(u) \leftarrow 0$;
27: for $u' \in nbr(u, G_i)$ do
28: if core($u'$) $> core(u) \vee u' \in V_i$ then
29: $deg^+(u) \leftarrow deg^+(u) + 1$;
30: recomute $deg^+(u')$;
31: append $u$ to the end of $O_i$;
32: $V_f \leftarrow V_{k-1}, O(G_i) \leftarrow O'$;
33: return the K-order $O(G_i)$ of $G_i$, and $V_f$.
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 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 6-21), we identify the vertices that need to be removed from their previous position of K-order $O'$ after the edge deletion. Specifically, we first update the graph $G_t$ 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., $mc_d(u) < core(u)$, into a queue $Q$. All vertices in $Q$ need to update their core numbers based on Lemma 4 (Lines 10-16). After that, the algorithm recursively probes each neighboring vertex $v$ of vertices in $Q$, and adds $v$ into the vertex set $V'$ if $mc_d(v) < core(v)$ (Lines 17-21). 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 24-31). In details, for each $u \in V_t$, we update the $deg^+(u)$ of $u$ and its neighbours, remove $u$ from $O'_t$, and insert $u$ to the end of $O'_t$. In the final step, we use $V_t$ to record the vertices that may become the potential followers for the anchored vertices, i.e., these vertices' core number becomes $k - 1$ in the new K-order $O'$ (Line 32).

5.3 The Incremental Algorithm

Base on the above K-order maintenance strategies and the impacted vertex sets $V_t$ and $V_{t-1}$, we propose an efficient incremental algorithm, IncAVT, for processing the AVT query. Algorithm 6 summarizes the major steps of IncAVT. Given an evolving graph $G = \{G_t\}_t$, the allocated size of selected anchored vertex set $l$, and a positive integer $k$, the IncAVT algorithm returns a series of anchored vertex set $S = \{S_t\}_t$ for each $G_t$ of $G$ by using Algorithm 1, and then compute the anchored vertex set $S_i$ of $G_i$ by using Algorithm 2 where $T$ is set as 1 (Lines 1-3). The while loop at lines 4-17, computes the anchored vertex set of each snapshot graph $G_t \in G$. $E^+$ and $E^-$ represent the edges insertion and edges deletion between $G_{t-1}$ to $G_t$ respectively, and we initialize the anchored vertex set $S_i$ of $G_t$ and $S_{t-1}$ (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 $V_t$ 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 $V_t$ 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 $S_i$'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 $\{V_t \cup V_{t-1} \cup nbor(V_t \cup V_{t-1}) \setminus C_{k}(G_t)\}$ based on Theorem 3 (Lines 9-14). If the number of followers of anchored vertex set $\{S_i \setminus u \cup v\}$ is bigger than $F$, we then update $S_i$ 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 $S$ (Line 17). The IncAVT algorithm finally returns the series of anchored vertex set $S$ as the final result (Line 18).

Algorithm 6. IncAVT

| Input: $G = \{G_t\}_t$ : an evolving graph, $l$: the allocated size of anchored vertex set, and $k$: degree constraint |
| Output: $S = \{S_t\}_t$ : the series of anchored vertex sets |
| 1: Build the K-order $O(G_t)$ of $G_t$ /* using Algorithm 1 */ |
| 2: Compute the anchored vertex set $S_i$ of $G_i$ with size $l$ using Algorithm 2 |
| 3: $S := \{S_t\}_t$ : $t := 2$ |
| 4: while $t < T$ do |
| 5: $G_t := G_{t-1} \oplus E^+$, $S_t$ := $S_{t-1}$ |
| 6: /* maintain K-order by using Algorithms 4 and 5 */ |
| 7: $(O', V_t) := EdgeInsert(G_t, O(G_{t-1}), E^+, k)$ |
| 8: $(O(G_i), V_t) := EdgeRemove(G_t, O', E^-, k)$ |
| 9: for each $u \in S_t$ do |
| 10: compute $F_t(S_i, G_i) \quad F_t := F_t(S_i, G_i)$ |
| 11: $max := 0, u := u$ |
| 12: for each /* Theorem 3 */ |
| 13: $\{v | v \in \{V_t \cup V_{t-1} \cup nbor(V_t \cup V_{t-1}) \setminus C_{k}(G_t)\}$ |
| 14: $\land \{\exists u \in \{v | \exists core(u) = k - 1 \land v \leq u\}$ |
| 15: if $max < F_t(S_i \setminus u \cup v, G_t)$ then |
| 16: $max := F_t(S_i \setminus u \cup v, G_t), u := v$ |
| 17: if $max > F_t$ then |
| 18: remove $u$ from $S_t$, add $u'$ to $S_t$ |
| 19: $S := S \cup S_t, t := t + 1$ |
| 20: return $S$ |

6 Experimental Evaluation

In this section, we present the experimental evaluation of our proposed approaches for the AVT problem: the Greedy algorithm that is optimized by two strategies mentioned in Section 4 (Greedy), and the incremental algorithm (IncAVT). The source codes of this work are available at https://github.com/IncAVT/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 compare with two baselines adapted from the existing works: (i) OLAK, which is proposed in [36] to find out the best anchored vertices at each snapshot graph, and (ii) RCM, which is the state-of-the-art anchored k-core algorithm proposed in [23], for tracking the best anchored vertices selection at each snapshot graph.

Datasets. We conduct the experiments using six publicly available datasets from the Stanford Network Analysis Project (SNAP): 1) email-Enron, Guuttela, Deezer, eu-core, mathoverflow, and CollegeMsg. The statistics of the datasets are shown in Table 2. As the original datasets (i.e., email-Enron, Guuttela, and Deezer) 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. Moreover, we further conduct our experiments using two real-world

1. http://snap.stanford.edu/
temporal network datasets from SNAP: en-core, mathoverflow, and CollegeMsg. Specifically, we have averagely divided these two datasets into T graph snapshots (e.g., $G_t = (V, E_t)$, $t \in [0, T]$), where V is the vertex and $E_t$ is the edges appearing in the time period of $t$ in each dataset. Besides, the edge insertion set $E^+$ of $G_t$ contains edges newly appears in $G_t$ but does not exist in $G_{t-1}$; Similarly, the edge deletion set $E^-$ of $G_t$ is the edges existed in $G_{t-1}$ but disappear in $G_t$. Note that an edge will be disppear if it keeps being inactive in a period of time (i.e., a time window $W = 365$ days in mathoverflow dataset).

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 2.60 GHz Intel Xeon CPU and 96 GB RAM.

### Efficiency Evaluation

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

#### Varying Core Number $k$

We compare the performance of different approaches by varying $k$. Due to the various average degree of six datasets, we set different $k$ for them. Figs. 3a, 3b, 3c, 3d, 3e, and 3f show the running time of OLAK, Greedy, IncAVT, and RCM, on the six 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 when $k$ is varied. We notice that OLAK visits more number of candidate anchored vertices than the other two approaches, and IncAVT shows the minimum number of visited candidate anchored vertices.

### Table 2: Dataset Statistics

| Dataset        | Nodes | Edges       | $d_{avg}$ | Days | Type         |
|----------------|-------|-------------|-----------|------|--------------|
| email-Enron    | 36,692| 183,831     | 10.02     | -    | Communication|
| Gnutella       | 62,586| 147,878     | 4.73      | -    | P2P Network  |
| Deezer         | 41,773| 125,826     | 6.02      | -    | Social Network|
| eu-core        | 986   | 332,334     | 25.28     | 803  | Email        |
| mathoverflow   | 13,840| 195,330     | 5.86      | 2,350| Question&Answer|
| CollegeMsg     | 1,899 | 59,835      | 10.69     | 193  | 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      |

Fig. 3. Time cost of algorithms with varying $k$.

Fig. 4. Number of candidate anchored vertices with varying $k$.
6.2.2 Varying Snapshot Size $T$

We also test our proposed algorithms by varying $T$ from 2 to 30. Specifically, Figs. 5a, 5b, and 5c present the running time with varied values of $T$ in email-Enron, Gnutella, and Deezer. 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. The above experimental results verify the excellent performance of our IncAVT when the network is smoothly evolving, which is claimed in the contributions part of Section 1 in this paper. Figs. 5d, 5e, and 5f show the running time of these approaches on three real-world temporal datasets eu-core, mathoverflow, and CollegeMsg when $T$ is varied. We observe that our optimized Greedy method always performs better than OLAK and RM for all varied $T$ values in eu-core and mathoverflow. As expected, in eu-core, when $T \leq 20$, the performance of IncAVT is significantly better than the other three methods; Besides, the running time of IncAVT significantly increases when $T = 21$, and then increased slowly with the increases of $T$. This is because the efficiency of $K$-order maintenance will downgrade when the percentage of updated edges is high (i.e., 17% percentage of edges updated at snapshot $T = 21$ in eu-core). In fact, the above phenomenon is the inherent character of the core maintenance technical strategy (e.g., Zhang et al. [39] reported that their core maintenance related method decreased above five times when the percentage of updated edges increasing from 1% to 5%). In addition, Figs. 5e and 5f show that even the performance of our IncAVT method decreases at $T = 16$ in mathoverflow and $T = 22$ in CollegeMsg, when many edges are updated in these two periods, IncAVT still performs better than OLAK for all values of $T$.

Figs. 6a, 6b, 6c, 6d, 6e, and 6f 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.

6.2.3 Varying Anchored Vertex Set Size $l$

Figs. 7a, 7b, 7c, 7d, 7e, and 7f 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 in email-Enron, Gnutella, Deezer, eu-core, mathoverflow, and CollegeMsg. 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.

Figs. 8a, 8b, 8c, 8d, 8e, and 8f 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.

6.3 Effectiveness Evaluation

In this experiment, we evaluate the total number of followers produced by the AVT problem with different datasets and approaches in Figs. 9, 10, and 11 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 Fig. 9c, the follower size in the Deezer dataset is about one thousand when $T = 2$ and goes up to 50,000 when $T = 30$. Similar pattern can also be found in Fig. 10 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 Fig. 11 for 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.

### 6.4 A Case Study on Anchored Vertex Tracking

We conduct a case study in this subsection to provide more insights into comparing our proposed methods with the brute-force method for the problem studied in this paper. Specifically, the brute-force method requires exhaustively enumerating all possible anchored sets with size \( l \). The time complexity is \( O(C_l^i \cdot |E|) \), which is cost-prohibitive and growing exponentially while \( l \) increases (e.g., the running time of brute-force in mathoverflow and eu-core by setting \( l = 2 \) and \( k = 3 \) are over 24 hours and 38,140 ms, respectively). In Fig. 12, we report the followers results of given anchored vertices at different snapshots in eu-core using IncAVT, Greedy, and brute-force method by varying \( T \) and setting \( l = 2 \) and \( k = 3 \). We observed that, the approximate results (i.e., number of followers) reported by the four approximate algorithms (i.e., OLAK, Greedy, IncAVT, and RCM) are very close to the exact result queried by brute-force algorithm.

Finally, we further show the selected anchored vertices and the related followers in detail at the first snapshot period in Table 4.

### 7 RELATED WORK

#### 7.1 k-Core Decomposition

The model of \( k \)-core was first introduced by Seidman et al. [32], and has been widely used as a metric for measuring the structure cohesiveness of a specific community in the topic of social contagion [33], user engagement [5], [12], [23], [27], [35], [37], [40]. 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. [12] 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 [39] proposed an efficient greedy algorithm by using the vertex deletion order in \( k \)-core decomposition, named OLAK. In the same year, another research [36] studied the anchored \( k \)-core problem, which aims to identify critical users that may lead a maximum \( k \)-core. Zhou et al. [40] 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 [23] 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. Considering that the topology of networks often evolves in real-world, we proposed and studied the anchored vertex tracking problem (AVT) in this paper, which is extended from the traditional anchored \( k \)-core problem [5], aiming to find out the optimal anchored vertices in each timestamp so as to fully maximize the community size at each period of evolving 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.

In addition, some other community models such as \( k \)-truss [17], [35] and \( k \)-plex [3] can be applied to measure the quality of user engagement dynamics in social networks. Compared with \( k \)-core, the \( k \)-truss model not only captures users with high engagement but also ensures strong tie strength among the users. However, the \( k \)-truss is

### 7.2 User Engagement

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], [12], [23], [27], [35], [37], [40]. 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. [12] 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 [39] proposed an efficient greedy algorithm by using the vertex deletion order in \( k \)-core decomposition, named OLAK. In the same year, another research [36] studied the anchored \( k \)-core problem, which aims to identify critical users that may lead a maximum \( k \)-core. Zhou et al. [40] 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 [23] 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. Considering that the topology of networks often evolves in real-world, we proposed and studied the anchored vertex tracking problem (AVT) in this paper, which is extended from the traditional anchored \( k \)-core problem [5], aiming to find out the optimal anchored vertices in each timestamp so as to fully maximize the community size at each period of evolving 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.

In addition, some other community models such as \( k \)-truss [17], [35] and \( k \)-plex [3] can be applied to measure the quality of user engagement dynamics in social networks. Compared with \( k \)-core, the \( k \)-truss model not only captures users with high engagement but also ensures strong tie strength among the users. However, the \( k \)-truss is

| Algorithms | Selected Anchored Vertices | Followers |
|------------|----------------------------|-----------|
| Brute-force | 469, 630                   | 163, 72, 630, 468, 469 |
| OLAK       | 630, 541                   | 630, 163, 72, 541, 531 |
| Greedy     | 541, 351                   | 541, 531, 351, 184    |
| IncAVT     | 541, 351                   | 541, 531, 351, 184    |
| RCM        | 552, 630                   | 552, 630, 72, 163, 320|
defined based on the triangle, a local concept, and may not fully represent the user’s cluster in a global view. Besides, the cohesiveness of the k-plex is higher than that in both k-core and k-truss. In other words, the users in k-plex have a tighter relationship than that in both k-core and k-truss. Nevertheless, finding a k-plex from a given graph for an integer k is NP-hard, leads to the unsuitability of the k-plex model in this work.

8 Conclusion

In this article, 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) reducing the potential anchored vertices that need probing; and (2) proposing 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.

Acknowledgments

Taotao Cai and Shuqiao Yang are the joint first authors.

References

[1] H. Aksu, M. Canim, Y. Chang, I. Korpeoglu, and O. Ulusoy, “Distributed k-core view materialization and maintenance for large dynamic graphs,” IEEE Trans. Knowl. Data Eng., vol. 26, no. 10, pp. 2439–2452, Oct. 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,” Netw. Heterogeneous Media, vol. 3, no. 2, pp. 371–393, 2008.

[3] B. Balasundaram, S. Butenko, and I. V. Hicks, “Clique relaxations in social network analysis: The maximum k-plex problem,” Operations Res., vol. 59, no. 1, pp. 133–142, 2011.

[4] V. Batagelj and M. Zaversnik, “An O(m) algorithm for cores decomposition of networks,” 2003, arXiv:cond-mat/0310049.

[5] K. Bhawalkar, J. M. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma, “Preventing unraveling in social networks: The anchored k-core problem,” in Proc. Int. Colloq. Automata Lang. Program., 2012, pp. 440–451.

[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., vol. 29, no. 3, pp. 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 Proc. IEEE 36th Int. Conf. Data Eng., 2020, pp. 409–420.

[8] C. V. Cannistraci, G. Alainis-Lobato, and T. Ravasi, “From link-prediction in brain connectomes and protein interactomes to the local-community-paradigm in complex networks,” Sci. Rep., vol. 3, no. 1, 2013, Art. no. 1613.

[9] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir, “A model of internet topology using k-shell decomposition,” Proc. Nat. Acad. Sci. USA, vol. 104, no. 27, pp. 11 150–11 154, 2007.

[10] D. Centola, “The spread of behavior in an online social network experiment,” Science, vol. 329, no. 5996, pp. 1194–1197, 2010.

[11] X. Chen, G. Song, X. He, and K. Xie, “On influential nodes tracking in dynamic social networks,” in Proc. SIAM Int. Conf. Data Mining, 2015, pp. 613–621.

[12] R. Chitnis, F. V. Fomin, and P. A. Golovach, “Parameterized complexity of the anchored k-core problem for directed graphs,” J. Graph., vol. 247, pp. 11–22, 2016.

[13] R. H. Chitnis, F. V. Fomin, and P. A. Golovach, “Preventing unraveling in social networks gets harder,” in Proc. 27th AAAI Conf. Artif. Intell., 2013, pp. 1085–1091.

[14] A. Das, M. Svendsen, and S. Tirthapura, “Incremental maintenance of maximal cliques in a dynamic graph,” VLDB J., vol. 28, no. 3, pp. 351–375, 2019.

[15] U. Feige, “A threshold of ln n for approximating set cover,” J. ACM, vol. 45, no. 4, pp. 634–652, 1998.

[16] C. Giatsidis, F. D. Miliatos, D. M. Thilikos, and M. Vazirgiannis, “CORECLUSTER: A degeneracy based graph clustering framework,” in Proc. 28th AAAI Conf. Artif. Intell., 2014, pp. 44–50.

[17] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, “Querying k-truss community in large and dynamic graphs,” in Proc. ACM SIGMOD Conf. Manage. Data, 2014, pp. 1311–1322.

[18] X. Jia et al., “Tracking community consistency in dynamic networks: An influence-based approach,” IEEE Trans. Knowl. Data Eng., vol. 33, no. 2, pp. 782–795, Feb. 2021.

[19] R. M. Karp, “Reducibility among combinatorial problems,” in Proc. Symp. Complexity Comput. Comput., 1972, pp. 85–103.

[20] M. Kitsak et al., “Identification of influential spreaders in complex networks,” Nature Phys., vol. 6, pp. 888–893, 2010.

[21] G. Kossinets and D. Watts, “Origins of homophily in an evolving social network,” Amer. J. Sociol., vol. 115, no. 2, pp. 405–450, 2009.

[22] G. Kossinets and D. J. Watts, “Empirical analysis of an evolving social network,” Science, vol. 311, no. 5787, pp. 88–90, 2006.

[23] R. Laishram, A. E. Sariyuce, T. Eliassi-Rad, A. Pinar, and S. Soodarajah, “Residual core maximization: An efficient algorithm for maximizing the anchored k-core,” in Proc. SIAM Int. Conf. Data Mining, 2020, pp. 325–333.

[24] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, “Microscopic evolution of social networks,” in Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2008, pp. 462–470.

[25] 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., vol. 320, pp. 512–523, 2018.

[26] R. Li, J. X. Yu, and R. Mao, “Efficient core maintenance in large dynamic graphs,” IEEE Trans. Knowl. Data Eng., vol. 26, no. 10, pp. 2453–2465, Oct. 2014.

[27] Q. Linghu, F. Zhang, X. Lin, W. Zhang, and Y. Zhang, “Global reinforcement of social networks: The anchored coreness problem,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2020, pp. 2211–2226.

[28] F. D. Malliaros and M. Vazirgiannis, “To stay or not to stay: Modeling engagement dynamics in social graphs,” in Proc. 22nd ACM Int. Conf. Inf. Knowl. Discov. Data Mining, 2013, pp. 469–478.

[29] M. Newman, “Clustering and preferential attachment in growing networks,” Phys. Rev. E, vol. 64, no. 2, 2001, Art. no. 026102.

[30] G. Rossetti, L. Pappalardo, R. Kikas, D. Pedreschi, F. Giannotti, and M. Dumas, “Community-centric analysis of user engagement in skype social network,” in Proc. IEEE/ACM Int. Conf. Adv. Soc. Netw. Anal. Mining, 2015, pp. 547–552.

[31] A. E. Sariyuce, B. Gedik, G. Jacques-Silva, K. Wu, and U. V. Catayurek, “Streaming algorithms for k-core decomposition,” Proc. VLDB Endowment, vol. 6, no. 6, pp. 433–444, 2013.

[32] S. B. Seidman, “Network structure and minimum degree,” Soc. Netw., vol. 5, no. 3, pp. 269–287, 1983.

[33] J. Ugander, L. Backstrom, R. Kumar, and A. Tomkins, “Querying community evolution in large and dynamic graphs,” in Proc. VLDB Endowment, vol. 6, no. 10, pp. 17–28, 2013.

[34] L. Weng, F. Menczer, and Y.-Y. Ahn, “Virality prediction and community structure in social networks,” Sci. Rep., vol. 3, no. 1, pp. 1–6, 2013.

[35] 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., vol. 32, no. 1, pp. 78–91, Jan. 2020.

[36] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin, “OLAK: An efficient algorithm to prevent unraveling in social networks,” Proc. VLDB Endowment, vol. 10, no. 6, pp. 649–660, 2017.
Taotao Cai received the PhD degree in computer science from Deakin University, Australia, in 2020. He is currently a postdoctoral research fellow with the School of Computing, Macquarie University. Before moving to Macquarie, he worked as an associate research fellow with the School of Information Technology, Deakin University. His research interests include graph data processing, social network analysis, and deep learning.

Shuqiao Yang received the BS degree from Shanghai Ocean University, China, the MS degree from Zhejiang University, China, and the PhD degree from the School of Information Technology, Deakin University, Australia. He is currently a postdoctoral research associate with the School of Computer Science and Engineering, University of New South Wales, Australia. His research interests include data analysis, text mining, machine learning, and deep learning.

Jianxin Li received the PhD degree in computer science from the Swinburne University of Technology, Australia, in 2009. He is an associate professor with the School of IT, Deakin University. His research interests include social computing, query processing and optimization, Big Data analytics, and intelligent computation in educational data. He has published 70 high quality research papers in top international conferences and journals, including Proceedings of the VLDB Endowment, IEEE ICDE, IEEE ICDM, EDBT, ACM Transactions on Knowledge and Data Engineering, and ACM WWW.

Quan Z. Sheng received the PhD degree in computer science from the University of New South Wales and did his postdoctoral as a research scientist with CSIRO ICT Centre. He is a full professor and head of the School of Computing, Macquarie University. Before moving to Macquarie, he spent 10 years with the School of Computer Science, The University of Adelaide, serving in a number of senior leadership roles including acting head and deputy head of the School of Computer Science. His research interests include the Internet of Things, Big Data analytics, knowledge discovery, and Internet technologies. He is ranked by Microsoft Academic as one of the Most Impactful Authors in Services Computing (ranked Top 5 All Time).

Jian Yang received the PhD degree from Australian National University, in 1995. She is a full professor and research director with the School of Computing, Macquarie University. Before she joined Macquarie University, she worked as an associate professor with Tilburg University, The Netherlands (2000-2003), a senior research scientist with the Division of Mathematical and Information Science, CSIRO, Australia (1998–2000), and as a lecturer (assistant professor) with the Department of Computer Science, the Australian Defence Force Academy, University of New South Wales (1993-1998).

Xin Wang (Member, IEEE) received the PhD degree in computer science and technology from Nankai University, China, in 2009. During June 2013 to September 2013, he worked as a visiting scholar with Griffith University, Australia. He is currently a full professor and vice-dean of the College of Intelligence and Computing, Tianjin University. His research interests include Semantic Web data management, graph databases, and large scale knowledge processing. He is a CCF senior member, ACM member, IEEE/CS member, CAAI (China Association for Artificial Intelligence) member, and CIPS (Chinese Information Processing Society of China) member. He and his group with Tianjin University have published more than 40 journal and conference papers.

Wei Emma Zhang received the PhD degree from the School of Computer Science, The University of Adelaide, in 2017. She is a lecturer with the School of Computer Science, the University of Adelaide, and an Honorary lecturer with the School of Computing, Macquarie University. Her research interests include text mining, natural language processing, information retrieval, and Internet of Things.

Longxiang Gao received the PhD degree in computer science from Deakin University, Australia. He is currently a professor with the Qilu University of Technology (Shandong Academy of Sciences) and Shandong Computer Science Center (National Supercomputer Center in Jinan). He was a senior lecturer Deakin University. His research interests include Fog/Edge computing, Blockchain, data analysis, and privacy protection.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.