Anchored Maximum Communities over Large Directed Graphs

Yang Huang
Hunan University
hy19@hnu.edu.cn

Xu Zhou*
Hunan University
zhxu@hnu.edu.cn

Yan Ding
Hunan University
ding@hnu.edu.cn

Qing Liu
Zhejiang University
qingliucs@zju.edu.cn

Haoxian Xu
Hunan University
xuhaoxian13@163.com

Kenli Li
Hunan University
lkl@hnu.edu.cn

ABSTRACT

User engagement is a powerful tool that analyzes the expansion or unraveling of social networks. There have been many researches on user engagement to anchor critical users for enhancing engagement. However, these researches neglect the inherent directed nature of real-world social networks, such as the unidirectional follower relationships on platforms like X (Twitter). Motivated by this, we introduce the **Anchored (k, l)-Core Maximization (ADCM)** problem over directed graphs for the first time. Given a directed graph G, degree constraints k and l, and a budget b, the goal is to find b vertices in G, whose sustained engagement can maximize the (k, l)-core. We prove the NP-hardness of the ADCM problem and propose the Greedy-based algorithm (Greedy-based) to process it effectively. After that, to mitigate the isolation effect caused by greediness, a vertex scoring function is designed to support core expansion. To boost the computational performance, we develop pruning techniques, candidate reduction strategies, and an upper-bound-based termination criterion, based on which we design the **F**ast **A**nchor **D**-core (FAD) algorithm. Extensive experiments on eight real-world datasets demonstrate the effectiveness and efficiency of our proposed algorithms. From experimental results, FAD achieves over 1000× speedup compared to Greedy-based when $b > 3$.

PVLDB Reference Format:
Yang Huang, Xu Zhou, Yan Ding, Qing Liu, Haoxian Xu, and Kenli Li. Anchored Maximum Communities over Large Directed Graphs. PVLDB, 19(6): 1345 - 1357, 2026.
doi:10.14778/3797919.3797939

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/sanxian13/FAD.

1 INTRODUCTION

Online social networks are inherently vulnerable to cascading failures, in which the departure of a few users can trigger a chain reaction, prompting weakly connected individuals in forums or groups to disengage from the platform, ultimately threatening the stability of the entire system [2]. This "domino effect," observed in the collapse of platforms such as Friendster[1] and Tianya[2], is known as network unraveling, where users lose active neighbors and gradually withdraw from interaction. Because incentivizing all users is infeasible, maintaining the engagement of critical users is essential to the health of online communities.

Table 1: Existing Anchoring Models

Works	Graph Type	Entity	Structure
[4, 22, 27, 34, 38, 39]	Undirected	Vertex	k-core
[38, 40, 41]	Undirected	Edge	k-truss
[7]	Undirected Attribute	Vertex	k-core
[25, 26, 36]	Undirected	Vertex	Coreness
[8]	Undirected Dynamic	Vertex	k-core
[14, 23, 30]	Directed	–	(k, l)-core
Our Problem	Directed	Vertex	(k, l)-core

Recent studies introduce "*anchor*" users under the anchored k-core model [4, 27, 35, 38–40]. Anchoring reinforces a small set of critical vertices with effectively infinite degrees, ensuring their persistent activity, which in turn stabilizes neighboring vertices, referred to as *followers*. This localized reinforcement prevents cascading removals, thereby preserving the overall structural stability of the network. In practice, anchoring can be interpreted as providing targeted incentives (e.g., visibility boosts, early-access privileges, or rewards for consistent posting) to ensure long-term participation of critical users.

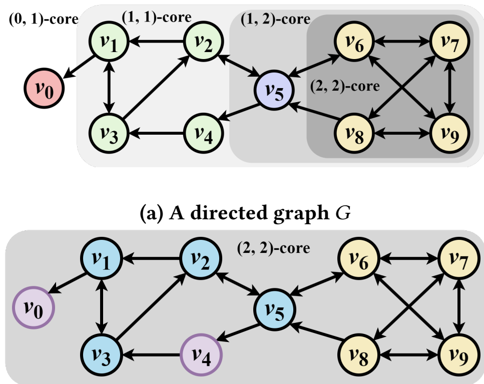
However, existing anchored models are designed for undirected networks, neglecting the asymmetric nature of many real-world systems. In networks such as X[3] (formerly Twitter), citation graphs, and epidemic spreading networks [29], relationships are inherently asymmetric, where user persistence depends jointly on outgoing and incoming interactions.

Motivated by this gap, we identify the **Anchored (k, l)-Core maximization (ADCM) problem** in directed graphs for the first time. The aim is to select up to b anchors—vertices whose in-/out-degrees are treated as infinite—to maximize the size of the resulting (k, l)-core, where each non-anchor vertex has at least k out-neighbors and l in-neighbors. Such a structure captures the engagement patterns observed in real platforms, where each user both follows and is followed by a sufficient number of active users. Table 1 summarizes the key differences between existing anchoring models and our proposed formulation.

*Xu Zhou is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 6 ISSN 2150-8097.
doi:10.14778/3797919.3797939

[1]https://friendster.com/
[2]https://www.tianya.cn/
[3]https://x.com/



(a) A directed graph G

(b) Anchored (2,2)-core with $b = 2$

Figure 1: An Example of the ADCM Problem.

Example 1. *Fig. 1 shows ADCM in an information propagation network (e.g., a technical forum) with 10 users $v_0, v_1, \ldots, v_9$. A directed edge $u \longrightarrow v$ represents user u propagating information to user v. When an active user stops posting, users who depend on their content lose their information sources, and downstream users lose their receivers, resulting in a cascading decline in engagement. Anchoring a small number of critical users can sustain bidirectional information flow within local communities and prevent such instability from spreading.*

We assume $k = l = 2$, meaning that each user must have at least two outgoing and two incoming neighbors to remain in the $(2, 2)$-core, reflecting the reciprocity characteristic of a stable community. As shown in Fig. 1(a), there are only four users, $C_{2,2}(G) = \{v_6, v_7, v_8, v_9\}$ (in yellow), forming a $(2, 2)$-core, while the remaining users v_0–v_5 have left the community group due to insufficient in-degree and out-degree. If the budget is $b = 2$, incentivizing v_0 and v_4 (in violet) jointly stabilizes $\{v_1, v_2, v_3, v_5\}$ (in blue), expanding the anchored $(2, 2)$-core to $C_{2,2}(G_A) = \{v_0, v_1, \ldots, v_9\}$, where G_A denotes the graph after anchoring the set $A = \{v_0, v_4\}$, as shown in Fig. 1(b). This example shows that anchoring a small set of critical users effectively mitigates cascading disengagement.

Applications. The ADCM problem arises in many real-world directed systems, including:

- **Supply–Demand Systems.** In logistics and production networks, facilities often rely on multiple upstream suppliers and downstream distribution partners to hedge against disruptions and maintain stable operations [9, 31]. The (k, l) condition models dual dependencies: each node maintains at least l active upstream and k downstream connections. Since disruptions tend to propagate through sparsely connected facilities, reinforcing a limited set of vulnerable suppliers or hubs (i.e., anchoring them) can mitigate cascading production halts and enhance supply-chain robustness.
- **Security–Control Systems.** Power grids and communication networks contain asymmetric operational dependencies (e.g., control–response, upstream–downstream) that make them vulnerable to cascading failures [5, 6]. Because large-scale infrastructures cannot feasibly reinforce all vulnerable components, operators often focus on a small set of key units whose guaranteed operability helps stabilize their dependent nodes. Modeling such systems as directed graphs, ADCM provides an abstract framework for selecting anchor vertices that maximize a resilient (k, l)-core, supporting tasks such as preventive maintenance, redundancy planning, and emergency stabilization.

Challenges. As the first to propose the ADCM problem, we theoretically prove its NP-hardness, non-submodularity, and inapproximability. Experimental results further demonstrate that the cost of implementing exact algorithms is prohibitively high, and even simple greedy algorithms are highly time-consuming.

- Unlike undirected settings, anchoring in directed graphs coupled in- and out-degree requirements. This asymmetry precludes the direct adaptation of existing k-core algorithms.
- Non-submodularity implies that early anchoring gains do not reliably predict future improvements; short-term increases in core size may fail to materialize into larger long-term gains, causing local decisions to be suboptimal.
- Since directed (k, l)-cores lack full nesting, candidate anchors cannot be pruned efficiently, and evaluating a single anchor may trigger cascading follower updates, resulting in substantial redundant exploration.

The bottlenecks are twofold: **Anchor candidate explosion**: Nodes outside the (k, l)-core are considered candidate anchors, and as k and l increase, the number of candidate anchors grows significantly. **Costly follower filtering**: Scanning potential followers for each candidate anchor takes $O(|V|)$ time, but most fail to meet the follower conditions, leading to substantial evaluation overhead.

Our solution. To address the challenges of the ADCM problem, we propose FAD, a greedy-based framework designed to efficiently compute candidate anchors and followers. The algorithm organizes potential followers using an Order-reachable Path structure, which is derived from the core decomposition, and employs a scoring function that takes into account both current and future follower gains. We begin by performing a layered core decomposition from the $(k-1, l-1)$-core to the (k, l)-core, utilizing a Disjoint Set Union structure to maintain connectivity information. This preprocessing step enables the efficient calculation of candidate followers and their corresponding scores. Since an anchor only influences the vertices along its order-reachable path, we refine the vertex space into different shells to prune ineffective anchors based on follower requirements. Furthermore, we introduce a Hierarchical Traceback Cascade Strategy, which refines the exploration of followers for each candidate anchor and further prunes followers. Guided by a greedy strategy, we select anchors based on a scoring function and optimize the computation by leveraging upper bounds on each anchor's score, thereby reducing unnecessary anchor computations. If the structure of a DSU component remains unchanged after anchoring, we reuse previously computed results via lazy updates.

Contributions. Key contributions are summarized as follows.

- We formulate the ADCM problem on directed graphs, prove its NP-hardness and non-submodularity, and propose the FAD algorithm to solve it (Sections 2.1 and 3.6).
- We develop a greedy framework guided by a novel scoring function for incremental gain (Sections 2.2 and 3.3).
- We exploit ADCM properties to prune anchor and follower candidates (Section 3.1).

- We develop a DSU-based hierarchical order-reachable path structure and a Hierarchical Traceback Cascade Strategy to accelerate follower computation (Section 3.2).
- We adopt a lazy reuse strategy to reduce redundancy and accelerate anchor selection (Section 3.5).
- We evaluate our method on eight real-world datasets to demonstrate its efficiency and effectiveness (Section 4).

Section 5 reviews related work, and Section 6 concludes the paper.

2 PRELIMINARIES

In this section, we formulate ADCM and prove its NP-hardness, inapproximability, and non-submodularity. Table 2 summarizes the main notations.

2.1 Problem Definition

Let $G = (V, E)$ be an unweighted directed graph, where V and E are the sets of vertices and edges, respectively. The cardinalities $|V|$ and $|E|$ denote their sizes. For a vertex $u \in V$, $N_{in}(u, G)$ and $N_{out}(u, G)$ are the in-neighbor and out-neighbor sets of u in G, i.e., $N_{out}(u, G) = \{v \in V \mid (u, v) \in E\}$, $N_{in}(u, G) = \{v \in V | (v, u) \in E\}$. The out-degree and in-degree of u in G are denoted by $\delta_G^{out}(u)$ and $\delta_G^{in}(u)$, i.e., $\delta_G^{out}(u) = |N_{out}(u, G)|$ and $\delta_G^{in}(u) = |N_{in}(u, G)|$. Here, if a vertex u is an anchor, we set $\delta_G^{out}(u) = \delta_G^{in}(u) = +\infty$, ensuring it is always retained in the (k, l)-core. For $A \subseteq V$, let G_A denote G with every $v \in A$ anchored, and write G_u for $G_{\{u\}}$. Next, we introduce the (k, l)-core model on directed graphs.

Definition 1 ((k,l)-core[15]). *Given a directed graph $G = (V, E)$ and integers k and l, a (k, l)-core (also called D-core) of G, denoted by $C_{k,l}(G)$, is a maximal subgraph $S \subseteq G$ such that every vertex $u \in S$ satisfies out-degree $\delta_S^{out}(u) \geq k$ and in-degree $\delta_S^{in}(u) \geq l$.*

Example 2. *The subgraph $S = \{v_5, \ldots, v_9\}$ in Fig. 1 is a maximal $(1, 2)$-core, since any vertex outside S has fewer than $k = 1$ out-degree or $l = 2$ in-degree within the resulting subgraph.*

Property 1 (Disconnected). *A D-core may not be connected, i.e., it may consist of multiple connected components.*

Property 2 (Partially Nested[14]). *Given a directed graph, for any two D-cores, i.e., (k_1, l_1)-core and (k_2, l_2)-core, one is nested in the other only if (i) $k_1 \geq k_2$ and $l_1 \geq l_2$, or (ii) $k_1 \leq k_2$ and $l_1 \leq l_2$.*

Definition 2 (Followers). *Given G and a vertex u, the followers of anchoring u are $F(u, G) = C_{k,l}(G_u) \setminus \big(C_{k,l}(G) \cup \{u\}\big)$. Similarly, for $A \subseteq V$, $F(A, G) = C_{k,l}(G_A) \setminus \big(C_{k,l}(G) \cup A\big)$.*

We write $F(u, G)$ as $F(u)$, $F(A, G)$ as $F(A)$, and $C_{k,l}(G)$ as $C_{k,l}$.

Definition 3 (Anchored (k,l)-core). *Given a directed graph G and an anchored vertex set $A \subseteq V(G)$, the anchored (k, l)-core, denoted by $C_{k,l}(G_A)$, is the (k, l)-core with vertices anchored in A.*

Problem Statement. Given a directed graph G, a budget b, and degree constraints $k, l \geq 0$, ADCM aims to select a set $A \subseteq V$, with $|A| \leq b$, that maximizes the increase in the (k, l)-core size excluding A, i.e.,

$$\arg \max_{A \subseteq V,\, |A| \leq b} g(A), \tag{1}$$

where $g(A) = |F(A, G)|$.

Table 2: Summary of Notations

Notation	Description
G	A directed graph.
A	The set of anchored vertices.
G_A	The graph obtained by anchoring vertices in A.
$F(u), F(A)$	Followers of an anchor u or an anchor set A.
$C_{k,l}(G)$	The (k, l)-core of G.
$H_{k,l}(G)$	The (k, l)-shell of G.
$N_{out}(u)$	Out-neighbors of u in G.
$N_{in}(u)$	In-neighbors of u in G.
$L(u)$	Layer index of u.
$u \rightsquigarrow v$	v is order-reachable from u.
$U(u)$	Order-reachable region of u.

Example 3. *In Fig. 1, the $(1, 2)$-core of the directed graph G is $C_{1,2}(G) = \{v_5, \ldots, v_9\}$. When $b = 1$, anchoring $A = \{v_4\}$ yields the largest gain, with followers $F(v_4) = \{v_1, v_2, v_3\}$ and $C_{1,2}(G_A) = \{v_1, \ldots, v_9\}$. Conversely, anchoring v_0, v_1,v_2 yields no followers, and anchoring v_3 activates only $\{v_1, v_2\}$.*

Theorem 1. *ADCM is NP-hard for any $k \geq 1$ or $l \geq 1$.*

Proof. We prove NP-hardness by reducing Max-3SAT [20] to ADCM. Given a 3-CNF formula with n variables and m clauses, construct a directed graph $G = (V, E)$ consisting: (i) literal vertices $\{x_i, \neg x_i\}$ for each variable; (ii) a clause vertex c_j for each clause C_j; and (iii) a constant-size auxiliary core C_{aux}.

Case 1: $k \geq 1$ and $l \geq 0$. Set the anchor budget $b = n$. Literal vertices have no outgoing edges except those of anchored literals, so when $k \geq 1$, every non-anchored literal is peeled and removed from the (k, l)-core. Thus an anchor set A encodes a full truth assignment over the variables. Each clause vertex c_j receives l incoming edges and $(k-1)$ outgoing edges from C_{aux}. The l incoming edges guarantee the in-degree constraint ($\delta^{in} \geq l$), while the $(k-1)$ outgoing edges create a deficit of exactly one required outgoing edge. For each literal ℓ in clause C_j, we add an edge $c_j \to \ell$. Formally,

$$c_j \in C_{k,l}(G_A) \iff C_j \text{ is satisfied by the assignment encoded by } A.$$

Therefore the size of the anchored core is linearly related to the number of satisfied clauses s: $|C_{k,l}(G_A)| = |C_{\text{aux}}| + s$, where s is the number of satisfied clauses. Maximizing the ADCM objective is thus equivalent to maximizing s. Since Max-3SAT is NP-hard, ADCM is NP-hard for all $k \geq 1$.

Case 2: $k \geq 0$ and $l \geq 1$. By symmetry, the result follows from Case 1 applied to the reverse graph G^R. ◻

Theorem 2. *The function $g(\cdot)$ of ADCM is not submodular.*

Proof. The objective function $g(\cdot)$ is submodular if $g(A \cup \{x\}) - g(A) \geq g(B \cup \{x\}) - g(B)$ holds for all $A \subseteq B$ and $x \notin B$. Consider the directed graph in Fig. 2 with k=l=1. Let $A = \emptyset$ and $B = \{u_1\}$, where $A \subset B$. It holds that $C_{1,1}(G) = \emptyset, C_{1,1}(G_A) = \emptyset$ and $C_{1,1}(G_B) = \{u_1\}$. Recall that $g(S)$ denotes the number of followers of S. Thus, $g(A) = |F(A)| = 0$ and $g(B) = |F(B)| = 0$. Now add $x = u_3$ to both A and B. We obtain $C_{1,1}(G_{A\cup\{x\}}) = \{u_3\}$ and $C_{1,1}(G_{B\cup\{x\}}) = \{u_1, u_2, u_3, u_4, u_5, u_6\}$. Accordingly, $g(A \cup \{x\}) - g(A) = 0 < 4 = g(B \cup \{x\}) - g(B)$, violating submodularity. ◻

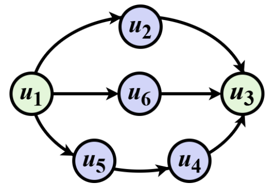


Figure 2: A directed graph for the submodular proof.

THEOREM 3. *ADCM cannot be approximated in polynomial time within a factor of* $\boldsymbol{\alpha} + \varepsilon$ *for any constant* $\varepsilon > 0$*, unless* P = NP.

A full proof is given in the technical report[19], based on a gap-preserving reduction from the Max-3SAT hardness [17].

2.2 Greedy Framework

A straightforward approach to ADCM is to enumerate all b-sized subsets of V and select the one maximizing $g(A)$. However, this exact strategy has exponential time complexity $O\left(\binom{|V|}{b} \cdot |E|\right)$, making it impractical given the NP-hardness and non-submodularity of ADCM. Therefore, we adopt a greedy-based framework with a score-based selection strategy to obtain an effective anchor set.

Algorithm 1 outlines the pseudocode of our Greedy-based algorithm framework. **Initialization**: In each iteration, candidate anchors violating the (k, l)-core constraint are identified via core decomposition (Lines 3-4). **Evaluation and Selection**: For each candidate $v_x \in cA$, compute its follower set (Lines 5-6) and select either the vertex with the highest score or the one with the largest follower set (Line 7). **Termination**: Once the budget b is exhausted, the algorithm returns the final anchor set A (Line 10).

Algorithm 1: Greedy-based Algorithm

Input: $G(V, E)$: directed graph, b: budget size, k: out-degree constraint, l: in-degree constraint
Output: A: an anchored set

1 $A \leftarrow \emptyset, cA \leftarrow \emptyset$;
2 **while** $b > 0$ **do**
3 $\quad C_{k,l} \leftarrow$ DECOMPOSE(G, k, l, A);
4 $\quad cA \leftarrow V \setminus C_{k,l}$;
5 $\quad$**foreach** $v \in cA$ **do**
6 $\quad\quad$Compute Follower(v, A, G, k, l);
7 $\quad v_x \leftarrow \arg\max_{v \in cA} \phi(v)$; // $\phi(v)$ **alternates between** $Score(v)$ **and** $F(v)$
8 $\quad A \leftarrow \{A \cup v_x\}$;
9 $\quad b \leftarrow b - 1$;
10 **return** A
11 **Function** *Decompose(G, k, l, A)*
12 $\quad$Initialize $S \leftarrow \emptyset$, degree $\delta^{in}, \delta^{out}$ for all $v \in V$;
13 $\quad$**foreach** $v \in A$ **do**
14 $\quad\quad$Set $\delta^{in}(v) \leftarrow +\infty, \delta^{out}(v) \leftarrow +\infty$;
15 $\quad$**while** *there exists* $v \in V \setminus (S \cup A)$ *with* $\delta^{out}(v) < k$ *or* $\delta^{in}(v) < l$ **do**
16 $\quad\quad S \leftarrow S \cup \{v\}$;
17 $\quad\quad$**foreach** *neighbor* u *of* v **do**
18 $\quad\quad\quad$Update $\delta^{in}(u)$ or $\delta^{out}(u)$ accordingly;
19 $\quad$**return** $V \setminus S$

3 FAST ANCHOR D-CORE OPTIMIZATION

Although Algorithm 1 is straightforward, its efficiency is limited by the large candidate space and redundant follower computation. Moreover, its reliance on immediate follower counts often overlooks interactions among anchors. To address these issues, we propose the **F**ast **A**nchor **D**-core (FAD) algorithm. FAD accelerates greedy selection by pruning unpromising candidates and localizing follower computation (Sections 3.1 and 3.2). It further improves solution quality and efficiency via a novel scoring function, upper-bound based termination, and reuse techniques (Sections 3.3–3.5).

3.1 Reducing Potential Anchors

Computing the follower set for every candidate anchor dominates the cost of Algorithm 1. By characterizing the structural prerequisites for anchored propagation, we prune all vertices that can never activate any follower, thereby reducing the candidate space. We further accelerate candidate identification through a parallel decomposition strategy.

DEFINITION 4 ((k, l)-SHELL). *The* (k, l)*-shell consists of the vertices in* $C_{k,l}(G)$ *that do not belong to any strictly higher* $(k+1, l)$*- or* $(k, l+1)$*-core:* $H_{k,l}(G) = C_{k,l}(G) \setminus \big(C_{k+1,l}(G) \cup C_{k,l+1}(G)\big)$.

EXAMPLE 4. *Revisiting Fig. 1, the* $(1, 1)$*-shell is* $H_{1,1} = C_{1,1} \setminus (C_{1,2} \cup C_{2,1}) = \{v_1, v_2, v_3, v_4\}$ *(as highlighted in green). For anchoring the* $(2, 2)$*-core, Theorem 4 gives the candidate follower set* $\{v_1, v_2, v_3, v_4, v_5\}$*. According to Theorem 5,* v_0 *is not a valid anchor since it contributes only one outgoing edge to* v_1*, which still fails to meet the* $(2, 2)$ *requirements. In contrast,* v_2 *is a feasible candidate anchor because it can supply the required incoming edge to* v_5.

THEOREM 4 (CANDIDATE FOLLOWER). *All followers of an anchor* a *must lie in* $C_{k-1,l-1}(G) \setminus C_{k,l}(G) = H_{k-1,l-1} \cup H_{k-1,l} \cup H_{k,l-1}$.

PROOF. Let $S' = C_{k,l}(G_a)$ be the anchored (k, l)-core after anchoring a. For any vertex $v \in S' \setminus \{a\}$, removing a decreases the in- and out-degrees of v by at most one. Hence $\delta^{out}_{S' \setminus \{a\}}(v) \geq k - 1$ and $\delta^{in}_{S' \setminus \{a\}}(v) \geq l - 1$, which implies $S' \setminus \{a\}$ is a feasible $(k - 1, l - 1)$-subgraph of G. By maximality of $C_{k-1,l-1}(G)$, we have $S' \setminus \{a\} \subseteq C_{k-1,l-1}(G)$. Therefore every follower lies in $C_{k-1,l-1}(G) \setminus C_{k,l}(G)$. □

THEOREM 5 (CANDIDATE ANCHORS). *The candidate anchor set for the* (k, l)*-core is* $cA = (N_{out}(H_{k-1,l-1}) \cap N_{in}(H_{k-1,l-1})) \cup N_{out}(H_{k-1,l}) \cup N_{in}(H_{k,l-1}) \setminus C_{k,l}$.

PROOF. Vertices in $H_{k-1,l-1}$ lack both in- and out-degree; the anchor must provide both. Vertices in $H_{k-1,l}$ lack only an out-neighbor; the anchor must provide an outgoing edge. Vertices in $H_{k,l-1}$ lack only an in-neighbor; the anchor must provide an incoming edge. □

Guided by the partial nesting property (Property 2), the shell hierarchy required by Theorem 5 can be computed efficiently via parallel decomposition. Algorithm 2 first computes $C_{k-1,l-1}$ (Line 1), and then decomposes this subgraph in parallel to obtain $C_{k,l-1}$, $C_{k-1,l}$, and $C_{k,l}$ (Lines 2–3). Each shell is derived through simple set-difference operations (Lines 4-7). Finally, the candidate anchor set cA is constructed using Theorem 5 and returned (Line 8). In practice, all core decompositions are implemented by Function 3.

Algorithm 2: CandidateAnchor(G, k, l, A)

Input: $G(V, E)$: directed graph, (k, l): degree constraint, A: anchored set
Output: cA : Candidate anchors

$C_{k-1,l-1} \leftarrow$ **Decompose**$(G, k-1, l-1, A)$;
for *each* $(i, j) \in \{(k-1, l), (k, l-1), (k, l)\}$ **do in parallel**
  $C_{i,j} \leftarrow$ **Decompose**$(C_{k-1,l-1}, i, j, A)$;
$H_{k-1,l-1} \leftarrow C_{k-1,l-1} \setminus (C_{k,l-1} \cup C_{k-1,l})$;
for *each* $(i, j) \in \{(k-1, l), (k, l-1)\}$ **do in parallel**
  $H_{i,j} \leftarrow C_{i,j} \setminus C_{k,l}$;
cA is calculated by Theorem 5;
return cA

Complexity. Algorithm 2 performs a constant number of core decompositions, each in $O(|E|)$ time. The shell computations and the scan over cA require $O(|V|)$ set operations. Thus, the overall worst-case running time is $O(|V| + |E|)$.

3.2 Hierarchy-based Follower Calculation

Although pruning candidate anchors greatly reduces the search space, computing followers remains expensive. To localize anchor influence and avoid exploring irrelevant regions, we introduce a hierarchy-based follower computation strategy, supported by an optimized shell decomposition that constructs a layered structure.

3.2.1 ***Layer Structure***. Since follower activation depends on how vertices obtain the missing degree support (Theorems 4 and 5), we introduce a layered structure that captures this propagation order.

Definition 5 (Layer structure). *Let $L(v)$ denote the index of the peeling round in which vertex v is removed during the (k, l)-core decomposition performed on the $(k-1, l-1)$-core subgraph. If vertex v is removed after vertex u, then $L(u) < L(v)$.*

Example 5. *As shown in Fig. 3, a layer structure is preconstructed from Fig. 1 for anchoring the $(2, 2)$-core. The process begins by removing $C_{0,0} \setminus C_{1,1} = \{v_0\}$, forming layer L_0. Then, a $(2, 2)$-core decomposition is performed within $C_{1,1}(G)$: $\{v_1, v_4\}$ are removed first to form L_1; their removal triggers violations in $\{v_2, v_3\}$, which are subsequently removed to form L_2; finally, v_5 is removed as L_3.*

Definition 6 (Order-reachable Path and Region). *A path $u = u_0 \rightsquigarrow u_1 \rightsquigarrow \dots \rightsquigarrow u_m = v$ is* order-reachable *if the layer index is non-decreasing along the path, i.e., $L(u_i) \le L(u_{i+1})$. The* ***order-reachable region*** *of u with respect to the D-core is $U(u) = \{x \in C_{k-1,l-1}(G) \mid \exists c \in C_{k,l}(G) : u \rightsquigarrow x \rightsquigarrow c$ is order-reachable$\}$.*

The layer-successor neighbors of u are defined as $\mathcal{N}^+(u) = \{v \in N_{in}(u) \cup N_{out}(u) \mid L(v) > L(u)\}$. We write $u \rightsquigarrow v$ if there exists an order-reachable path from u to v.

Example 6. *Revisiting Example 5, there exists at least one order-reachable path from v_0 to the $(2, 2)$-core via v_1, such as $\{v_0 \rightsquigarrow v_1 \rightsquigarrow v_3 \rightsquigarrow v_2 \rightsquigarrow v_4 \rightsquigarrow v_5 \rightsquigarrow C_{2,2}(G)\}$, and $\{v_1 \rightsquigarrow v_0 \rightsquigarrow v_2 \rightsquigarrow v_5 \rightsquigarrow C_{2,2}(G)\}$. Since no vertex in G has v_4 as a layer-successor neighbor, i.e., $v_4 \notin \mathcal{N}^+(u)$ for any $u \in G$, we have $U(v_0, C_{2,2}(G)) = \{v_1, v_2, v_3, v_5\}$.*

Theorem 6. *If $u \notin U(v)$, then u cannot be a follower of anchor v.*

Proof. Consider a vertex $u \notin U(v)$. During the $(k-1, l-1)$-core decomposition, either $L(v) \le L(u)$, so v is removed no later than u and cannot increase u's degree, or $L(v) \ge L(u)$, so u and v are disconnected in the $(k-1, l-1)$-core and v still cannot contribute to u's degree. In either case, u cannot be a follower of v. □

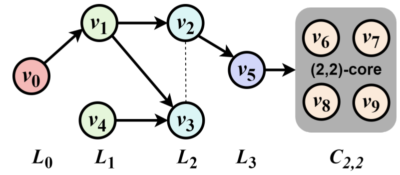


Figure 3: An Order-reachable Path Diagram of Fig. 1 **(dashed lines denote intra-layer links; solid arrows indicate cross-layer order-reach paths.)**

Theorem 7. *If vertex $v \in F(u, G_A)$, then $F(v, G_A) \subseteq F(u, G_A)$.*

Proof. If $v \in F(u)$, there exists an order-reachable path $u \rightsquigarrow v$ in G_A. For any $w \in F(v)$, there exists $v \rightsquigarrow w$. Concatenating these paths gives $u \rightsquigarrow w$, preserving the (k, l)-degree constraints. Thus, $w \in F(u)$, and $F(v, G_A) \subseteq F(u, G_A)$. □

These structural properties restrict follower computation to the order-reachable region of a candidate anchor, which prunes invalid followers and significantly accelerates the computation.

3.2.2 ***Cascade Components via DSU***. Property 1 shows that a D-core may split into multiple parts, and Theorem 6 further restricts the effect of an anchor to its own order-reachable region. To capture these local deletion dependencies, we integrate a Disjoint Set Union (DSU) into the shell-decomposition process: whenever the removal of a vertex v triggers the deletion of a neighbor u, we union u and v. The resulting cascade-connected components summarize the peeling dependencies and are later used in the reuse step (Section 3.5) to identify candidates unaffected by the newly selected anchor.

Function 3: OptShellDec(G, k, l)

Initialize $Q, V' \leftarrow \emptyset, V \leftarrow V(G)$, DSU over V, $layerNum \leftarrow 1$;
Initialize $d_{in}(v) \leftarrow |N_{in}(v)|$, $d_{out}(v) \leftarrow |N_{out}(v)|$ for all $v \in V$;
foreach $v \in V$ **do**
  if $d_{out}(v) < k$ ***or*** $d_{in}(v) < l$ **then**
    Q.enqueue(v); $V' \leftarrow V' \cup \{v\}$; $L(v) \leftarrow layerNum$;
while Q *not empty* **do**
  $size \leftarrow |Q|$;
  for $i \leftarrow 1$ **to** $size$ **do**
    $v \leftarrow Q$.dequeue();
    foreach $x \in N_{out}(v)$ ***and*** $x \notin V'$ **do**
      $d_{in}(x) \leftarrow d_{in}(x) - 1$;
      if $(d_{out}(x) < k$ ***or*** $d_{in}(x) < l)$ **then**
        Q.enqueue(x); $V' \leftarrow V' \cup \{x\}$;
        $L(x) \leftarrow layerNum$; DSU.union(x, v);
    foreach $x \in N_{in}(v)$ ***and*** $x \notin V'$ **do**
      $d_{out}(x) \leftarrow d_{out}(x) - 1$;
      if $(d_{out}(x) < k$ ***or*** $d_{in}(x) < l)$ **then**
        Q.enqueue(x); $V' \leftarrow V' \cup \{x\}$;
        $L(x) \leftarrow layerNum$; DSU.union(x, v);
  $layerNum \leftarrow layerNum + 1$;
return $V \setminus V', L, DSU$;

Function 3. Each vertex v with in-/out-degree less than k/l is enqueued into Q, and we set $L(v) = layerNum$ (Lines 3–5). In Lines 7–18, the current batch with the smallest layer index is dequeued (Line 9), identifies u whose degree constraints are violated due to removals in the current batch, assigns them the layer index $L(u) = layerNum$, and unites them with their triggering vertices via DSU (Lines 10-18), which implicitly constructs a sequentially reachable path U among removed vertices. The *layerNum* is incremented, and the process repeats until Q is empty (Line 19). The algorithm outputs the (k, l)-core, vertex layer indices, and DSU components.

Complexity. Function 3 runs in $O(|V|+|E|\alpha(V))$, where $O(|V|+|E|)$ is from BFS traversal and $O(|E|\alpha(V))$ from union-find. Here, $\alpha(V)$ denotes the inverse Ackermann function in DSU.

3.2.3 *Hierarchical Traceback Cascade Strategy.*

The core of our follower computation is a two-phase mechanism: Priority-based Forward Expansion and Traceback Cascade. To minimize redundant checks, we follow the layer hierarchy defined by the peeling process. Starting from the anchor a, vertices are explored along order-reachable paths. By processing vertices in non-decreasing layer order, each vertex is evaluated only after the support provided by lower-layer vertices has been stabilized. Whenever a vertex under examination violates the (k, l)-core constraints, a traceback cascade is triggered to immediately revoke all dependent vertices in higher layers, preventing the accumulation of invalid support and ensuring the maximality of the resulting anchored core.

Algorithm. Algorithm 4 presents the procedure for computing the follower set of a candidate anchor a. First, the follower set F, destroyed vertex set DE, and the pending deletion queue Q^D are initialized. If anchor a is covered by previously processed anchors, the computation terminates immediately (Lines 2–3). Line 4 computes the order-reachable region U, i.e., all vertices above a in the layer hierarchy. Line 5-6, all layer-successor neighbors of a are inserted into a min-priority queue Q, which expands vertices in non-decreasing layer order. The visited set V stores all vertices already inserted into Q. Lines 7–9 initialize temporary in-/out-degree supports $d'_{in}[\cdot]$ and $d'_{out}[\cdot]$ for each vertex in U, counting only neighbors within U and the current core. In the main loop (Lines 10–27), the algorithm always pops from Q the vertex with the minimum layer index. If v has already been deleted, it is skipped (Lines 12-13). Otherwise, its temporary degree supports are checked against the (k, l)-core constraints (Lines 14-18). If both constraints are satisfied, v is tentatively added to the follower set (Line 15), and all its successor-layer neighbors in U are inserted into Q if they have not been visited (Lines 16-18). Otherwise, v is marked as deleted and placed into the deletion queue Q^D (Lines 19-20). Lines 21–27 perform the Hierarchical Traceback Cascade. the algorithm iteratively dequeues a vertex u from Q^D, removes it from the follower set F, and updates the structure information of its neighbors N_u (Lines 23-25). If either degree falls below the required threshold, x is also added to the deletion queue, propagating the deletion effect. Finally, When the priority queue Q becomes empty, the remaining vertices in F constitute the follower set of anchor a.

Algorithm 4: FOLLOWER(a, A, G, k, l, L)

Input: a: anchor, A: current anchor set, G: directed graph, (k, l): degree constraints, L: layer structure
Output: F: follower set of a

1 Initialize $F \leftarrow \emptyset$, $DE \leftarrow \emptyset$, $Q^D \leftarrow \emptyset$;
2 **if** *a is covered by previous anchors in A (Theorem 7)* **then**
3   **return** F;
4 Compute order-reachable region $U \leftarrow U(a)$ (Definition 6);
5 Min-priority queue $Q \leftarrow \{(L(v), v) \mid v \in \mathcal{N}^+(a) \cap U\}$;
6 Visited set $V \leftarrow \{v \mid (L(v), v) \in Q\}$;
7 **foreach** $v \in U$ **do**
8     $d'_{in}[v] = \left|N_{in}(v) \cap \left(U \cup C_{k,l}(G_A)\right)\right|$;
9     $d'_{out}[v] = \left|N_{out}(v) \cap \left(U \cup C_{k,l}(G_A)\right)\right|$;
10 **while** *Q not empty* **do**
11     $(L(v), v) \leftarrow Q.\text{POP\_MIN\_L}()$;
12   **if** $v \in DE$ **then**
13   **continue**
14   **if** $d'_{in}[v] \geq l$ ***and*** $d'_{out}[v] \geq k$ **then**
15   Add v to F;
16   **foreach** $u \in (N_{in}(v) \cup N_{out}(v)) \cap U$ **do**
17   **if** $u \notin V$ **then**
18                 $Q.\text{PUSH}((L(u), u))$; $V \leftarrow V \cup \{u\}$;
19   **else**
20   Add v to DE and Q^D;
21   **while** *Q^D not empty* **do**
22         $u \leftarrow Q^D.pop()$; Remove u from F;
23   **foreach** $x \in (N_{in}(u) \cup N_{out}(u)) \cap F$ **do**
24             $d'_{out}[x] \mathrel{-}= [u \in N_{out}(x)]$;
25             $d'_{in}[x] \mathrel{-}= [u \in N_{in}(x)]$;
26   **if** $d'_{in}[x] < l$ ***or*** $d'_{out}[x] < k$ **then**
27   Add x to DE and Q^D;
28 **return** F

EXAMPLE 7. *Figs. 1 and 3 show the follower computation for anchor $a = v_0$ in the $(2, 2)$-core. Initially, the layer-successor v_1 of v_0 is inserted into the priority queue $Q = \{v_1\}$. Since v_1 temporarily satisfies the degree constraints ($d'_{out}(v_1) = |\{v_0, v_3\}| = 2$ and $d'_{in}(v_1) = |\{v_2, v_3\}| = 2$), it is added to the follower set $F = \{v_1\}$. Its layer-successors v_2 and v_3 are then enqueued. Next, v_3 is dequeued and found to violate the in-degree constraint ($d'_{in}(v_3) = |\{v_2\}| = 1 < 2$), since its other in-neighbor v_4 is invalid. The removal of v_3 triggers a cascade that reduces the degree supports of v_1 to one, causing v_1 to be removed from F. Similarly, v_2 is dequeued and also violates the degree constraints ($d'_{in}(v_2) = d'_{out}(v_2) = |\{v_5\}| = 1$). With Q empty, the process terminates with $F(v_0) = \emptyset$.*

Complexity. Algorithm 4 has a time complexity of $O(|E| \log |V|)$ in the worst case. In practice, fewer vertices are typically explored due to strict order-reachable conditions and early termination by the Traceback Cascade Strategy.

3.3 Scoring Function

Although hierarchy-based methods improve computational efficiency of follower identification, they suffer from a fundamental limitation in anchor selection due to their myopic nature. For example, methods such as OLAK [39] iteratively select anchors based on local optima, overlooking global topology and ignoring potential synergies among anchors. This calls for a more topology-aware anchor selection strategy balancing local gain with global cohesion.

Fig. 2 illustrates the limitation of the local optimization strategy in the anchored $(1, 1)$-core problem with $b = 2$. In the first round, anchoring any vertex $u \in G$ yields no followers; u_6 is selected based on descending ID order, resulting in $A = \{u_6\}$ and $F = \emptyset$. In the second round, no remaining vertex contributes any followers, and the process terminates with no gain. In contrast, the optimal anchor set $A = \{u_3, u_1\}$ leads to $F(A) = \{u_2, u_4, u_5, u_6\}$.

Motivated by this, we design a scoring function that not only accounts for the current number of followers but also captures the potential impact on future non-anchor (k, l)-core vertices, thereby balancing local gain with structural influence.

Intuitively, a neighboring vertex in the non-anchored core that is removed earlier during the decomposition process (i.e., at a lower layer) requires a higher budget to remain as a follower. Such vertices are less likely to be retained later, and their influence is accordingly down-weighted in the scoring. The formal scoring function is defined as follows.

Definition 7 (Scoring Function). *The score of an anchor a on the anchored directed graph G_A is defined as:*

$$Score(a, G_A) = |F(a, G_A)| + \sum_{v_i \in \bar{N}_a} \lg\left(1 + \frac{1}{L_{\max} - L(v_i)}\right), \quad (2)$$

where $F(a, G_A)$ is the follower set of a, $\bar{N}_a = (N_{in}(a) \cup N_{out}(a)) \cap (G \setminus C_{k,l}(G_A))$ is the set of non-core neighbors of a, and $L_{\max}$ is the layer index assigned to all vertices in $C_{k,l}(G_A)$.

The first term reflects the direct gain from anchoring a, while the second estimates its potential influence on non-core neighbors v_i. If $v_i \notin C_{k-1,l-1}$, we assign $L(v_i) = 0$.

Example 8. *Revisiting Fig. 2, we compute scores for the anchored $(1, 1)$-core with budget $b = 2$. The peeling layers are $l_1 = \{v_1, v_3, v_4\}$ and $l_2 = \{v_2, v_5, v_6\}$. Candidate anchor v_1 has no followers, and $\bar{N}_{v_1} = \{v_2, v_5, v_6\}$. The second term evaluates to $3 \cdot \lg\left(1 + \frac{1}{3-2}\right)$, giving $Score(v_1) \approx 0.9030$. Similarly, $Score(v_2) = Score(v_5) \approx 0.9030$, $Score(v_6) \approx 0.3522$, $Score(v_3) \approx 0.7781$, and $Score(v_4) \approx 0.4771$. The highest-scoring vertex v_1 is selected as the anchor.*

Next round, v_3 yields followers $\{v_4, v_5, v_6\}$ and has $\bar{N}_{v_3} = \{v_2\}$, giving $Score(v_3) \approx 3.3010$, the highest among all. Thus, v_3 is anchored, and the final anchor set $A = \{v_1, v_3\}$ matches the optimal solution.

3.4 Refined Upper Bound Calculation

Although structurally invalid candidate anchors have been removed in Subsection 3.1, computing the exact score $\text{Score}(x)$ for every candidate anchor is still expensive, because evaluating $|F(x)|$ requires executing Algorithm 4. To avoid unnecessary score evaluations, we estimate an upper bound on the potential score of each candidate. If the estimated bound of a vertex v is lower than the best actual score obtained so far, v is pruned immediately without computing its true score.

Existing bottom-up upper-bound estimators for anchored cores often severely overestimate follower counts due to redundant accumulation. For example, OLAK computes $\text{UB}(x) = \sum_{u \in U}(\text{UB}(u)+1)$, which counts intermediate vertices multiple times along overlapping paths. As shown in Fig. 4, v_0 is assigned an upper bound of 73, while only 22 vertices are actually reachable. This motivates a refined two-hop estimation method that yields substantially tighter bounds for directed graphs.

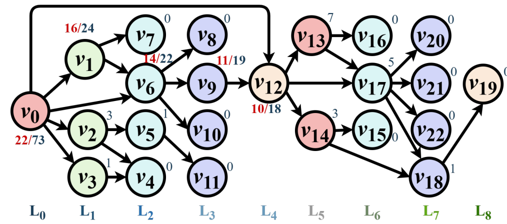


Figure 4: ***Illustration of the refined upper bound with 24 order-reachable vertices from layer L_0 to L_8; vertices $\{v_4, v_5, v_8, \ldots, v_{22}\}$ are initialized with UB$(v_i) = 0$.***

Definition 8 (Order-reachable 1-hop neighbors). *The order-reachable 1-hop set $\mathcal{N}^1(u)$ of vertex u is a maximal subset of $\mathcal{N}^+(u)$ such that for vertices $v_1, v_2 \in \mathcal{N}^1(u)$, $v_1 \notin \mathcal{N}^+(v_2)$ and $v_2 \notin \mathcal{N}^+(v_1)$.*

Definition 9 (Order-reachable 2-hop neighbors). *The order-reachable 2-hop set $\mathcal{N}^2(u)$ of a vertex u consists of all vertices w such that: (i) $w \in \mathcal{N}^+(v)$ for some $v \in \mathcal{N}^1(u)$; and (ii) For any two distinct vertices $w_1, w_2 \in \mathcal{N}^2(u)$, $w_1 \notin \mathcal{N}^+(w_2)$ and $w_2 \notin \mathcal{N}^+(w_1)$.*

Using Definitions 8–9, the refined follower upper bound of a candidate anchor x is defined recursively as:

$$UB'(x) = \begin{cases} \sum_{w \in \mathcal{N}^2(x)} UB(w) + |\mathcal{N}^1(x) \cup \mathcal{N}^2(x)|, & |\mathcal{N}^1(x)| > 0; \\ 0, & \text{otherwise.} \end{cases}$$

The first term aggregates the bounds of 2-hop neighbors, and the second counts 1-/2-hop vertices. Intuitively, if $\mathcal{N}^1(x) = \emptyset$, meaning that all order-reachable successors of x already lie in $C_{k,l}(G_A)$, then x cannot contribute any additional followers and thus $\text{UB}(x) = 0$.

Example 9. *In Fig. 4, the refined upper bound reduces* $\text{UB}(v_0)$, $\text{UB}(v_1)$, $\text{UB}(v_6)$, $\text{UB}(v_9)$, *and* $\text{UB}(v_{12})$ *substantially compared with OLAK. For instance, $\mathcal{N}^1(v_{12}) = \{v_{13}, v_{14}\}$ and $\mathcal{N}^2(v_{12}) = \{v_{15}, v_{16}, v_{17}\}$, yielding:* $\text{UB}(v_{12}) = \min\big(5 + \text{UB}(v_{15}) + \text{UB}(v_{16}) + \text{UB}(v_{17}),\ 10\big) = 10$, *whereas OLAK gives 18. Across the graph, v_0 decreases from 73 to 22, v_1 from 24 to 16, v_6 from 22 to 14, and v_9 from 19 to 11.*

Let $|S_x^*|$ be the number of such vertices produced by Function 3 Thus the final upper bound is:

$$UB(x) = \min(UB'(x), |S_x^*|). \quad (3)$$

The score upper bound $UB_{sco}(x)$ of x is estimated as:

$$\begin{aligned} UB_{sco}(x) &= UB(x) + \sum_{v_i \in \bar{N}_x} \lg\left(1 + \frac{1}{L_{\max} - L(v_i)}\right) \\ &\leq UB(x) + |\bar{N}_x| \cdot \lg(2) \quad (4) \end{aligned}$$

Complexity. Computing $UB(x)$ for all candidate anchors $x \in cA$ takes $O(|E| + |V|\log|V| + \sum|S_x^*|)$ time, which simplifies to $O(|E| + |V|\log|V|)$ in practice, since $\sum|S_x^*| = O(|V|)$.

3.5 Reuse Follower Computation

According to Theorem 6, an anchor only influences vertices on its order-reachable path, i.e., within the same cascade-connected component. However, the greedy strategy anchors one vertex per round, causing repeated evaluation of anchors in disjoint components and resulting in redundant computation. To avoid this, we cache unaffected candidate anchors and their followers in each round via $Reuse(x)$, and reuse follower upper bounds for vertices unaffected by the current anchor (see Section 3.4). These cached results are directly used in subsequent rounds to eliminate recomputation.

Algorithm. Algorithm 5 reuses previously computed information for candidates unaffected by the newly selected anchor a. Line 1 initializes the cache and identifies the cascade component of a. Lines 2–3 scan all candidates and select those whose components differ from a's. If the previous cache stores the exact follower set of such a candidate v (Line 4), all stored information is reused (Line 5); otherwise, only its upper-bound values are reused (Lines 6–7). The updated cache is returned in Line 8.

Algorithm 5: Reuse(a, cA, DSU, R_{prev})

```
Input: a: selected anchor; cA: candidate anchors; DSU: cascade
       components; R_prev: cached results
Output: R: cache for next round
1 R ← ∅;   C_a ← DSU.Find(a);
2 foreach v ∈ cA do
3     if DSU.Find(v) ≠ C_a then
4         if R_prev contains exact followers of v then
5             R[v] ← R_prev[v] ; // reuse F(v), UB(v), UB_sco(v)
6         else if R_prev contains only an upper bound for v then
7             R[v] ← R_prev[v] ;    // reuse (UB(v), UB_sco(v))
8 return R
```

3.6 The FAD Algorithm

Algorithm 6 presents the full workflow of FAD, which integrates shell decomposition, upper-bound estimation, follower computation, and reuse-based acceleration into a unified greedy framework. The algorithm initializes the anchor set A and the reuse cache R (Line 1). For each of the b anchoring rounds (Lines 2-20), the procedure begins by computing the shell-based hierarchical structure of the graph via Function 3, which returns the (k, l)-core C, its surrounding shells S, the layer index array L, and the cascade-connected components maintained by a DSU structure (Line 3). Subsequently, the candidate anchor set cA is extracted following Theorem 5 (Line 4). In Lines 5-7, the algorithm refines or reuses the follower upper bounds for all candidates. If cached information is available in R, it is reused directly; otherwise, the upper bound is recomputed using Eq. (3) (Line 5). The corresponding score upper bound $\text{UB}_{sco}(v)$ for each candidate v is then obtained using Eq. (4) (Lines 6–7). The main greedy selection loop begins at Line 9. Candidates are processed in decreasing order of their score upper bounds. If the current candidate v has $\text{UB}_{sco}(v)$ lower than the best score found so far, the loop terminates early due to the monotonicity of the ordering (Lines 10–11). For each remaining candidate, FAD either reuses its exact follower set from the previous iteration (Lines 12–13) or computes it using Algorithm 4 (Lines 14–15). The actual score of v is then computed (Line 16), and if it exceeds the current maximum, both the best candidate a and the score threshold $maxS$ are updated (Lines 17–18). Once the best anchor a of this round is determined, FAD updates the reuse cache via Algorithm 5 (Line 19), adds a to the anchor set A, and decreases the remaining budget (Line 20). After completing all b iterations, the final anchor set $A$is returned (Line 21).

Complexity. The runtime of Algorithm 6 is $O(b|cA||E|\log|V|)$. In each iteration, ShellDec and candidate computation take $O(|V|+|E|\alpha(|V|))$ time, the refined upper-bound computation costs $O(|E|+|V|\log|V|)$ time, and the selection loop requires $O(|cA|\cdot|E|\log|V|)$ in the worst case, as Follower runs in $O(|E|\log|V|)$ per candidate.

Algorithm 6: FAD: Followers-Aware Directed Anchoring

```
Input: G(V, E): directed graph; (k, l): core constraints; b: budget
Output: A: selected anchors
1  A ← ∅, R ← ∅;
2  while b > 0 do
3      C, L, DSU ← OptShellDec(G, k, l) ;          // Function 3
4      cA ← CandidateAnchor(G, k, l, A) ;          // Theorem 5
5      ComputeUB(cA, R, L) ; // Reuse UB or recompute Eq. 3
6      foreach v ∈ cA do
7          UB_sco(v) ← Eq. (4) ;                   // Score upper bound
8      a ← ∅, maxS ← −∞;
9      foreach v ∈ cA in decreasing UB_sco(v) do
10         if UB_sco(v) < maxS then
11             break;                              // Pruning
12         if v ∈ R and R[v] stores F(v) then
13             F(v) ← R[v].F ;       // Reuse exact followers
14         else
15             F(v) ← Follower(v, A, G, k, l, L) ; // Algorithm 4
16         Score(v) ← Score(F(v), L) ;       // Definition (7)
17         if Score(v) > maxS then
18             maxS ← Score(v), a ← v
19     R ← Reuse(a, cA, DSU, R) ;                  // Algorithm 5
20     A ← A ∪ {a}; b ← b − 1;
21 return A
```

4 EXPERIMENTS

4.1 Experimental Setting

Datasets and Parameters. We evaluate all algorithms on eight real-world directed networks collected from SNAP [4], including social networks soc-LiveJournal1 (SL), soc-pokec (SP), and twitter-combined (TC), communication networks Wiki-Vote (WV), email-EuAll (EE), and WikiTalk (WT), as well as interaction networks p2p-Gnutella06 (PG) and Amazon0302 (AM). These datasets vary in scale and density and have been widely used in prior studies on directed (k, l)-core models [23, 28]. Table 3 summarizes the basic statistics of each dataset together with the parameter settings used in our experiments. Following prior work such as OLAK and CLOCK [27, 39], we set the default anchor budget $b = 20$ and typically use $k = l = 20$. When a $(20, 20)$-core does not exist or is excessively large, we adapt the degree thresholds based on $k_{\max}$ and $l_{\max}$. Here, $k_{\max}$ and $l_{\max}$ denote the maximum values such that $C_{k,0}(G) \neq \emptyset$ and $C_{0,l}(G) \neq \emptyset$, respectively.

Algorithm. To our knowledge, prior work has not studied ADCM on directed graphs. Besides our algorithm, six other algorithms are implemented for comparison to validate result effectiveness, as shown in Tab. 4: Stoch, Deg, Deg-C, Smn, Loc-Opt and Exact.

[4]https://snap.stanford.edu/

Table 3: Dataset statistics and default (k, l) parameters

Dataset	Vertices	Edges	Avg. Degree	$k_{\max}$	$l_{\max}$	Default (k, l)
WV	7,115	103,689	14.58	15	19	(10,10)
PG	8,717	31,525	3.61	2	2	(1,1)
TC	81,306	1,768,149	21.74	53	55	(20,20)
AM	262,111	1,234,877	4.7	5	5	(4,4)
EE	265,214	420,045	1.59	28	28	(20,20)
SP	1,632,803	30,622,564	18.75	31	32	(27,27)
WT	2,394,385	5,021,410	2.1	88	87	(20,20)
SL	4,847,571	68,993,773	14.24	254	253	(100,100)

Table 4: Algorithm Descriptions

Algorithm	Description
Stoch	Stochastically selects b vertices from $V \setminus C_{k,l}(G)$.
Deg	Selects the b highest degrees vertices in $V \setminus C_{k,l}(G)$.
Deg-C	Selects the b vertices maximizing $\lvert N_{in}(v) \cup N_{out}(v) \rvert - \lvert C_{k,l}(G) \cap N(v) \rvert$.
Smn	Selects b vertices with the most neighbors in $C_{k,l}(G) \setminus C_{k-1,l-1}(G)$.
Exact	Exhaustively obtains optimal solution.
Loc-Opt	Roundwise top-follower anchor (OLAK [39])
Greedy-based	Iteratively calculate the followers of each $v \in G \setminus C_{k,l}(G_A)$ and select the best anchor using Alg. 1.
OPT-RCA	Incorporate the candidate anchor-follower reduction strategy (Section 3.1) into the Greedy-based algorithm.
OPT-HTC	Apply the hierarchical traceback cascade strategy (Sections 3.2-3.3) to OPT-RCA.
FAD	Apply the upper-bound-based termination strategy and the reuse strategy (Sections 3.4–3.5) to OPT-HTC.
NoR	FAD without the reuse strategy
NoU	FAD without the upper-bound-based termination strategy.
NoC	FAD without the candidate-anchor reduction strategy.

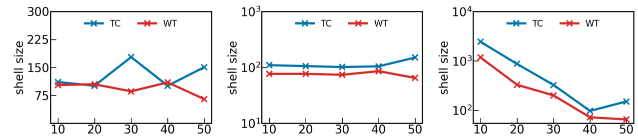
Table 5: Number of followers under b=100

Alg.	Dataset							
	AM	EE	PG	SL	SP	TC	WV	WT
Stoch	0	3	16	0	0	10	5	1
Deg	14	25	86	3	230	312	127	44
Deg-C	14	31	45	3	229	385	146	43
Smn	16	46	39	168	280	381	133	119
Loc-Opt	520	50	178	183	461	711	149	182
FAD	865	60	220	223	520	885	189	229

These algorithms employ different anchor selection strategies to obtain the number of followers. An ablation study is conducted to progressively evaluate the efficiency of our techniques, comparing Greedy-based, OPT-RCA, OPT-HTC, and the final advanced FAD algorithm. When multiple candidates have the same maximum follower count or score, one is randomly selected for anchoring. In addition to reading and saving the data, all other operations are performed online.

Setup. All experiments are run in GNU C++ on a server with an Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz and 256GB RAM, compiled with GCC (-O3). The runtime and follower count are averaged over 20 independent runs.

We observe consistent trends across all eight datasets. Due to space limitations, we present the main results on two representative datasets, WT (large-scale) and TC (small-scale), and the complete results are provided in the technical report [19]. Due to the high time complexity of the exact algorithm, two smaller datasets, WV and PG, are used for comparison.



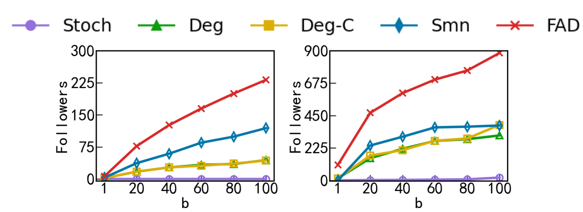
(a) Varying k, l=50 (b) Varying l, k=50 (c) Varying (k, l), k=l

Figure 5: Relative size of $C_{k,l}$ to $C_{k-1,l-1}$ under varying k or l.

Candidate-Space Analysis. Fig. 5 shows that the size of $C_{k,l} \setminus C_{k-1,l-1}$ does not monotonically decrease as k or l increases. When varying only one parameter (Figs. 5a,b), WT and TC exhibit different and non-monotonic trends. A clear reduction appears only when increasing both k and l simultaneously (Fig. 5c). This indicates that the search space is governed by structural differences between consecutive directed cores rather than by the magnitude of k and l alone, as D-cores often evolve in highly non-monotonic ways.

4.2 Effectiveness

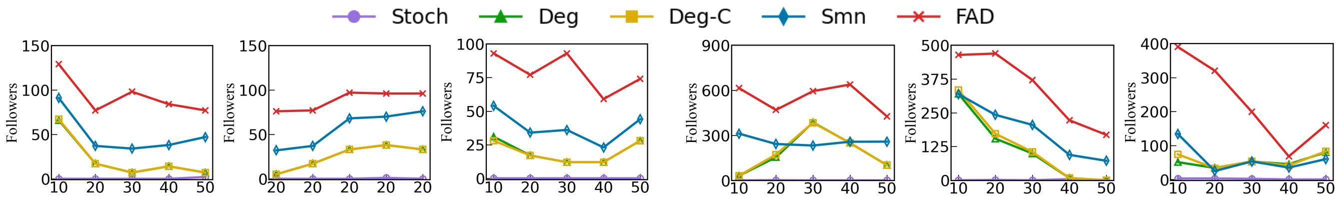
Different Datasets. Table 5 lists the number of followers attracted by anchors identified using the FAD, Stoch, Deg, Deg-c, Smn and Loc-Opt strategies across different datasets under a budget of b = 100. Among them, FAD consistently yields the largest follower sets, as it not only considers the current followers but also anticipates the impact on subsequent vertices. In contrast, the Loc-Opt algorithm focuses solely on local optimization, leading to suboptimal anchor selection. For Stoch, 100 independent runs confirm its poor effectiveness, with the number of followers often being 0 or very low. This is because b is much smaller than $|V \setminus C_{k,l}(G)|$, reducing the probability of selecting effective anchors. The Deg, Deg-c, and Smn strategies primarily select high-degree vertices as anchors. However, the high overlap of neighbors between these high-degree nodes leads to many shared followers across anchors, limiting their effectiveness and thus weakening their overall performance.



(a) WT (b) TC

Figure 6: Number of followers with varying b

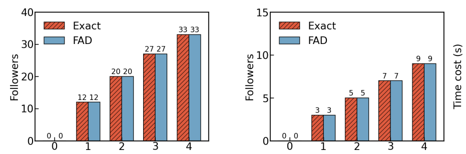
Impact of Parameters b on Effectiveness. Fig. 6 shows that the follower count increases as the budget b grows on the WT and TC



(1) Varying k, l=20 (2) Varying l, k=20 (3) Varying (k, l), k=l
(a) WT

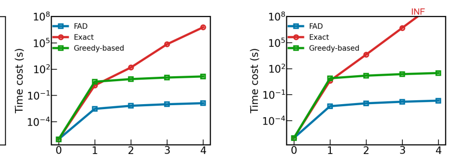
(1) Varying k, l=20 (2) Varying l, k=20 (3) Varying (k, l), k=l
(b) TC

Figure 7: Number of followers as k or l varies on different datasets



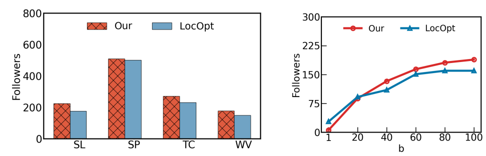
(a) PG, (1,1)-core (b) WV, (1,1)-core

Figure 8: Follower counts vs. budget b



(a) PG, (1,1)-core (b) WV, (1,1)-core

Figure 9: Runtime vs. budget b



(a) Varying datasets (b=100) (b) Varying b in WV

Figure 10: Followers vs. scoring methods

datasets. A larger budget relaxes core constraints, enhances connectivity, and induces cascade effects, thereby allowing more vertices to satisfy the k/l-degree conditions. FAD consistently achieves the highest number of followers across all budget settings. The follower count increases monotonically with b, but the marginal gain of each additional anchor does not necessarily increase, reflecting the non-submodular nature of ADCM. When b becomes sufficiently large, the marginal gain naturally diminishes, as most reachable vertices have already been stabilized by previously selected anchors. Moreover, as illustrated in Fig. 1, even selecting all vertices as anchors may fail to form a $(4, 4)$-core due to structural limitations.

Impact of Parameters k and l on Effectiveness. Fig. 7 reports the number of followers obtained by different anchor-selection strategies on the WT and TC datasets under varying out-degree and in-degree constraints. Compared with the other strategies, FAD exhibits distinct and more pronounced fluctuations when varying k and l. This difference arises from the anchor-selection mechanisms of these methods. The degree-based and random strategies (Stoch, Deg, Deg-c, Smn) rely solely on static local features, such as degree or immediate (k, l)-core neighborhoods,which remain structurally stable as k and l change. Consequently, their results exhibit similar overall patterns across different parameter settings. In contrast, FAD integrates both local (k,l)-core information and global propagation-path signals during anchor scoring. This joint consideration makes FAD highly sensitive to variations in the candidate layer and to dataset-specific core structures, leading to more diverse curve shapes and stronger fluctuations under different (k,l values. In addition, consistent with Fig. 5, the follower count broadly follows the size of the candidate layer.

Follower and Runtime Analysis: FAD vs. Exact. Figs. 8 and 9 compare follower counts and runtime of FAD and Exact on small datasets PG and WV, with degree $k = l = 1$ and budget b from 0 to 4. Larger k and l greatly increase candidate anchors and followers, making Exact's computation costly, so evaluation is limited to this range. Fig. 8 shows FAD's follower counts closely match Exact's, validating result correctness. Fig. 9 adds runtime comparison with the greedy algorithm. Exact's runtime grows exponentially with b; at $b = 4$, it takes at least 18 hours on PG and 56 hours on WV. Cases with $b > 4$ are omitted due to expected runtimes over two days. The greedy algorithm is over 1,000× slower than FAD. These results confirm FAD's effectiveness and efficiency.

Impact of Scoring Function on Effectiveness. As shown in Fig. 10a, when $b = 100$, our method achieves higher follower counts than LocOpt on the SL, SP, TC and WV datasets. Taking WV as an example, Fig. 10b shows that when b is greater than 20, as the budget b increases, our score-based selection consistently outperforms LocOpt, proving the effectiveness of our method under larger b. Our method alternates between selecting the highest-scoring anchor for odd b and the vertex with the most followers for even b, balancing immediate and future gains. In contrast, LocOpt focuses on short-term benefits. Although LocOpt may excel at very small budgets, our approach generally achieves better overall performance by considering both immediate and downstream influence.

4.3 Efficiency

In this subsection, we evaluate the efficiency of the FAD algorithm mainly through ablation studies.

Runtime Summary over All Datasets. Fig. 11 summarizes the runtime of all seven algorithms, including the ablation variants, on eight datasets with a budget of 100 under their default parameter settings. Across all datasets, each optimization step in FAD contributes substantially to reducing runtime. The largest gains come from candidate-anchor and follower pruning, which dramatically shrink the search space. The reuse strategy and the upper-bound early termination further accelerate computation by eliminating redundant work and cutting off unproductive exploration. These optimizations collectively contribute to the steady decrease in FAD's overall runtime. These optimizations enable FAD to be more than 1000× faster than Greedy-based on small datasets such as WV and PG, while Greedy-based often fails to finish within two days on

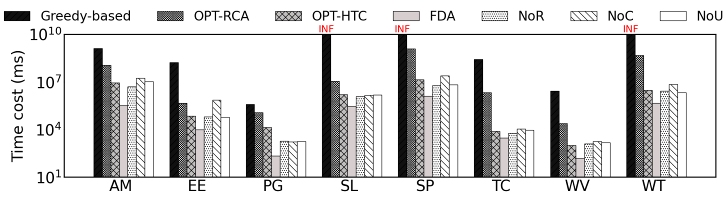


Figure 11: Execution time under different datasets (b = 100)

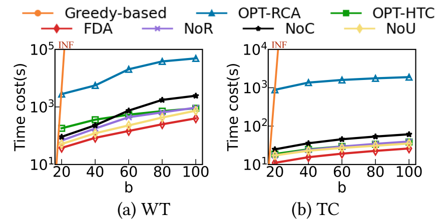


Figure 12: Time cost under varying b

large datasets (marked as INF). In contrast, FAD consistently completes within 35 minutes. Although FAD performs slightly worse than OPT-HTC on the TC and WV datasets, limited opportunities for anchor set and follower reuse cause extra overhead during reuse. Importantly, as evidenced by the ablation study, these optimizations exhibit a cumulative effect: each component yields a large speedup when added during algorithm development, yet removing any single component from the fully optimized pipeline causes only modest degradation because the remaining optimizations still eliminate most redundant work.

Impact of Parameter b on Efficiency. In Fig. 12, the running time of all algorithms on the WT and TC datasets increases with the anchor budget b, but the growth rates differ substantially across methods. The Greedy-based algorithm suffers from high computational overhead since it recomputes the entire follower set all nodes in the graph from scratch at each iteration. In particular, FAD exhibits the most stable growth trend as b increases. These observations are fully consistent with the overall runtime analysis in Fig. 11, where the cumulative effect of pruning, reuse, and upper-bound termination enables FAD to maintain strong efficiency across all datasets.

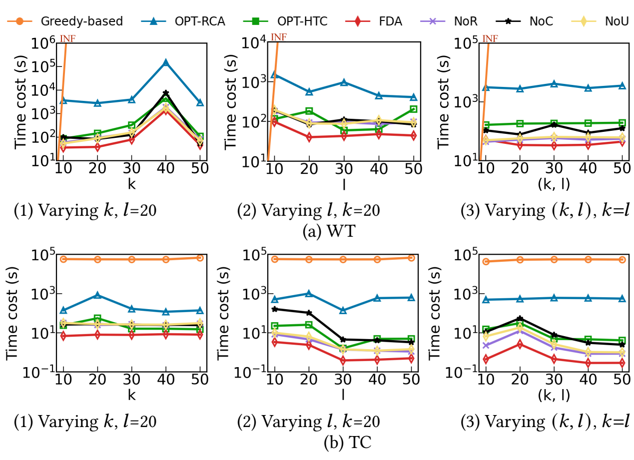


Figure 13: Time cost under varying k or l

Impact of Parameters k or l on Efficiency. As shown in Fig. 13, we evaluate the runtime of all algorithms, including the optimized variants, on the WT and TC datasets with b = 20 while varying the values of k and l. All algorithms exhibit only minor runtime changes when varying k or l individually, but show more noticeable variations when both parameters change jointly. This behavior follows the structure of the candidate-follower layer (Fig. 5): larger candidate layers lead to more candidate anchors and deeper propagation chains, increasing computational overhead, whereas jointly increasing k and l reduces the search space due to the power-law nature of social networks. These results are consistent with the overall runtime summary in Fig. 11, further confirming that underlying graph structure dominates the runtime behavior of all methods.

Memory Usage. Table 6 shows the maximum memory usage for each function during the FAD experiment on the TC and WT datasets, related to computing and maintaining information. "Reuse" refers to the memory occupied by the data cache from the previous round. "DSU" indicates the memory used for storing connectivity component information, which helps determine connected components in a graph or network. "Shell" encompasses the memory for hierarchical structure data. "OR" represents the memory usage of the Order-reachable structure. "Other" includes memory used by temporary variables and for reading the entire graph. The results show that as the graph size increases, memory consumption grows linearly. In particular, Reuse helps avoid recomputing the same results, saving both memory and computation time.

Table 6: Memory Footprint Breakdown of TC and WT

Component	TC (MB, %)	WT (MB, %)
Reuse	5.5 (12.5%)	87.1 (13.7%)
DSU	2.8 (6.3%)	82.2 (12.9%)
Shell	0.9 (2.0%)	25.1 (3.9%)
OR	2.2 (5.0%)	1.8 (0.3%)
Other	32.7 (74.1%)	441.3 (69.2%)
Total	44.1 (100%)	637.6 (100%)

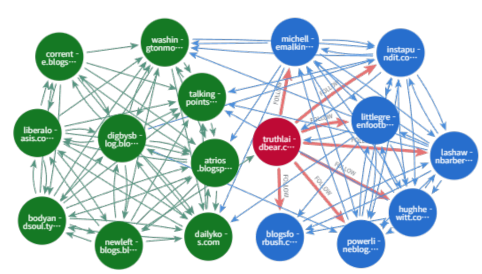


Figure 14: Case study

4.4 Case Study

The case study is implemented using FAD on the political blog network dataset [1], with visualization and querying supported by the Neo4j system without modifying the underlying data structure (see GitHub[5]). Nodes represent political blogs and directed edges denote hyperlinks or citations. The network is highly polarized, with approximately 91% of links occurring within ideologically similar communities, while cross-community connections are sparse. Information flow is largely confined within communities, and the D-core structure exhibits weak connectivity and is prone to fragmentation.

Fig. 14 shows ADCM on the $(1, 4)$-core with budget $b = 1$. FAD selects the conservative blog *truthlaidbear.com* as the anchor, which acts as a structurally critical bridging vertex between ideological communities. Although this vertex does not have the largest local degree, it lies close to the $(1, 4)$-core boundary and connects multiple order-reachable regions. Anchoring this vertex enables its conservative neighbors to become followers, providing sufficient in-degree support to the liberal community while aggregating out-degree information from the conservative side.

4.5 Extension

Extension to Richer Graph Models. FAD can be extended to attributed and labeled property graphs by incorporating node and edge attributes as weighting or filtering factors in the scoring and pruning steps. This adaptation enhances the selection of anchored cores by integrating both structural and semantic relevance, aligning with the *attribute-core model* proposed by Cai et al. [7].

FAD for Dynamic Graphs: Potential Extensions. The anchoring operation in FAD can be viewed as a localized structural update, suggesting potential for incremental maintenance in dynamic graphs. This allows FAD to efficiently reuse computed results when handling dynamic changes, such as vertex and edge insertions or deletions. By integrating snapshot-based incremental graph updates with FAD's reuse mechanism, we can handle dynamic changes locally (e.g., adjusting cores from $C_{k-1,l-1}$ to $C_{k,l}$), inspired by Cai et al.'s work on undirected dynamic graphs [8]. This makes FAD scalable and suitable for real-time dynamic network analysis.

5 RELATED WORK

Core Decomposition. Core decomposition is a fundamental graph computation problem with applications in diverse domains, such as social networks, information propagation, recommendation systems, and cybersecurity[10, 11]. It was initially proposed by Seidman[33] for the k-core problem in undirected graphs and later extended by Batagelj[3] to the D-core in directed graphs.

The peeling operation is the most commonly used technique for core decomposition, particularly in undirected graphs. Guo *et al.* [16] improved its efficiency on large heterogeneous networks via sparse matrix multiplication. Chu *et al.* [12] proposed a parallel hierarchical core decomposition method with $\mathcal{P}$-complexity for high-quality subgraph search. Liu *et al.* [28] designed a local algorithm for distance-generalized core decomposition on dynamic graphs, avoiding full-graph traversal. Lin *et al.* [24] addressed cascading effects and reduced iterative updates in evolving graphs. For directed graphs, the asymmetric nature of edges poses additional challenges. Luo *et al.* [30] combined D-shells with implicit vertex removal to support parallel D-core computation. Liao *et al.* designed distributed H-index and D-index frameworks to improve runtime and communication overhead. Beyond peeling, other techniques and variants have been explored. Victor *et al.* [37] proposed α-core for weighted directed graphs, integrating vertex attributes without needing multiple thresholds. Kim *et al.* [21] extended the (p, n)-core model to signed networks, enabling targeted community queries.

User Engagement. Ensuring long-term stability and scalability in social networks is a key goal, where core decomposition plays a vital role in evaluating structural cohesion of networks [42].

A prominent line of work focuses on the anchoring core problem, which seeks to incentivize certain users to stay permanently, triggering beneficial cascading effects. Zhang *et al.* [39] proposed OLAK using greedy selection and onion-layer decomposition for improved efficiency. Teng *et al.* [36] introduced a time-dependent anchoring method, while He *et al.* [18] developed a filter-verification strategy on bipartite graphs[32] to avoid redundant computation. Linghu *et al.* [25, 26] addressed anchored coreness, designing a tree-based structure and applying work-stealing in distributed environments to enhance performance. Cai *et al.* [7] studied keyword-based anchoring in attributed graphs, proposing a multiway tree with keyword-aware indexing. They also tackled dynamic anchoring via incremental tracking [8]. Beyond vertex anchoring, edge-based strategies have been explored. Zhou *et al.* [42] proposed a sequential structure for the edge k-core problem. Do and Shin [13] explored enhancing network resilience by adding hyperedges after vertex or hyperedge deletions in hypergraphs. Sun *et al.* [34] introduced FastCM+ to partition graphs and enable parallel edge addition strategies.

Despite extensive studies on core anchoring in undirected graphs, these methods do not suit directed networks, where edge asymmetry poses new structural challenges. To maintain stability and scalability in directional communities, exploring anchoring strategies tailored to directed graphs is crucial.

6 CONCLUSION AND FUTURE WORK

In this paper, we study the Anchored (k, l)-Core Maximization (ADCM) problem to enhance engagement in directed social networks. Due to the NP-hardness of the problem, we employ a greedy heuristic with a vertex scoring function to facilitate core expansion. To improve efficiency, we introduce candidate reduction, pruning techniques, and an upper-bound-based termination criterion. Further optimizations, including reuse techniques, enhance computational performance. Extensive experiments on eight real-world datasets demonstrate the effectiveness and efficiency of our proposed methods. In the future, large language models (LLMs) can be combined with graph structures to capture complex dependencies in directed or dynamic graphs and efficiently solve the ADCM problem under network changes.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Grant Nos. U23A20317, 62576051 and 62572182), the Creative Research Groups Program of the National Natural Science Foundation of China (Grant No. 62321003), and the Natural Science Foundation of Hunan Province (Grant No. 2023JJ10016).

[5]https://github.com/sanxian13/FAD

REFERENCES

[1] Lada A. Adamic and Natalie Glance. 2005. The political blogosphere and the 2004 U.S. election: divided they blog. In *Proceedings of the 3rd International Workshop on Link Discovery* (Chicago, Illinois) *(LinkKDD '05)*. Association for Computing Machinery, New York, NY, USA, 36–43.

[2] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. 2006. Group formation in large social networks: membership, growth, and evolution *(KDD '06)*. Association for Computing Machinery, New York, NY, USA, 44–54.

[3] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) Algorithm for Cores Decomposition of Networks. *ArXiv* cs.DS/0310049 (2003).

[4] Kshipra Bhawalkar, Jon Kleinberg, Kevin Lewi, Tim Roughgarden, and Aneesh Sharma. 2015. Preventing Unraveling in Social Networks: The Anchored k-Core Problem. *SIAM J. Discret. Math.* 29, 3 (Jan. 2015), 1452–1475.

[5] Charles D Brummitt, Raissa M D'Souza, and E A Leicht. 2012. Suppressing cascades of load in interdependent networks. *Proceedings of the National Academy of Sciences of the United States of America* 109, 12 (2012), 4345–4346.

[6] Sergey V. Buldyrev, Roni Parshani, Gerald Paul, H. Eugene Stanley, and Shlomo Havlin. 2010. Catastrophic cascade of failures in interdependent networks. *Nature* 464, 7291 (April 2010), 1025–1028.

[7] Taotao Cai, Jianxin Li, Nur Al Hasan Haldar, Ajmal Mian, John Yearwood, and Timos Sellis. 2020. Anchored Vertex Exploration for Community Engagement in Social Networks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 409–420.

[8] Taotao Cai, Shuiqiao Yang, Jianxin Li, Quan Z. Sheng, Jian Yang, Xin Wang, Wei Emma Zhang, and Longxiang Gao. 2023. Incremental Graph Computation: Anchored Vertex Tracking in Dynamic Social Networks. *IEEE Trans. Knowl. Data Eng.* 35, 7 (2023), 7030–7044.

[9] Abhijit Chakraborty, Tobias Reisch, Christian Diem, Pablo Astudillo-Estévez, and Stefan Thurner. 2024. Inequality in economic shock exposures across the global firm-level supply network. *Nature Communications* 15, 1 (2024), 3348.

[10] Xueqin Chang, Jiajie Fu, Qing Liu, Yunjun Gao, and Baihua Zheng. 2025. Time-Aware Influence Minimization via Blocking Social Networks . In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 557–570.

[11] Xueqin Chang, Qing Liu, Yunjun Gao, Baihua Zheng, Yi Cai, and Qing Li. 2025. The Most Influenced Community Search on Social Networks . In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 2601–2614.

[12] Deming Chu, Fan Zhang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2022. Hierarchical Core Decomposition in Parallel: From Construction to Subgraph Search. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 1138–1151.

[13] Manh Tuan Do and Kijung Shin. 2023. Improving the core resilience of real-world hypergraphs. *Data Min. Knowl. Discov.* 37, 6 (2023), 2438–2493.

[14] Yixiang Fang, Zhongran Wang, Reynold Cheng, Hongzhi Wang, and Jiafeng Hu. 2019. Effective and Efficient Community Search Over Large Directed Graphs (Extended Abstract). In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 2157–2158.

[15] Christos Giatsidis, Dimitrios M. Thilikos, and Michalis Vazirgiannis. 2011. D-cores: Measuring Collaboration of Directed Graphs Based on Degeneracy. In *2011 IEEE 11th International Conference on Data Mining*. 201–210.

[16] Yucan Guo, Chenhao Ma, and Yixiang Fang. 2024. Efficient Core Decomposition Over Large Heterogeneous Information Networks. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 2393–2406.

[17] Johan Håstad. 1997. Some Optimal Inapproximability Results. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, Frank Thomson Leighton and Peter W. Shor (Eds.). ACM, 1–10.

[18] Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2022. Efficient Reinforcement of Bipartite Networks at Billion Scale. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 446–458.

[19] Yang Huang. [n.d.]. *Technical Report for "Anchored Maximum Communities over Large Directed Graphs"*. Technical Report. https://github.com/good-hy/FAD/blob/main/technicalReport.pdf

[20] H. Karloff and U. Zwick. 1997. A 7/8-approximation algorithm for MAX 3SAT?. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*. 406–415. https://doi.org/10.1109/SFCS.1997.646129

[21] Junghoon Kim, Hyun Ji Jeong, Sungsu Lim, and Jungeun Kim. 2023. Effective and efficient core computation in signed networks. *Inf. Sci.* 634, C (July 2023), 290–307.

[22] Ricky Laishram, Ahmet Erdem Sar, Tina Eliassi-Rad, Ali Pinar, and Sucheta Soundarajan. 2020. Residual core maximization: An efficient algorithm for maximizing the size of the k-core. In *Proceedings of the 2020 SIAM International Conference on Data Mining*. SIAM, 325–333.

[23] Xuankun Liao, Qing Liu, Jiaxin Jiang, Xin Huang, Jianliang Xu, and Byron Choi. 2022. Distributed D-core Decomposition over Large Directed Graphs. *Proc. VLDB Endow.* 15, 8 (2022), 1546–1558.

[24] Zhe Lin, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Zhihong Tian. 2021. Hierarchical core maintenance on large dynamic graphs. *Proc. VLDB Endow.* 14, 5 (Jan. 2021), 757–770.

[25] Qingyuan Linghu, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2020. Global Reinforcement of Social Networks: The Anchored Coreness Problem. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 2211–2226.

[26] Qingyuan Linghu, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2022. Anchored coreness: efficient reinforcement of social networks. *VLDB J.* 31, 2 (2022), 227–252.

[27] Kaixin Liu, Sibo Wang, Yong Zhang, and Chunxiao Xing. 2021. An Efficient Algorithm for the Anchored k-Core Budget Minimization Problem. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 1356–1367.

[28] Qing Liu, Xuliang Zhu, Xin Huang, and Jianliang Xu. 2021. Local algorithms for distance-generalized core decomposition over large dynamic graphs. *Proc. VLDB Endow.* 14, 9 (May 2021), 1531–1543.

[29] James O. Lloyd-Smith, Sebastian J. Schreiber, Peter E. Kopp, and Wayne M. Getz. 2005. Superspreading and the effect of individual variation on disease emergence. *Nature* 438, 7066 (November 2005), 355–359. https://doi.org/10.1038/nature04153

[30] Wensheng Luo, Yixiang Fang, Chunxu Lin, and Yingli Zhou. 2024. Efficient Parallel D-Core Decomposition at Scale. *Proc. VLDB Endow.* 17, 10 (June 2024), 2654–2667.

[31] Xin Ouyang, Litao Liu, Wu Chen, Chao Wang, Xin Sun, Canfei He, and Gang Liu. 2024. Systematic Risks of the Global Lithium Supply Chain Network: From Static Topological Structures to Cascading Failure Dynamics. *Environmental Science & Technology* 58, 50 (2024), 22135–22147.

[32] Dong Pan, Xu Zhou, Wensheng Luo, Zhibang Yang, Qing Li, Yunjun Gao, and Kenli Li. 2025. Accelerating maximum biplex search over large bipartite graphs. *VLDB J.* 34, 1 (2025), 1.

[33] Stephen B. Seidman. 1983. Network structure and minimum degree. *Social Networks* 5 (1983), 269–287.

[34] Xin Sun, Xin Huang, and Di Jin. 2022. Fast algorithms for core maximization on large graphs. 15, 7 (March 2022), 1350–1362.

[35] Xin Sun, Xin Huang, Zitan Sun, and Di Jin. 2021. Budget-constrained Truss Maximization over Large Graphs: A Component-based Approach. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong (Eds.). ACM, 1754–1763.

[36] Siyi Teng, Jiadong Xie, Fan Zhang, Can Lu, Juntao Fang, and Kai Wang. 2024. Optimizing Network Resilience via Vertex Anchoring. In *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, Tat-Seng Chua, Chong-Wah Ngo, Ravi Kumar, Hady W. Lauw, and Roy Ka-Wei Lee (Eds.). ACM, 606–617.

[37] Friedhelm Victor, Cuneyt G. Akcora, Yulia R. Gel, and Murat Kantarcioglu. 2021. Alphacore: Data Depth based Core Decomposition. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 1625–1633. https://doi.org/10.1145/3447548.3467322

[38] Fan Zhang, Conggai Li, Ying Zhang, Lu Qin, and Wenjie Zhang. 2020. Finding Critical Users in Social Communities: The Collapsed Core and Truss Problems. *IEEE Trans. Knowl. Data Eng.* 32, 1 (2020), 78–91.

[39] Fan Zhang, Wenjie Zhang, Ying Zhang, Lu Qin, and Xuemin Lin. 2017. OLAK: an efficient algorithm to prevent unraveling in social networks. *Proc. VLDB Endow.* 10, 6 (feb 2017), 649–660. https://doi.org/10.14778/3055330.3055332

[40] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2018. Efficiently Reinforcing Social Networks over User Engagement and Tie Strength. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 557–568.

[41] Zhongxin Zhou, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Chen Chen. 2019. K-core maximization: an edge addition approach. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (Macao, China) *(IJCAI'19)*. AAAI Press, 4867–4873.

[42] Zhongxin Zhou, Wenchao Zhang, Fan Zhang, Deming Chu, and Binghao Li. 2022. VEK: a vertex-oriented approach for edge k-core problem. 25, 2 (March 2022), 723–740.