# Efficient Reinforcement of Bipartite Networks at Billion Scale

Yizhang He[†], Kai Wang[†], Wenjie Zhang[†], Xuemin Lin[†], Ying Zhang[⋆]
[†]University of New South Wales, [⋆]University of Technology Sydney
{yizhang.he, kai.wang}@unsw.edu.au, {zhangw, lxue}@cse.unsw.edu.au, ying.zhang@uts.edu.au

***Abstract*—Bipartite networks, which model relationships between two different types of entities, are prevalent in many real-world applications. On bipartite networks, the cascading node departure undermines the networks' ability to provide sustainable services, which makes reinforcing bipartite networks a vital problem. Although network reinforcement is extensively studied on unipartite networks, it remains largely unexplored on bipartite graphs. On bipartite networks, $(\alpha, \beta)$-core is a stable structure that ensures different minimum engagement levels of the vertices from different layers, and we aim to reinforce bipartite networks by maximizing the $(\alpha, \beta)$-core. Specifically, given a bipartite network $G$, degree constraints $\alpha$ and $\beta$, budgets $b_1$ and $b_2$, we aim to find $b_1$ upper layer vertices and $b_2$ lower layer vertices as anchors and bring them into the $(\alpha, \beta)$-core s.t. the number of non-anchor vertices entering in the $(\alpha, \beta)$-core is maximized. We prove the problem is NP-hard and propose a heuristic algorithm FILVER to solve the problem. FILVER runs $b_1 + b_2$ iterations and choose the best anchor in each iteration. Under a filter-verification framework, it reduces the pool of candidate anchors (in the filter stage) and computes the resulting $(\alpha, \beta)$-core for each anchor vertex more efficiently (in the verification stage). In addition, filter-stage optimizations are proposed to further reduce "dominated" anchors and allow computation-sharing across iterations. To optimize the verification stage, we explore the cumulative effect of placing multiple anchors, which effectively reduces the number of running iterations. Extensive experiments on 16 real-world datasets and a billion-scale synthetic dataset validate the effectiveness and efficiency of our proposed techniques.**

## I. Introduction

Bipartite networks are naturally used to model the relationships between two distinct types of entities, which are widely prevalent in many real-world scenarios such as customer-product networks in E-commerce [1], user-page networks in social analysis [2], and people-activity networks in event recommendation [3]. Notably, the structural stability of these networks is crucial since it reflects the ability of such networks to provide sustainable services [4], [5]. For example, on a customer-product network of an online shopping platform (e.g., *Amazon*, *Alibaba*, and *eBay*), the continuing and frequent connections between customers and products usually lead to not only more revenue from sales but also more web traffic to attract advertisements [6], [7], [8]. As node departures have the snowball effect of causing more nodes to drop out [9] and even cause a total network breakdown [5], it is vital to reinforce a bipartite network and maintain its stability.

In the literature, a common and practical approach for network reinforcement is *core maximization*, which aims to enlarge the stable structure captured by the $k$-core model [10], [11], [12], [13], [14], [15]. Here $k$-core is the maximal subgraph where each vertex has at least $k$ neighbors, which is considered as a feasible indicator of network stability [5], [16]. However, the approaches proposed in these existing works focus on general (unipartite) networks and are not suitable for reinforcing bipartite graphs, where the vertices of two different layers represent different types of entities, and their degrees can be of different scales.

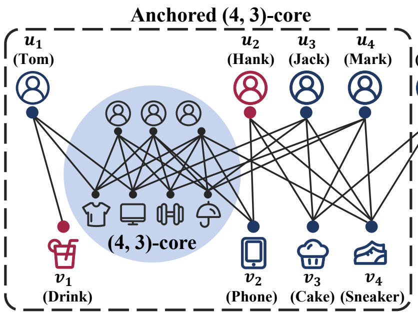


Fig. 1: A user-item network

On bipartite networks, the $(\alpha, \beta)$-core model [17], [18], [19] naturally extends the $k$-core model by having different degree constraints ($\alpha$ and $\beta$) on two different vertex layers (i.e., the upper/lower layers). In an $(\alpha, \beta)$-core, the vertices on upper and lower layers are guaranteed to have minimum engagement levels of $\alpha$ and $\beta$ respectively [18], which are unlikely to leave the bipartite network and cause it to break down. Motivated by the above reasons, we aim to reinforce bipartite networks by maximizing the $(\alpha, \beta)$-core. Specifically, if we select some vertices as *anchors* and let them enter the $(\alpha, \beta)$-core (e.g., setting their degrees to $+\infty$ or add more connections to them), then more vertices (i.e., *followers*) that were not in the $(\alpha, \beta)$-core will be also included by it. The expanded $(\alpha, \beta)$-core is also called the *anchored* $(\alpha, \beta)$*-core*. In this paper, given a bipartite network $G$, and two integers $b_1$ and $b_2$, we study the *anchored* $(\alpha, \beta)$*-core problem* that aims to find $b_1$ upper layer vertices and $b_2$ lower layer vertices as anchors to maximize the number of vertices in the anchored $(\alpha, \beta)$-core (i.e., maximize the number of followers). Consider the example in Figure 1. The closely connected community with 3 people and 4 items is the $(4, 3)$-core. Given $b_1 = b_2 = 1$, to expand this community as much as possible, the best solution is to choose $u_2$ ("Hank") and $v_1$ ("Drink") as anchors, which allows all vertices to be included in the anchored $(4, 3)$-core except $u_5$ ("Joey").

**Applications.** The anchored $(\alpha, \beta)$-core problem on bipartite graphs has many real-world applications, and we present two representative scenarios as follows.

- *Maintaining social groups.* In user-item networks (e.g., customer-product networks in *Amazon*, user-page networks in *Facebook*, and user-movie networks in *IMDb*), social groups formed by people with common interests, habits, and beliefs are widely prevalent. As these groups are often fragile and prone to fall apart [20], [21], maintaining the variety and stability of these groups is vital to the activity and prosperity of networks. For example, in a customer-product network, the underlying platform can offer special discounts on certain items, or sponsor some users like social media influencers to achieve this goal. Therefore, studying the anchored $(\alpha, \beta)$-core problem can effectively identify these critical users and items for maintaining such social groups.

• *Reinforcing mutualistic networks.* In ecosystems, mutually beneficial interactions between two types of bio-entities are naturally modeled as bipartite networks called mutualistic networks. Many of these networks are formed by plants and animals [22]. Such reciprocal relationships are vital to the ecosystem because the extinction of some plants or animals can result in extinction cascades [23]. In the $(\alpha, \beta)$-core of a plant-animal network, each plant can depend on at least $\alpha$ animals to pollinate and disperse seeds, while each animal can live on food sources and shelters provided by at least $\beta$ plants. These plants and animals are more resilient and less vulnerable to extinction. Such stable units of the ecosystem can be expanded and strengthened by improving the habitats of certain species (i.e., "anchor" some plants or animals to expand the induced $(\alpha, \beta)$-core).

**Motivations.** Despite the applications mentioned above, the anchored $(\alpha, \beta)$-core problem is NP-hard and APX-hard. Existing studies of core maximization to reinforce unipartite networks propose greedy algorithms for NP-hard problems like the anchored *k*-core problem [11], [12], the anchored coreness problem [13], [16], and the anchored budget minimization problem [15]. However, these algorithms do not apply to our problem for two reasons: (1) these algorithms are based on *k*-core, which ignores the characteristics of bipartite networks, i.e., the vertices in different layers have different types; (2) the optimization objectives are inherently different. Besides, as nowadays bipartite networks can reach billion-scale, it is cost-prohibitive to devise exact algorithms to solve the anchored $(\alpha, \beta)$-core problem on large bipartite networks.

Motivated by the above observations, in this paper, we resort to greedy heuristics to solve the problem. Intuitively, we can run $b_1 + b_2$ iterations and for each iteration, we search for the best anchor which can produce the largest number of followers. The number of followers of each candidate anchor is computed by using the $(\alpha, \beta)$-core computation algorithm in [18]. Experimental results show that this greedy approach can produce commensurate numbers of followers as the exact algorithm. However, it suffers from long running time (e.g., it cannot finish within 24 hours on the `WC` dataset with 3.8 million edges), which cannot be used to handle large-scale bipartite networks. This compromised efficiency is due to the following reasons. Firstly, in each iteration of the greedy algorithm, it needs to process lots of candidate anchors, the number of which is linear to the number of vertices in the whole graph. Secondly, to build the anchor set, we need to compute the followers of each candidate anchor, which needs to traverse the whole graph in the worst case. In addition, it is difficult to explore computation-sharing opportunities across iterations for the greedy algorithm since the anchored $(\alpha, \beta)$-core can expand a lot in each iteration.

**Our approaches.** Firstly, we propose the *upper/lower deletion orders* based on the vertex deletion order during $(\alpha, \beta)$-core computation to capture the dependencies among anchors and followers. By utilizing these orders, the FILVER algorithm is proposed, which follows a filter-verification framework to find the best anchor in each iteration. In the filter stage, FILVER leaves out unpromising candidate anchors that do not produce any followers. In the verification stage, FILVER computes the followers for each promising anchor in a local manner, which is much faster than computing globally using the $(\alpha, \beta)$-core computation algorithm [18].

Secondly, we propose a set of optimizations to further speed up the filter stage. We define *follower signature* to filter more "dominated" anchors, whose follower set can be covered by other anchors'. The follower signature of an anchor $x$ is the set of its neighbors that are deleted after $x$ when computing the $(\alpha, \beta)$-core, which can be considered as the starting point for computing the followers. An important feature of follower signatures is that if the follower signature of an anchor $x_1$ is covered by that of another anchor $x_2$, then the follower set of $x_1$ must be covered by that of $x_2$. By leveraging the bipartite graph structure, a two-hop filtering algorithm is devised to efficiently eliminate such "dominated" anchors and drastically reduce the candidate anchor pool. In addition, since the anchored $(\alpha, \beta)$-core expands in each iteration, the upper/lower deletion orders need to be correctly updated at the beginning of the filter stage. To reuse information across iterations, we explore the nested property of $(\alpha, \beta)$-core to identify the affected scope of each placed anchor and reduce the parts of the orders that need recomputation.

Thirdly, we optimize the verification stage by exploring the cumulative effect of placing multiple anchors and propose the FILVER$^{++}$ algorithm. Specifically, while scanning through the candidate anchors (after filtering) in each iteration, FILVER$^{++}$ maintains a set of $t$ ($t > 1$) representative anchors. By devising effective anchor set maintenance techniques, we constantly update this set of anchors and ensure that the sets of followers brought by these anchors are largely non-overlapping. Since FILVER$^{++}$ finds $t$ anchors rather than one per iteration, it requires fewer iterations till termination. Experiment results validate that FILVER$^{++}$ brings in a similar number of followers as FILVER and further pushes the efficiency boundary.

**Contribution.** Here we summarize our principal contributions.

- We propose and explore the anchored $(\alpha, \beta)$-core problem to reinforce bipartite networks. We also prove the problem is NP-hard and APX-hard.
- We propose a heuristic filter-verification framework to solve the problem, which can largely reduce the computation cost.
- We propose a set of filter-stage optimizations to further filter "dominated" anchors and allow computation-sharing across iterations.
- We propose verification-stage optimizations by exploring

the cumulative effect of placing multiple anchors to reduce the number of processed iterations.

- We conduct extensive experiments on 17 real and synthetic bipartite networks to validate the effectiveness and efficiency of the proposed techniques. Experimental results show that the FILVER algorithm is efficient and produces a similar number of followers as the exact algorithm. Also, the FILVER$^{++}$ algorithm with optimizations in both stages significantly outperforms FILVER in efficiency by up to 44× and is scalable to the billion-scale dataset.

**Organization.** The rest of the paper is organized as follows. Section 2 defines the problem and introduces the straightforward solutions. Section 3 presents the filter-verification framework and the FILVER algorithm. Section 4 and 5 propose filter-stage and verification-stage optimizations, respectively. Section 6 reports the experimental results. Section 7 reviews the related work, and Section 8 concludes the paper.

## II. Preliminaries

TABLE I: Summary of Notations

| Notation | Definition |
|---|---|
| $G$ | a bipartite graph |
| $A$ | the set of anchors |
| $N(u,G)$ | the neighbors of $u$ in $G$ |
| $d(u,G)$ | the degree of $u$ in $G$ |
| $C_{\alpha,\beta}(G_A)$ | the vertex set of anchored $(\alpha,\beta)$-core |
| $\mathcal{S}_{up}(G)$, $\mathcal{S}_{low}(G)$ | the upper and lower shell of $G$ |
| $F(A)$ | the followers of anchor set $A$ |
| $\mathcal{O}_U(u)$, $\mathcal{O}_L(u)$ | the upper/lower deletion order of a vertex $u$ |
| $rf(x)$ | the order-reachable vertices from an anchor $x$ |
| $core_U(u)$, $core_L(u)$ | the upper/lower deletion order of a vertex $u$ |
| $AG_U(x)$, $AG_L(x)$ | the upper/lower affected graph an anchor $x$ |

In this section, we present important notations and introduce the anchored $(\alpha,\beta)$-core model. Then, we formally define the anchored $(\alpha,\beta)$-core problem and prove its hardness.

### A. Problem definition

In this paper, we use an unweighted bipartite graph $G(V = (U, L), E)$ to model a bipartite network. $V = U \cup L$ denotes the set of vertices where $U$ and $L$ represent the upper and lower layer, respectively. The vertices in $U$ and $L$ are called upper vertices and lower vertices. $E \subseteq U \times L$ denotes the set of edges. We use $n = |V|$ to denote the number of vertices and $m = |E|$ to denote the number of edges in $G$ ($m > n$). The set of neighbors of a vertex $u$ in $G$ is denoted by $N(u,G)$. Given a vertex set $J$, $N(J)$ denotes the union of the neighbors of vertices in $J$ (i.e., $N(J) = \bigcup_{u\in J} N(u,G)$). In addition, the degree of $u$ is denoted by $d(u,G) = |N(u,G)|$. When the context is clear, we omit the input graph $G$ in notations.

**Definition 1** ($(\alpha,\beta)$-core)**.** *Given a bipartite graph G and degree constraints α and β, a subgraph $G'$ is the $(\alpha,\beta)$-core, if (1) all vertices in $G'$ satisfy the degree constraints, i.e., $d(u,G') \geq \alpha$ for each $u \in U(G')$ and $d(v,G') \geq \beta$ for each $v \in L(G')$; and (2) $G'$ is maximal, i.e., any supergraph $G'' \supseteq G'$ is not an $(\alpha,\beta)$-core. The vertex set of the $(\alpha,\beta)$-core is denoted by $C_{\alpha,\beta}(G)$.*

Now we introduce the anchored $(\alpha,\beta)$-core model. In this paper, if a vertex $u$ is **anchored**, it always stays in the $(\alpha,\beta)$-core regardless of its degree (or equivalently, we set $d(u,G)$ to $+\infty$). Such vertices are called **anchor vertices** or **anchors**.

**Definition 2** (Anchored $(\alpha,\beta)$-core)**.** *Consider a bipartite graph G, degree constraints α, β, and a vertex set $A \subseteq V(G)$. The graph G with vertices in A anchored is denoted by $G_A$ and the anchored $(\alpha,\beta)$-core is the corresponding $(\alpha,\beta)$-core of $G_A$. The vertex set of the anchored $(\alpha,\beta)$-core is denoted by $C_{\alpha,\beta}(G_A)$.*

Apart from the vertices in $C_{\alpha,\beta}(G) \cup A$, there exist vertices able to satisfy the degree constraints and stay in the anchored $(\alpha,\beta)$-core due to the presence of the anchors, which are the **followers** of $A$. We formally define the followers as below.

**Definition 3** (Follower)**.** *Given a bipartite graph G, degree constraints α, β and a set of anchors A, the vertices in $C_{\alpha,\beta}(G_A) \setminus (C_{\alpha,\beta}(G) \cup A)$ are called **followers** of A, denoted by $F(A)$. The upper and lower vertices in $F(A)$ are called **upper followers** and **lower followers**, denoted by $F_U(A)$ and $F_L(A)$ respectively.*

**Problem Statement.** Given a bipartite graph $G$, degree constraints $\alpha$, $\beta$, and budgets $b_1$, $b_2$, the *anchored $(\alpha,\beta)$-core problem* aims to find a sets of anchors $A = A_1 \cup A_2$ ($A_1 \subseteq U(G)$ and $A_2 \subseteq L(G)$ with $|A_1| = b_1$ and $|A_2| = b_2$) such that the number of followers ($|F(A)|$) is maximized.

### B. Problem complexity

**Theorem 1.** *The anchored $(\alpha,\beta)$-core problem is NP-hard except the cases when $1 \leq \alpha = \beta \leq 2$.*

*Proof.* When $\alpha = \beta = 1$, the whole graph is in the $(\alpha,\beta)$-core and the problem is trivially polynomial-time solvable. When $\alpha = \beta = 2$, the $(2,2)$-core is the same as *k*-core when $k = 2$ which can be solved using an efficient algorithm [10] that maximizes the anchored 2-core.

When $\alpha = 1$ and $\beta > 1$, the $(\alpha,\beta)$-core includes exactly the lower vertices with degrees at least $\beta$ and their neighbors. Placing anchors on upper vertices does not incur any followers because all lower vertices satisfying the degree constraints are already in the $(\alpha,\beta)$-core. Thus, we only consider finding $b_2$ lower vertices as anchors. In such case, the anchored $(\alpha,\beta)$-core problem is equivalent to a maximum cover (hereafter called MC) problem [24], which aims to find at most $b$ sets to cover the most number of elements and is NP-hard. Likewise, the problem is also NP-hard when $\beta = 1$ and $\alpha > 1$.

We still need to prove the NP-hardness when (1) $\alpha \geq 3$ and $\beta \geq 2$ and (2) $\beta \geq 3$ and $\alpha \geq 2$. Symmetrically, we only need to handle case (1). As the anchored $(\alpha,\beta)$-core problem can be reduced from itself with a restriction that only selecting upper anchors, we only need to prove the NP-hardness of such case by reducing it from the MC problem. Consider an arbitrary instance of the MC problem with $c$ sets $\{T_i\}$ ($1 \leq i \leq c$), and $d$ elements $\{e_1, e_2, \ldots e_d\} = \cup_{1\leq i\leq c} T_i$. We construct a corresponding instance of the anchored $(\alpha,\beta)$-core problem.

We first construct a bipartite graph $B$ with $(\alpha-1)\times(\beta-1)$ upper vertices and $2\times(\alpha-1)$ lower vertices. The lower vertices of $B$ has two parts, $L^*$ and $L'$, each of size $\alpha - 1$. Each vertex in $L^*$ connects to all upper vertices in $B$. Each vertex in $L'$ connects to $\beta - 1$ upper vertices. Note that only vertices in $L'$ do not meet the degree constraints $\alpha$ and $\beta$. We also construct a bipartite graph $R$, which is a tree with an upper vertex as its root. In $R$, each lower vertex has $\alpha - 1$ upper vertices as children, and each upper vertex that is not a leaf node has $\beta - 1$ lower vertices as children. The tree is grown until it has at least $\max_{1 \leq i \leq c} T_i$ upper vertices as leaf nodes.

We make $d$ copies of $B$ denoted by $B_i$ ($1 \leq i \leq d$), each corresponding to an element $e_i$. We also make $c$ copies of $R$ denoted by $R_j$ ($1 \leq j \leq c$), each corresponding to a set $T_i$. The roots of $R_i$ are denoted by $u_j$ ($1 \leq j \leq c$). Then, we connect $B_i$ and $R_j$ in the following way: if $e_i \in T_j$, we connect the lower vertices with degree $\beta - 1$ in $B_i$ to the $i_{th}$ leaf node (which is an upper vertex) in $R_j$. If a leaf node in $R_j$ does not connect to any $B_i$, we connect it to a lower vertex in a biclique $J$ with $\beta$ upper vertices and $\alpha$ lower vertices. When searching for the optimal anchor set, we can only consider roots $u_j \in R_j$ ($1 \leq j \leq c$) as anchors intuitively. When $u_j$ is anchored, all the vertices in $B_i$ become followers if $B_i$ connects to $R_j$. In this way, the optimal selection of sets within budget $b$ in the MC problem corresponds to the optimal assignment of anchors in the anchored $(\alpha, \beta)$-core problem. Fig. 2 shows the reduction process when $\alpha = 3$ and $\beta = 2$. Since the MC problem is NP-hard, the anchored $(\alpha, \beta)$-core problem is also NP-hard when $\alpha \geq 3$ and $\beta \geq 2$. □

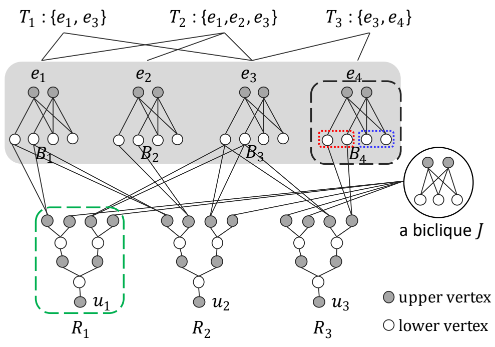


Fig. 2: Illustrating the reduction when $\alpha = 3$ and $\beta = 2$.

**Theorem 2.** *For any $\epsilon > 0$, it is NP-hard to approximate the anchored $(\alpha, \beta)$-core problem within a ratio of $(1 - 1/e + \epsilon)$ except the cases when $1 \leq \alpha = \beta \leq 2$.*

*Proof.* Theorem 1 shows that the MC problem can be reduced to the anchored $(\alpha, \beta)$-core problem for all $\alpha$ and $\beta$ except when $1 \leq \alpha = \beta \leq 2$. Thus, this theorem holds since the MC problem cannot be approximated by a polynomial-time algorithm with a ratio of $(1 - 1/e + \epsilon)$ [25]. □

### C. Warm up

**An exact solution.** To solve the anchored $(\alpha, \beta)$-core problem, a basic approach is to go through all possible combinations of anchor assignments and select the anchor set $A$ that maximizes the number of followers. The number of followers can be computed by setting the degrees of anchors to $+\infty$ and applying the $(\alpha, \beta)$-core computation algorithm [18], which iteratively removes the vertices without enough degrees in $O(m)$ time. Although this brute force method guarantees an optimal solution, the time complexity $O(\binom{n_1}{b_1}\binom{n_2}{b_2}m)$ is cost-prohibitive. Here $n_1$ and $n_2$ are the number of upper and lower vertices in $G$, respectively.

**A naive greedy approach.** Due to the above reason, we resort to greedy heuristics to solve the problem, and a naive greedy approach Naive is described as follows. Given budgets $b_1$ and $b_2$, Naive runs $b_1 + b_2$ iterations and finds one best anchor in each iteration. Firstly, the anchor set $A$ is initialized to empty. Then, in each iteration, we consider the vertices that are not in the vertex set of the anchored $(\alpha, \beta)$-core (i.e., $C_{\alpha,\beta}(G_A)$) as candidate anchors. After going through all the candidate anchors, we choose the one with the largest number of followers and add it into $A$. The time complexity of Naive is $O((b_1 + b_2)nm)$ since there are $b_1 + b_2$ iterations, and it needs $O(nm)$ time to compute the followers of $O(n)$ candidate anchors in total in each iteration.

## III. A FILTER-VERIFICATION FRAMEWORK

When handling large-scale bipartite graphs, Naive is still too time-consuming due to its large candidate anchor pool and inefficient follower computation process. Since the vertex deletion order during core decomposition [26], [27], [28] is used to accelerate the computation process when solving many $k$-core related problems on unipartite graphs [11], [29], [30], [14], we investigate if a new greedy paradigm can be designed by considering the vertex deletion order during the $(\alpha, \beta)$-core computation process.

### A. Exploring the vertex deletion order

First, we present the following definitions and observations.

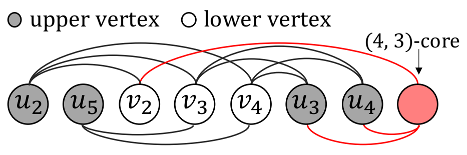
**Definition 4. Upper/lower shell.** *Given a bipartite graph $G$ and degree constraints $\alpha$ and $\beta$, the upper shell refers to the vertices in $C_{\alpha,\beta-1}(G) \setminus C_{\alpha,\beta}(G)$, denoted by $\mathcal{S}_{up}(G)$. Similarly, the lower shell refers to the vertices in $C_{\alpha-1,\beta}(G) \setminus C_{\alpha,\beta}(G)$, denoted by $\mathcal{S}_{low}(G)$.*

It follows immediately that when a upper vertex $x \in U(G)$ is anchored, its followers (i.e., $F(x)$) must come from the upper shell $\mathcal{S}_{up}(G)$. This is because anchoring one upper vertex can only cause the vertices in the $(\alpha, \beta - 1)$-core to enter the anchored $(\alpha, \beta)$-core. Likewise, if a lower vertex $x \in L(G)$ is anchored, the followers must come from the lower shell $\mathcal{S}_{low}(G)$. Thus, we define the potential followers as follows.

**Definition 5. Potential followers.** *Given a bipartite graph $G$ and the degree constraints $\alpha$ and $\beta$, we call the vertices in $\mathcal{S}_{up}(G) \cup \mathcal{S}_{low}(G)$ potential followers.*

A straightforward observation is that a candidate anchor cannot have any followers if it does not connect to any potential followers. This helps us prune out the unpromising anchors and identify the promising anchors defined as follows.

**Definition 6. Promising anchors.** *Given a bipartite graph $G$ and degree constraints $\alpha$ and $\beta$, the promising anchors are the upper vertices which connect to $\mathcal{S}_{up}(G)$ and not in*



| | $u_2$ | $u_5$ | $v_2$ | $v_3$ | $v_4$ | $u_3$ | $u_4$ |
|---|---|---|---|---|---|---|---|
| $O_U(u)$: | 0 | 0 | 1 | 2 | 3 | 4 | 5 |
| $r\text{-}score(u)$: | 7 | 6 | 0 | 2 | 2 | 0 | 0 |
| $\lvert sig(u)\rvert$: | 3 | 2 | - | - | - | 0 | 0 |

Fig. 3: The upper deletion order of the bipartite graph in Fig. 1 ($\alpha = 4, \beta = 3$). An red edge represents two edges.

*the $(\alpha, \beta)$-core (i.e., in $N(\mathcal{S}_{up}(G)) \setminus C_{\alpha,\beta}(G)$) and the lower vertices which connect to $\mathcal{S}_{low}(G)$ and not in the $(\alpha, \beta)$-core (i.e., in $N(\mathcal{S}_{low}(G)) \setminus C_{\alpha,\beta}(G)$). The other vertices in $G$ are called unpromising anchors.*

**Upper/lower deletion order.** To efficiently identify the promising anchors and their followers, we propose the **upper deletion order** (denoted by $\mathcal{O}_U$) and the **lower deletion order** (denoted by $\mathcal{O}_L$). For the upper deletion order, we compute the $(\alpha, \beta)$-core from the $(\alpha, \beta-1)$-core and assign $\mathcal{O}_U(u) = i$ ($i > 1$) for the $i^{th}$ deleted vertex $u$. In addition, for the other upper vertices that connect to some vertices already in the order and not in the $(\alpha, \beta)$-core, we also include them in the order and assign their order numbers as 0. Likewise, the lower deletion order is derived from the vertex deletion order when computing the $(\alpha, \beta)$-core from the $(\alpha - 1, \beta)$-core similarly. Note that these orders can be computed in $O(m)$ time and we will show the details later.

**Example 1.** *Consider maximizing the $(4, 3)$-core of the graph in Fig. 1 by choosing an upper anchor vertex. The upper deletion order of $v_2$, $v_3$, $v_4$, $u_3$, and $u_4$ are computed by iteratively deleting vertices from the $(4, 2)$-core until the $(4, 3)$-core is found. $\mathcal{O}_U(u_2)$ and $\mathcal{O}_U(u_5)$ are zero because $u_2$ and $u_5$ are not in the $(4, 2)$-core. Note that since $u_1$ is not connected to any potential followers, it is excluded from $\mathcal{O}_U$ and is not a promising anchor. $v_1$ is also excluded from $\mathcal{O}_U$ since it is neither an upper vertex nor a potential follower.*

## *B. The filter-verification framework*

Based on the above deletion orders, we propose a filter-verification framework to find the best anchor in one iteration.

**Filter stage.** In this stage, the unpromising anchors are filtered out in two steps. (1) We filter out the vertices that are not in either of the orders. These vertices are either in the $(\alpha, \beta)$-core or outside of the $(\alpha, \beta)$-core but not connected to any potential followers. (2) For each remaining candidate anchor $x$, we compute an order-based upper bound of the number of its followers (i.e., $|F(x)|$) and leave out those whose upper bounds are zero.

To derive an upper bound for $|F(x)|$, we first introduce the following concepts.

**Definition 7. Order-reachable.** *Given the upper (lower) deletion order $\mathcal{O}_U$ ($\mathcal{O}_L$) and a vertex $x$ in $U(G)$ ($L(G)$), a vertex $u$ is order-reachable from $x$, if there exists a path from $x$ to $u$ such that for any adjacent vertices $x_1$ and $x_2$ on the path $\mathcal{O}_U(x_1) < \mathcal{O}_U(x_2)$ ($\mathcal{O}_L(x_1) < \mathcal{O}_L(x_2)$).*

**Lemma 1.** *For an anchor $x$, the followers of $x$ must be order-reachable from $x$.*

*Proof.* Let $u$ be a follower of $x$. It is immediate that $x$ must precede $u$ in the upper (lower) deletion order. Otherwise, $u$ will be deleted before $x$ during the anchored $(\alpha, \beta)$-core computation. Also, since $u$ must gain support from $x$ to stay in the anchored $(\alpha, \beta)$-core, $u$ must be order-reachable from $x$. Thus, this lemma holds. □

Based on Lemma 1, $F(x)$ must be contained by the set of order-reachable vertices from $x$, denoted as *rf*$(x)$. Thus, $|$*rf*$(x)|$ is an upper bound for $|F(x)|$. However, computing $|$*rf*$(x)|$ for all remaining anchors needs $O(nm)$ time since for each anchor, it takes $O(m)$ time to traverse all the order-reachable vertices using BFS. To reduce the complexity, we resort to a coarser and recursive definition of upper bound as follows.

$$r\text{-}score(x) = \begin{cases} \sum_{u \in W(x)} (r\text{-}score(u) + 1) & |W(x)| > 0, \\ 0 & otherwise \end{cases}$$

Here $W(x)$ denotes the set of neighbors of $x$ that are order-reachable. Note that $r\text{-}score(x)$ allows us to pass the estimates of $|$*rf*$(x)|$ through neighbors recursively. By processing vertices in a reverse deletion order, the above upper bounds of all promising anchors can be computed easily using dynamic programming in $O(m)$ time. For example, as shown in Fig. 3, $r\text{-}score(u_3) = r\text{-}score(u_4) = 0$ because no vertices are order-reachable from them. By the above definition, $r\text{-}score(v_4) = (r\text{-}score(u_3) + 1) + (r\text{-}score(u_3) + 1) = 2$. If $r\text{-}score(x) = 0$, $x$ is an unpromising anchor and can be pruned since $|F(x)| \leq r\text{-}score(x)$ (e.g., $u_3$ and $u_4$ in Fig. 3). In addition, we will show that $r\text{-}score(x)$ can also guide the greedy algorithm when searching for the best anchor since it reflects the anchor's potential to produce followers.

**Verification stage.** In this stage, we compute the followers for the promising anchors and find the best anchor. By Lemma 1, for each promising anchor, we only consider the order-reachable vertices of it as candidate followers. Thus, instead of computing $F(x)$ globally from the input graph $G$, we can compute $F(x)$ based on the upper/lower deletion orders in a local manner as outlined in Algorithm 1. Without loss of generality, we present the techniques in the context of anchoring an upper vertex.

To compute the followers of an upper anchor $x$, we traverse the vertices that are order-reachable from $x$ in ascending order of upper deletion order (by iteratively adding them into $V'$). In this process, the vertices in $\mathcal{O}_U$ are divided into three groups:

- the set of unvisited vertices, denoted by $V_c$;
- the set of discarded vertices, i.e., the visited vertices that are proven not followers of $x$, denoted by $V_d$;
- the set of survived vertices, i.e., the visited vertices that satisfy the degree constraints and can be followers of $x$, denoted by $V_s$.

For each potential follower $u \in \mathcal{O}_U$, to determine if $u$ is a follower of $x$, we need to evaluate which neighbors of $u$ can stay in the anchored $(\alpha, \beta)$-core. Note that the neighbors of $u$ in $V_s$

**Algorithm 1:** Compute Followers

```
Input: $x$: an upper anchor; $\mathcal{O}_U$: the upper deletion order
Output: $F(x)$: the followers of $x$
1  $V' \leftarrow \{x\}$; $V_s \leftarrow \{x\}$; $V_d \leftarrow \emptyset$;
2  while $V'$ is not empty do
3      $u \leftarrow$ the vertex in $V'$ s.t. $\mathcal{O}_U(u)$ is minimal;
4      compute $d_{ub}(u)$ accordingly;
5      if $d_{ub}(u)$ meets the degree constraints then
6          $V_s \leftarrow V_s \cup \{u\}$; /* mark $u$ as survived */
7          foreach $v \in N(u)$ such that $v$ is unvisited and
             $\mathcal{O}_U(u) < \mathcal{O}_U(v)$ do
8              $V' \leftarrow V' \cup \{v\}$;
9              mark $v$ as visited;
10     else
11         $V_d \leftarrow V_d \cup \{u\}$; /* mark $u$ as discarded */
12         update $d_{ub}(v)$ for each vertex $v \in \mathcal{O}_U$;
13         add the vertices in $V'$ that violate the degree
             constraints into $V_d$;
14     $V' \leftarrow V' \setminus \{u\}$;
15 return the survived vertices in $V_s \setminus \{x\}$ as $F(x)$;
```

and $V_c$ can be in the anchored $(\alpha, \beta)$-core while the ones in $V_d$ cannot. All neighbors of $u$ in the $(\alpha, \beta)$-core are also included. Thus, the upper bound of the degree of $u$ in the anchored $(\alpha, \beta)$-core is $|N(u) \cap V_c| + |N(u) \cap V_s| + |N(u) \cap C_{\alpha,\beta}(G)|$, denoted by $d_{ub}(u)$ (Line 4). If $d_{ub}(u)$ satisfies the degree constraints, then $u$ is marked as "survived", and the unvisited, order-reachable neighbors of $u$ are added into $V'$ as candidate followers (Lines 5-9). Otherwise, $u$ is marked as discarded and added into $V_d$. Subsequently, the degree upper bounds of other potential followers in $\mathcal{O}_U$ are updated recursively and those violating the degree constraints are marked as discarded as well (Lines 10-13). When the algorithm terminates, the vertices in $V_s$ satisfy the degree constraints and are the followers of $x$. Note that if $x$ is a lower anchor, we use $\mathcal{O}_L$ to compute $F(x)$ instead of $\mathcal{O}_U$ in Algorithm 1.

**The FILVER algorithm.** Based on the above techniques, we propose the FILVER algorithm as shown in Algorithm 2. Each iteration of FILVER follows the filter-verification framework, which includes (1) a filter stage where the promising anchors are identified and ranked; (2) a verification stage where the anchors' followers are computed and the best anchor is chosen.

In the filter stage (Lines 3-6), the upper/lower deletion orders are computed by calling the `OrderComputation` procedure. Firstly, it computes the upper deletion order by iteratively removing the vertices that violate the degree constraints (in $P$) until the $(\alpha, \beta)$-core is found (Lines 17-22). For each vertex $u$ deleted in the $i^{th}$ iteration, $\mathcal{O}_U(u)$ is set to $i$. In Line 23, for each upper promising anchor $u$ not in $\mathcal{S}_{up}(G)$, $\mathcal{O}_U(u)$ is set to 0. In Lines 24-25, lower deletion order is computed with the same time complexity. Then, the promising anchor set PA is initialized to be the vertices in either of these two orders (Line 4). For each $x \in$ PA, we compute $r\text{-}score(x)$ and exclude $x$ from PA if $r\text{-}score(x) = 0$ (Lines 5-6).

In the verification stage (Lines 7-13), we explore the promising anchors in non-increasing order of their upper bounds and record the current best anchor as $x^*$. For each explored anchor $x$, we compute $F(x)$ by calling Algorithm 1. Note that we do not need to compute $F(x)$ in the following two situations.

- If there exists a visited anchor $x'$ such that $x \in F(x')$,

**Algorithm 2:** FILVER

```
Input: $G$: a bipartite graph; $\alpha, \beta$: the degree constraints;
       $b_1, b_2$: budgets
Output: $A$: the set of anchors
1  $A \leftarrow \emptyset$;
2  while $|A \cap U(G)| < b_1$ or $|A \cap L(G)| < b_2$ do
3      compute upper/lower deletion orders with Procedure
         Order Computation;
4      PA $\leftarrow$ the vertices in the upper/lower deletion order;
5      compute $r\text{-}score(x)$ for each $x$ in PA;
6      remove the vertices with $r\text{-}scores$ being 0 from PA;
7      $x^* \leftarrow$ a vertex in PA with the largest $r\text{-}score$;
8      foreach $x \in$ PA with non-increasing r-score do
9          if $x$ is not dominated by $x^*$ or any previous
             explored anchors then
10             compute $F(x)$ using Algorithm 1;
11             if $|F(x)| > |F(x^*)|$ then
12                 $x^* \leftarrow x$;
13     $A \leftarrow A \cup \{x^*\}$;
14 return $A$;
15 Procedure OrderComputation($G$, $\alpha$, $\beta$);
16 $\mathcal{G} \leftarrow$ the $(\alpha, \beta - 1)$-core of $G$; $i \leftarrow 0$;
17 $P \leftarrow$ the vertices in $\mathcal{G}$ violating the degree constraints;
18 while $P$ is not empty do
19     foreach $u \in P$ do
20         remove $u$ and its incident edges from $\mathcal{G}$;
21         $i \leftarrow i + 1$ and $\mathcal{O}_U(u) \leftarrow i$;
22     $P \leftarrow$ vertices in $\mathcal{G}$ violating the degree constraints;
23 for each $u \in U(G)$ connecting to $\mathcal{S}_{up}(G)$ and not in
     $C_{\alpha,\beta-1}(G)$, let $\mathcal{O}_U(u) \leftarrow 0$;
24 run Lines 16-22 with $\mathcal{G}$ replaced by the $(\alpha - 1, \beta)$-core and
     $\mathcal{O}_U$ replaced by $\mathcal{O}_L$;
25 for each $u \in L(G)$ connecting to $\mathcal{S}_{low}(G)$ and not in
     $C_{\alpha-1,\beta}(G)$, let $\mathcal{O}_L(u) \leftarrow 0$;
26 return $\mathcal{O}_U$ and $\mathcal{O}_L$
```

then we skip computing $F(x)$ since $|F(x)| \leq |F(x')|$;
- If $r\text{-}score(x) \leq |F(x^*)|$, we also skip $x$.

When $|F(x)| > |F(x^*)|$, we update the best anchor $x^*$ to $x$. At the end of each iteration, we add $x^*$ to $A$ and update $G$ by $G_A$, which is the graph with vertices in $A$ anchored.

**Complexity analysis.** FILVER needs to run $b_1 + b_2$ iterations. In each iteration, the upper/lower deletion order computation takes $O(m)$ time. Computing $r\text{-}score(x)$ for one anchor $x$ takes $O(deg(x))$ time, so computing all upper bounds needs $\sum_x O(deg(x, G)) = O(m)$. For each anchor $x$, computing $F(x)$ using Algorithm 1 takes $O(m)$ time. Thus, the total time complexity of FILVER is $O((b_1 + b_2)(n' \times m))$, where $n'$ is the number of anchors we compute followers for in each iteration, which is much smaller than $n$ in practice. The space complexity is $O(m)$ because the input graph takes $O(m)$ space and the upper/lower deletion orders take $O(n)$ space.

## IV. Filter stage optimizations

**Motivations.** Although the FILVER algorithm achieves significant speedup compared to the Naive algorithm, the filter stage of FILVER still lacks pruning power for the following reasons.

- In FILVER, some unfiltered anchors may be "dominated" by other anchors, which means that their followers are fully covered by others'.

- When a new anchor is placed, the upper/lower deletion orders can change a lot and need to be recomputed from scratch in each iteration of FILVER.

In this section, we propose filter-stage optimizations to address these issues. Firstly, we unveil the dominating relationships among candidate anchors and leverage bipartite graph structure to efficiently filter many "dominated" anchors. Secondly, we exploit the nested property of the $(\alpha, \beta)$-core to identify which parts of the upper/lower deletion orders needs recomputation.

## *A. Discover dominating relationships among anchors*

As shown in Fig. 3, the follower set of $u_5$ ($\{u_3, u_4, v_3, v_4\}$) is contained by the follower set of $u_2$ ($\{u_3, u_4, v_2, v_3, v_4\}$). In this case, $u_5$ cannot become the best anchor in the current iteration since it is "dominated" by $u_2$, and we wish to filter such anchors earlier. However, it is difficult to predict and exploit such relationships in the filter stage, when the followers are not yet computed. To effectively filter more "dominanted" anchors, we introduce the anchors' follower signatures.

**Definition 8. Follower signature.** *Given an anchor $x$ in $\mathcal{O}_U$ or $\mathcal{O}_L$, the follower signature of $x$ is the set of its neighbors that are order-reachable from $x$, denoted by $sig(x)$.*

The follower signature can be considered as the starting point to compute the followers of an anchor as shown in Algorithm 1. For example, in Fig. 3, $v_2$, $v_3$, and $v_4$ are the neighbors of $u_2$ that are order-reachable from it, so $sig(u_2) = \{v_2, v_3, v_4\}$. Intuitively, if two anchors have the same follower signature, they must produce the same followers. The following lemma depicts a more general picture.

**Lemma 2.** *Consider two anchors $x_1$ and $x_2$ that are both in $U(G)$ or $L(G)$. If $sig(x_1) \subseteq sig(x_2)$, then $F(x_1) \subseteq F(x_2)$.*

*Proof.* Let $G_1$ and $G_2$ be the anchored $(\alpha, \beta)$-core w.r.t. $x_1$ and $x_2$, respectively. Since $sig(x_1) \subseteq sig(x_2)$, if we replace $x_1$ by $x_2$ in $G_1$ (denoted as $G_1'$), $G_1'$ will still satisfy the degree constraints of $(\alpha, \beta)$-core. By Definition 2, $G_2$ is the maximal subgraph with $x_2$ anchored that satisfies the degree constraints, which means that $G_1'$ must be a subgraph of $G_2$. Thus, $F(x_1) \subseteq F(x_2)$, and the lemma holds. □

Motivated by the above lemma, the anchors whose follower signatures are dominated by others can be safely pruned. To implement this, a naive approach is to compute the follower signatures of all promising anchors and perform pairwise comparisons. This is obviously too time-consuming due to the vast number of follower signatures.

By Definition 8, given an anchor $x$, the vertices whose follower signatures can dominate or be dominated by $sig(x)$ must be the two-hop neighbors of $x$. In addition, according to the bipartite graph structure, the search scope for these two-hop neighbors is limited to the layer $x$ resides on. For example, in Fig. 3, the two-hop neighbors of $u_2$ in $\mathcal{O}_U$ are $u_3$, $u_4$, an $u_5$, which are all upper layer vertices. Here we formally define these vertices as the order-obeying two-hop neighbors.

**Definition 9. Order-obeying two-hop neighbors.** *Consider an anchor $x$ in $U(G)$ ($L(G)$). If there exists a path $x \rightsquigarrow v \rightsquigarrow w$ such that $\mathcal{O}_U(x) < \mathcal{O}_U(v)$ and $\mathcal{O}_U(w) < \mathcal{O}_U(v)$ ($\mathcal{O}_L(x) < \mathcal{O}_L(v)$ and $\mathcal{O}_L(w) < \mathcal{O}_L(v)$), then $w$ is an order-obeying two-hop neighbor of $x$.*

Based on Definition 9, we devise a two-hop filtering algorithm to detect the dominating relationships among the anchors on the same layer. Without loss of generality, the algorithm is shown in Algorithm 3 in the context of filtering upper anchors.

**Algorithm 3:** Two-hop Filtering Algorithm

```
Input: $G$: a bipartite graph; $\alpha, \beta$: the degree constraints;
Output: PA: the set of remaining anchors
1  PA $\leftarrow \emptyset$;
2  foreach $x \in U(G) \cap \mathcal{O}_U$ with non-decreasing $|sig(x)|$ do
3      mark $x$ as visited;
4      $v_1 \leftarrow$ the vertex in $sig(x)$ with the minimum degree;
5      $D \leftarrow N(v_1)$;
6      foreach vertex $v \in sig(x) \setminus \{v_1\}$ do
7          if $|D| \cdot log(deg(v)) < deg(v)$ then
8              binary search for each vertex $w \in D$ in $N(v)$;
9          else
10             for each vertex $w$ in $N(v)$ check if $w \in D$;
11         $D \leftarrow D \cap N(v)$;
12     if $D$ is not empty then
13         discard $x$;
14     else
15         Add $x$ to PA and compute $|rf(x)|$;
16 return PA
```

**The two-hop filtering algorithm.** The two-hop filtering algorithm visits the upper anchors in $\mathcal{O}_U$ in non-decreasing order of the sizes of their follower signatures (Line 2). In this manner, for each anchor $x$, we only need to check if $x$ is dominated by any unvisited anchor, i.e., an anchor with a follower signature at least as large as $sig(x)$ (Line 3-15). Evidently, the unpromising anchors with empty follower signatures do not need to be considered.

For any anchor $y$ dominating $x$, $y$ must be an order-obeying two-hop neighbor via all vertices in $sig(x)$, i.e., $y \in \cap_{v \in sig(x)} N(v) \setminus \{x\}$. We use $D$ to store the vertices that may dominate $x$ and initialize $D$ as the neighbor set of the vertex with the smallest degree ($v_1$) in $sig(x)$ (Lines 4-5). Note that we only include the unvisited anchors whose upper deletion order is higher than $\mathcal{O}_U(v)$ in $D$. Then, we visit each vertex $v \in sig(x) \setminus \{v_1\}$ and update $D$ as $D \cap N(v)$ (Lines 6-11). To compute $D \cap N(v)$, we can have the following two methods.

- We visit each neighbor $w$ of $v$ and check whether $w \in D$.
- Suppose $N(v)$ is sorted beforehand, we can conduct a binary search on $N(v)$ for each $w \in D$.

The first method takes $O(deg(v))$ time and the second needs $O(|D| \cdot log(deg(v)))$ time. Thus, for each $v \in sig(x)$, we can pre-compute $deg(v)$ and $|D| \cdot log(deg(v))$ to choose which method is more efficient (Line 7). Note that if $D$ becomes empty at any point, it means that no anchor can dominate $x$ and we terminate the for-loop of Lines 6-11 early. Otherwise, we discard $x$ (Line 13). The time complexity of Algorithm 3 Lines 1-13 is $O(\sum_{(u,v) \in E(G)} min(deg(v), min_{w \in N(v)}(deg(w)) \cdot log(deg(v))))$. Note that since |PA| is drastically reduced compared to FILVER, we can afford to compute $|rf(x)|$ as the upper bound for $|F(x)|$ for each remaining anchor $x \in$ PA, by using a BFS from $x$ in $O(\sum_{u\in rf(x)\cup\{x\}} deg(u))$ time (Line 15). To filter lower anchors with dominated follower signatures, we use $L(G)\cap\mathcal{O}_L$ instead of $U(G)\cap\mathcal{O}_U$.

**Lemma 3.** *The two-hop filtering algorithm correctly excludes all anchors whose follower signatures are being "dominated".*

*Proof.* For any $x_1$ and $x_2$ such that $sig(x_1)\subseteq sig(x_2)$, $|sig(x_1)|\le|sig(x_2)|$. (1) $|sig(x_1)|<|sig(x_2)|$. In this case, the algorithm processes $x_1$ first and $x_2$ will occur in $D$ after Line 11. (2) $|sig(x_1)|=|sig(x_2)|$. In this case, $x_1$ and $x_2$ must share the same follower signature, and the algorithm will keep one of $x_1$ and $x_2$. Thus, the lemma holds. □

**Example 2.** *We process the upper anchors of $\mathcal{O}_U$ in Fig. 3 in non-decreasing order of their follower signatures. $u_3$ and $u_4$ are pruned since $sig(u_3)=sig(u_4)=\emptyset$. We visit $u_5$ first and go through the vertices in $sig(u_5)=\{v_3,v_4\}$. Since $u_2\in N(v_3)\cap N(v_4)$ and $|sig(u_2)|>|sig(u_5)|$, $u_5$ is dominated by $u_2$ and thus can be safely pruned.*

**Algorithm 4:** Order Maintenance Algorithm

```
Input: $x^*$: the best anchor chosen in the last iteration
Output: $\mathcal{O}_U$: the upper deletion order
1 $G^*\leftarrow\emptyset$;
2 $V(G^*)\leftarrow$ the vertices visited by BFS from $x^*$ restricted to
    vertices with $core_U(u)\ge core_U(x^*)$;
3 restore the edges in $G^*$ from $E(G)$;
4 foreach $core_U(x^*)\le\beta'\le\beta-1$ do
5     compute the $(\alpha,\beta')$-core component containing $x^*$;
6     for each deleted vertex $u$, $core_U(u)\leftarrow\beta'$;
7 Invoke Procedure OrderComputation from Algorithm 2
    to compute $\mathcal{O}_U(u)$ for all $u\in V(G^*)$;
8 Properly assign $core_U(u)$ for all $u\in V(G^*)$;
9 return $\mathcal{O}_U$;
```

**Algorithm 5:** FILVER$^+$

```
Input: $G$: a bipartite graph; $\alpha,\beta$: the degree constraints;
    $b_1,b_2$: budgets;
Output: $A$: the set of anchors
1 $A\leftarrow\emptyset$;
2 while $|A\cap U(G)|<b_1$ or $|A\cap L(G)|<b_2$ do
3     if $A$ is empty then
4         compute $\mathcal{O}_U$ and $\mathcal{O}_L$ from $G$ with the Order
            Computation procedure;
5     else
6         maintain $\mathcal{O}_U$ and $\mathcal{O}_L$ with Algorithm 4;
7     PA $\leftarrow$ the vertices in the upper/lower deletion order;
8     filter the anchors in PA by Algorithm 3;
9     $x^*\leftarrow$ a vertex in PA with the largest $|rf(x)|$;
10    foreach $x\in$ PA with non-increasing $|rf(x)|$ do
11        if $x$ is not dominated by $x^*$ or any previous
            explored anchors then
12            compute $F(x)$ using Algorithm 1;
13            if $|F(x)|>|F(x^*)|$ then
14                $x^*\leftarrow x$;
15    $A\leftarrow A\cup\{x^*\}$;
16 return $A$;
```

## B. Connectivity-based order maintenance

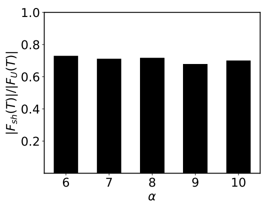
In the FILVER algorithm, the upper/lower deletion orders are recomputed from the input graph in each iteration, which is time-consuming. An intuitive alternative is to only rebuild the parts of these orders that are affected by the last chosen anchor (hereafter denoted by $x^*$). To achieve this, we first introduce the following concept.

**Definition 10. Upper/lower core numbers.** *Given a bipartite graph G and degree constraints α and β, the upper core number of a vertex u is the largest integer k such that $u\in(\alpha,k)$-core, i.e., $core_U(u)=\max\{k|u\in(\alpha,k)\text{-core}\}$. Likewise, the lower core number of u is the largest integer k such that $u\in(k,\beta)$-core, i.e., $core_L(u)=\max\{k|u\in(k,\beta)\text{-core}\}$.*

The upper/lower core numbers allow us to measure the influential scope of $x^*$ on $\mathcal{O}_U$ and $\mathcal{O}_L$. We denote the upper core number of $x^*$, $core_U(x^*)$ as $\beta^*$. Since $x^*$ can only affect the upper core numbers and $\mathcal{O}_U$ in the $(\alpha,\beta')$-core with $\beta'\ge\beta^*$, we identify the connected component of $(\alpha,\beta^*)$-core containing $x^*$ as the **upper affected graph** of $x^*$, denoted by $AG_U(x^*)$. Note that we only need to recompute $\mathcal{O}_U$ for the vertices in $AG_U(x^*)$. Likewise, if $core_L(x^*)=\alpha^*$, then the connected component of $(\alpha^*,\beta)$-core containing $x^*$ is the **lower affected graph** of $x^*$, denoted by $AG_L(x^*)$. We only need to recompute $\mathcal{O}_L$ for the vertices in $AG_L(x^*)$. Based on these observations, we propose the order maintenance algorithm, as outlined in Algorithm 4.

**The order maintenance algorithm.** Without loss of generality, Algorithm 4 is presented in the context of updating $\mathcal{O}_U$. $G^*$ is used to represent the upper affected graph of $x^*$ (Line 1). The vertices of $G^*$ is found by conducting a BFS from $x^*$ and only visiting the vertices with upper core numbers no less than $core_U(x^*)$ (Line 2). Since updating $\mathcal{O}_U$ locally demands the $(\alpha,\beta\text{-}1)$-core, we compute it from $G^*$ by gradually increasing the degree constraint $\beta'$ and correctly update the upper core numbers in $G^*$. Note that due to the nested property of $(\alpha,\beta)$-core, we always compute $(\alpha,\beta')$-core component from $(\alpha,\beta'-1)$-core component in Line 5. Then, since only the part of $\mathcal{O}_U$ in $G^*$ is changed, we apply the OrderComputation procedure from Algorithm 2 on $G^*$ (Line 7). To properly assign the upper core number in $G^*$ (Line 8), we set $core_U(u)$ to $\beta-1$ for each deleted vertex $u$ in Line 7. For the remaining vertices in the anchored $(\alpha,\beta)$-core, we assign their upper core numbers as $\beta$. Note that Algorithm 4 can be used to update $\mathcal{O}_L$ by replacing upper core numbers and $\beta-1$ in Line 4 to lower core numbers and $\alpha-1$, respectively. The time cost of Algorithm 4 is linear to the size of the upper/lower affected graph.

**The FILVER$^+$ algorithm.** Following the filter-verification framework, we propose the FILVER$^+$ algorithm based on the above filter-stage optimizations, as outlined in Algorithm 5. Specifically, we only compute $\mathcal{O}_U$ and $\mathcal{O}_L$ from the input graph $G$ in the first iteration when $A$ is empty (Line 4). Otherwise, we maintain $\mathcal{O}_U$ and $\mathcal{O}_L$ based on the best anchor of the last iteration (Line 6). Also, we use the two-hop filtering algorithm to eliminate the anchors with dominated follower signatures from PA and compute a tight upper bound for the remaining anchors (Line 8). The logic of the verification stage (Lines 10-15) remains the same as the FILVER algorithm.


(a) varying $\alpha$ on WC, $\beta = 7$

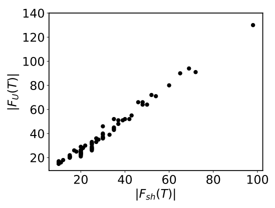

(b) $\alpha = 6$ and $\beta = 7$ on WC

Fig. 4: Comparing $|F_{sh}(T)|$ with $|F(T)|$, where $|T| = 5$

## V. Verification stage optimizations

**Motivation.** With filter-stage optimizations, FILVER$^+$ significantly outperforms FILVER. To further speed up FILVER$^+$, here we investigate verification-stage optimizations. An intuitive idea is to reduce the number of iterations (in the verification stage) by placing multiple $t$ ($t > 1$) anchors in each iteration. However, the problem of finding multiple anchors per iteration is the anchored $(\alpha, \beta)$-core problem with smaller budgets and thus is NP-hard. It is cost-prohibitive to search through all possible combinations of $t$ anchors. In this section, we propose a new strategy to effectively find a set $T$ of anchors which can bring more followers in one iteration.

### *A. Anchor set maintenance*

In each iteration, we maintain an anchor set $T$ of size $t$ and try to update $T$ with a new incoming anchor $x$ such that $|F(T)|$ is increased. For each $x$, we need to check all possible ways to replace one anchor in $T$ by $x$, which requires $O(t \times m)$ time in total. Thus, it does not provide much efficiency improvement if we anchor $t$ vertices per iteration in this way. To improve the efficiency while ensuring the quality of anchor vertices, we aim to find a metric to identify the anchor with "the least contribution" in $T$ and only compare this anchor with the new incoming anchor. We first show that the anchors' followers in $T$ can be computed collectively as follows.

$$F(T) = \bigcup_{x \in T} F(x) \cup \bigcup_{1 < i \leq |T|} (f_T(i) \setminus f_T(i-1)) \qquad (1)$$

where $f_T(i)$ denotes the set of followers induced by any anchor set $X$ with size equal to $i$ (i.e., $f_T(i) = \bigcup_{X \subseteq T, |X| = i} F(X)$). In this formula, the first term $\bigcup_{x \in T} F(x)$ is the union of the followers independently brought by each anchor in $T$, which is hereafter called the *in-shell follower set* of $T$ and denoted as $F_{sh}(T)$. The second term represents the collective effects of multiple anchors.

According to Equation 1, $F_{sh}(T)$ is a subset of $F(T)$ and thus $|F_{sh}(T)|$ can be considered as a lower bound of $|F(T)|$. As shown in Fig. 4(a), we randomly select 100 sets of anchors on a representative dataset WC and compute the ratio $|F_{sh}(T)|/|F(T)|$. It can be observed that $|F_{sh}(T)|$ is around $0.7 \times |F(T)|$ and is a tight lower bound of $|F(T)|$. For each randomly selected anchor set $T$ when $\alpha = 6$ and $\beta = 7$, we also report the sizes of $F_{sh}(T)$ and $F(T)$ in Fig. 4(b), which indicates that $|F_{sh}(T)|$ and $|F(T)|$ are highly correlated. Now we can define the following concepts based on $|F_{sh}(T)|$ to measure the contribution of each anchor in $T$.

**Definition 11. Exclusive follower set.** *Given the anchor set $T$, for each $x \in T$, the exclusive follower set of $x$ in $T$, denoted by $F_{ex}(x, T)$, is the set of followers that are not followers of any other anchors in $T$, i.e., $F_{ex}(x, T) = F(x) \setminus F_{sh}(T \setminus \{x\})$.*

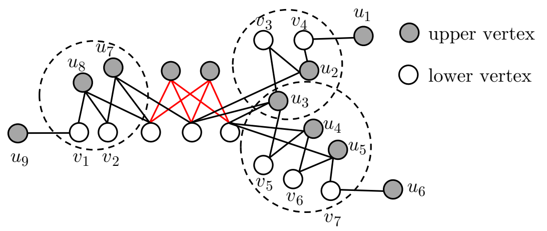


Fig. 5: Maintaining an anchor set

**Definition 12. Least-contribution-anchor.** *Given a set of anchors $T$, the least-contribution-anchor is the anchor $x \in T$ s.t. $|F_{ex}(x, T)|$ is minimized, denoted by $x_{min}(T)$.*

Based on Definition 11, if an anchor $x' \in T$ has a large size of exclusive follower set, we consider $x'$ has a high contribution to $T$ because many followers in $F_{sh}(T)$ depend solely on $x'$. On the contrary, the anchor with the smallest number of exclusive followers in $T$, i.e., $x_{min}(T)$, can be viewed as the anchor with the least contribution to $T$. Therefore, given a new anchor $x$, we can replace $x_{min}(T)$ with $x$ in $O(|F(x)|)$ time according to the following lemma.

**Lemma 4.** *Given an anchor set $T$ and a new anchor $x$, suppose $T'$ is the anchor set after replacing $x_{min}(T)$ with $x$, i.e., $T' = (T \setminus x_{min}(T)) \cup \{x\}$. If $F_{ex}(x, T') > F_{ex}(x_{min}(T), T)$, then $|F_{sh}(T')| > |F_{sh}(T)|$.*

*Proof.* Let $x = F_{sh}(T \setminus \{x_{min}(T)\})$. Then, $|F_{sh}(T')| = |x \cup F_{ex}(x, T')| = |x| + |F_{ex}(x, T')|$. Also, $|F_{sh}(T)| = |x \cup F_{ex}(x_{min}(T), T)| = |x| + |F_{ex}(x_{min}(T), T)|$. Thus, $|F_{sh}(T')| - |F_{sh}(T)| = F_{ex}(x, T') - F_{ex}(x_{min}(T), T)$ and this lemma holds. □

**Algorithm 6:** Anchor Set Maintenance

```
Input: $T$: the current anchor set; $x$: a new incoming anchor;
       $b_1, b_2$: budgets; $A$: the set of selected anchors
Output: $T$: the updated anchor set
1 if $|T| < t$ then
2     if $T \cup \{x\}$ satisfies the budget constraints then
3         $T \leftarrow T \cup x$; /* insert $x$ into $T$ */
4 else
5     $T' \leftarrow (T \setminus x_{min}(T)) \cup \{x\}$;
6     if $|F_{sh}(T')| > |F_{sh}(T)|$ and $T'$ satisfies the budget
      constraints then
7         $T \leftarrow T'$; /* replace $x_{min}(T)$ with $x$ */
8 return $T$
```

The anchor set maintenance algorithm is outlined in Algorithm 6. When $|T| < t$, we keep inserting the anchors into $T$ (Lines 1-3). When $|T| = t$, we replace $x_{min}(T)$ with $x$ if the new anchor set $T' = T \setminus x_{min}(T)) \cup \{x\}$ has a larger in-shell follower set (i.e., $|F_{sh}(T')| > |F_{sh}(T)|$) (Lines 5-7). By Lemma 4, $|F_{sh}(T')| > |F_{sh}(T)|$ is equivalent to $|F_{ex}(x, T')| > |F_{ex}(x_{min}(T), T)|$, so we only need to compute and compare the sizes of $F_{ex}(x, T')$ and $F_{ex}(x_{min}(T), T)$ in Line 6. Note that $T$ must always satisfy the budget constraints (Line 2 and Line 6) that $|T \cap U(G)| + |A \cap U(G)| \leq b_1$ and $|T \cap L(G)| + |A \cap L(G)| \leq b_2$.

**Complexity analysis of Algorithm 6.** The dominating cost of Algorithm 6 occurs when computing $|F_{ex}(x, T')|$ (Line 6) and replacing $x_{min}(T)$ with $x$ (Line 7). Computing $|F_{ex}(x, T')|$ can be done in $O(|F(x)|)$ time by visiting each element $u'$ in $F(x)$ and checking if $u'$ is covered by any anchor in $T \setminus x_{min}(T)$. Replacing $x_{min}(T)$ with $x$ can also be implemented in $O(|F(x)|)$ time by using a hash table that stores which vertices in $T$ have the exclusive follower sets of size $k$ for possible values of $k$. Therefore, the time complexity of Algorithm 6 is $O(|F(x)|)$.

**Example 3.** *In Fig. 5, the follower sets of anchors $u_1$, $u_6$, and $u_9$ are $\{u_2, u_3, v_3, v_4\}$, $\{u_3, u_4, u_5, v_5, v_6, v_7\}$, and $\{u_7, u_8, v_1, v_2\}$, respectively. When $t = 2$, let the current anchor set $T = \{u_1, u_6\}$. $u_1$ is the least-contribution-anchor of $T$ (i.e., $u_1 = x_{min}(T)$). Suppose $u_9$ is a newly processed anchor. Let $T'$ be the anchor set with $u_1$ replaced by $u_9$, i.e., $T' = \{u_6, u_9\}$. Since $|F_{ex}(u_9, T')| = 4 > |F_{ex}(u_1, T)| = 3$, we replace $u_1$ with $u_9$.*

```
Algorithm 7: FILVER$^{++}$
Input: $G$: a bipartite graph; $\alpha, \beta$: the degree constraints;
       $b_1, b_2$: budgets; $t$: the number of anchors placed in
       each iteration
Output: $A$: the set of anchors
1  $A \leftarrow \emptyset$;
2  while $|A \cap U(G)| < b_1$ or $|A \cap L(G)| < b_2$ do
3      run Lines 3-8 of Algorithm 5;
4      $T \leftarrow \emptyset$; /* initialize the anchor set */
5      foreach $x \in$ PA with non-increasing $|rf(x)|$ do
6          if $x$ is not dominated by any previous anchors and
           $|rf(x)| > |F_{ex}(x_{min}(T), T)|$ then
7              compute $F(x)$ using Algorithm 1;
8              update $T$ using Algorithm 6;
9      $A \leftarrow A \cup T$;
10     $G \leftarrow G_A$;
11 return $A$
```

## B. *The* FILVER$^{++}$ *algorithm*

Based on the verification-stage techniques, we propose the FILVER$^{++}$ algorithm as outlined in Algorithm 7. The filter stage of FILVER$^{++}$ is the same as Algorithm 5 (Line 3) except that when $A$ is not empty, FILVER$^{++}$ needs to maintain $\mathcal{O}_U$ and $\mathcal{O}_L$ against $t$ new anchors. Algorithm 4 can be extended to handle $t$ anchors in batch. To update $\mathcal{O}_U$, we visit each $x' \in T$ in non-decreasing order of $core_U(x')$ and invoke Algorithm 4 to update the part of $\mathcal{O}_U$ in $AG_U(x')$. Note that if an unvisited anchor $x''$ exists in $AG_U(x')$, then $AG_U(x'') \subseteq AG_U(x')$ since anchors with smaller upper core numbers are visited earlier. Hence, we do not need to maintain $\mathcal{O}_U$ w.r.t. $x''$. The lower deletion order $\mathcal{O}_L$ can be maintained in batch similarly.

In the verification stage (Lines 4-9), FILVER$^{++}$ explores the anchors that survived the filter stage in non-increasing order of their upper bounds. After computing $F(x)$ for an anchor $x$ by calling Algorithm 1 (Line 7), Algorithm 6 is invoked to maintain the anchor set $T$ (Line 8). Note that we do not need to compute $F(x)$ in the following two situations.

- If there exists a visited anchor $x'$ such that $x \in F(x')$, then we skip computing $F(x)$ since $|F(x)| \leq |F(x')|$;
- If $|rf(x)| \leq |F_{ex}(x_{min}(T), T)|$, $x$ cannot possibly improve $T$ and we also skip $x$.

At the end of each iteration, the anchors in $T$ are added to $A$.

**Complexity analysis of FILVER$^{++}$.** The time complexity of the filter stage of FILVER$^{++}$ is the same as that of FILVER$^{+}$. In the verification stage, for each remaining anchor $x$, it takes $O(m)$ time to compute $F(x)$ and $O(|F(x)|)$ time to maintain the anchor set $T$ using Algorithm 6. As $t$ anchors are chosen at a time, Algorithm 7 requires $\lceil \frac{b_1+b_2}{t} \rceil$ iterations til termination. Since the verification stage incurs the dominating cost, the total time complexity of FILVER$^{++}$ algorithm is $O(\lceil \frac{b_1+b_2}{t} \rceil (n'' \times m))$, where $n''$ is the number of anchors we compute followers for. Apart from the $O(m)$ storage cost of the input graph, the anchor set maintenance algorithm and storing the upper/lower deletion orders and upper/lower core numbers takes $O(n)$ space. The total space complexity of Algorithm 7 is $O(m)$ (if we suppose $m > n$).

## VI. Experimental evaluation

In this section, we evaluate the anchored $(\alpha, \beta)$-core model and the proposed algorithms through empirical studies.

### A. *Experimental Settings*

TABLE II: Summary of Datasets

| Dataset | $\|E\|$ | $\|U\|$ | $\|L\|$ | $d_{max}$ | $\delta$ |
|---|---|---|---|---|---|
| Unicode (UL) | 1.26K | 0.87K | 0.25K | 141 | 4 |
| Cond-mat (AC) | 58.60K | 38.74K | 16.73K | 116 | 8 |
| Writers (WR) | 144.34K | 135.57K | 89.36K | 246 | 6 |
| Producers (PR) | 207.27K | 187.68K | 48.83K | 512 | 6 |
| Movies (ST) | 281.40K | 157.18K | 76.10K | 321 | 7 |
| BookCrossing (BX) | 1.15M | 445.8K | 105.3K | 13,601 | 41 |
| Stack-Overflow (SO) | 1.30M | 545.2K | 96.7K | 6,119 | 22 |
| Taobao (TB) | 1.02M | 5.16M | 2.015M | 1393 | 10 |
| Wiki-en (WC) | 3.80M | 2.04M | 1.85M | 11,593 | 18 |
| Amazon (AZ) | 5.74M | 2.15M | 1.23M | 12,180 | 26 |
| DBLP (DB) | 8.65M | 1.43M | 4.00M | 951 | 10 |
| Epinions (ER) | 13.67M | 876.3K | 120.5K | 162,169 | 152 |
| Wiki-de (DE) | 57.32M | 3.62M | 425.8K | 278,998 | 156 |
| Delicious (DUI) | 101.80M | 34.61M | 833.1K | 29,240 | 184 |
| LiveJournal (LG) | 112.31M | 3.20M | 7.49M | 1,053,676 | 109 |
| Orkut (OG) | 327.04M | 11.51M | 2.78M | 318,240 | 467 |
| Synthetic (SN) | 1919.93 M | 5M | 5M | 36,360 | 359 |

**Datasets.** We use 16 real datasets in our experiments. 15 of them are from KONECT (http://konect.cc/), and `Taobao` is a user-item network from the Taobao user-behaviour dataset (https://tianchi.aliyun.com/), where each edge represents that a user has bought an item. To further challenge scalability, we generate the `Synthetic` dataset with about two billion edges using the well-known graph generator GTgraph under the Erdős–Rényi model (http://www.cse.psu.edu/~kxm85/software/GTgraph/). Table II shows the statistics of these datasets. $|U|$ and $|L|$ are the number of vertices in the upper and lower layers. $|E|$ is the number of edges in the graph. $d_{max}$ is the maximum degree in the graph, and $\delta$ is the maximum integer such that the $(\delta, \delta)$-core exists in the graph.

**Algorithms.** We evaluate the following greedy algorithms: Naive (the naive greedy algorithm) in Section II, FILVER (the basic filter-verification algorithm) in Section III, FILVER$^{+}$ (FILVER with the filter-stage-optimizations) in Section IV, and FILVER$^{++}$ (FILVER with both the filter-stage and verification-stage optimizations) in Section V. We also test FILVER against the exact algorithm Exact in Section II and other approaches

(Random, Top-Degree, and Degree-Greedy which will be described later). The algorithms are implemented in C++, and the experiments are run on a Linux server with an Intel Xeon E5-2698 processor and 512GB memory. *We terminate an algorithm if its running time exceeds* $10^5$ *seconds.*

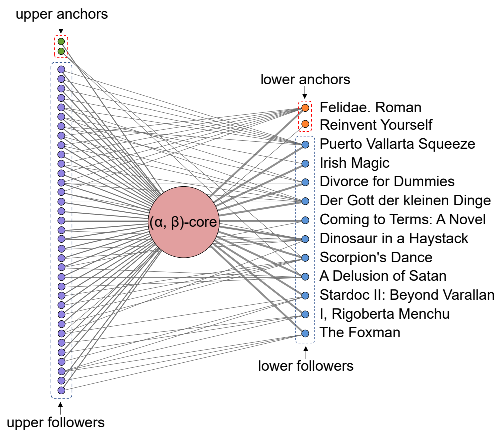
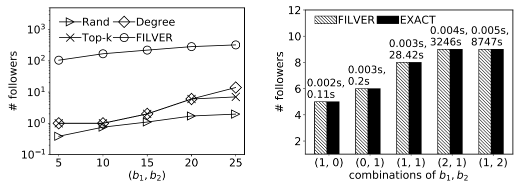


Fig. 6: A case study on BX



(a) WC, varying $b_1, b_2$ (b) UL, varying $b_1, b_2$

Fig. 7: Effectiveness of FILVER

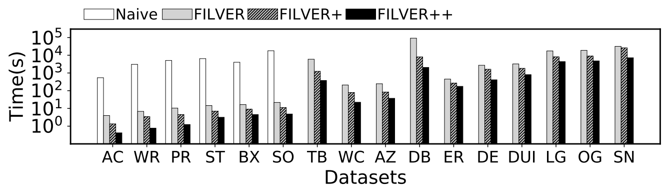


Fig. 8: Performance on different datasets

## B. Effectiveness Evaluation

Here we evaluate the effectiveness of the anchored $(\alpha, \beta)$-core model and the proposed filter-verification framework.

**Evaluate the effectiveness of anchored $(\alpha, \beta)$-core model.** Firstly, we conduct a case study to evaluate the effectiveness of the anchored $(\alpha, \beta)$-core model on a user-book dataset (BX) [31] by running FILVER. Here each edge indicates that a user has read a book. Fig. 6 shows the anchored $(3, 20)$-core with 2 upper anchors and 2 lower anchors. Due to these four anchors, there are 35 upper followers and 11 lower followers entering in the anchored (3, 20)-core. Note that some followers are not connected to any anchor vertices in the result since vertices can gain support from not only anchors but also other followers.

**Evaluate the effectiveness of FILVER.** We evaluate the effectiveness of FILVER by comparing the number of followers generated with other approaches. Note that we omit the results from Naive algorithm since it yields the same anchors and followers as FILVER. In Fig. 7(a), we show the number of followers produced by 4 algorithms (Random, Top-Degree, Degree-Greedy and FILVER) as budgets $b_1$ and $b_2$ increase from 5 to 25 on WC. $\alpha$ and $\beta$ are set to 10 and 7, respectively. Random assigns $b_1$ anchors in $U(G)$ and $b_2$ anchors in $L(G)$ arbitrarily. Top-Degree assigns the $b_1$, $b_2$ anchors with the top-$b_1$ and top-$b_2$ largest degrees in $U(G)$ and $L(G)$, respectively. Degree-Greedy follows a greedy strategy and selects the anchor with the largest degree from $V(G) \setminus C_{\alpha,\beta}(G_A)$ iteratively until there is no budget left. We observe that the degree-based algorithms slightly outperform Random. Also, FILVER produces significantly more followers than other algorithms.

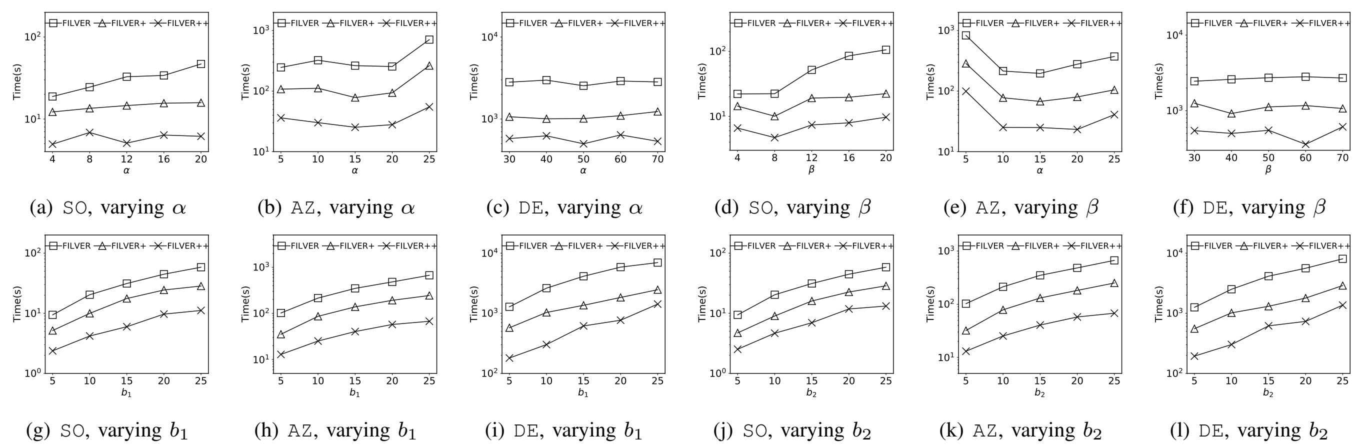
We also compare FILVER to Exact, which checks all combinations of candidate anchors and finds the optimal solution. Fig. 7(b) shows the number of followers produced by FILVER and Exact on UL w.r.t. different $b_1$ and $b_2$, where $\alpha = 4$, $\beta = 3$. FILVER can find the optimal solution in all these settings. Note that the running time of Exact grows exponentially, so we only test FILVER against Exact on a small dataset with limited budgets.

## C. Performance Evaluation

In this part, we evaluate the performance of Naive, FILVER, FILVER$^+$, and FILVER$^{++}$. First, we show the running time of these algorithms on 16 datasets (UL is omitted since it is too small). Then, we investigate the effect of $\alpha, \beta, b_1, b_2$ and $t$. By default, $\alpha$ and $\beta$ are set to $0.6\delta$ and $0.4\delta$, respectively. $b_1, b_2$ are set to 10, and $t$ is set to 5.

**Evaluate performance on different datasets.** Fig. 8 shows the running time of Naive, FILVER, FILVER$^+$, and FILVER$^{++}$ on 16 datasets with default parameters. We observe that the algorithms under the filter-verification framework (i.e., FILVER, FILVER$^+$, and FILVER$^{++}$) are much faster than Naive on all datasets. For instance, Naive cannot finish within the time limit on datasets larger than SO, while FILVER, FILVER$^+$, and FILVER$^{++}$ are scalable to the graph SN with about two billion edges. This is because Naive identifies a very large pool of candidate anchors and needs to traverse the whole graph to compute followers for each of them. We also observe that both the filter-stage and verification-stage optimizations significantly improve the efficiency of FILVER. By applying the filter-stage optimizations, FILVER$^+$ outperforms FILVER on all the datasets. In addition, with optimizations in two stages, FILVER$^{++}$ outperforms FILVER by about one order of magnitude on AC, WR, PR, and WC. Especially, on TB and DB, the relative speedup of FILVER$^{++}$ versus FILVER are $15\times$ and $44\times$, respectively. Note that we omit Naive in the remaining experiments since it is not scalable.

**Evaluate the effect of degree constraints $\alpha$ and $\beta$.** In the first row of Fig. 9, we report the running time of FILVER, FILVER$^+$, and FILVER$^{++}$ for different values of $\alpha$ and $\beta$ on SO, AZ, and DE. We can observe that the running time does not necessarily change as $\alpha$ or $\beta$ varies, because the degree constraints do not contribute to the time complexity of these algorithms. A clear pattern depicted in the figures is that FILVER$^+$ consistently outperforms FILVER, while FILVER$^{++}$ outperforms FILVER$^+$. This again validates the effectiveness of the proposed optimizations and the efficiency of FILVER$^{++}$.

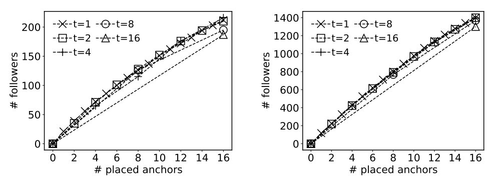


(a) SO, varying $\alpha$ (b) AZ, varying $\alpha$ (c) DE, varying $\alpha$ (d) SO, varying $\beta$ (e) AZ, varying $\beta$ (f) DE, varying $\beta$

(g) SO, varying $b_1$ (h) AZ, varying $b_1$ (i) DE, varying $b_1$ (j) SO, varying $b_2$ (k) AZ, varying $b_2$ (l) DE, varying $b_2$

Fig. 9: Effect of $\alpha$, $\beta$, $b_1$, and $b_2$

**Evaluate the effect of budgets $b_1$ and $b_2$.** The second row of Fig. 9 shows the performances of FILVER, FILVER$^+$, and FILVER$^{++}$ when budgets $b_1$ and $b_2$ vary from 5 to 25 on SO, AZ, and DE. We can see that as $b_1$ or $b_2$ increases, the running time of the three algorithms increases. This is because more budgets result in more iterations for all these algorithms, which drive up the time cost.



(a) WC, varying $t$ (b) DB, varying $t$

Fig. 10: Effect of $t$ on the number of followers

TABLE III: Effect of $t$ on the running time

| $t$ | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| WC | 65.58 | 33.68 | 19.67 | 12.56 | 7.19 |
| DB | 5997.72 | 3434.09 | 1900.85 | 1271.16 | 586.10 |

**Evaluate the effect of $t$.** We evaluate the effect of $t$ (i.e., the number of anchors chosen per iteration) in FILVER$^{++}$ on the datasets WC and DB. Here $t$ takes on the values of 1, 2, 4, 8 and 16. $\alpha = 0.6\delta$, $\beta = 0.4\delta$ and $b_1 = b_2 = 8$. For FILVER$^{++}$ with different $t$, Fig. 10 depicts the accumulating number of followers as the number of placed anchors approaches $b_1 + b_2$. We can observe that when $t$ is small ($t < 8$), FILVER$^{++}$ and FILVER$^+$ generate similar numbers of followers. When $t$ approaches $b_1 + b_2$, FILVER$^{++}$ yields slightly fewer followers than FILVER$^+$. The running time of FILVER$^+$ and FILVER$^{++}$ are reported in Table III. As expected, FILVER$^{++}$ incurs smaller running time because it places $t$ ($t > 1$) anchors in each iteration, while FILVER$^+$ only finds one. As $t$ increases, the running time of FILVER$^{++}$ decreases because more anchors placed per iteration means fewer iterations.

## VII. Related Work

Below we review related works of network reinforcement and cohesive subgraph models on bipartite graphs.

**Network reinforcement.** The motivation for network reinforcement goes back to the literature of engagement dynamics of networks [32], [10]. Bhawalkar and Kleinberg [10] observe the phenomenon of network unraveling and introduce the anchored *k*-core model to for network reinforcement. Efficient solutions are proposed to solve this problem [11], [12]. Following the anchored *k*-core model, many similar problems are studied, such as the anchored and collapsed coreness problem [13], [16], [4], anchored *k*-truss problem [33], collpased *k*-core and *k*-truss problem [34], anchored vertex exploration [30], budget minimization for anchored *k*-core [15], and *k*-core maximization by edge addition [14]. The algorithms proposed in these works focus on unipartite graphs and are not suitable to reinforce bipartite networks. A recent work [35] studies collapsed $(\alpha, \beta)$-core problem on bipartite graphs, which aims to minimize the size of the $(\alpha, \beta)$-core by removing a set of edges from it. However, it focuses on the edges inside the $(\alpha, \beta)$-core instead of actively expanding it.

**Cohesive subgraph models on bipartite networks.** On bipartite networks, various cohesive subgraph models are proposed. Extended from *k*-core, $(\alpha, \beta)$-core [17], [18], [19], [36] uses different degree constraints to ensure the engagement levels of the vertices of different layers. The $k$-bitruss model is proposed [37], [38], [39] which requires each edge in the subgraph is contained in at least $k$ butterflies. In addition, biclique [40], [41] is a well-known cohesive subgraph model which is a complete bipartite graph. However, existing works on bipartite graphs mainly focus on the efficient computation of the cohesive subgraphs rather than network reinforcement.

## VIII. Conclusion

In this paper, we study the anchored $(\alpha, \beta)$-core problem to reinforce bipartite networks. Due to the NP-hardness of the problem, we resort to greedy heuristics. By considering the vertex deletion orders during $(\alpha, \beta)$-core computation, we propse a filter-verification framework to improve efficiency. We further push the efficiency boundary with optimizations in both filter and verification stages. Extensive experiments on 16 real-world graphs and one synthetic graph at billion-scale verify the effectiveness of the proposed techniques.

## References

[1] J. Wang, A. P. De Vries, and M. J. Reinders, "Unifying user-based and item-based collaborative filtering approaches by similarity fusion," in *SIGIR*. ACM, 2006, pp. 501–508.

[2] C. M. O'Connor, J. U. Adams, and J. Fairman, "Essentials of cell biology," *Cambridge, MA: NPG Education*, vol. 1, p. 54, 2010.

[3] J. C. Brunson, "Triadic analysis of affiliation networks," *arXiv preprint arXiv:1502.07016*, 2015.

[4] F. Zhang, J. Xie, K. Wang, S. Yang, and Y. Jiang, "Discovering key users for defending network structural stability," *World Wide Web*, pp. 1–23, 2021.

[5] F. Morone, G. Del Ferraro, and H. A. Makse, "The k-core as a predictor of structural collapse in mutualistic ecosystems," *Nature physics*, vol. 15, no. 1, pp. 95–102, 2019.

[6] R. M. Dewan, M. L. Freimer, and J. Zhang, "Management and valuation of advertisement-supported web sites," *Journal of Management Information Systems*, vol. 19, no. 3, pp. 87–98, 2002.

[7] R. Benbunan-Fich and E. M. Fich, "Effects of web traffic announcements on firm value," *International Journal of Electronic Commerce*, vol. 8, no. 4, pp. 161–181, 2004.

[8] H. Sun, J. Chen, and M. Fan, "Effect of live chat on traffic-to-sales conversion: Evidence from an online marketplace," *Production and Operations Management*, 2021.

[9] X. Huang, I. Vodenska, S. Havlin, and H. E. Stanley, "Cascading failures in bi-partite graphs: model for systemic risk propagation," *Scientific reports*, vol. 3, p. 1219, 2013.

[10] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma, "Preventing unraveling in social networks: the anchored k-core problem," *SIAM Journal on Discrete Mathematics*, vol. 29, no. 3, 2015.

[11] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin, "Olak: an efficient algorithm to prevent unraveling in social networks," *PVLDB*, vol. 10, no. 6, 2017.

[12] R. Laishram, A. Erdem Sar, T. Eliassi-Rad, A. Pinar, and S. Soundarajan, "Residual core maximization: An efficient algorithm for maximizing the size of the k-core," in *SIAM*. SIAM, 2020, pp. 325–333.

[13] Q. Linghu, F. Zhang, X. Lin, W. Zhang, and Y. Zhang, "Global reinforcement of social networks: The anchored coreness problem," in *SIGMOD*, 2020.

[14] Z. Zhou, F. Zhang, X. Lin, W. Zhang, and C. Chen, "K-core maximization: An edge addition approach," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 2019, pp. 4867–4873.

[15] K. Liu, S. Wang, Y. Zhang, and C. Xing, "An efficient algorithm for the anchored k-core budget minimization problem," in *ICDE*. IEEE, 2021, pp. 1356–1367.

[16] Q. Linghu, F. Zhang, X. Lin, W. Zhang, and Y. Zhang, "Anchored coreness: efficient reinforcement of social networks," *The VLDB Journal*, pp. 1–26, 2021.

[17] A. Ahmed, V. Batagelj, X. Fu, S.-H. Hong, D. Merrick, and A. Mrvar, "Visualisation and analysis of the internet movie database," in *2007 6th International Asia-Pacific Symposium on Visualization*. IEEE, 2007.

[18] D. Ding, H. Li, Z. Huang, and N. Mamoulis, "Efficient fault-tolerant group recommendation using alpha-beta-core," in *CIKM*. ACM, 2017.

[19] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient ($\alpha$, $\beta$)-core computation: an index-based approach," in *WWW*, 2019.

[20] D. Garcia, P. Mavrodiev, and F. Schweitzer, "Social resilience in online communities: The autopsy of friendster," in *Proceedings of the first ACM conference on Online social networks*, 2013, pp. 39–50.

[21] K. Seki and M. Nakamura, "The collapse of the friendster network started from the center of the core," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2016, pp. 477–484.

[22] G. A. Pavlopoulos, P. I. Kontou, A. Pavlopoulou, C. Bouyioukos, E. Markou, and P. G. Bagos, "Bipartite graphs in systems biology and medicine: a survey of methods and applications," *Gigascience*, vol. 7, no. 4, p. giy014, 2018.

[23] J. Bascompte and P. Jordano, "Plant-animal mutualistic networks: the architecture of biodiversity," *Annu. Rev. Ecol. Evol. Syst.*, vol. 38, 2007.

[24] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972.

[25] U. Feige, "A threshold of ln n for approximating set cover," *JACM*, vol. 45, no. 4, 1998.

[26] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.

[27] V. Batagelj and M. Zaversnik, "An o (m) algorithm for cores decomposition of networks," *arXiv preprint cs/0310049*, 2003.

[28] Z. Lin, F. Zhang, X. Lin, W. Zhang, and Z. Tian, "Hierarchical core maintenance on large dynamic graphs," *PVLDB*, vol. 14, no. 5, pp. 757–770, 2021.

[29] Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin, "A fast order-based approach for core maintenance," in *ICDE*. IEEE, 2017, pp. 337–348.

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

[31] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *WWW*, 2005.

[32] F. D. Malliaros and M. Vazirgiannis, "To stay or not to stay: modeling engagement dynamics in social graphs," in *CIKM*, 2013.

[33] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "Efficiently reinforcing social networks over user engagement and tie strength," in *ICDE*. IEEE, 2018, pp. 557–568.

[34] F. Zhang, C. Li, Y. Zhang, L. Qin, and W. Zhang, "Finding critical users in social communities: The collapsed core and truss problems," *TKDE*, 2018.

[35] C. Chen, Q. Zhu, Y. Wu, R. Sun, X. Wang, and X. Liu, "Efficient critical relationships identification in bipartite networks," *World Wide Web*, pp. 1–21, 2021.

[36] Y. He, K. Wang, W. Zhang, X. Lin, and Y. Zhang, "Exploring cohesive subgraphs with vertex engagement and tie strength in bipartite graphs," *Information Sciences*, vol. 572, pp. 277–296, 2021.

[37] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Efficient bitruss decomposition for large-scale bipartite graphs," in *ICDE*. IEEE, 2020.

[38] Z. Zou, "Bitruss decomposition of bipartite graphs," in *DASFAA*. Springer, 2016.

[39] A. E. Sarıyüce and A. Pinar, "Peeling bipartite networks for dense subgraph discovery," in *WSDM*. ACM, 2018, pp. 504–512.

[40] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou, "Maximum biclique search at billion scale." *PVLDB*, vol. 13, no. 9, 2020.

[41] S. Lehmann, M. Schwartz, and L. K. Hansen, "Biclique communities," *Physical review E*, vol. 78, no. 1, p. 016108, 2008.