Quantifying Node Importance over Network Structural Stability

Fan Zhang
Guangzhou University
fanzhang.cs@gmail.com

Qingyuan Linghu
University of New South Wales
q.linghu@unsw.edu.au

Jiadong Xie
Chinese University of Hong Kong
xiejiadong0623@gmail.com

Kai Wang
Antai College of Economics and Management, Shanghai Jiao Tong University
cskaelwang@gmail.com

Xuemin Lin
Antai College of Economics and Management, Shanghai Jiao Tong University
xuemin.lin@sjtu.edu.cn

Wenjie Zhang
University of New South Wales
zhangw@cse.unsw.edu.au

ABSTRACT

Quantifying node importance on engagement dynamics is critical to support network stability. We can motivate or retain the users in a social platform according to their importance s.t. the network is more sustainable. Existing studies validate that the *coreness* of a node is the "best practice" on network topology to estimate the engagement of the node. In this paper, the importance of a node is the effect on the engagement of other nodes when its engagement is strengthened or weakened. Specifically, the importance of a node is quantified via two novel concepts: the *anchor power* to measure the engagement effect of node strengthening (i.e., the overall coreness gain) and the *collapse power* to measure the engagement effect of node weakening (i.e., the overall coreness loss). We find the computation of the two concepts can be naturally integrated into a shell component-based framework, and propose a unified static algorithm to compute both the anchored and collapsed followers. For evolving networks, efficient maintenance techniques are designed to update the follower sets of each node, which is faster than redoing the static algorithm by around 3 orders of magnitude. Extensive experiments on real-life data demonstrate the effectiveness of our model and the efficiency of our algorithms.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**.

KEYWORDS

Core decomposition; Node importance; Network stability

ACM Reference Format:
Fan Zhang, Qingyuan Linghu, Jiadong Xie, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2023. Quantifying Node Importance over Network Structural Stability . In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599480

KDD '23, August 6–10, 2023, Long Beach, CA, USA.

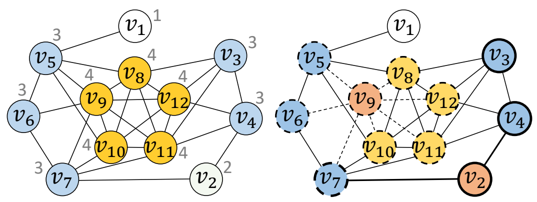
ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599480

1 INTRODUCTION

The structural stability of a network indicates the ability of the network to maintain a sustainable service and/or to defend the attacks from competitors. The leave of some nodes (aka the vertices or the users) in a network can be contagious, causing the continuous departure of other affected nodes and eventually breaking the stability of the network, e.g., Friendster is suspended due to engagement decline [17, 49]. Thus, it is essential to quantify the importance of each node in a network and enhance the engagement of the nodes according to their importance data [31, 38].

Complex networks are naturally modeled as graphs, in which the k-core is defined as a (connected) maximal subgraph with each vertex having at least k neighbors (i.e., a degree of at least k) in the subgraph [39, 48]. Accordingly, every vertex has a unique *coreness* value: the largest k s.t. the k-core contains the vertex. Given a graph, *core decomposition* [2] iteratively removes every vertex with the smallest degree in current graph s.t. the k-cores and the coreness of each node can be computed. Existing works often adopt the k-core model on capturing node engagement, e.g., [3, 9, 31, 36, 38, 43, 54–57], because its degeneration property can naturally capture the engagement dynamics in real-life networks. That is, the iterative removal of the nodes with the smallest degree well models the leaving sequence of users in the decline of a network. In [38], the coreness of a node is first demonstrated as the "best practice" on network structure for estimating its engagement. In [31], the coreness of a node is validated as positively correlated with its engagement data (its number of check-ins) in the Gowalla network. Thus, the structural stability of a network can be expressed by the coreness aggregation in the network.

Nevertheless, a node with high engagement is not certainly important for sustaining network stability, since motivating or protecting such a node may have a small effect on the engagement of other nodes. Thus, for node strengthening, the importance of a node u is quantified by its *anchor power*, i.e., the coreness gain of all other nodes if u is anchored (the degree of u is regarded as $+\infty$ and it will not be deleted in any batch of core decomposition) [3]. We may give different incentives to the users according to their anchor powers to motivate overall user engagement. Besides, for node weakening, the importance of a node u is quantified by its *collapse power*, i.e., the coreness loss of all other nodes if u is collapsed (the degree of u is regarded as 0 and it will be deleted in the first batch of core decomposition) [56]. The leave of the users with different collapse powers would break the network stability to



(a) Vertex Coreness (b) Anchor v_2 / Collapse v_9

Figure 1: Node Engagement (a) and Node Importance (b)

different extents. We should protect (encourage) the participation of the users accordingly. The two powers are on different views: applying the anchor power is to actively strengthen the engagement of users while applying the collapse power is to protect the user engagement from a defense view. To further motivate the study, the performance of the above models on graph structure is validated with real-life node importance data (Section 3).

Node motivation/protection by node importance can be applied in various domains such as social networks, financial networks, computer networks, and ecological networks. For instance, in marketing campaigns across different social platforms, we can offer incentives to users according to their importance, such as rewarding user interactions. We may also improve/sustain the stability of a target community if we regard it as a network.

Example 1.1. Figure 1a shows a graph of 12 vertices and their connections, the coreness of each vertex is marked near the vertex, e.g., the coreness of v_9 is 4. The k-core is induced by all the vertices with coreness of at least k, e.g., the 3-core is induced by $v_3, v_4, ..., v_{11}$ and v_{12}. We can see a vertex with a higher coreness is better engaged in the graph. For node importance, as Figure 1b shows, the user v_2 has 3 anchored followers that are v_3, v_4 and v_7 (coreness increased by anchoring v_2, marked in bold circles). The user v_9 has 7 collapsed followers that are $v_5, v_6, v_7, v_8, v_{10}, v_{11}$ and v_{12} (coreness decreased by collapsing v_9, marked in dashed circles). As the numbers of followers are relatively large, v_2 and v_9 are more important regarding anchor power and collapse power, respectively.

The anchor power (resp. collapse power) of a node u equals the number of *anchored followers* (resp. *collapsed followers*) of u, where a follower of u is a node with coreness changed for anchoring u (resp. collapsing u). It is because the anchoring/collapsing of one node can only change the coreness of another node by at most 1 [31, 54]. Thus, we aim to investigate and efficiently compute the importance of each node regarding its follower sets. Besides, many real networks are evolving (e.g., new friend relations, new web links, etc) [28, 30, 35] and the change of anchor/collapse power can be drastic for a simple graph update. Thus, we also aim to efficiently maintain the follower sets for each node against graph dynamics.

Challenges. To the best of our knowledge, no existing work focuses on the importance study of every single node for network stability. Core maintenance [58] is a streaming algorithm to update the coreness value of each vertex, after an edge is inserted into or removed from the graph. The computation of anchored/collapsed followers can be simulated to core maintenance by adding edges to the anchored vertex or removing edges from the collapsed vertex. However, it is cost-prohibitive because one execution of core maintenance can compute at most one follower set and we have to repeatedly execute core decomposition many times.

Recent works on user engagement study aim to find the optimal combination of b anchored vertices [31] or b collapsed vertices [54] and focus on pruning unpromising combinations, where b is a given budget. Our problem is budget-free as we compute the importance of every single user, which is favored in the applications regarding motivating/protecting a large percentage or all of the users in a network. Different actions may be applied to the users with different importance levels on network stability. Our problem also addresses the scenarios when there is no specified budget value or the budget is not fixed with time evolves. The advanced algorithms in above studies can be adapted as a baseline solution (Algorithm 1) to compute the follower set of every vertex on a static graph, while this is still not efficient enough on large graphs.

Regarding graph dynamics, after an edge insertion or removal, both the coreness values and the anchor/collapse powers of many vertices may change. As validated in our preliminary experiment, a large portion of the vertices for anchoring/collapsing in the graph will have their follower sets changed after a random removal of 1000 edges (up to 13% on the datasets in Section 6). Thus, it is non-trivial to locate the candidate vertices for updating their anchored/collapsed followers, and we need to carefully track the importances of all the vertices when the network evolves.

Our Solution. To address the challenges, we propose a novel framework to compute and maintain the follower sets of each vertex. We first divide the graph into multiple *shell components*[1] formed by all the maximal connected subgraphs where each vertex has the same coreness. Thus, a follower set consists of some vertex subsets from the shell components. For every vertex as a follower, the candidate sets of its anchored/collapsed vertices can be tracked oppositely. Therefore, we enumerate each shell component and compute the followers in the component for each candidate anchored/collapsed vertex (Algorithm 4) s.t. the proposed algorithm is more efficient and has a lower time complexity compared with the baseline.

We also propose a novel maintenance algorithm to efficiently update the follower sets of every vertex. When an edge is inserted into or removed from the graph, we first apply core maintenance [58] to update the coreness of each vertex. Then, the new induced shell components are carefully collected, and the follower sets of any vertex can be updated only based on the new shell components, where the scale of new shell components for one inserted or removed edge is often small. Due to the independent follower computation of each candidate on the corresponding shell components, any parallel architecture can be utilized to accelerate both the static and the dynamic algorithms.

Contributions. The main contributions of this paper are as follows.

1) We motivate the anchor/collapse power on quantifying node importance over network structural stability by validating its performance on real data (Section 3).

2) We formally prove that the candidate sets for anchoring or collapsing can be located in fine granularity, where the follower sets can be independently computed on each shell component and efficiently parallelized (Section 5.1).

[1]The shell components are more fine-grained than the core tree used in existing studies [30, 31, 39, 46, 47, 54] where a tree node may correspond to multiple shell components.

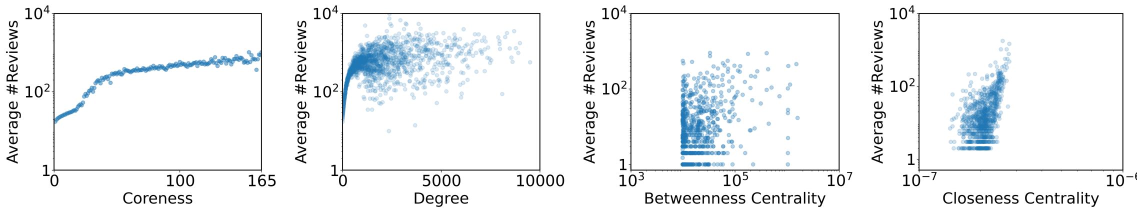
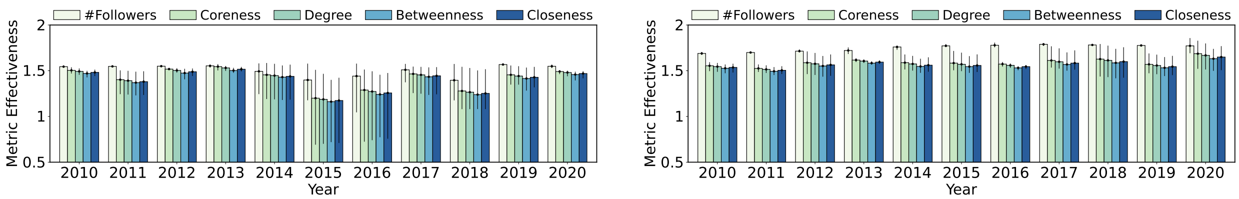


Figure 2: Different Ranking Metrics v.s. Ground-truth Node Engagement (#Reviews in Yelp)



(a) Importance of Anchored Users (Top 100 / The Rest)

(b) Importance of Collapsed Users (Top 100 / The Rest)

Figure 3: Effectiveness of Different Metrics by Relative Importance of the Selected Users (Dynamic of #Reviews in Yelp)

3) A dynamic algorithm is proposed to update the followers of each node against edge insertion/removal (Section 5.2). We also optimize the follower computation of one node (Section 5.3).

4) The experiments conducted on 9 real-life datasets validate that the model of anchor/collapse power is effective and our proposed algorithms are efficient (Section 6).

2 RELATED WORK

The k-core [39, 48] is studied in various areas, e.g., community discovery [13–15, 26, 27], spreader identification [20, 29, 37, 51], and the applications of biology and ecology [1, 4, 43]. Core decomposition algorithms are also studied under different computation environments [2, 8, 42, 53]. Network robustness metrics are surveyed in [16, 33, 44] where the focuses are different from our model, e.g., the centrality measures are built on information flow [10, 24, 41, 52] while our anchor/collapse power is based on user engagement. A node with a large centrality may be influential in the information cascade while a node with a large anchor/collapse power is important in sustaining the overall user engagement of a network. As validated later, the coreness-based metrics can well match the ground-truth node importance.

The anchor problems aim to find and enhance a small set of nodes to improve coreness aggregation level [3, 31, 32]. The collapse problems hold a different view where a set of nodes are protected to sustain a certain level of coreness aggregation [54, 56]. Anchoring is also studied in motivating the smallest set of users s.t. an improvement quota on stability is satisfied [34]. A variant of coreness loss is to delete a user set leading to the maximum number of coreness-changed users [12]. Besides, edge manipulation is considered on network stability which adds promising new edges or protects key existing links, e.g., [6, 23, 40, 50, 59, 60]. Different to above network-level study, some works focus on the stability problems on communities, e.g., [5, 7]. Nevertheless, none of the above works explore the importance of every node in a network and study its efficient computation on both static and dynamic graphs.

3 ANALYSIS OF REAL DATA

To better motivate the study, we analyze the real data from Yelp [11] to check whether the coreness of a node can well estimate its engagement level and whether the anchor/collapse power of a node can well match the importance of the node on sustaining overall user engagement. The Yelp data contains a social network G including the users (vertices) and their friend relations (edges) and the reviews (for Yelp POIs) written by the users with timestamps. The evaluation on Brightkite data is in Appendices.

Case Study on Node Engagement. We first analyze the correlation between node engagement and different vertex ranking metrics on the structure of G including coreness, degree, betweenness centrality [18] and closeness centrality [10]. The *ground-truth engagement* of a user is represented by his/her number of reviews for all the POIs in Yelp. Figure 2 depicts the average number of reviews for the vertices with the same coreness (resp. degree, betweenness, or closeness). Since computing the betweenness or closeness centrality is costly, we randomly sample 1000 vertices to show their correlation with node engagement. For degree and coreness, we compute on all the vertices. We find that the engagement values may differ a lot for the vertices with close degrees (resp. betweenness or closeness values), while coreness is clearly in a positive correlation with node engagement. It is because core decomposition well models the leaving sequence of users in the degeneration of a network. The result of coreness is also better than other metrics, e.g., the variants of centrality metrics [16, 33, 44]. The outperformance is similar if we normalize the scores to the same granularity for every metric. Therefore, for networks with the same scale (decided by the number of vertices), a network with a larger coreness sum of all the nodes is considered more stable on structure[2].

Case Study on Node Importance. We further validate the match of anchor/collapse power (i.e., the number of anchored/collapsed followers) and node importance. To find the real importance data,

[2]For networks with different scales, it is more appropriate to compare the average coreness of the nodes.

we need to check the behavior of real anchored/collapsed users. Thus, for every two consecutive months in the Yelp data, we say a user is an *anchored user* (resp. *collapsed user*) if his/her number of reviews in the latter month is larger (resp. smaller) than the former month. Recall that the importance of a node is quantified by the effect on the engagement of other nodes if its engagement is strengthened/weakened, i.e., the engagement dynamic of its anchored/collapsed followers. Therefore, the *ground-truth importance* of an anchored user (resp. collapsed user) is measured by the average variation of review numbers of his/her anchored (resp. collapsed) followers between two consecutive months.

Then, we evaluate the anchored/collapsed users for each metric, i.e., the number of anchored followers, the number of collapsed followers, coreness, degree, betweenness, and closeness, respectively. For each metric, the first group contains the top 100 users[3] according to their scores on the metric (e.g., the corenesses), and the second group contains the rest users (793 users on average). The *metric effectiveness* (*ME*) of a metric is measured by the gap of overall node importance between the users selected by the metric and the other users, i.e., the average ground-truth importance of the first group divided by that of the second group.

As the node importance is based on the data on consecutive months (the engagement dynamic of the followers), we report the *ME* of each metric for all the consecutive months. Figure 3 shows the *ME* from different metrics where each bar is the average value from all consecutive months during one year and the error bar shows the variance of *ME* from every month. Note that the anchor/collapse power (i.e., #Followers) performs the best for *every* two consecutive months among the 11-year real data. It shows that, for strengthening (resp. weakening) the users with large anchor (resp. collapse) powers, the effect on other users (i.e., the importance) is larger than strengthening (resp. weakening) the users from other metrics. Note that the result of anchor/collapse power is also better than other variants, e.g., authority/hub centrality [21] on bidirectional graphs. Thus, the anchor/collapse power is superior for quantifying node importance on network structural stability.

4 PRELIMINARIES

We consider an unweighted and undirected graph $G = (V, E)$. The notations are summarized in Table 1. We may omit G in notations when the context is clear, e.g., using $deg(u)$ instead of $deg(u, G)$.

Definition 4.1 (k-core[4]). Given a graph G and a positive integer k, a subgraph G' is the k-core of G, denoted by $C_k(G)$, if (*i*) G' satisfies the degree constraint, i.e., $deg(u, G') \geq k$ for each $u \in V(G')$; and (*ii*) G' is maximal, i.e., any supergraph $G'' \supset G'$ is not a k-core.

Given two integers k and k' with $k \geq k'$, the k-core is always a subgraph of the k'-core, i.e., $C_k(G) \subseteq C_{k'}(G)$. In addition, each vertex in G has a unique coreness value.

Definition 4.2 (coreness). Given a graph G, the coreness of a vertex $u \in V(G)$, denoted by $c(u, G)$, is the largest k such that $C_k(G)$ contains u, i.e., $c(u, G) = max\{k \mid u \in C_k(G)\}$.

[3]As the reviews with timestamps are "recommended reviews" selected by Yelp [11], the average number of anchored/collapsed users in one month is only 893.

[4]In this paper, we do not require the k-core be connected as in [39, 48] and use k-core to represent all the connected subgraphs satisfying Definition 4.1.

Table 1: Summary of Notations

Notation	Definition
G	an unweighted and undirected graph
$V(G); E(G)$	the vertex set of G; the edge set of G
$n; m$	$\|V(G)\|$; $\|E(G)\|$ (assume $m > n$)
$N(u, G)$	the set of neighbors of u in G
$deg(u, G)$	$+\infty$ if u is anchored, 0 if u is collapsed, and $\|N(u, G)\|$ otherwise
$C_k(G)$	the k-core of G
$c(u, G)$	the original coreness of u in G
$c^+_x(u, G); c^-_x(u, G)$	the coreness of u in G with anchoring/collapsing x
$^+\mathcal{F}(x, G); {}^-\mathcal{F}(x, G)$	the anchored/collapsed follower set of x in G
$H_k(G)$	the k-shell of G
$\mathcal{SC}[v]$	the only shell component containing v
$S; S.V; S.E; S.c$	a shell component and its vertex set, edge set and coreness value, respectively
$\mathcal{A}[S]; C[S]$	the anchor/collapser candidate set of S
$^+F[x][S]; {}^-F[x][S]$	the anchored/collapsed follower set of x in S

Given a graph G, *core decomposition* is to compute the coreness for each vertex $v \in V(G)$, which can be computed in $O(m)$ time by recursively deleting the vertex with the smallest degree in G [19].

In this paper, once a vertex x is *anchored*, its degree is regarded as positive infinity (i.e., $deg(x, G) = +\infty$); once a vertex x is *collapsed*, its degree is regarded as zero (i.e., $deg(x, G) = 0$). Note that the existence of anchored/collapsed vertices does not change the neighbor set of any vertex. An anchored vertex is also called an *anchor*, and a collapsed vertex is also called a *collapser*, respectively.

In core decomposition, an anchored vertex will not be removed as its degree is positive infinity; for every collapsed vertex, it will be deleted at the first iteration of core decomposition as its degree is zero. Thus, anchoring (resp. collapsing) a vertex may increase (resp. decrease) the corenesses of other vertices.

When a vertex x is anchored, we use $c^+_x(u, G)$ to denote the coreness of u in G with $deg(x, G) = +\infty$. A vertex with coreness increased is called the *anchored follower* of x.

Definition 4.3 (anchored follower set). Given a graph G and an anchor vertex x, the anchored follower set of x in G is denoted by $^+\mathcal{F}(x, G)$, formed by every vertex except x with its coreness increased after anchoring x in G, i.e., $^+\mathcal{F}(x, G) = \{u \in V(G) \mid u \neq x \wedge c^+_x(u, G) > c(u, G)\}$.

When a vertex x is collapsed, we use $c^-_x(u, G)$ to denote the coreness of u in G with $deg(x, G) = 0$. A vertex with coreness decreased is called the *collasped followers* of x.

Definition 4.4 (collapsed follower set). Given a graph G and a collapser vertex x, the collapsed follower set of x in G is denoted by $^-\mathcal{F}(x, G)$, formed by every vertex except x with its coreness decreased after collapsing x in G, i.e., $^-\mathcal{F}(x, G) = \{u \in V(G) \mid u \neq x \wedge c^-_x(u, G) < c(u, G)\}$.

In this paper, we aim to efficiently compute/maintain the anchored/collapsed follower sets for each vertex to capture its node importance over network structural stability.

Problem Statement. Given a graph G, for each $v \in V(G)$, we compute $^+\mathcal{F}(v, G)$ and $^-\mathcal{F}(v, G)$. When an edge is inserted into or removed from G, for each $v \in V(G)$, we maintain $^+\mathcal{F}(v, G)$ and $^-\mathcal{F}(v, G)$.

Algorithm 1: **Baseline(G)**

Input : G : the graph
Output : ${}^{+}\mathcal{F}(v,G)$ and ${}^{-}\mathcal{F}(v,G)$ for each $v \in V(G)$

1 $c(u) \leftarrow$ the coreness of each $u \in V(G)$;
2 **for** each $u \in V(G)$ in parallel **do**
3   Compute ${}^{+}\mathcal{F}(u)$ by Algorithm 4 of [31];
4 **for** each $u \in V(G)$ with at least 1 follower [54] in parallel **do**
5   Compute ${}^{-}\mathcal{F}(u)$ by Algorithm 3 of [54];
6 **return** ${}^{+}\mathcal{F}(v)$ and ${}^{-}\mathcal{F}(v)$ for each $v \in V(G)$

Baseline Algorithm. The baseline algorithm is based on state-of-the-art algorithms on computing/maintaining the coreness of every vertex and computing the anchored/collapsed followers. Algorithm 1 shows the pseudo-code of the baseline. We first compute the coreness of each vertex, by core decomposition [19] in the static scenario, or by core maintenance in the dynamic scenario (Line 1). Then, we compute the anchored follower sets for each vertex u (Lines 2-3) by Algorithm 4 of [31]. We enumerate each vertex with at least 1 collapsed follower (Lines 4-5) and compute by Algorithm 3 of [54] to find collapsed followers. Algorithm 1 can be easily parallelized since the computation on each vertex from Line 2 or Line 4 is independent.

As the worst-case time cost of Algorithm 4 in [31] is $O(m)$, and Algorithm 3 in [54] is $O(d_{max} \cdot m)$, the time complexity of Algorithm 1 is $O(n \cdot d_{max} \cdot m)$ where d_{max} is the largest vertex degree in G. However, the baseline algorithm is still costly since it has redundant computations when searching for followers. Besides, for dynamic graphs, it needs to re-compute the followers for all the vertices, which is cost-prohibitive.

5 OUR SOLUTION

We first introduce the shell component used in our algorithms.

Definition 5.1 (k-shell). Given a graph G and a positive integer k, the k-shell, denoted by $H_k(G)$, is the set of vertices in G with their corenesses exactly equal to k, i.e., $H_k(G) = V(C_k(G)) \setminus V(C_{k+1}(G))$.

Definition 5.2 (shell component). Given a graph G and the k-shell $H_k(G)$, a subgraph S is a shell component of $H_k(G)$, if S is a maximal connected component of $G[H_k(G)]$.

A k-shell is formed by the vertices of a series of non-overlapping shell components. For each vertex in G, there exists only one shell component S containing v. Besides, in core decomposition, for a given integer k, the deletion sequence of the shell components of $H_k(G)$ can be arbitrary.

Example 5.3. In Figure 4a, we have $H_1(G) = \{v_1\}$, $H_2(G) = \{v_2\}$, $H_3(G) = \{v_3, v_4, v_5, v_6, v_7\}$ and $H_4(G) = \{v_8, v_9, v_{10}, v_{11}, v_{12}\}$. As circled in the figure, we have 5 shell components from S_1 to S_5, e.g., S_3 and S_4 are two shell components of $H_3(G)$. In core decomposition of G, the deletion sequence of shell components can be either $(S_1, S_2, S_3, S_4, S_5)$ or $(S_1, S_2, S_4, S_3, S_5)$.

For a shell component S of $H_k(G)$, we denote $S.V$, $S.E$ and $S.c$ as the vertex set, edge set and the coreness of the vertices in S, i.e., $S.V = V(S)$, $S.E = E(S)$ and $S.c = c(v,G)$ for any $v \in S.V$. We use the structure $\mathcal{SC}$ to index the shell components for all the vertices. For each $v \in V(G)$, $\mathcal{SC}[v]$ is the only shell component with $v \in \mathcal{SC}[v].V$.

Algorithm 2: **ShellDecomp(G)**

Input : G : the graph
Output : $\mathcal{SC}$: the index of shell components in G

1 $c(u,G)$ of each $u \in V(G)$ by core decomposition [19];
2 **for** each *unassigned* $u \in V(G)$ **do**
3     $S \leftarrow$ a new shell component;
4     $S.c \leftarrow c(u,G)$; $S.V \leftarrow S.V \cup \{u\}$;
5   u is set *assigned*; **ShellConnect**(u, S, $\mathcal{SC}$);
6     $\mathcal{SC}[u] \leftarrow S$;
7 **return** $\mathcal{SC}$

Algorithm 3: **ShellConnect(u, S, $\mathcal{SC}$)**

Input : u : a vertex, S : the shell component containing u, $\mathcal{SC}$: the shell component index

1 **for** each $v \in N(u,G)$ with $c(v) = c(u)$ **do**
2     $S.E \leftarrow S.E \cup \{(u,v)\}$;
3   **if** v is *unassigned* **then**
4         $S.V \leftarrow S.V \cup \{v\}$;
5   v is set *assigned*; **ShellConnect**(v, S, $\mathcal{SC}$);
6         $\mathcal{SC}[v] \leftarrow S$;

Shell Component Computation. The shell components can be computed in $O(m)$ time by traversing the graph by a constant number of times. Algorithms 2 and 3 illustrate the process of decomposing each vertex into its shell component.

In Algorithm 2, firstly we need to conduct core decomposition [19] on G to get the coreness of each vertex (Line 1). We traverse all the vertices with each vertex marked *unassigned* by default (Line 2). Each time meeting an *unassigned* vertex $u \in V(G)$, we create a new shell component S (Line 3), set the related domains of S and set u as *assigned* (Lines 4-5). Then we call Algorithm 3 to recursively collect all the vertices which should exist in S (Line 5). After that, we set $\mathcal{SC}[u]$ by S (Line 6). When all the vertices are set *assigned* (by Algorithms 2 or 3), we can get $\mathcal{SC}$.

In Algorithm 3, for the vertex u, we traverse each neighbor $v \in N(u,G)$. If $c(v) = c(u)$, we add the edge (u,v) into $S.E$ (Line 2). Note that (u,v) and (v,u) are the same in our setting. Only if v is *unassigned* (Line 3), we add v into $S.V$ and recursively call Algorithm 3 to find all the vertices of S (Lines 4-6).

5.1 Static Follower Computation

Theorems for Candidate Sets. For each shell component, its candidate anchor/collapser sets are limited by the following theorems.

Lemma 5.4. *If a vertex x is anchored in G, the coreness of any $u \in V(G) \setminus \{x\}$ will not decrease and may increase by at most 1 [31].*

Lemma 5.5. *If a vertex x is collapsed in G, the coreness of any $u \in V(G) \setminus \{x\}$ will not increase and may decrease by at most 1 [54].*

We define two affiliated structures for each shell component S, denoted by the anchor candidate set $\mathcal{A}[S]$ and the collapser candidate set $C[S]$. The anchor candidate set (resp. collapser candidate

Algorithm 4: **StaticFollowerComputation**(G)

Input : G : the graph
Output : ${}^{+}\mathcal{F}(v,G)$ and ${}^{-}\mathcal{F}(v,G)$ for each $v \in V(G)$

1 $\hat{\mathcal{S}}, \mathcal{A}[\cdot], C[\cdot] \leftarrow$ **ShellDecomp**(G);
2 **for** each pair (v, S) in parallel with $v \in \mathcal{A}[S] \cup C[S]$ and $S \in \hat{\mathcal{S}}$ **do**
3   **if** $v \in \mathcal{A}[S]$ **then**
4     ${}^{+}F[v][S] \leftarrow$ **FindAnchoredFollowers**(v, S);
5     ${}^{+}\mathcal{F}(v,G) \leftarrow {}^{+}\mathcal{F}(v,G) \cup {}^{+}F[v][S]$;
6   **else if** $v \in C[S]$ **then**
7     ${}^{-}F[v][S] \leftarrow$ **FindCollapsedFollowers**(v, S);
8     ${}^{-}\mathcal{F}(v,G) \leftarrow {}^{-}\mathcal{F}(v,G) \cup {}^{-}F[v][S]$;
9 **return** ${}^{+}\mathcal{F}(v,G)$ and ${}^{-}\mathcal{F}(v,G)$ for each $v \in V(G)$

set) is the set of vertices that may affect the coreness of vertices in $S.V$ if they are anchored (resp. collapsed).

Definition 5.6 (anchor candidate set). Given a shell component S in graph G, the anchor candidate set of S, denoted by $\mathcal{A}[S]$, is $\mathcal{A}[S] = S.V \cup \{v \mid v \in V(G) \wedge c(v,G) < S.c \wedge N(v,G) \cap S.V \neq \emptyset\}$.

Definition 5.7 (collapser candidate set). Given a shell component S in graph G, the collapser candidate set of S, denoted by $C[S]$, is $C[S] = S.V \cup \{v \mid v \in V(G) \wedge c(v,G) > S.c \wedge N(v,G) \cap S.V \neq \emptyset\}$.

We show that all the vertices in the same shell component have the same anchor candidate set and collapser candidate set.

Theorem 5.8. *If a vertex $u \in S.V$ is an anchored follower of vertex x, we have $x \in \mathcal{A}[S]$.*

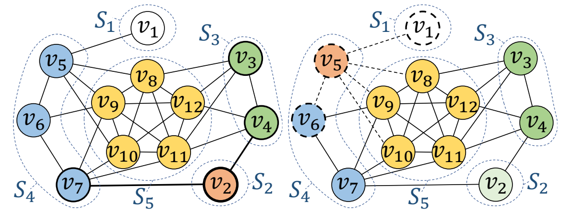
Proof. The sketch: We assume $x \notin \mathcal{A}[S]$ is a candidate anchor and $u \in S.V$ is an anchored follower of x. Then we consider the deletion sequence of vertices in core decomposition without anchoring any vertex and with anchoring x, respectively, to check the coreness change of every vertex. This will prove u is not an anchored follower of x, which contradicts our assumption. The idea also applies to the case of collapsing vertices. □

Note that the full proofs are in the appendices for all theorems.

Theorem 5.9. *If a vertex $u \in S.V$ is a collapsed follower of vertex x, we have $x \in C[S]$.*

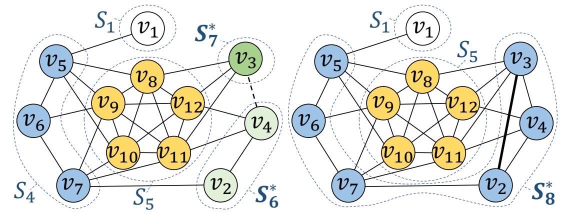
Static Follower Computation. Here we present our follower computation algorithm for static graphs in Algorithm 4. We first run Algorithm 2 to obtain all the shell components of G and record them in $\hat{\mathcal{S}}$ (Line 1). For each $S \in \hat{\mathcal{S}}$, we also get $\mathcal{A}[S]$ and $C[S]$ accordingly, which can be accumulated when traversing the neighbors of each vertex in computing the shell components.

Then, for each $v \in V(G)$ and a shell component S, we define ${}^{+}F[v][S] = S.V \cap {}^{+}\mathcal{F}(v,G)$ and ${}^{-}F[v][S] = S.V \cap {}^{-}\mathcal{F}(v,G)$ so as to compute v's followers in each atom unit, i.e., the shell component S. We know that for each shell component S, its valid anchors and collapsers (with at least one follower in S) can only from $\mathcal{A}[S]$ (Theorem 5.8) and $C[S]$ (Theorem 5.9), respectively. Thus, in Algorithm 4, we compute ${}^{+}F[v][S]$ (resp. ${}^{-}F[v][S]$) w.r.t. every shell component S in G (Lines 2-8). The followers of one vertex (Lines 4 and 7) can be computed by core decomposition with anchors/collapsers, and the optimized algorithms are given in Section 5.3.



(a) Anchor v_2 **(b) Collapse v_5**

Figure 4: Follower Computation



(a) Remove (v_3, v_4) **(b) Insert (v_2, v_3)**

Figure 5: Follower Maintenance

As the shell component does not overlap with each other, we can directly utilize shared-memory parallelization techniques to compute ${}^{+}F[v][S]$ (resp. ${}^{-}F[v][S]$) for each vertex v on each corresponding shell component S without any conflicts. Note that we apply the parallelization at the vertex level rather than the shell component level to reduce the workload imbalance.

Complexity Analysis. In Algorithm 4 , the accumulated size of $\mathcal{A}[S] \cup C[S]$ for every S in G is no larger than $O(m)$ (Line 2). The time cost of follower computation on one vertex by Line 4 or 7 is $O(|S|_{max})$ where $|S|_{max}$ is the largest size (the number of edges) of a shell component. Thus, the time complexity of Algorithm 4 is $O(m \cdot |S|_{max})$. It is lower than the $O(n \cdot d_{max} \cdot m)$ time complexity of Algorithm 1 because $|S|_{max}$ is often smaller than n and is certainly smaller than $n \cdot d_{max}$.

Example 5.10. Figure 4a shows a graph with 5 shell components. As there are 3 shell components S_2, S_3 and S_4 with $v_2 \in \mathcal{A}[S_2]$, $v_2 \in \mathcal{A}[S_3]$ and $v_2 \in \mathcal{A}[S_4]$, for the anchoring of v_2, we can compute ${}^{+}F[v_2][S_2]$, ${}^{+}F[v_2][S_3]$ and ${}^{+}F[v_2][S_4]$. Then, we have ${}^{+}F[v_2][S_2] = \emptyset$, ${}^{+}F[v_2][S_3] = \{v_3, v_4\}$ as $c^{+}_{v_2}(v_3) = c^{+}_{v_2}(v_4) = 4$, and ${}^{+}F[v_2][S_4] = \{v_7\}$ as $c^{+}_{v_2}(v_7) = 4$. So, ${}^{+}\mathcal{F}(v_2, G) = \{v_3, v_4, v_7\}$.

Example 5.11. In Figure 4b, for the collapsing of v_5, we compute ${}^{-}F[v_5][S_1]$ and ${}^{-}F[v_5][S_4]$, as $v_5 \in C[S_1]$ and $v_5 \in C[S_4]$. Since $v_5 \notin C[S_5]$, we know there is no follower of collapsing v_5 in S_5. Then, we have ${}^{-}F[v_5][S_1] = \{v_1\}$ as $c^{-}_{v_5}(v_1) = 0$, and ${}^{-}F[v_5][S_4] = \{v_6\}$ as $c^{-}_{v_5}(v_6) = 2$. So, ${}^{-}\mathcal{F}(v_5, G) = \{v_1, v_6\}$.

5.2 The Maintenance w.r.t. Edge Streaming

We consider the insertion/removal of a single edge which can also be used to handle the streaming of multiple edges and vertices. We maintain the anchored follower set ${}^{+}\mathcal{F}(v,G)$ and the collapsed follower set ${}^{-}\mathcal{F}(v,G)$ for each $v \in V(G)$ by Algorithm 5. Specifically, before inserting/removing the edge (v_s, v_t), the current $\mathcal{SC}[\cdot]$, $\mathcal{A}[\cdot]$ and $C[\cdot]$ are stored in $\mathcal{SC}^*[\cdot]$, $\mathcal{A}^*[\cdot]$ and $C^*[\cdot]$, respectively

Algorithm 5: **FollowerMaintenance**((v_s, v_t), G)

Input : (v_s, v_t) : an edge to insert or remove, G : the graph before inserting/removing (v_s, v_t), $\mathcal{SC}[\cdot]$, $\mathcal{A}[\cdot]$ and $C[\cdot]$ of G
Output : ${}^{+}\mathcal{F}(v, G)$ and ${}^{-}\mathcal{F}(v, G)$ for each v with changed followers

1 $\mathcal{SC}^*[\cdot]$, $\mathcal{A}^*[\cdot]$, $C^*[\cdot] \leftarrow \mathcal{SC}[\cdot]$, $\mathcal{A}[\cdot]$, $C[\cdot]$;
2 $\hat{\mathcal{S}} \leftarrow$ the new shell component set; $V^* \leftarrow \emptyset$;
3 **if** $c(v_s, G) = \min\{c(v_s, G), c(v_t, G)\}$ **then** $V^* \leftarrow V^* \cup \mathcal{SC}^*[v_s].V$;
4 **if** $c(v_t, G) = \min\{c(v_s, G), c(v_t, G)\}$ **then** $V^* \leftarrow V^* \cup \mathcal{SC}^*[v_t].V$;
5 (v_s, v_t) is inserted into or removed from G;
6 Update $c(u, G)$ for each $u \in V(G)$ by core maintenance [58];
/* compute new shell components */
7 **for** each *unassigned* $u \in V^*$ **do**
8 $\quad S^* \leftarrow$ a new shell component;
9 $\quad S^*.c \leftarrow c(u, G)$; $S^*.V \leftarrow S^*.V \cup \{u\}$;
10 $\quad u$ is set *assigned*; **ShellConnect**(u, S^*, $\mathcal{SC}^*$);
11 $\quad$ Replace $\mathcal{SC}^*[u]$ by S^* in $\mathcal{SC}^*$;
12 $\quad \hat{\mathcal{S}} = \hat{\mathcal{S}} \cup \{S^*\}$;
13 $\mathcal{A}^*[\cdot]$ and $C^*[\cdot]$ are updated while doing Lines 7-12;
14 $U \leftarrow \bigcup_{S^* \in \hat{\mathcal{S}}} S^*.V$;
/* remove expired followers on old shell components */
15 **for** each pair (v, S) in parallel with $v \in \mathcal{A}[S] \cup C[S]$ and $S \in \bigcup_{u \in U}\{\mathcal{SC}[u]\}$ **do**
16 $\quad$ **if** $v \in \mathcal{A}[S]$ **then**
17 $\quad\quad {}^{+}\mathcal{F}(v, G) \leftarrow {}^{+}\mathcal{F}(v, G) \setminus {}^{+}F[v][S]$;
18 $\quad$ **else if** $v \in C[S]$ **then**
19 $\quad\quad {}^{-}\mathcal{F}(v, G) \leftarrow {}^{-}\mathcal{F}(v, G) \setminus {}^{-}F[v][S]$;
/* compute new followers on new shell components */
20 Run **Lines 2-9 of Algorithm 4** with $\hat{\mathcal{S}}$, $\mathcal{A}^*[\cdot]$ and $C^*[\cdot]$;
21 $\mathcal{SC}[\cdot]$, $\mathcal{A}[\cdot]$, $C[\cdot] \leftarrow \mathcal{SC}^*[\cdot]$, $\mathcal{A}^*[\cdot]$, $C^*[\cdot]$;
22 **return** the result of Line 20

(Line 1), and they will be properly updated. Then we compute the set V^* which contains all the vertices to update corenesses for the insertion/removal of (v_s, v_t) (Lines 2-4). We use $\hat{\mathcal{S}}$ to record all the new shell components and each vertex is set to *unassigned* initially.

After inserting/removing (v_s, v_t) (Line 5), we adopt the state-of-the-art algorithm of core maintenance in [58] to update $c(u, G)$ for each $u \in V(G)$ (Line 6). Lines 7-12 collect the new shell components into $\hat{\mathcal{S}}$. Without the need to call Algorithm 2 again for the whole graph, the maintenance on shell components only starts from each vertex u in V^* (Line 7). Lines 8-12 do the same operations as Lines 3-6 of Algorithm 2 to compute the new shell component set $\hat{\mathcal{S}}$. Please note that $\mathcal{A}^*[\cdot]$ and $C^*[\cdot]$ can be updated straightforwardly during Lines 7-12 according to their definitions (Line 13), because only $\mathcal{A}^*[S^*]$ and $C^*[S^*]$ for each $S^* \in \hat{\mathcal{S}}$ need to be updated. We denote an updated vertex set as $U = \bigcup_{S^* \in \hat{\mathcal{S}}} S^*.V$ (Line 14). By traversing all the old shell components containing the vertices in U, we can remove all the expired anchored followers and collapsed followers (Lines 15-19). At last, for the new shell component set $\hat{\mathcal{S}}$, we call Algorithm 4 on $\hat{\mathcal{S}}$ along with the updated $\mathcal{A}^*[\cdot]$ and $C^*[\cdot]$ (Line 20) to compute the new followers. Finally, we put $\mathcal{SC}^*[\cdot]$, $\mathcal{A}^*[\cdot]$ $\mathcal{SC}[\cdot]$ back into $\mathcal{SC}[\cdot]$, $\mathcal{A}[\cdot]$ and $C[\cdot]$ for using in the next edge updating request. As in Algorithm 4, Lines 15-20 of Algorithm 5 can be straightforwardly parallelized.

Algorithm 6: **FindCollapsedFollowers**(x, S)

Input : x : a collapser in $C[S]$, S : a shell component
Output : ${}^{-}F[x][S]$: the collapsed follower set of x in S

1 $Q \leftarrow$ a queue;
2 **If** $x \in S.V$ **then** x is set *discarded*; $Q.push(x)$;
3 **else**
4 $\quad HS(u) \leftarrow HS(u) - 1$; $Q.push(u)$ **for** each $u \in N(x, S)$
5 **while** $Q \neq \emptyset$ **do**
6 $\quad u \leftarrow Q.pop()$;
7 $\quad$ **if** $u \neq x$ **then**
8 $\quad\quad d^+(u) \leftarrow HS(u) + |\{v \mid v \in N(u, S) \land v \textit{ is not discarded}\}|$;
9 $\quad\quad$ **If** $d^+(u) < S.c$ **then** u is set *discarded*;
10 $\quad$ **if** u is *discarded* **then**
11 $\quad\quad$ **for** each $v \in N(u, S)$ with v is not *discarded* and $v \notin Q$ **do**
12 $\quad\quad\quad Q.push(v)$;
13 ${}^{-}F[x][S] \leftarrow$ *discarded* vertices in $S.V \setminus \{x\}$;
14 **return** ${}^{-}F[x][S]$

Example 5.12. In Figure 5a, we remove the edge (v_3, v_4) from the graph. In the update of shell components (Lines 7-12), S_2 and S_3 are replaced (Line 11) with the new shell components, i.e., the new S_6^* with $S_6^*.V = \{v_2, v_4\}$ and $S_6^*.c = 2$, and the new S_7^* with $S_7^*.V = \{v_3\}$ and $S_7^*.c = 3$. The anchor and collapser candidate sets of S_1, S_4 and S_5 remain the same, and the vertices in the above two sets have the unchanged followers on S_1, S_4 and S_5. Thus, we only need to recompute the followers of $\mathcal{A}^*[S_6^*]$ and $C^*[S_6^*]$ on S_6^*, and the followers of $\mathcal{A}^*[S_7^*]$ and $C^*[S_7^*]$ on S_7^*, e.g., we compute ${}^{+}F[v_2][S_6^*] = \{v_4\}$ and we know ${}^{+}F[v_2][S_4]$ is not changed.

Example 5.13. In Figure 5b, we insert the edge (v_2, v_3) into the graph. S_2, S_3 and S_4 are replaced with the new S_8^*. We have $S_8^*.V = \{v_2, v_3, v_4, v_5, v_6, v_7\}$ and $S_8^*.c = 3$. The anchor and collapser candidate sets of S_1 and S_5 remain the same, and the vertices in the above two sets have unchanged followers on S_1 and S_5. Only the followers of $\mathcal{A}^*[S_8^*]$ and $C^*[S_8^*]$ on S_8^* need to be recomputed.

Theorem 5.14. *For the insertion or removal of (v_s, v_t), Algorithm 5 correctly updates ${}^{+}\mathcal{F}(v, G)$ and ${}^{-}\mathcal{F}(v, G)$ for each $v \in V(G)$.*

Complexity Analysis. In Algorithm 5, Lines 1-19 can be finished in $O(m)$ time since they traverse G by a constant number. So, the time complexity of Algorithm 5 is $O(m + m^* \cdot |S^*|_{max})$ where m^* is the accumulated size of $\mathcal{A}^*[S^*] \cup C^*[S^*]$ for every S^* in $\hat{\mathcal{S}}$ and $|S^*|_{max}$ is the largest size of a shell component in $\hat{\mathcal{S}}$.

5.3 Computation of Followers for One Vertex

Let $HS(u)$ denote *higher coreness support* of a vertex u, i.e., the number of u's neighbors with larger corenesses than u. We have $HS(u) = |\{v \mid v \in N(u, G) \land c(v) > c(u)\}|$.

Collapsed Followers Computation. We use Algorithm 6 to compute ${}^{-}F[x][S]$ which utilizes a queue Q (Line 1) to explore the collapsed followers starting from the collapser vertex x. If $x \in S.V$, x is set *discarded* and pushed into Q (Line 2). Note that, all the vertices in $S.V$ are not *discarded* initially, and any *discarded* vertex (except x) becomes a collapsed follower. If $x \notin S.V$, for each $u \in N(x, S)$, we reduce $HS(u)$ by 1 and push u into Q (Lines 3-4).

Algorithm 7: **FindAnchoredFollowers**(x, S)

Input : x : an anchor in $\mathcal{A}[S]$, S : a shell component
Output : ${}^{+}F[x][S]$: the anchored follower set of x in S

1 $H \leftarrow$ a min heap w.r.t. the layer value of each vertex;
2 **If** $x \in S.V$ **then** x is set *survived*; $H.push(x)$;
3 **else**
4 $\quad HS(u) \leftarrow HS(u) + 1$; $H.push(u)$ **for** each $u \in N(x, S)$;
5 **while** $H \neq \emptyset$ **do**
6 $\quad u \leftarrow H.pop()$;
7 $\quad V^{\leq} \leftarrow \{v \mid v \in N(u, S) \wedge l(v) \leq l(u) \wedge (v \textit{ is survived} \vee v \in H)\}$;
8 $\quad V^{>} \leftarrow \{v \mid v \in N(u, S) \wedge l(v) > l(u) \wedge v \textit{ is not discarded}\}$;
9 $\quad$ **if** $u \neq x$ **then**
10 $\quad\quad d^{+}(u) \leftarrow HS(u) + |V^{\leq}| + |V^{>}|$;
11 $\quad\quad$ **If** $d^{+}(u) \geq S.c + 1$ **then** u is set *survived*;
12 $\quad$ **if** u is *survived* **then**
13 $\quad\quad$ **for** each $v \in N(u, S)$ with $l(v) > l(u)$ and $v \notin H$ **do**
14 $\quad\quad\quad H.push(v)$;
15 $\quad$ **else**
16 $\quad\quad u$ is set *discarded*; **Shrink**(u, S) (Algorithm 8) ;
17 ${}^{+}F[x][S] \leftarrow$ *survived* vertices in $S.V \setminus \{x\}$;
18 **return** ${}^{+}F[x][S]$

Algorithm 8: **Shrink**(u, S)

Input : u : the vertex to shrink, S : a shell component

1 **for** each *survived* vertex $v \in N(v, S)$ with $v \neq x$ **do**
2 $\quad d^{+}(v) \leftarrow d^{+}(v) - 1$;
3 $\quad$ **If** $d^{+}(v) < S.c + 1$ **then** v is set *discarded*; $T \leftarrow T \cup \{v\}$;
4 **Shrink**(v) **for** each $v \in T$;

Then we traverse Q until it becomes empty (Line 5). Each time when we pop a vertex u (Line 6), if $u \neq x$, we need to decide whether it should be *discarded* (Lines 7-9). If u is set *discarded*, we push each undiscarded $v \in N(u, S) \setminus Q$ into Q (Lines 10-12). After traversing Q, all the *discarded* vertices in $S.V$ except x form ${}^{-}F[x][S]$.

Anchored Followers Computation. We use Algorithm 7 to compute ${}^{+}F[x][S]$ which adapts Algorithm 4 of [31]. The main idea is utilizing the *layer value* of each vertex, i.e., the deletion batch (layer) in core decomposition. We use L_k^i to denote the i-layer of the k-shell. Specifically, when $i = 1$, we have $L_k^1 = \{u \mid u \in C_k(G) \wedge deg(u, C_k(G)) < k+1\}$; when $i > 1$, we have $L_k^i = \{u \mid u \in G_k^i \wedge deg(u, G_k^i) < k+1\}$ where $G_k^i = G[V(C_k(G)) \setminus \bigcup_{1 \leq j \leq i-1} L_k^j]$. We denote $l(u) = i$ if $u \in L_k^i$ as the unique *layer value* of u.

In Algorithm 7, we utilize a min heap H (Line 1) to traverse the potential followers (except x), where the key is the layer value of each vertex[5]. Please note that any vertex in $S.V$ is neither *discarded* or *survived* unless it is explicitly set so, and all the *survived* vertices form the follower set except x. Note that a *survived* vertex may still be *discarded* later due to the deletion cascade.

If $x \in S.V$, x is set *survived* and pushed into H (Line 2). If $x \notin S.V$, for each $u \in N(x, S)$, we increment $HS(u)$ by 1 and push u into H (Lines 3-4). Then we traverse H until it becomes empty (Line 5). When we pop a vertex u (Line 6), if $u \neq x$, we need to decide whether it can be set *survived* (Lines 7-11), where we compute a degree upper bound $d^{+}(u)$ at Line 10. The vertices in $V^{\leq}$ or $V^{>}$ are the neighbors of u which may become its followers. If u is *survived*, we push each unvisited potential follower into H (Lines 12-14). Once a vertex is set *discarded*, it may cause a cascade of vertex discarding, computed by Algorithm 8 (Lines 15-16). After traversing H, all the *survived* vertices in $S.V$ except x form ${}^{+}F[x][S]$.

[5] The complexity can be optimized by using a set of buckets to simulate H.

Complexity Analysis. The neighbor set $N(\cdot, S)$, higher coreness support $HS(\cdot)$ and layer value $l(\cdot)$ can be incidentally computed when traversing the neighbors of each vertex in Algorithm 2, as the visiting sequence can be decided by different shell components. Either Algorithm 6 or 7 traverses the graph by a constant number. Thus, the time complexity is $O(|S|)$ for both algorithms, where $|S|$ is the number of edges in S.

Example 5.15. To compute ${}^{+}F[v_2][S_3]$ on the graph in Figure 4a, we execute Algorithm 7. As $v_2 \notin S_3.V$, we have $HS(v_4) = 2 + 1 = 3$ (originally $HS(v_4) = 2$ w.r.t. v_{11} and v_{12}) and v_4 is pushed into H (Line 4). After v_4 is popped (Line 6), we compute $d^{+}(v_4)$ (Line 10). Because $l(v_4) = 1$, $l(v_3) = 2$ and v_3 is not *discarded*, we have $V^{\leq} = \emptyset$ and $V^{>} = \{v_3\}$. So, $d^{+}(v_4) = 3 + 0 + 1 = 4$. Since $S_3.c = 3$, v_4 is set *survived* (Line 11) and v_3 is pushed into H (Line 14). Then v_3 is popped and $d^{+}(v_3) = 3 + 1 + 0 = 4$ as $HS(v_3) = 3$, $V^{\leq} = \{v_4\}$ and $V^{>} = \emptyset$. So v_3 is set *survived* and no more vertex is pushed into H. Finally, we return ${}^{+}F[v_2][S_3] = \{v_3, v_4\}$.

6 EXPERIMENTAL EVALUATION

Table 2: Statistics of Datasets

Dataset	Nodes	Edges	d_{max}	k_{max}	$\|S\|_{max}$
Brightkite	58,228	214,078	1,134	52	11,838
Github	37,700	289,003	9,458	34	20,976
Gowalla	196,591	950,327	14,730	51	14,060
NotreDame	325,729	1,090,108	10,721	155	215,052
Stanford	281,903	1,992,636	38,625	71	89,688
Youtube	1,134,890	2,987,624	28,754	51	72,726
DBLP	1,566,919	6,461,300	1,522	118	89,276
Yelp	1,032,416	17,971,548	6,367	165	269,238
Orkut	3,072,441	117,185,083	33,313	253	6,889,284

Datasets. We use 9 real-life datasets, where `NotreDame`, `Stanford` and `DBLP` are from [22], `Yelp` is from [11], and the others are from [25]. We clean self loops and multiple edges in the datasets. Each directed edge is regarded as undirected. Table 2 shows the statistics of the datasets, where d_{max} is the largest vertex degree, k_{max} is the largest vertex coreness and $|S|_{max}$ is the largest number of edges in a shell component. We may abbreviate the name of a dataset by the bold and underlined characters as shown in the table.

Algorithms. We evaluate the following algorithms: (i) `Baseline`: computing the follower sets of each vertex based on the state-of-the-art (Algorithm 1); (ii) `Static`: our new framework to compute the follower sets with shell components (Algorithm 4); (iii) `Maintenance`: our maintenance algorithm to update the follower sets of each vertex (Algorithm 5).

Environments. The experiments are conducted on a CentOS Linux server (Release 7.5.1804) equipped with a Quad-Core Intel Xeon

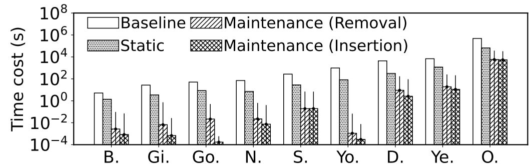


Figure 6: Performance of Different Algorithms

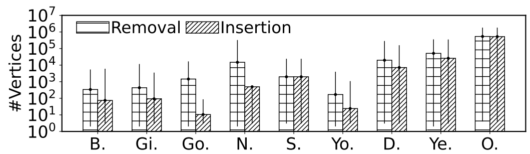
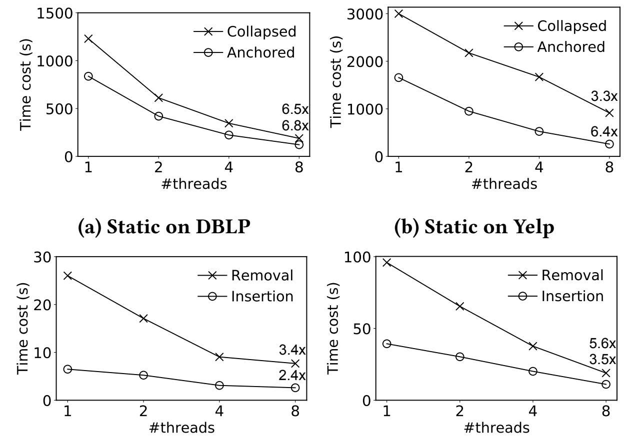


Figure 7: #Candidate Vertices to Update Follower Sets



(a) Static on DBLP (b) Static on Yelp

(c) Maintenance on DBLP (d) Maintenance on Yelp

Figure 8: Parallelization on `Static` and `Maintenance`

CPU (E5-2640 v4 @ 2.20GHz, 25MB Cache) and 128GB memory. All algorithms are implemented using C++11, with the source code compiled by GCC (7.3.0) under -O3 optimization. For shared memory multiprocessing programming, we employed OpenMP.

Our source code is shared at https://github.com/Xiejiadong/.

6.1 Main Experimental Results

Effectiveness of Anchor/Collapse Power. For the evaluation of model effectiveness, please refer to the results in Section 3. Some additional results on effectiveness are in the appendices.

Time Cost of Different Algorithms. In Figure 6, we report the runtime of `Baseline`, `Static` and `Maintenance`. We use 8 threads for each algorithm. For sequential performance, the runtime gap between different algorithms is very similar to the gap with 8 threads. To test real structure difference, we randomly remove 100 edges and insert them back into each dataset, for the maintenance algorithms. Each time an edge is removed or inserted, we record the time cost of computing both the anchored and collapsed followers for all the vertices and report the average runtime of 100 edges.

Figure 6 shows that `Static` is faster than `Baseline` by around 1 order of magnitude. It is consistent with the lower time complexity of `Static` compared with `Baseline`, as analyzed at the end of Section 5.1. Besides, `Maintenance` outperforms the static algorithms by orders of magnitude. This is more significant than the difference in time complexities, because the search space of `Maintenance` is often much smaller than the worst case. We also add the error bars to show the variance of time cost among the 100 edge removals and 100 edge insertions, respectively. We find that in the worst cases, the time cost of `Maintenance` is still less than other algorithms.

In the experiments, edge insertion is often faster than removal in `Maintenance`. It is because a shell component S may be split into S_1 and S_2 by edge removal, and a vertex in anchor candidate set $A[S]$ may be visited twice in both $A^*[S_1]$ and $A^*[S_2]$ at Line 20 of Algorithm 5. Thus, the number of candidate vertices to update/check their anchored followers can be larger after edge removal, because $|A^*[S_1]| + |A^*[S_2]| >= |A[S]|$. Correspondingly, we also have $|C^*[S_1]| + |C^*[S_2]| >= |C[S]|$. The computation on split shell components is more costly and edge insertion will not split a shell component. The gap in candidate number and time cost is consistent with the result in Figures 6 and 7, respectively.

Number of Candidate Vertices to Update Follower Sets. On each dataset, we randomly remove 100 edges and then insert 100 edges back. Each time an edge is removed or inserted, we record the number of candidate vertices whose follower sets may be updated, i.e., $\mathcal{A}^*[\cdot]$ and $C^*[\cdot]$ in Algorithm 5. In Figure 7, the bars show the average number of candidate vertices w.r.t one edge insertion or removal. We can find that a single edge update can cause 10^1 to 10^4 vertices to update their followers. We also add the error bar on each box to show the variance of the number of candidates to update among the removal and insertion of 100 edges, respectively.

Performance of Parallelization. Figures 8(a-b) report the performance of `Static` regarding collapsed follower computation (Algorithm 6) and anchored follower computation (Algorithm 7), respectively. We can find that the time cost with 8 threads is much smaller than the sequential cost, where the speedup is up to 6.8x for anchoring and 6.5x for collapsing, respectively. Figures 8(c-d) show the performance of `Maintenance` regarding edge insertion or removal. We mark the speedup ratio between 1 thread and 8 threads in each subfigure, which shows the advantage of our algorithms in separating the computation into atom units.

7 CONCLUSION

In this paper, we study the model of anchored/collapsed power. We validate the coreness value is positively correlated with node engagement, and the match between the anchored/collapsed power and node importance over network structural stability. A novel framework is proposed to compute the anchored/collapsed power of every node on both static and dynamic graphs, with well-designed optimizations. The experiments on 9 real-life datasets demonstrate the effectiveness of the model and the efficiency of our algorithms.

ACKNOWLEDGMENTS

This work is partially supported by the National Natural Science Foundation of China (62002073, U2241211) and the Guangdong Basic and Applied Basic Research Foundation (2023A1515012603, 2019B1515120048). We thank the anonymous reviewers for their insightful suggestions. Kai Wang is the corresponding author.

REFERENCES

[1] Gary D Bader and Christopher WV Hogue. 2003. An automated method for finding molecular complexes in large protein interaction networks. *BMC bioinformatics* 4, 1 (2003), 1–27.

[2] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[3] Kshipra Bhawalkar, Jon M. 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 (2015), 1452–1475.

[4] Michał Bola and Bernhard A Sabel. 2015. Dynamic reorganization of brain functional networks during cognition. *Neuroimage* 114 (2015), 398–413.

[5] 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 *ICDE*. 409–420.

[6] Chen Chen, Qiuyu Zhu, Renjie Sun, Xiaoyang Wang, and Yanping Wu. 2021. Edge manipulation approaches for k-core minimization: metrics and analytics. *IEEE Trans. Knowl. Data Eng.* (2021).

[7] Huiping Chen, Alessio Conte, Roberto Grossi, Grigorios Loukides, Solon P. Pissis, and Michelle Sweering. 2021. On Breaking Truss-Based Communities. In *KDD*. 117–126.

[8] Deming Chu, Fan Zhang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2022. Hierarchical Core Decomposition in Parallel: From Construction to Subgraph Search. In *ICDE*. 1138–1151.

[9] Michael Suk-Young Chwe. 2000. Communication and coordination in social networks. *The Review of Economic Studies* 67, 1 (2000), 1–16.

[10] Pierluigi Crescenzi, Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. 2016. Greedily Improving Our Own Closeness Centrality in a Network. *ACM Trans. Knowl. Discov. Data* 11, 1 (2016), 9:1–9:32.

[11] Yelp Open Dataset. 2022. https://www.yelp.com/dataset.

[12] Palash Dey, Suman Kalyan Maity, Sourav Medya, and Arlei Silva. 2021. Network Robustness via Global k-cores. In *AAMAS*. 438–446.

[13] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-Core Community Search over Labeled Graphs. *Proc. VLDB Endow.* 14, 11 (2021), 2006–2018.

[14] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. 2009. Extraction and classification of dense implicit communities in the web graph. *ACM Transactions on the Web (TWEB)* 3, 2 (2009), 1–36.

[15] Yixiang Fang, Reynold Cheng, Yankai Chen, Siqiang Luo, and Jiafeng Hu. 2017. Effective and efficient attributed community search. *VLDB J.* 26, 6 (2017), 803–828.

[16] Scott Freitas, Diyi Yang, Srijan Kumar, Hanghang Tong, and Duen Horng Chau. 2022. Graph Vulnerability and Robustness: A Survey. *IEEE Trans. Knowl. Data Eng.* (2022).

[17] David García, Pavlin Mavrodiev, and Frank Schweitzer. 2013. Social resilience in online communities: the autopsy of friendster. In *Conference on Online Social Networks*. 39–50.

[18] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. 2015. Fully Dynamic Betweenness Centrality Maintenance on Massive Networks. *Proc. VLDB Endow.* 9, 2 (2015), 48–59.

[19] Wissam Khaouid, Marina Barsky, Venkatesh Srinivasan, and Alex Thomo. 2015. K-core decomposition of large networks on a single PC. *Proc. VLDB Endow.* 9, 1 (2015), 13–23.

[20] Maksim Kitsak, Lazaros K Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H Eugene Stanley, and Hernán A Makse. 2010. Identification of influential spreaders in complex networks. *Nature physics* 6, 11 (2010), 888–893.

[21] Jon M. Kleinberg. 1998. Authoritative Sources in a Hyperlinked Environment. In *ACM-SIAM Symposium on Discrete Algorithms*. 668–677.

[22] Jérôme Kunegis. [n.d.]. The KONECT Project. http://konect.cc/.

[23] Ricky Laishram, Ahmet Erdem Sariyüce, Tina Eliassi-Rad, Ali Pinar, and Sucheta Soundarajan. 2018. Measuring and Improving the Core Resilience of Networks. In *WWW*. 609–618.

[24] Glenn Lawyer. 2015. Understanding the influence of all nodes in a network. *Scientific reports* 5, 1 (2015), 1–9.

[25] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[26] Rong-Hua Li, Lu Qin, Fanghua Ye, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. 2018. Skyline Community Search in Multi-valued Networks. In *SIGMOD*. 457–472.

[27] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential Community Search in Large Networks. *Proc. VLDB Endow.* 8, 5 (2015), 509–520.

[28] Rong-Hua Li, Jeffrey Xu Yu, and Rui Mao. 2014. Efficient Core Maintenance in Large Dynamic Graphs. *IEEE Trans. Knowl. Data Eng.* 26, 10 (2014), 2453–2465.

[29] Jian-Hong Lin, Qiang Guo, Wen-Zhao Dong, Li-Ying Tang, and Jian-Guo Liu. 2014. Identifying the node spreading influence with largest k-core values. *Physics Letters A* 378, 45 (2014), 3279–3284.

[30] Zhe Lin, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Zhihong Tian. 2021. Hierarchical Core Maintenance on Large Dynamic Graphs. *Proc. VLDB Endow.* 14, 5 (2021), 757–770.

[31] Qingyuan Linghu, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2020. Global Reinforcement of Social Networks: The Anchored Coreness Problem. In *SIGMOD*. 2211–2226.

[32] 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.

[33] Jing Liu, Mingxing Zhou, Shuai Wang, and Penghui Liu. 2017. A comparative study of network robustness measures. *Frontiers Comput. Sci.* 11, 4 (2017), 568–584.

[34] Kaixin Liu, Sibo Wang, Yong Zhang, and Chunxiao Xing. 2021. An Efficient Algorithm for the Anchored k-Core Budget Minimization Problem. In *ICDE*. 1356–1367.

[35] 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 (2021), 1531–1543.

[36] Junjie Luo, Hendrik Molter, and Ondrej Suchý. 2021. A Parameterized Complexity View on Collapsing k-Cores. *Theory Comput. Syst.* 65, 8 (2021), 1243–1282.

[37] Fragkiskos D Malliaros, Maria-Evgenia G Rossi, and Michalis Vazirgiannis. 2016. Locating influential nodes in complex networks. *Scientific reports* 6, 1 (2016), 1–10.

[38] Fragkiskos D Malliaros and Michalis Vazirgiannis. 2013. To stay or not to stay: modeling engagement dynamics in social graphs. In *CIKM*. 469–478.

[39] David W Matula and Leland L Beck. 1983. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)* 30, 3 (1983), 417–427.

[40] Sourav Medya, Tiyani Ma, Arlei Silva, and Ambuj K. Singh. 2020. A Game Theoretic Approach For Core Resilience. In *IJCAI*. 3473–3479.

[41] Sourav Medya, Arlei Silva, Ambuj K. Singh, Prithwish Basu, and Ananthram Swami. 2018. Group Centrality Maximization via Network Design. In *SDM*. 126–134.

[42] Alberto Montresor, Francesco De Pellegrini, and Daniele Miorandi. 2013. Distributed k-Core Decomposition. *IEEE Trans. Parallel Distributed Syst.* 24, 2 (2013), 288–300.

[43] Flaviano Morone, Gino Del Ferraro, and Hernán A Makse. 2019. The k-core as a predictor of structural collapse in mutualistic ecosystems. *Nature physics* 15, 1 (2019), 95–102.

[44] Milena Oehlers and Benjamin Fabian. 2021. Graph Metrics for Network Robustness—A Survey. *Mathematics* 9, 8 (2021), 895.

[45] Ahmet Erdem Sariyüce, Bugra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V. Çatalyürek. 2013. Streaming Algorithms for k-core Decomposition. *Proc. VLDB Endow.* 6, 6 (2013), 433–444.

[46] Ahmet Erdem Sariyüce and Ali Pinar. 2016. Fast Hierarchy Construction for Dense Subgraphs. *Proc. VLDB Endow.* 10, 3 (2016), 97–108.

[47] Ahmet Erdem Sariyüce, C. Seshadhri, and Ali Pinar. 2018. Local Algorithms for Hierarchical Dense Subgraph Discovery. *Proc. VLDB Endow.* 12, 1 (2018), 43–56.

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

[49] Kazunori Seki and Masataka Nakamura. 2017. The mechanism of collapse of the Friendster network: What can we learn from the core structure of Friendster? *Social Network Analysis and Mining* 7, 1 (2017), 10.

[50] Xin Sun, Xin Huang, and Di Jin. 2022. Fast Algorithms for Core Maximization on Large Graphs. *Proc. VLDB Endow.* (2022).

[51] Johan Ugander, Lars Backstrom, Cameron Marlow, and Jon Kleinberg. 2012. Structural diversity in social contagion. *Proceedings of the National Academy of Sciences* 109, 16 (2012), 5962–5966.

[52] Casper van Elteren, Rick Quax, and Peter Sloot. 2022. Dynamic importance of network nodes is poorly predicted by static structural features. *Physica A: Statistical Mechanics and its Applications* (2022), 126889.

[53] Dong Wen, Lu Qin, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. 2016. I/O efficient core graph decomposition at web scale. In *ICDE*. 133–144.

[54] Fan Zhang, Jiadong Xie, Kai Wang, Shiyu Yang, and Yu Jiang. 2022. Discovering key users for defending network structural stability. *World Wide Web* 25, 2 (2022), 679–701.

[55] 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 (2017), 649–660.

[56] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2017. Finding Critical Users for Social Network Engagement: The Collapsed k-Core Problem. In *AAAI*. 245–251.

[57] Hengda Zhang, Xiaofei Wang, Hao Fan, Taotao Cai, Jianxin Li, Xiuhua Li, and Victor C. M. Leung. 2020. Anchor Vertex Selection for Enhanced Reliability of Traffic Offloading Service in Edge-Enabled Mobile P2P Social Networks. *J. Commun. Inf. Networks* 5, 2 (2020), 217–224.

[58] Yikai Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. 2017. A Fast Order-Based Approach for Core Maintenance. In *ICDE*. 337–348.

[59] Zhongxin Zhou, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Chen Chen. 2019. K-Core Maximization: An Edge Addition Approach. In *IJCAI*. 4867–4873.

[60] Weijie Zhu, Chen Chen, Xiaoyang Wang, and Xuemin Lin. 2018. K-core Minimization: An Edge Manipulation Approach. In *CIKM*. 1667–1670.

8 APPENDICES

8.1 Proofs of Theorems

Proof of Theorem 5.8. We prove it by contradiction. We assume $x \notin \mathcal{A}[S]$ and there is a vertex $u \in S.V$ with coreness increased by anchoring x (i.e., u is an anchored follower of x). According to the definition of $\mathcal{A}[S]$, we have the following possible situations: 1) $c(x, G) > S.c$; 2) $c(x, G) = S.c \wedge x \notin S.V$; 3) $c(x, G) < S.c \wedge N(x, G) \cap S.V = \emptyset$.

For situation 1), no matter x is anchored or not, x is always deleted after u in core decomposition [19], i.e., the vertex deletion sequence before removing u is same. So we have $c^+_x(u) = c(u)$ which contradicts with our assumption.

For situation 2), we first consider the vertex deletion sequence in core decomposition without anchoring x. After all the vertices with coreness less than $S.c$ are deleted, for each $S' \neq S$ and $S'.c = S.c$, it is feasible to delete $S.V$ before each $S'.V$ in core decomposition. After deleting $S.V$, we denote the deleted vertex set so far as V^d. Because $c(x) = S.c$ and $x \notin S.V$, we have $x \notin V^d$. Then, we consider the core decomposition with anchoring x. As $x \notin V^d$, we can still delete V^d following the same vertex deletion sequence. Thus for each $v \in V^d$, $c^+_x(v) = c(v)$. Since $u \in S.V \subseteq V^d$, u is not an anchored follower of x, which contradicts with our assumption.

For situation 3), we first consider the core decomposition without anchoring x. For each $v \in S.V$, we denote $N^<(v) = \{w \mid w \in N(v, G) \wedge c(w) < S.c\}$ and $N^>(v) = \{w \mid w \in N(v, G) \wedge c(w) > S.c\}$; we have $|N(v, S) \cup N^>(v)| < S.c + 1$ as $c(v) = S.c$. Then, we consider the core decomposition with anchoring x. For each $w \in N^<(v)$ subject to each $v \in S.V$, we have $w \neq x$ by situation 3) and $c^+_x(w) < S.c + 1$ by Lemma 5.4. Thus each above w is not in the $(S.c + 1)$-core. Now consider each $v \in S.V$. Let W denote the set consists of each $w \in N^<(v)$ with $c^+_x(w) = S.c$. The vertices in $N^<(v) \setminus W$ is still deleted before the removal of v. Then, it is feasible to delete W before v since W is not in the $(S.c + 1)$-core. Then we still have $|N(v, S) \cup N^>(v)| < S.c + 1$. Thus, v is not an anchored follower of x. As $u \in S.V$, this contradicts with our assumption. □

Proof of Theorem 5.9. We prove it by contradiction. We assume $x \notin C[S]$ and there is a vertex $u \in S.V$ with coreness decreased by collapsing x (i.e., u is an collapsed follower of x). According to the definition of $C[S]$, we have the following possible situations: 1) $c(x, G) < S.c$; 2) $c(x, G) = S.c \wedge x \notin S.V$; 3) $c(x, G) > S.c \wedge N(x, G) \cap S.V = \emptyset$.

For situation 1), no matter x is collapsed or not, x is always deleted before u in core decomposition [19], so that $c^-_x(u) = c(u)$ which contradicts with our assumption.

For situation 2), we first consider the vertex deletion sequence in core decomposition without collapsing x. After all the vertices with coreness less than $S.c$ are deleted, for each $S' \neq S$ and $S'.c = S.c$, it is feasible to delete $S'.V$ before $S.V$ in core decomposition. After the deletion of every above S', the remaining graph G' still has $deg(v, G') \geq S.c$ for each $v \in S.V$. We denote the deleted (resp. remaining) vertex set so far as V^d (resp. V^r). For each $v \in V^r \setminus S.V$, we have $c(v) > S.c$. Because $c(x) = S.c$ and $x \notin S.V$, we can ensure $x \in V^d$. Then, we consider the core decomposition with collapsing x. We first delete V^d including x, and the remaining graph is still induced by V^r satisfying $deg(v) \geq S.c$ for each $v \in V^r$. Since $u \in S.V \subseteq V^r$, u is not a collapsed follower of x, which contradicts with our assumption.

For situation 3), we first consider the core decomposition without collapsing x. For each $v \in S.V$, we denote $N^>(v) = \{w \mid w \in N(v, G) \wedge c(w) > S.c\}$; we have $|N(v, S) \cup N^>(v)| \geq S.c$ as $c(v) = S.c$. Then, we consider the core decomposition with collapsing x, which firstly deletes each $v \in V(G)$ with $c^-_x(v) < S.c$. For each $w \in N^>(v)$ subject to each $v \in S.V$, we have $w \neq x$ by situation 3) and $c^-_x(w) \geq S.c$ by Lemma 5.5. For each $v \in S.V$, the vertices of $N^>(v)$ exist in current remaining graph, so we still have $|N(v, S) \cup N^>(v)| \geq S.c$. Thus, v is not a collapsed follower of x. As $u \in S.V$, this contradicts with our assumption. □

Proof of Theorem 5.14. We first prove that Lines 1-12 correctly update the shell components. Assume $c(v_s) \leq c(v_t)$ where the coreness is from the graph before inserting or removing (v_s, v_t). For the insertion or removal of (v_s, v_t), only the vertices with corenesses equal to $c(v_s)$ and are reachable from v_s via a path consists of vertices with corenesses equal to $c(v_s)$ may have their corenesses changed by at most 1 [45, 58], i.e., only the vertices in V^* may change their corenesses. After the insertion or removal, the new shell components in $\hat{\mathcal{S}}$ are correctly computed by Algorithm 3 at Lines 7-12. For any shell component $S'' \notin \hat{\mathcal{S}}$, i.e., S'' is not connected with any new shell component $S' \in \hat{\mathcal{S}}$ among the edges in the updated $\mathcal{SC}^*$, we have the coreness of each vertex in S'' keeps unchanged. It is because, in previous and current core decompositions, the degree of each vertex in S'' keeps the same in the remaining graph when S'' is the next shell component to be deleted. Otherwise, S'' will be updated by Lines 7-12. Thus, each above S'' keeps the same and the shell components are correctly updated. By Theorems 5.8 and 5.9, the follower sets regarding the expired shell components should be removed (Lines 15-19) and the follower sets regarding the new shell components are correctly computed (Line 20). □

8.2 Additional Experiments

Different Metrics v.s. Ground-truth Node Engagement. Similar to the engagement study in Section 3, we further validate the correlation between different metrics and node engagement in `Brightkite` data from [25]. Note that there are only a few datasets available for the validation, because we need the ground-truth engagement of each user and the engagement dynamic between different time periods to check the ground-truth node importance.

The setting on `Brightkite` is the same as in the study on `Yelp`, except that the ground-truth engagement of a user here is the number of his/her check-ins in `Brightkite`. As shown in Figures 9, the performance is similar to `Yelp`: the engagement values may differ a lot for the vertices with close degrees (resp. betweenness or closeness values), while coreness is clearly in a positive correlation with node engagement. Note that coreness also outperforms other variants of centrality metrics, because core decomposition well models the leaving sequence of users in the degeneration of a network. Thus, the users with large corenesses are less likely to leave the network, compared with the users with small corenesses. Therefore, it is promising to study the engagement dynamics with the coreness-based models, i.e., the anchor/collapse power.

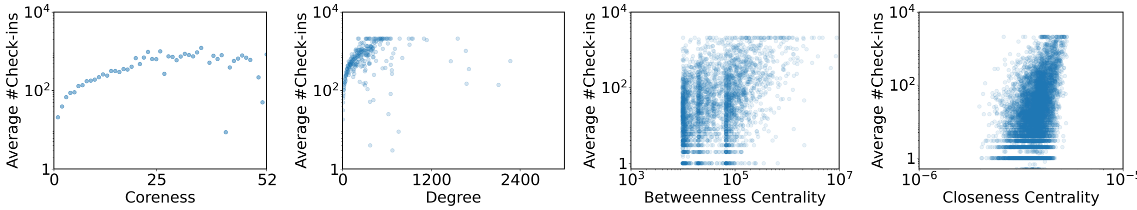
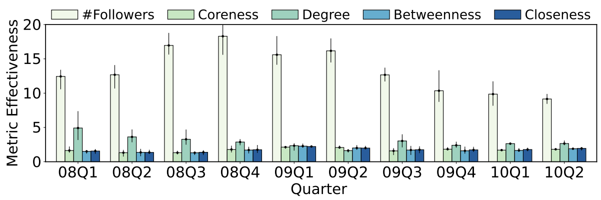
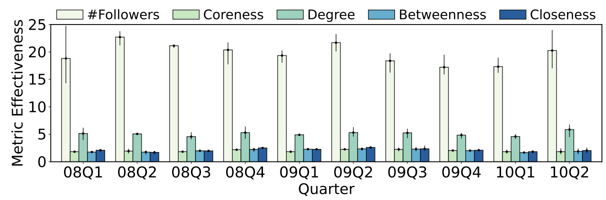


Figure 9: Different Vertex Ranking Metrics v.s. Ground-truth Node Engagement (#Check-ins in `Brightkite`)



(a) Importance of Anchored Users (Top 500 / The Rest)



(b) Importance of Collapsed Users (Top 500 / The Rest)

Figure 10: Effectiveness of Different Metrics by Relative Importance of the Selected Users (Dynamic of #Check-ins in Brightkite)

Different Metrics v.s. Ground-truth Node Importance. Similar to the importance study in Section 3. The setting is the same as in the study on `Yelp`, except that 1) the ground-truth node importance here is represented by the effect on user check-ins, 2) the first group here contains 500 users with the highest scores, and 3) the second group contains the rest users (5737 on average). Compared to the case study of `Yelp` (Figure 3), the gaps between #Followers and other metrics are larger, because of the different settings on user groups and the large number of check-ins in `Brightkite` compared with the number of reviews in `Yelp`.