Residual Core Maximization: An Efficient Algorithm for Maximizing the Size of the k-Core

Ricky Laishram* Ahmet Erdem Sarıyüce† Tina Eliassi-Rad‡ Ali Pinar§
Sucheta Soundarajan*

Abstract

In many online social networking platforms, the participation of an individual is motivated by the participation of others. If an individual chooses to leave a platform, this may produce a cascade in which that person's friends then choose to leave, causing their friends to leave, and so on. In some cases, it may be possible to incentivize key individuals to stay active within the network, thus preventing such a cascade. This problem is modeled using the *anchored k-core* of a network, which, for a network G and set of anchor nodes A, is the maximal subgraph of G in which every node has a total of at least k neighbors between the subgraph *and* anchors. In this work, we propose `Residual Core Maximization (RCM)`, a novel algorithm for finding b anchor nodes so that the size of the anchored k-core is maximized. We perform a comprehensive experimental evaluation on numerous real-world networks and compare `RCM` to various baselines. We observe that `RCM` is more effective and efficient than the state-of-the-art methods: on average, `RCM` produces anchored k-cores that are 1.65 times larger than those produced by the baseline algorithm, and is approximately 500 times faster on average.

1 Introduction

The participation of a person in social networking platforms is often motivated by the participation of others. People take part in such platforms in order to engage with others; and in return, they produce content that appeals to others. In other words, people's incentives for participation on a platform depend partially on the number of people to whom they can reach. When these incentives are low, people may leave the platform. This decreased participation may affect the participation of others, further decreasing the incentives for participation. Considering the social-networking platform as a complex network among people, locally decreased participation may cause a cascading exodus from the platform. Finding (and incentivizing) the critical individuals whose active participation are key to the larger participation in the network is an essential problem.

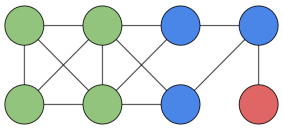


Figure 1: An anchored k-core example. The green nodes form a 3-core. If the red node is *anchored*, the entire graph becomes an anchored 3-core.

Motivated by this problem, we study the *anchored k-core problem*. The k-core of a graph is the maximal subgraph such that all nodes within the subgraph have degree at least k [26]. In the anchored k-core problem [5], one seeks to find a set of nodes to 'anchor,' or retain within the anchored k-core, even if their degree within the k-core subgraph is less than k: other nodes in the anchored k-core must thus have at least k connections either to other nodes in the subgraph or to the anchors. The objective of the anchored k-core problem is to maximize the size of the resulting *a*nchored k-core [6], in hopes of preventing a cascading exodus. An example is given in Figure 1. Here, the green nodes are in the 3-core, and if the red node is anchored, each blue node will have 3 neighbors within the subgraph, so the entire graph becomes an anchored 3-core.The anchored k-core problem is known to be NP-hard for $k > 2$ [5].

The algorithmic challenge behind the anchored k-core problem lies in the ability to foresee cumulative effect of groups of anchor nodes, not just individual nodes. It is possible that the addition of the first few anchor nodes make no difference, but the addition of one more anchor makes a drastic difference. A good algorithm should be able to foresee the big future pay-off even when the immediate benefits are small.

*Syracuse University, Syracuse, NY. (*rlaishra, susounda@syr.edu*)
†University at Buffalo, Buffalo, NY. (*erdem@buffalo.edu*)
‡Northeastern University, Boston, MA. (*eliassi@ccs.neu.edu*)
§Sandia National Laboratories, Livermore, CA. (*apinar@sandia.gov*) Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

We propose `Residual Core Maximization (RCM)`, a novel algorithm for the anchored k-core problem. `RCM` selects anchors based on two measures – *Anchor Score* and *Residual Degree*. If the number of anchors needed to convert a connected component is more than the anchor budget available, the anchors are selected based on the anchor score. Otherwise, the anchor selection depends on the residual degree.

The contributions of this paper are as follows:

1. We propose `Residual Core Maximization`, a novel algorithm for selecting anchor nodes to maximize the size of the anchored k-core.
2. We demonstrate that across various real social network datasets, `RCM` outperforms existing algorithms by finding an average of 1.65 times more followers[1], while also being 500 times faster.
3. We experimentally show that the `RCM` solutions values are close to the optimal solution values, while being multiple orders of magnitude faster.

All datasets used are publicly available, and we provide source code for `RCM`.[2]

2 Related Work

There has been extensive work on characterizing and analyzing the k-core decomposition of networks. We refer the reader to [24] for a more detailed survey.

2.1 k-core Decomposition Seidman defined the k-core of a graph as the maximal connected subgraph in which each node is connected to at least k other nodes [26] in the subgraph. The maximum k such that a node belongs in that k-core is called the *coreness* or *core number* of that node. Matula and Beck proposed a method for finding the k-cores in a graph [19]. Batagelj and Zaversnik proposed an efficient algorithm to find the k-core decomposition, in which nodes of degree less than k are iteratively removed until none are left [4].

The k-core decomposition has been used in numerous applications, including network visualization [2, 29, 31], studying the topology of large networks (such as the Internet) [3, 7], accelerating community detection [22], and studying the resilience of communities [10].

Laishram *et al.* proposed a measure for the resilience of the k-core structure and a method of inserting edges to improve the resilience [16]. Medya *et al.* [20] also studied the resilience of k-cores from a game-theoritic perspective. In [27], Shin *et al.* developed a method to find anomalous nodes in a social network based on their coreness and degree. In [1, 15, 18], the k-core decomposition is used to identify influential spreaders in social networks.

There has been a great deal of interest in k-core decomposition in large graphs. In [8, 14], researchers proposed out-of-core algorithms for k-core decomposition on large graphs, and in [21], distributed algorithms are introduced. For dynamic graph, various methods of maintaining the k-core structure in the case of streaming data has also been proposed [23, 17, 30, 9].

There has been a lot of works on extending the notion of k-cores to other network settings. Sariyuce *et al.* generalized k-cores to higher order structures [25], and Giatsidis *et al.* adapted the idea of k-cores to directed and weighted graphs [11, 12].

2.2 Anchored k-core Problem The *anchored k-core* problem was introduced by Bhawalkar *et al.* in 2012 [5]. The problem was inspired by the observation that a user in a social network is motivated to stay only if her neighborhood meets some minimal level of engagement: in k-core terms, she will stay if k friends are also in the network. Bhawalkar *et al.* defined the anchored k-core as the subgraph that is computed using the usual k-core decomposition algorithm, but with the modification that selected 'anchor' nodes are not deleted during the process. These anchored nodes may represent, for example, nodes that are recruited to remain active in the network, even if their friends are inactive. The anchored k-core problem, then, is the problem of selecting a specified number b anchor nodes such that the number of nodes in the anchored k-core is maximized. Bhawalkar *et al.* showed that for a general graph the anchored k-core problem is solvable in polynomial time for $k \leq 2$, but is NP-hard for $k > 2$ [6].

Zhang *et al.* proposed a greedy algorithm, called `OLAK`, for the anchored k-core problem [28]. `OLAK` operates over b iterations, where b is the maximum number of anchor nodes allowed. In each iteration, a node that is not in the anchored k-core but which would generate the largest number of followers if anchored is selected as the next anchor. Because only a single anchor node is considered at a time, and only nodes from the $(k-1)$-shell[3] can become followers when anchoring a single node, `OLAK` considers only follower nodes from the anchored $(k-1)$-shell during each iteration.

Zhou *et al.* [32] studied a problem that is close to the anchored k-core problem – which edges should be added to maximize the size of the k-core. However, this is fundamentally different from the anchored k-core problem because the graph cannot be modified in the anchored k-core problem.

[1] The followers are the nodes (excluding anchors) that are not in the k-core originally, but are in the anchored k-core.

[2] `https://github.com/rlaishra/RCM/`

[3] The k-shell is the subgraph of the k-core $\setminus$ $(k-1)$-core.

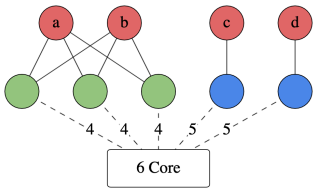


Figure 2: In this example, we seek to maximize the size of the anchored 6-core. The red nodes are the candidate anchors, the green nodes are in 4-shell and blue nodes are in 5-shell. The edges between the 6-core and the rest of the nodes are shown with dashed lines and the number represents the number of edges.

3 Problem Definition

Consider a graph $G = \langle V, E\rangle$, and let $N(u)$ denote the set of neighbors of node u in G. *The k-core of G is the maximal subgraph $G_k = \langle V_k, E_k\rangle$ such that every node in V_k has at least k neighbors in V_k* [4]. We use $\overline{G_k}$ to refer to the subgraph induced by $\overline{V_k} = V \setminus V_k$.

The k-core of a graph can be computed through a 'peeling' technique in which nodes with degree less than k are iteratively removed until all surviving nodes have degree at least k in the remaining subgraph.

Consider $A \subset \overline{V_k}$. The *anchored k-core* of G with anchors A is the maximal subgraph $G_{k,A} = \langle V_{k,A}, E_{k,A}\rangle$ such that $\forall u \in V_{k,A}$ one of the following holds:

(1) u is an anchor node, i.e., $u \in A$,
(2) u has at least k neighbors in $V_{k,A}$, i.e., $|N(u) \cap V_{k,A}| \geq k$.

The anchored k-core of a graph can be computed like the usual k-core – but with the nodes in A kept in the graph even if their degree is below k. In many applications, there is a bound on the number of anchor nodes. We denote this *anchor budget* by b. The '*followers*' are the non-anchor nodes that are not in the k-core but are in the anchored k-core, and are denoted by $\mathcal{F}(G, k, A)$, where

$$\mathcal{F}(G, k, A) = V_{k,A} \setminus (V_k \cup A). \tag{3.1}$$

For brevity, we will use $\mathcal{F}(A)$ when the G and k are clear from the context.

The anchored k-core problem was introduced in [6] as follows: *If we are given an anchor budget of b_m, which nodes should be anchored so that the number of followers is maximized?* Formally, the objective is to find the set A^* such that

$$A^* = \underset{A \subseteq \left[\overline{V_k}\right]^b}{\arg\max} |\mathcal{F}(k, A)| \tag{3.2}$$

where $\left[\overline{V_k}\right]^b = \left\{X \subseteq \overline{V_k} : |X| = b\right\}$.

The state-of-the-art methods for the anchored k-core problem are greedy methods that, in each step, select the anchor that brings in the most followers at each step. To demonstrate the shortcomings of this greedy approach, consider the example in Figure 2. In this example, we seek to maximize the size of the anchored 6-core by anchoring 2 nodes. The red nodes are the candidate anchors, the green nodes are in the 4-shell, and the blue nodes are in the 5-shell. For visual clarity, the edges between the 6-core and the rest of the nodes are represented by dotted lines, and the number represents the number of edges. It is clear that a greedy approach will select nodes c and d as anchors, resulting in 2 new followers. However, had a and b been anchored, there would have been 3 new followers.

4 Methodology

In this paper, we propose an anchor node selection algorithm called *Residual Core Maximization* (`RCM`). We describe the components that makes up `RCM` and how they combine together to select the anchor nodes in this section.

4.1 Candidate Followers and Anchors

We begin by deriving the necessary conditions for a node to be a *candidate follower* from the definition of k-core, and then use that to find the *candidate anchors*.

By definition of anchored k-core, it is easy to see that for $v \in \overline{V_{k,A}}$, if $|N(v)| < k$, it is not possible for v to become a follower. So, the set of *candidate followers* is given by,

$$C_f \stackrel{\text{def}}{=} \{v_f \in \overline{V_{k,A}} : |N(v_f)| \geq k\}. \tag{4.3}$$

Next, let $A' \subseteq \overline{V_{k,A}}$ denote the additional anchor nodes selected. Then for $v_a \in A'$, if $N(v_a) \cap C_f = \emptyset$, it is not possible for v_a to bring in new follower (either by itself or combination with other nodes). For efficiency we should not select such nodes as anchors. So the set of *candidate anchors* is,

$$C_a \stackrel{\text{def}}{=} \{v_a \in \overline{V_{k,A}} : |N(v_a) \cap C_f| > 0\} \tag{4.4}$$

and we should ensure that $A' \subseteq C_a$. We can discard any nodes not in $C_f \cup C_a$ from consideration.

4.2 Residual Degree

For nodes in C_f, we need to quantify how 'far' they are from becoming followers. So, we introduce the *Residual Degree* of a node $v_f \in C_f$ given anchor nodes A as,

$$\delta(v|A) \stackrel{\text{def}}{=} k - |N(v) \cap V_{k,A}|. \tag{4.5}$$

After new anchors A' are added, if $\delta(v_f|A \cup A') \leq 0$, it is easy to see from the definition of anchored k-core that v_f should also be in the anchored k-core

Algorithm 1 `FindResidualCore()`

$G_f \leftarrow$ Subgraph of G induced by C_f
$\mathcal{G} \leftarrow$ Connected components in G_f
$\mathcal{G} \leftarrow \{S \in \mathcal{G} : (\exists v \in S : N(v) \cap A' \neq \emptyset)\}$
$X \leftarrow$ Nodes in all the subgraphs in $\mathcal{G}$
repeat
  $Y \leftarrow \{v \in X : |N(v) \cap (X \cup A')| < \delta(v)\}$
  $X \leftarrow X \setminus Y$
until $Y = \emptyset$
return X

with anchors $A \cup A'$. We can also see that $\delta(v_f|A) \geq \delta(v_f|A \cup A')$ for any A'. So, intuitively the residual degree tells us how 'far' a candidate follower is from becoming a follower – nodes with lower values can be converted to new followers more easily. In the rest of the discussion, for brevity, we will denote the residual degree with $\delta(v_f)$ if A is clear from the context.

4.3 Residual Core When nodes $A' \subseteq C_a$ are added to A an additional anchors, which nodes in C_f become followers? To answer this we define the *Residual Core* subgraph. The residual core subgraph (with respect to the new anchors A') is defined as the maximal subgraph such that every node in the subgraph has at least as many neighbors in the subgraph or A' as its residual degree. We denote the residual core of A' with $R^*_{A'}$.

The residual core subgraph gives us all the new followers due to A' (Lemma 4.1), and it can be found efficiently as described in Algorithm 1.[4]

LEMMA 4.1. $\mathcal{F}(A \cup A') \setminus \mathcal{F}(A) = R^*_{A'}$.

Proof. Let $Y = \mathcal{F}(A \cup A') \setminus \mathcal{F}(A)$. Consider $v \in Y$. Then, $|V_{k,A\cup A'} \cap N(v)| \geq k$ and $|V_{k,A} \cap N(v)| < k$. We know that, $V_{k,A\cup A'} = V_{k,A} \cup A' \cup Y$, where $V_{k,A}$ and $A' \cup Y$ are mutually exclusive by definition. So $\forall v \in Y$,

$$|V_{k,A\cup A'} \cap N(v)| \geq k$$
$$|(V_{k,A} \cup A' \cup Y) \cap N(v)| \geq k$$
$$|((A' \cup Y) \cap N(v))| \geq \delta(v|A).$$

By definition of residual core, we can now see that Y is the residual core with anchors A'. Therefore, $\mathcal{F}(A \cup A') \setminus \mathcal{F}(A) = R^*_{A'}$. □

4.4 Bounds on the Number of Anchors Let G_f be the graph induced from G by the nodes in C_f, and let $\mathcal{G}$ be the set of connected components in G_f. If nodes in $G' \in \mathcal{G}$ become followers, they cannot effect the residual degree of nodes in other components. Thus, we can consider each component separately.

[4]Refer to the supplemental material for the proof of the correctness of the algorithm.

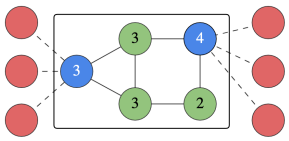


Figure 3: The nodes inside the box form G', and the number represents their residual degrees. The red nodes are the nodes in $C_a \setminus C_f$. The green nodes and blue nodes are V'_i and V'_o respectively.

For $G' \in \mathcal{G}$, let V'_i be the set of nodes in G' that can become follower without relying on nodes not in G' (if the other nodes become followers), and let V'_o be the set of nodes in G' that need anchors not in G' to become followers. That is, $V'_i = \{v \in V' : |N(v, G')| \geq \delta(v)\}$ and $V'_o = V' \setminus V'_i$ where V' is the set of nodes in subgraph G', and $N(v, G')$ is the set of neighbors of v in G'.

If anchors A' are selected such that all the nodes in V'_o become followers, G' become a residual core, and converts V'_i to followers (Lemma 4.1).

As an example, consider Figure 3. In this example, the G' is indicated by the rectangle, and the numbers inside the nodes are the residual degrees of the nodes. The red nodes are nodes in C_a. We can see that the green nodes have at least $\delta(*)$ neighbors within G'; but the blue nodes need anchors from among the red nodes. So, the green and blue nodes form V'_i and V'_o, respectively. If the blue nodes are converted to followers, the G' becomes a residual core.

By construction, the only neighbors of V'_o not in G' are in $C_a \setminus C_f$. It can be seen that each node $v \in V'_o$ needs $\delta(v) - |N(v, G')|$ anchors from $C_a \setminus C_f$ to become followers. We denote it by $\delta'(v)$. Then consider,

$$\beta^{\perp}(G') \stackrel{\text{def}}{=} \max_{v \in V'_o} \delta'(v).$$
$$\beta^{\top}(G') \stackrel{\text{def}}{=} \sum_{v \in V'_o} \delta'(v).$$
$$\beta^{*}(G') \stackrel{\text{def}}{=} \min_{v \in V'_o} \delta'(v).$$

If the remaining anchor budget is b', we have:

- $b' \geq \beta^{\top}(G')$. All the nodes in C' *can* be converted to followers.
- $b' < \beta^{\perp}(G')$. The budget is not enough to convert all the nodes in G' to followers. But it might be possible for some nodes to become followers.
- $b' < \beta^{*}(G')$. No node in G' can become a follower.

For a given component, depending on these case, we need different anchor selection strategies.

4.5 Residual Anchor Selection If the anchor budget remaining is enough to convert all nodes in G' to followers, we need to select the minimum number of an-

Algorithm 2 `ResidualAnchors()`

$A' \leftarrow \emptyset$
$T \leftarrow V'_o$
while $T \neq \emptyset$ **do**
  $v \leftarrow \underset{u \in C_a \setminus (C_f \cup A')}{\arg\max} |N(u) \cap T|$
  $A' \leftarrow A' \cup \{v\}$
  $T \leftarrow \{u \in T : \delta'(u, G') > |N(u) \cap A'|\}$
end while
return $\{(A', V')\}$

chors needed. Since the nodes in V'_i already have enough neighbors in G', it is enough to consider only V'_o.

We thus need to select the minimum number of anchors from $C_a \setminus C_f$ such that each node $v \in V'_o$ is connected to at least $\delta'(v)$ anchors.

Finding the minimum number of residual anchors is NP-hard,[5] and so we propose a heuristic algorithm for this task (Algorithm 2). At each step, the algorithm selects the node from $C_a \setminus (C_f \cup A')$ that has the most neighbors in T, and adds it to A'. Here T is the set of nodes such that all the nodes in T still requires additional anchors to become followers.

4.6 Anchor Score based Anchors Selection If the anchor budget is not enough to convert all the nodes in G' to followers, we want to convert as many as possible. To quantify the quality of a candidate anchor node with respect to maximizing the number of followers we propose a node-level measure called the *Anchor Score*. Denote all the nodes in G' by C'_f, and consider C'_a such that $C'_a = \{v \in C_a : N(v) \cap C'_f \neq \emptyset\}$.

Then, we define the Anchor Score of $v \in C'_f \cup C'_a$ as

$$\alpha(v) \stackrel{\text{def}}{=} 1 + \sum_{u \in C_f \cap N(v)} \frac{\alpha(u)}{\delta(u)}. \tag{4.6}$$

The intuition is that nodes that are connected to others with high anchor score and low residual degree are important themselves. If nodes with high anchor scores are anchored, this helps in converting its neighbors into followers, which may themselves also be important.

To calculate the anchor scores of all nodes in $C'_f \cup C'_a$, we have $|C'_f \cup C'_a|$ equations:

$$\mathbf{q} = \mathbf{1} + \boldsymbol{D}\mathbf{q}, \tag{4.7}$$

where $\mathbf{q}$ is the vector of anchor scores, $\mathbf{1}$ is a vector of 1's, and $\boldsymbol{D}$ is a matrix such that $\boldsymbol{D}_{i,j} = \frac{1}{\delta(j)}$ if edge (i, j) exist, otherwise 0.

Depending on the membership of a node in C'_a and/or C'_f, we have the following conditions:

[5] Please refer to the supplementary material for the proof.

Algorithm 3 `ASAnchors()`

$A', F', \mathbb{S} \leftarrow \emptyset, \emptyset, \emptyset$
while $|A'| < b$ **do**
  Calculate the Anchor Scores $\alpha(*)$
  $v \leftarrow \underset{u \in C'_f \cup C'_a}{\arg\max}\ \alpha(u)$
  $R \leftarrow$ `FindResidualCore`$(A' \cup \{u\})$
  $A' \leftarrow A' \cup \{v\}$
  $F' \leftarrow F' \cup R$
  $\mathbb{S} \leftarrow \mathbb{S} \cup \{(A', F')\}$
  Remove R and v from C'_a and C'_f
  Update $\delta(*)$
end while
return $\mathbb{S}$

1. $v \in C'_f \setminus C'_a$. Since $C'_f \cap N(v) = \emptyset$ by definition, $\alpha(v) = 1$.
2. $v \in C'_f \cap C'_a$. In this case, $\alpha(v)$ appears on both sides of equation 4.7.
3. $v \in C'_a \setminus C'_f$. Here, v cannot appear on the right of the equation. So, $\alpha(v)$ is simple to calculate once the other two cases have been calculated.

To compute anchor scores, we first set the score for $C'_f \setminus C'_a$ to 1. We next restrict computation of Equation (4.7) to only the nodes in $C'_f \cap C'_a$, and calculate the anchor scores. Finally, we calculate the anchor scores of $C'_a \setminus C'_f$ using Equation (4.6) and the previously calculated anchor scores.

After calculating the anchor scores, the node with the highest value is selected as the next anchor. The process repeats as long as there is budget left. Algorithm 3 describes this process.

4.7 Residual Core Maximization In this section, we put together the pieces of our proposed algorithm *Residual Core Maximization* (`RCM`). The main idea of `RCM` is to divide the graph into multiple connected components of C_f, and then to find anchors for these subgraphs separately depending on $\beta^{\top}(G')$ (Section 4.4). Algorithm 4 describes `RCM` in detail.

The first step is to generate $\mathcal{G}$, the connected components of the subgraph induced with C_f. `RCM` then generates the (anchors, followers) tuples for the components, denoted by $\mathbb{S}$. This step can be performed in parallel. Next, we need to find a set A such that,

$$\hat{\mathbb{S}} = \left\{ \mathbb{S}' \subseteq \mathbb{S} : \left| \bigcup_{S \in \mathbb{S}'} S\texttt{[0]} \right| \leq b \right\}$$

$$\mathbb{S}^* = \underset{\mathbb{S} \in \hat{\mathbb{S}}}{\arg\max} \left| \bigcup_{S \in \mathbb{S}} S\texttt{[1]} \right|,$$

where $S\texttt{[}i\texttt{]}$ denotes the i-the element in the tuple S. This

Algorithm 4 `ResidualCoreMaximization()`

$A, \mathbb{S} \leftarrow \emptyset, \emptyset$
Find C_a, C_f and calculate $\delta(*)$
$\mathcal{G} \leftarrow$ Connected components in G_f
for G' in $\mathcal{G}$ **do**
  if $\beta^*(G') > b$ **then**
    continue
  else if $\beta^{\perp}(G') > b$ **then**
    $\mathbb{S} \leftarrow \mathbb{S} \cup$ `ASAnchors`(G')
  else if $\beta^{\perp}(G') \leq b$ **then**
    $\mathbb{S} \leftarrow \mathbb{S} \cup$ `ResidualAnchors`(G')
  else
    $\mathbb{S} \leftarrow \mathbb{S} \cup$ `ResidualAnchors`(G')
    $\mathbb{S} \leftarrow \mathbb{S} \cup$ `ASAnchors`(G')
  end if
end for
$A \leftarrow$`SolutionSelection`$(\mathbb{S}, b)$
return A

Algorithm 5 `SolutionSelection()`

$A, F \leftarrow \emptyset, \emptyset$
while $|A| < b$ **do**
  $S^* \leftarrow \arg\max_{S \in \mathbb{S}} \frac{|S[1] \setminus F|}{|S[0] \setminus A|}$
  $\mathbb{S}$`.remove`(S^*)
  if $|A \cup S^*[0]| \leq b$ **then**
    $A \leftarrow A \cup S^*[0]$
    $F \leftarrow F \cup S^*[1]$
  end if
end while
return A

problem is close to the set union knapsack problem.[6] So, we use a greedy algorithm that selects $S^* \in \mathbb{S}$ that maximizes $\frac{|S^*[1] \setminus F|}{|S^*[0] \setminus A|}$, where A and F are the sets of anchors selected so far and the followers as a result. This is described in Algorithm 5.

After S^* (or the approximation) is computed, `RCM` selects anchors as, $A = \bigcup_{S \in \mathbb{S}^*} S[0]$. The source code of `RCM` is publicly available.[7]

Running Time: If E_{fa} is the set of edges in the subgraph induced from G with the nodes $C_f \cup C_a$, the running time of `RCM` is given by $O(|E_{fa}|)$[8].

5 Experiments

We evaluate the performance of `RCM` against various baselines both in finding followers and efficiency in doing that. We also compare to the optimal algorithm described by Bhawalkar *et al.* [5] for $k = 2$.

[6] The set union knapsack problem is a generalization of the knapsack problem in which the weight is calculated based on union of sets rather than sum of numbers [13]. In our problem, the value is also calculated based on set unions.

[7] `https://github.com/rlaishra/RCM`

[8] Please refer to the supplemental material for details.

Table 1 lists the real-world networks used in our experiments. These datasets are available at Network Repository[9] and SNAP.[10] We consider social, web, and collaboration networks of various sizes – ranging from a few thousands to more 1 million edges. All the graphs are treated as undirected. Unless otherwise stated, we use only the sequential version of `RCM` in the following discussion and results.

5.1 Comparison Against Baseline Algorithms

We consider three baseline algorithms for finding anchor nodes. The first is `OLAK`, the current state-of-the-art algorithm for anchor nodes selection [28]. `OLAK` greedily selects one anchor node at a time, and recomputes the anchored k-core decomposition in each step. `OLAK` has been demonstrated to work well on a number of real-world networks. For fair running time comparison, we implement `OLAK` in Python. The second baseline is *Maximum Degree* (`MD`) in which a node from C_a that has the maximum number of neighbors in C_f is selected as anchor as anchor at each step. The third baseline is *Random* (`RND`), which selects anchors randomly from C_f. In all baselines, after an anchor node has been selected, the new anchor and followers are removed from C_a and C_f. We set k to the median core number of the network (given in Table 1) and vary the anchor budget from 50 to 250 in increments of 50. Results for different values of k are included in the supplementary material.

Figure 4a shows the number of followers for varying budgets for some selected networks and Figure 4b shows the followers at $b = 250$ for `RCM` and the best baseline on all networks. `RCM`, shown in red, clearly outperforms all the baselines. As expected, the results are closer to `OLAK` for lower budgets, but the difference increases for higher budgets. Among the baselines, no single algorithm is always the best. The results for all baselines are in the supplementary material. We observe that `RCM` outperform the baselines in all the cases considered.

To compare the runtime efficiency of the various algorithms, we consider the time to find each follower. Figure 5a shows the time to to find a follower against the budget and Figure 5b shows the result for `RCM` and the best baseline[11] for all the network at $b = 250$. In all the cases `RCM` is much faster than all the baselines. Note that in many algorithms, the average time to find a follower drops as the budget increases because the size of C_a and C_f drops (as nodes become followers and anchors).

5.2 Comparison with Optimal Solution

In this section, we compare the performance of `RCM` against

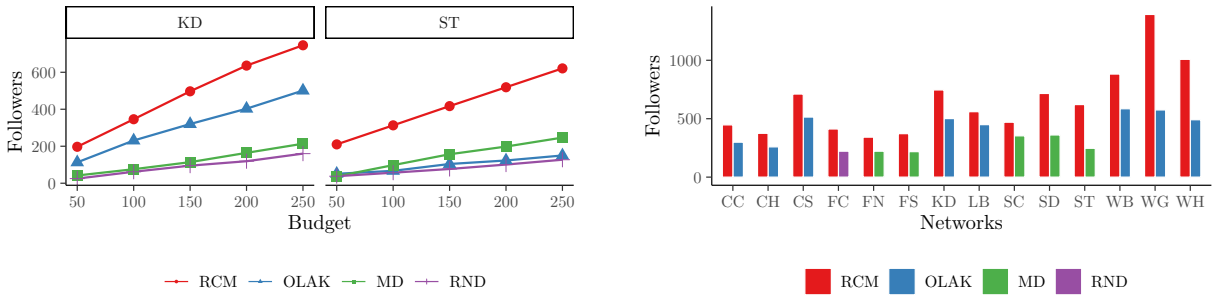
[9] `http://networkrepository.com`

[10] `https://snap.stanford.edu/data/index.html`

[11] Results for all the baselines are in the supplementary material.

Network	Abbr.	$\lvert V\rvert$	$\lvert E\rvert$	k_{max}	k_{mid}	$\lvert C_a\rvert$	$\lvert C_f\rvert$	$\lvert E_{fa}\rvert$	$\lvert\mathcal{G}\rvert$
socfb-combined[10]	FC	4.0×10^3	8.8×10^4	115	17	1289	501	7029	13
ca-CondMat[10]	CC	2.3×10^4	9.3×10^4	25	4	2892	1179	3739	685
ca-HepPh[10]	CH	1.2×10^4	1.1×10^5	238	4	1487	634	1901	362
loc-Brightkite[10]	LB	5.8×10^4	2.1×10^5	52	2	3288	2365	3004	2006
socfb-Northeastern19[9]	FN	1.4×10^4	3.8×10^5	43	33	4978	1246	18473	1
socfb-Syracuse56[9]	FS	1.4×10^4	5.4×10^5	75	46	5522	1417	37698	2
ca-citeseer[1]	CS	2.2×10^5	8.1×10^4	86	3	18486	8493	20187	5991
loc-Gowalla[10]	LG	1.9×10^5	9.5×10^5	51	3	17890	10263	17706	7479
com-DBLP[10]	KD	3.1×10^5	1.0×10^6	113	3	23182	11010	25144	8240
web-Google[10]	WG	8.7×10^5	4.3×10^6	44	4	198014	46891	245188	20471
soc-Catster[9]	SC	1.5×10^5	5.4×10^6	419	21	5285	2003	8428	1054
soc-Dogster[9]	SD	4.2×10^5	8.5×10^6	248	12	20887	8750	26438	5339
soc-TwitterHiggs[9]	ST	4.5×10^5	1.3×10^7	125	17	27146	9234	40651	3493
web-Hudong[9]	WH	2.0×10^6	1.4×10^7	266	5	82791	40160	83886	29687
web-BaiduBaike[9]	WB	2.0×10^6	1.7×10^7	78	3	51659	32501	50222	27735

Table 1: Statistics of the real-world networks used in our experiments. $|V|$ and $|E|$ are the number of nodes and edges respectively; k_{max} and k_{mid} are the maximum and median values of the coreness of all the nodes. $|C_a|$ and $|C_f|$ are sizes of the candidate anchors and followers for k_{mid}. $|E_{fa}|$ is the number of edges in the subgraph induced with $C_f \cup C_a$, and $|\mathcal{G}|$ is the number of connected components.



(a) Followers vs Budget.

(b) Followers at $b = 250$.

Figure 4: Number of followers found by RCM and various baselines (at k fixed at the median value). In Figure 4a, the number of followers against the budget is shown for some selected networks. In 4b, the number of followers at $b = 250$ for all the networks considered is shown. Only RCM and the best baseline is shown. We can see that RCM selects the anchors that result in the largest number of followers in all cases. (**Higher values are better.**)

the optimal solution. Bhawalkar *et al.* [5] proposed an algorithm for finding the optimal solution for $k \leq 2$. We also include OLAK in the comparison. For these experiments we consider $k = 2$ and $b = 50$.

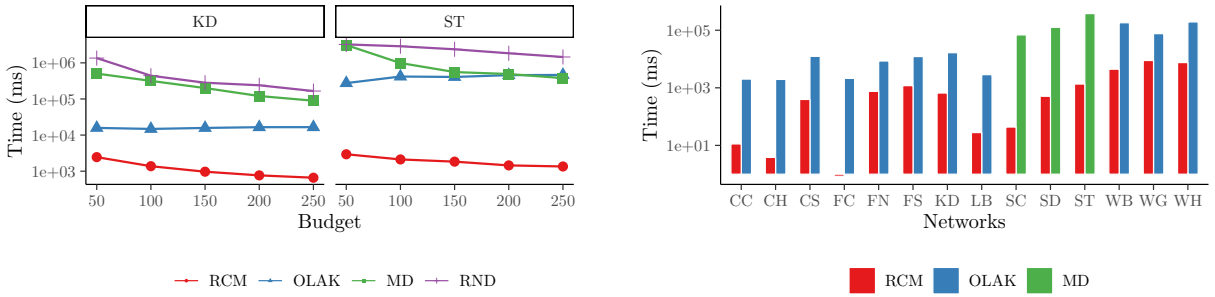
Table 2 shows the comparison between RCM, OPT and OLAK. In all cases, the number of followers due to RCM is very close to that found by OPT. The followers due to OLAK are much fewer in all the networks. Additionally, RCM is around 100 times faster than OPT.

5.3 Experimental Analysis of RCM In this section, we evaluate the various aspects of RCM – (a) the contribution of AnchorScore() and ResidualAnchors() to the overall performance, (b) the speedup due to parallelization, and (c) scalability with network size.

We evaluate the contribution of ResidualAnchors() and ASAnchors() by designing versions of RCM that use only one of them. We denote these as RCM-RC and RCM-AS respectively. Results are shown in Figure 6a. We observe that results are clearly better when we use both ResidualCore() and ASAnchors(). Additionally, RCM-RC outperforms RCM-AS in two out of the three networks. RCM-AS outperforms RCM-RC in the network FS because $|\mathcal{G}| = 2$ and the budget is not enough to completely convert any component to followers.

To evaluate the speedup due to parallelization (Section 4.7), we limit the number of CPU cores available and compare the computation time.[12] Figure 6b shows the results of this experiment. In most networks RCM achieves significant speedup with CPU cores. However,

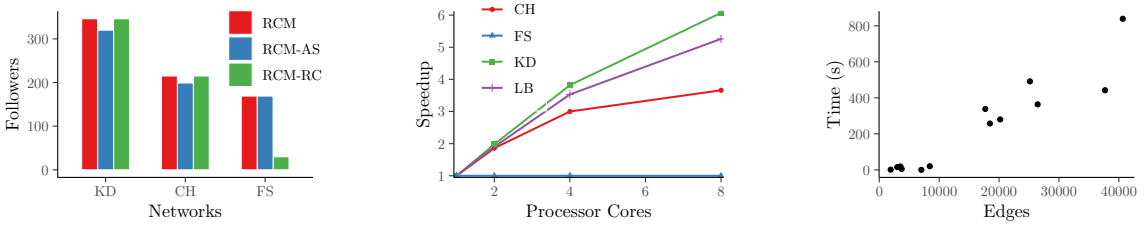
[12]The k value is given in Table 1 and $b = 100$.



(a) Time to find a follower vs budget.

(b) Time to find a follower at $b = 250$.

Figure 5: Average time to find a follower by RCM and baselines. In Figure 5a, the the time at different budgets is given for selected networks, and in Figure 5b the time at $b = 250$ is shown for RCM and the best baseline. The value of k is given in Table 1. RCM is much faster than the baselines in all the cases. (**Lower values are better.**)



(a) Follower count due to RCM, RCM-RC and RCM-AS.

(b) Speed up due to parallel computation.

(c) Running time of RCM against $|E_{fa}|$.

Figure 6: Experimental results for analysis of RCM. Figure 6a shows the contribution of different parts of RCM, Figure 6b shows the speedup due to parallel computation, and Figure 6c shows the running time against $|E_{fa}|$.

Network	k	b	Alg.	Followers	Time (ms)
KD	2	50	RCM	114	1.2×10^2
			OPT	115	1.9×10^4
			OLAK	97	1.5×10^4
LG	2	50	RCM	150	3.5×10^2
			OPT	152	2.9×10^4
			OLAK	133	9.6×10^3
WG	2	50	RCM	180	3.9×10^3
			OPT	186	2.6×10^5
			OLAK	95	6.2×10^4

Table 2: Comparison of RCM, OPT and OLAK. Observe that in all the cases, RCM is very close the OPT while being multiple magnitudes faster.

in the case of FS network there is no speedup. This is because there are only two components – a large one and a very small one, making parallelization ineffective.

As described in Section 4.7 the runtime of RCM is given by $O(|E_{fa}|)$, where E_{fa} is the set of edges in the subgraph induced by $C_f \cup C_a$. Figure 6c shows the running time of RCM against $|E_{fa}|$ for all the networks in Table 1. As expected, the observed runtime is approximately linear in $|E_{fa}|$.

6 Conclusions

In tis paper, we addressed the anchored k-core problem: *given an anchor budget, what is the set of anchor nodes that should be selected to maximize the number of followers?* We proposed a method, called *Residual Core Maximization* (RCM). Through extensive experimental analysis, we demonstrate that RCM performs significantly better than the state-of-the-art algorithms. On average, RCM finds 1.65 times the followers found by the best baseline method, while being 500 times faster. We also compared RCM against the optimal solution (for $k = 2$) and observed that the number of followers found by RCM is very close to the optimal; and the time to find each follower is around 100 times faster.

Acknowledgments

Laishram and Soundarajan were supported by Army Research Office award W911NF-18-1-0047. Soundarajan was additionally supported by NSF award 1908048. Sariyuce was supported by NSF-1910063. Eliassi-Rad was supported in parts by NSF CNS-1314603, NSF IIS-1741197, the Combat Capabilities Development Command Army Research Laboratory under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Se-

curity CRA), and the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes not withstanding any copyright notation here on.

References

[1] M. A. Al-garadi, K. D. Varathan, and S. D. Ravana, *Identification of influential spreaders in online social networks using interaction weighted k-core decomposition method*, Physica A, 468 (2017).

[2] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, *Large scale networks fingerprinting and visualization using the k-core decomposition*, in NIPS, 2005.

[3] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, *K-core decomposition of Internet graphs: hierarchies, self-similarity and measurement biases*, Networks and Heterogeneous Media, 3 (2008).

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

[5] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma, *Preventing unraveling in social networks: the anchored k-core problem*, in ICLAP, 2012.

[6] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma, *Preventing unraveling in social networks: the anchored k-core problem*, SIDMA, 29 (2015).

[7] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir, *A model of internet topology using k-shell decomposition*, PNAS, 104 (2007), pp. 11150–11154.

[8] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, *Efficient core decomposition in massive networks*, in ICDE, 2011.

[9] H. Esfandiari, S. Lattanzi, and V. S. Mirrokni, *Parallel and streaming algorithms for k-core decomposition*, in ICML, 2018.

[10] D. Garcia, P. Mavrodiev, and F. Schweitzer, *Social resilience in online communities: The autopsy of friendster*, in COSN, 2013.

[11] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis, *Evaluating cooperation in communities with the k-core structure*, in ASONAM, 2011.

[12] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis, *D-cores: measuring collaboration of directed graphs based on degeneracy*, KAIS, 35 (2013).

[13] O. Goldschmidt, D. Nehme, and G. Yu, *Note: On the set-union knapsack problem*, Naval Research Logistics (NRL), 41 (1994).

[14] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo, *K-core decomposition of large networks on a single pc*, PVLDB, 9 (2015).

[15] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse, *Identification of influential spreaders in complex networks*, Nature physics, 6 (2010).

[16] R. Laishram, A. E. Sariyüce, T. Eliassi-Rad, A. Pinar, and S. Soundarajan, *Measuring and improving the core resilience of networks*, in WWW, 2018.

[17] R.-H. Li, J. X. Yu, and R. Mao, *Efficient core maintenance in large dynamic graphs*, TKDE, 26 (2014).

[18] Y. Liu, M. Tang, T. Zhou, and Y. Do, *Core-like groups result in invalidation of identifying super-spreader by k-shell decomposition*, Scientific reports, 5 (2015).

[19] D. Matula and L. Beck, *Smallest-last ordering and clustering and graph coloring algorithms*, JACM, 30 (1983).

[20] S. Medya, T. Ma, A. Silva, and A. Singh, *K-core minimization: A game theoretic approach*, arXiv preprint arXiv:1901.02166, (2019).

[21] A. Montresor, F. De Pellegrini, and D. Miorandi, *Distributed k-core decomposition*, TPDS, 24 (2013).

[22] C. Peng, T. G. Kolda, and A. Pinar, *Accelerating community detection by using k-core subgraphs*, arXiv preprint arXiv:1403.2226, (2014).

[23] A. E. Sariyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek, *Streaming algorithms for k-core decomposition*, PVLDB, 6 (2013).

[24] A. E. Sariyüce and A. Pinar, *Fast hierarchy construction for dense subgraphs*, PVLDB, 10 (2016).

[25] A. E. Sariyuce, C. Seshadhri, A. Pinar, and U. V. Catalyurek, *Finding the hierarchy of dense subgraphs using nucleus decompositions*, in WWW, 2015.

[26] S. B. Seidman, *Network structure and minimum degree*, Social networks, 5 (1983).

[27] K. Shin, T. Eliassi-Rad, and C. Faloutsos, *Corescope: Graph mining using k-core analysis - patterns, anomalies and algorithms*, in ICDM, 2016.

[28] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin, *Olak: an efficient algorithm to prevent unraveling in social networks*, PVLDB, 10 (2017).

[29] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, *Finding critical users for social network engagement: The collapsed k-core problem*, in AAAI, 2017.

[30] Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin, *A fast order-based approach for core maintenance*, in ICDE, 2017.

[31] F. Zhao and A. K. Tung, *Large scale cohesive subgraphs discovery for social network visual analysis*, PVLDB, 6 (2012).

[32] Z. Zhou, F. Zhang, X. Lin, W. Zhang, and C. Chen, *K-core maximization: An edge addition approach*, in AAAI, 2019.