Fast Algorithms for Core Maximization on Large Graphs

Xin Sun
College of Intelligence and Computing, Tianjin University
Tianjin, China
sun_xin@tju.edu.cn

Xin Huang
Hong Kong Baptist University
Hong Kong, China
xinhuang@comp.hkbu.edu.hk

Di Jin
College of Intelligence and Computing, Tianjin University
Tianjin, China
jindi@tju.edu.cn

ABSTRACT

Core maximization, that enlarges the k-core as much as possible by inserting a few new edges into a graph, is particularly useful for social group engagement and network stability improvement. However, the core maximization problem has been theoretically proven to be NP-hard even APX-hard for $k \geq 3$. Existing heuristic approaches suffer from the limitation of inefficiency on large graphs. To address this limitation, in this paper, we revisit this challenging yet important problem of core maximization, that is, given a graph G, a number k, and a budget b, to insert b new edges into G such that the corresponding k-core is maximized. We propose a novel algorithm FastCM+ based on several fast search strategies. The core idea is to apply graph partition to divide $(k-1)$-shell into different components. Then, FastCM+ considers each $(k-1)$-shell component independently to convert different layered vertices into k-core, in two manners of completely and partially. Based on the complete/partial conversions, FastCM+ is generalized to further handle $(k-\lambda)$-shell conversions for $2 \leq \lambda \leq k$. Leveraging dynamic programming combinations of different components' potential answers, FastCM+ finds a good-quality answer for edge insertions. Experimental results on eleven datasets demonstrate that our algorithm runs much faster than state-of-the-art methods on large graphs meanwhile achieving better answers.

PVLDB Reference Format:
Xin Sun, Xin Huang, and Di Jin. Fast Algorithms for Core Maximization on Large Graphs. PVLDB, 15(7): 1350 - 1362, 2022.
doi:10.14778/3523210.3523214

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/sunxin18/FastCM.

1 INTRODUCTION

One fundamental task of graph analytics is detecting cohesive substructures on various complex networks, e.g., social networks, biological networks, transportation networks, and so on [18, 42]. Among different notions of cohesive subgraphs in the literature, a well-studied definition of k-core is the largest subgraph of a given graph such that every vertex has at least k neighbors in this subgraph [26, 36]. The k-core discovery enjoys an efficient computational property in linear time of graph size, which has many applications, e.g., community detection and search [12, 14, 18, 33, 41], group influence maximization [20, 35], user engagement [39], graph visualization [1, 9, 42], anomaly detection [32], interdisciplinary collaboration search [11], structural collapse prediction [28], and also network robustness analytics [10, 22]. Recently, the problem of core maximization has been studied to maximize the node size of k-core by inserting b new edges into a graph [8].

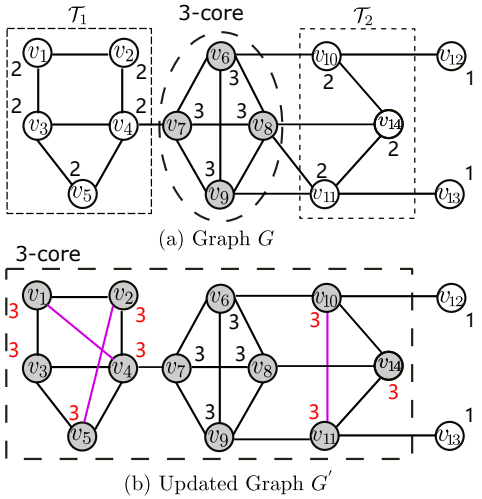


Figure 1: An example of k-core maximization in graph G. Here, $k = 3$ and the budget $b = 3$. The updated graph G' is G inserted with three new edges (v_2, v_5), (v_1, v_4), and (v_{10}, v_{11}). The 3-core has eight new vertices $\mathcal{F} = \{v_1, v_2, v_3, v_4, v_5, v_{10}, v_{11}, v_{14}\}$.

Motivating example. Figure 1(a) illustrates a graph G with 13 vertices. The whole graph G is the 1-core where every vertex has at least one neighbor. The coreness of each vertex is marked with a value nearby the vertex label in Figure 1, e.g., the coreness of v_1 is 2. The 3-core has four vertices $\{v_6, v_7, v_8, v_9\}$. Assume that $k = 3$ and the budget $b = 3$, the core maximization aims at enlarging the 3-core of G by inserting three new edges into graph G. An optimal solution is to insert three new edges $\{(v_1, v_4), (v_2, v_5), (v_{10}, v_{11})\}$ into G. The updated graph G' and the new 3-core are shown in Figure 1(b), which has eight new 3-core vertices $\mathcal{F} = \{v_1, v_2, v_3, v_4, v_5, v_{10}, v_{11}, v_{14}\}$.

Core maximization has been shown to be useful in many real applications via *adding new edges*, including the social group engagement in social networks by *creating new friendships* via a function of friend recommendation [5, 23], identifying missing defense

Di Jin is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 15, No. 7 ISSN 2150-8097.
doi:10.14778/3523210.3523214

links in military networks, topology extension in distributed computing networks, enhancing stability for resource exchanging in P2P networks [8, 44], and also improving the connectivity of flight networks by *adding new airlines* as shown in Exp-8 in Section 7.

Challenges and Contributions. In this paper, we revisit an important problem of k-core maximization by adding b new edges into graph G [8, 44, 45]. The core maximization problem is theoretically shown to be NP-hard and even APX-hard for $k \geq 3$ [44]. A greedy approach EKC proposed in [44] inserts the best candidate edges into graph G one by one until using up all of b budget. However, EKC has a high time complexity of $O(bn^2m)$ where n and m are the sizes of vertices and edges in G. Due to a large number of *edge candidates* to be considered, EKC cannot easily handle large graphs. To improve the efficiency, a faster approach VEK [45] is proposed by considering *vertex candidates* rather than *edge candidates* in $O(bnm)$ time. However, both methods suffer from another quality limitation by using a greedy strategy of *inserting one edge* or *converting one vertex* at each round of insertions, further improvements on quality and efficiency remain open.

To address the above limitations, we propose a novel algorithm FastCM+ based on $(k-1)$-shell partition and dynamic programming techniques, which allows inserting multiple edges to convert a batch of vertices at each round of insertions. The key idea is to identify candidate vertices outside of k-core and consider converting them into k-core followers level by level. First, we identify a substructure of $(k-1)$-shell that is very close to becoming k-core. We then propose two insertion strategies of *complete conversion* and *partial conversion*, which convert these $(k-1)$-shell vertices to k-core vertices under a budget b. After exploring all potential answers for each $(k-1)$-shell component, FastCM+ finally applies a dynamic programming optimization to select edge insertions and obtain good answers. If there is a remaining budget, it converts vertices from other $(k-\lambda)$-shells to followers for $\lambda \geq 2$. In summary, we make the main contributions in this paper as follows.

- We review the existing algorithms for k-core maximization and analyze their pros and cons, which motivates the design of our improved algorithms for fast core maximization (Section 4).
- We analyze the structural properties of $(k-1)$-shell and identify $(k-1)$-collapse as good candidates for edge insertion. Then, we leverage the $(k-1)$-shell partition to identify all $(k-1)$-shell components that are independent of each other. Finally, we propose the FastCM algorithm to convert each $(k-1)$-shell component to k-core *completely* (Section 5).
- We propose another novel edge insertion strategy of partially converting $(k-1)$-shell component to k-core when the budget b is insufficient for complete conversion. We propose to anchor the low-layered vertices to followers by a well-designed function of follower gain. Then, we integrate the dynamic programming techniques to combine all $(k-1)$-shell partitions' results and return the answer of edge selections. Moreover, we also extend the complete/partial $(k-1)$-shell conversions to handle $(k-\lambda)$-shell conversion for $2 \leq \lambda \leq k$ (Section 6).
- We conduct extensive experiments on large real-world graphs. The results show that our proposed approach FastCM+ can effectively enlarge k-core meanwhile running 34,000**x** and 391**x** faster than the state-of-the-art EKC [44] and VEK [45], respectively (Section 7).

We discuss related work in Section 2 and give the problem formulation in Section 3. Finally, we conclude the paper in Section 8.

2 RELATED WORK

Our work is related to *core maintenance*, *core maximization*, and *core minimization*.

Core maintenance. Core maintenance computes the vertex coreness efficiently over dynamic graphs where vertices/edges are inserted and removed, which is particularly useful for studying core structure modification [31]. Existing studies [17, 19, 31, 40] theoretically prove that the coreness change of each vertex is at most one after inserting/deleting one edge. Zhang et al. [40] propose a novel order-based approach to efficiently update the vertex corenesses for a changed graph. Core maintenance algorithms are also developed for edge-weighted graphs [24, 43]. Parallel algorithms for core maintenance are studied in [17, 19].

Core maximization and minimization. Several recent studies focus on changing the graph structure to maximizes/minimizes the size of k-core [5, 21, 22, 25, 27, 37, 38, 45, 46]. Zhang et al. [38] study the collapsed k-core problem to find critical users for social network engagement. The core minimization is to remove a small set of edges from a network such that it minimizes the k-core structure [27, 46]. [22] measures the stability of k-core under random edge/node deletions. [10] introduces a network robustness measure based on k-cores and evaluates the stability of the network under node removals. On the other hand, the anchored k-core maximization seeks to enlarge k-core by anchoring a set of nodes outside of k-core, even if these anchored vertices have less than k neighbors in the k-core [5, 21, 37]. In addition, there exist various studies that manipulate other topology patterns, e.g., reducing polarized bubble radius [16], enhancing communicability [2], improving betweenness centrality [4], coverage centrality maximization [13], eccentricity minimization [29], and truss maximization [7, 34].

3 PROBLEM FORMULATION

We consider an undirected and unweighted graph $G(V, E)$ with the set of vertices V and the set of edges E where $|V| = n$ and $|E| = m$. For a vertex $u \in V$ and a subgraph $H \subseteq G$, we denote $N_H(u)$ as the set of u's neighbors in graph H, i.e., $N_H(u) = \{v \in V(H) : (v, u) \in E(H)\}$. Alternatively, for a vertex set $P \subseteq V$, we also use the notation $N_P(u)$ to represent the set of u's neighbors in P, i.e., $N_P(u) = \{v \in P : (v, u) \in E\}$. Moreover, the cardinality $|N_H(u)|$ is called the degree of u in H. Given a vertex set $S \subseteq V$, the induced subgraph of G by S is denoted as G_S. Without loss of generality, we assume that graph G is connected, implying that $m \geq n - 1$.

3.1 K-Core and K-Shell

In the following, we give the definitions of k-core and coreness.

Definition 1 (k-core). *Given a graph G and an integer $k \in \mathbb{Z}$, the k-core, denoted by Ψ_k, is the largest subgraph of G such that each vertex has at least k neighbors within Ψ_k i.e., $\forall u \in \Psi_k, |N_{\Psi_k}(u)| \geq k$.*

Definition 2 (coreness). *The coreness of a vertex $v \in V$ is defined as the largest value k such that there exists a non-empty k-core Ψ_k containing v, denoted as $c(v) = \max\{k \in \mathbb{Z} : v \in \Psi_k\}$.*

Consider the graph G in Figure 1(a). The 3-core of G is $\Psi_3 = \{v_6, v_7, v_8, v_9\}$. There exists no 4-core in G. Thus, the coreness of vertex v_6 is $c(v_6) = 3$. The core decomposition [3] is to compute the coreness of all vertices v in graph G. The general idea of core decomposition is to iteratively remove the smallest-degree vertex and set its coreness as its degree at the time of removal. The vertices with the same coreness form the k-shell as follows.

Definition 3 ($(k-1)$-shell). *The $(k-1)$-shell of graph G, denoted by S_{k-1}, is the set of vertices that have exactly the coreness of $k-1$, i.e., $S_{k-1} = \{v \in V : c(v) = k-1\}$.*

By Def. 3, a vertex that belongs to $(k-1)$-shell, appears in $(k-1)$-core but not in k-core. Consider the graph G in Figure 1(a), there are the 1-shell $S_1 = \{v_{12}, v_{13}\}$, the 2-shell $S_2 = \{v_1, v_2, v_3, v_4, v_5, v_{10}, v_{11}, v_{14}\}$, and the 3-shell $S_3 = \{v_6, v_7, v_8, v_9\}$.

3.2 Problem Formulation

In this paper, we investigate the problem of core maximization, which aims at inserting new edges $\hat{E}$ into $G(V, E)$ such that the k-core of the new graph $G(V, \hat{E} \cup E)$ becomes larger.

Problem 1 (**Core Maximization**). *Given a graph $G(V, E)$, an integer $k \in \mathbb{Z}^+$ and a budget $b \in \mathbb{Z}^+$, the problem is to insert a set of new edges $\hat{E}$ into G such that the k-core of new graph $G(V, E \cup \hat{E})$ (denoted as $\hat{\Psi}_k$) is the largest. Equivalently,*

$$\hat{E} = \underset{|\hat{E}| \le b, \hat{E} \cap E = \emptyset}{\arg\max} |\hat{\Psi}_k| - |\Psi_k|.$$

Candidates and followers. The edges $\hat{E}$ that may enlarge the k-core after insertions are called *candidate edges*. For a vertex $v \in V$, if v has the coreness $c(v) < k$ in graph G, but v has the coreness $\hat{c}(v) \ge k$ in the new graph $G(V, E \cup \hat{E})$, we say that v is a *k-core follower*. The set of followers is defined as $\mathcal{F} = \hat{\Psi}_k \setminus \Psi_k = \{v \in V : v \notin \Psi_k, v \in \hat{\Psi}_k\}$. Alternatively, the problem of core maximization is to increase the most k-core followers $\mathcal{F}$.

Example 1. *Assume that the graph G in Figure 1, $k = 3$, and $b = 2$. The best answer is to insert $\hat{E} = \{(v_1, v_4), (v_2, v_5)\}$ edges into G, which brings five 3-core followers $\mathcal{F} = \hat{\Psi}_k \setminus \Psi_k = \{v_1, v_2, v_3, v_4, v_5\}$ as $\Psi_3 = \{v_6, v_7, v_8, v_9\}$ and $\hat{\Psi}_3 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$. However, another solution $\hat{E} = \{(v_1, v_4), (v_2, v_3)\}$ only brings four followers, which is a suboptimal solution. Even worse, the insertion of two edges $\hat{E} = \{(v_1, v_4), (v_2, v_6)\}$ cannot enlarge the 3-core at all.*

Hardness. We present the problem hardness as follows.

Corollary 1. *For two integers $k \ge 3$ and $b \ge 1$, the problem of core maximization is NP-*hard *and even APX-*hard*, which cannot be approximated in polynomial time within a ratio of $(1 - \frac{1}{e} + \epsilon)$ for any small $\epsilon > 0$, unless $P = NP$ [44].*

Given the *APX*-hardness of core maximization problem and the non-submodularity of its objective function $f_G(\hat{E}) = |\hat{\Psi}_k| - |\Psi_k|$ [44], it shows non-trivial challenges to develop polynomial-time algorithms with approximate quality guarantee.

4 EXISTING ALGORITHMS AND OUR IDEAS

In this section, we review the existing heuristic solutions for k-core maximization and identify their limitations [44, 45]. Then, we present an overview of our solution framework.

4.1 Limitations of Existing Algorithms

EKC [44] and VEK [45] address the core maximization problem by inserting new edges. EKC [44] adopts a greedy strategy to iteratively find the best candidate edge. Although several optimization techniques are proposed to improve efficiency, EKC is inefficient on large-scale graphs [45]. Thus, Zhou et al. [45] propose a vertex-oriented algorithm VEK faster than EKC. The idea of VEK is to convert one best candidate vertex to a follower by their proposed scoring function in each iteration. However, both methods still suffer from two major limitations as follows.

Limitation 1: high time complexities. EKC [44] takes $O(bmn^2)$ time and the faster algorithm VEK [45] still takes $O(bmn)$ time. The time complexities may be hard to be scalable on large-scale graphs with millions of vertices and edges. The reason is two-fold. EKC needs to consider a huge number of valid candidate edges in $V \times V \setminus E$. For example, there exist 3,114,351 candidate edges in EKC for enlarging the 10-core of a social network Twitter as shown in Table 2. Second, computing followers for each candidate edge/vertex by EKC and VEK is also time-consuming.

Limitation 2: poor-quality results and a limited pool of target followers. EKC adopts the greedy strategy to insert edges one by one, *without considering inserting multiple edges simultaneously*, which may lead to a poor-quality answer. For example, consider the graph G in Figure 1(a), $k = 3$ and $b = 2$. In the first iteration, EKC finds the best candidate edge $e = \{(v_{10}, v_{11})\}$ and three followers $\{v_{10}, v_{11}, v_{14}\}$. However, no more candidate edge can be chosen in the next iteration. Similarly, VEK seeks to convert vertices one by one, *without considering converting multiple vertices simultaneously*. VEK returns four followers $\mathcal{F} = \{v_4, v_{10}, v_{11}, v_{14}\}$. Thus, both EKC and VEK *fail to find the best answer* as shown in Example 1. Moreover, both approaches only consider the target followers of vertices in $(k-1)$-shell, without the consideration of converting other $(k-\lambda)$-shells into k-core for $2 \le \lambda \le k$. The limited target pool of EKC and VEK also neglects that a special instance may happen with an empty $(k-1)$-shell for large k and b. Therefore, the core maximization results of EKC and VEK can be poor.

To address the above shortcomings, it motivates us to develop faster and more effective core maximization algorithms.

4.2 Our Proposed Framework

In this section, we present our framework of fast core maximization (FastCM+). In general, FastCM+ is a heuristic algorithm focusing on the development of fast search strategies to identify important vertices and convert them to followers in batch.

Overview. The FastCM+ algorithm has four phases:

(1) $(k-1)$-shell partition phase: This phase partitions $(k-1)$-shell into several disjoint components. Each component can be dealt with edge insertions for core conversion independently. Moreover, it identifies a few key vertices in $(k-1)$-collapse for complete k-core conversion (Sections 5.1& 5.2).

(2) Converting $(k-1)$-shell to k-core phase: This phase explores to add new edges between the vertices of each $(k-1)$-shell component to convert these vertices to followers. We propose efficient insertion strategies that consume a small budget to complete the conversion. However, the input budget b may still be not enough for a full conversion in some components. Specifically, we consider two cases:
   - Case I: converting a $(k-1)$-shell component to k-core completely (Section 5.3).
   - Case II: converting a partial of vertices in a $(k-1)$-shell component to k-core partially (Sections 6.1& 6.2).

(3) Dynamic programming based edge selection phase: Based on the above exploration of conversion cases, this phase uses b new edges to carefully select the components for edge insertions, leveraging dynamic programming optimizations to achieve the most followers (Section 6.3).

(4) Converting $(k-\lambda)$-shell to k-core phase: This phase extends the techniques of $(k-1)$-shell conversion to handle $(k-\lambda)$-shell conversion for $\lambda \geq 2$. It further enlarges the k-core for a budget surplus after the complete conversation of whole $(k-1)$-shell (Section 6.4).

Integrating all the above techniques, the whole FastCM+ algorithm is presented in Section 6.5.

5 FastCM USING COMPLETE CONVERSION

In this section, we present a fast algorithm of core maximization, which first finds the edge candidates in $(k-1)$-shell and then inserts b edges for full conversions to enlarge k-core.

5.1 $(k-1)$-Collapse

We first analyze the structural properties of $(k-1)$-shell and identify a good candidate of $(k-1)$-collapse.

The vertices $v \in S_{k-1}$ with the coreness $c(v) = k-1$ are good candidates to be inserted with new edges, as they have only one coreness difference gap to become k-core follower in a very close way. Instead of randomly inserting edges associated with $(k-1)$-shell vertices, it needs to carefully select the edge candidates for effectively increasing the degrees of $(k-1)$-shell vertices and their corresponding corenesses. Specifically, we discuss the degree of $(k-1)$-shell vertex $v \in S_{k-1}$ in $(k-1)$-core Ψ_{k-1}, i.e., $|N_{\Psi_{k-1}}(v)|$. First, it clearly holds that $|N_{\Psi_{k-1}}(v)| \geq k-1$, as the coreness $c(v) \geq k-1$. On the other hand, according to k-core in Def. 1, the vertices of k-core have at least k neighbors in k-core. As the $(k-1)$-shell vertex $v \notin \Psi_k$, we have $|N_{\Psi_k}(v)| < k$. However, it is common that there exist $(k-1)$-shell vertices having enough k neighbors within $(k-1)$-core Ψ_{k-1}, i.e., $|N_{\Psi_{k-1}}(v)| \geq k$. For example, consider the vertices v_3 and v_4 in Figure 1(a) and $k=3$, both v_3 and v_4 have three neighbors within 2-core of graph G, e.g., $|N_{\Psi_2}(v_3)| = |\{v_1, v_4, v_5\}| \geq 3$. However, they fail to be contained in 3-core due to the peeling property of other 2-shell neighbors v_1, v_2, and v_5, as they have exact 2 neighbors within 2-core, e.g., $|N_{\Psi_2}(v_1)| = |\{v_2, v_3\}| = 2 < 3$. Therefore, we divide the $(k-1)$-shell vertices $v \in S_{k-1}$ into two categories: $|N_{\Psi_{k-1}}(v)| = k-1$ and $|N_{\Psi_{k-1}}(v)| \geq k$. The first one is that the $(k-1)$-shell vertex has *exact* $k-1$ *neighbors* within Ψ_{k-1}, defined as $(k-1)$-*collapse* as follows.

Definition 4 ($(k-1)$-Collapse). *The $(k-1)$-collapse of graph G, denoted by D_{k-1}, is a set of $(k-1)$-shell vertices that have exactly $k-1$ neighbors in $(k-1)$-core, i.e., $D_{k-1} = \{v \in S_{k-1} : |N_{\Psi_{k-1}}(v)| = k-1\}$.*

Example 2. *Consider the graph G in Figure 1(a) and $k=3$. The $(k-1)$-collapse is $D_{k-1} = \{v_1, v_2, v_5, v_{10}, v_{11}\}$.*

The $(k-1)$-collapse D_{k-1} is the first batch to be deleted from $(k-1)$-shell S_{k-1} in the process of k-core decomposition. The departure of D_{k-1} causes the departure of other vertices $u \in S_{k-1}$ with $|N_{\Psi_{k-1}}(u)| \geq k$. The key idea of fast k-core maximization is to insert news edges incident to vertices D_{k-1}.

5.2 $(k-1)$-Shell Partition

Next, we develop a novel strategy of $(k-1)$-shell partition, which divides the $(k-1)$-shell into multiple independent components based on the graph connectivity. The $(k-1)$-shell partition is effective for *inserting candidate edges associated with $(k-1)$-collapse*, based on the following three important observations.

- **Observation 1**: There exists a large number of edge candidates associated with $(k-1)$-collapse in $D_{k-1} \times V \setminus E$, with regard to a limited budget b, i.e., $b \ll |D_{k-1} \times V \setminus E|$. This desires a significant pruning of edge candidates in the search space.
- **Observation 2**: If a $(k-1)$-collapse vertex has no increment in degree after b edge insertions, it may cause a cascade degree decrease, leading to a waste of all b edge insertions. For example, consider $k=3$ and $b=1$, an edge insertion of (v_1, v_5) into G in Figure 1(a), leads to no k-core follower. As the $(k-1)$-collapse vertex v_2 still has the degree of 2, its departure from 3-core leads the further departure of v_1, v_3, v_4, and v_5.
- **Observation 3**: The edge insertions have heavy *locality* for k-core maximization. Continue the graph G in Figure 1(a), if we insert two edges $\{(v_1, v_4), (v_2, v_5))\}$ into G, v_{10}, v_{11}, and v_{14} cannot become k-core followers. This suggests that we can partition $(k-1)$-shell into different components and consider the edge insertion in each component independently.

Motivated by these observations, we make a $(k-1)$-shell partition on the induced subgraph of G by S_{k-1} as $G_{S_{k-1}}$ below.

Definition 5 ($(k-1)$-Shell Partition). *The $(k-1)$-shell partition is the set of h components in graph $G_{S_{k-1}}$, denoted as $\mathcal{P}(S_{k-1}) = \{\mathcal{T}_1, \ldots, \mathcal{T}_h\}$, where $\bigcup_{i=1}^{h} \mathcal{T}_i = S_{k-1}$ and $|\mathcal{P}(S_{k-1})| = h$.*

For two different components $\mathcal{T}_i$ and $\mathcal{T}_j$ where $1 \leq i, j \leq h$, $\mathcal{T}_i$ and $\mathcal{T}_j$ are disconnected in $G_{S_{k-1}}$. Thus, the two vertex sets $\mathcal{T}_i$ and $\mathcal{T}_j$ are also disjoint, i.e., $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset$.

Example 3. *Consider graph G in Figure 1(a) and $k=3$. The 2-shell is $S_2 = \{v_1, v_2, v_3, v_4, v_5, v_{10}, v_{11}, v_{14}\}$. Base on $(k-1)$-shell partition, S_2 is divided into two components $\mathcal{P}(S_2) = \{\mathcal{T}_1, \mathcal{T}_2\}$, where $\mathcal{T}_1 = \{v_1, v_2, v_3, v_4, v_5\}$ and $\mathcal{T}_2 = \{v_{10}, v_{11}, v_{14}\}$.*

Independent insertions on $(k-1)$-shell components. The $(k-1)$-shell partition enjoys an important property of *insertion independence* for core maximization as follows.

Theorem 1. *For two $(k-1)$-shell components $\mathcal{T}_i$ and $\mathcal{T}_j$, an edge insertion (u, v) where $v, u \in \mathcal{T}_i$, leads to no k-core follower in $\mathcal{T}_j$.*

Proof. Assume that a vertex $w \in \mathcal{T}_j$ is a new k-core follower for an edge insertion (u, v). Obviously, $c(u) = c(v) = c(w) = k-1$.

Algorithm 1 Complete Component Conversion

Input: a $(k-1)$-shell component $\mathcal{T}_i$, budget b
Output: C_i: the selected edge insertions, $\mathcal{F}_i$: the followers in $\mathcal{T}_i$
1: The $(k-1)$-collapse vertices of $\mathcal{T}_i$: $L_0 \leftarrow \mathcal{T}_i \cap D_{k-1}$;
2: **if** $|L_0| > 2 * b$ **then return** $\emptyset$;
3: Initialization: $C_i \leftarrow \emptyset$, $\mathcal{F}_i \leftarrow \emptyset$;
4: **while** $\exists$ two vertices $\{u, v\} \subseteq L_0$ and $(u, v) \notin E$ **do**
5: $\quad C_i \leftarrow C_i \cup \{(u, v)\}$; $L_0 \leftarrow L_0 \setminus \{u, v\}$;
6: **while** $\exists$ vertex $u \in L_0$ **do**
7: $\quad$ Select an arbitrary vertex v from $\mathcal{T}_i \cup \Psi_k$ for $(u, v) \notin E \cup C_i$;
8: $\quad C_i \leftarrow C_i \cup \{(u, v)\}$; $L_0 \leftarrow L_0 \setminus \{u\}$;
9: $\mathcal{F}_i \leftarrow \mathcal{T}_i$;
10: **return** $\{(C_i, \mathcal{F}_i)\}$;

According to the k-core maintenance rules [31], the vertex $w \in V$ has the coreness increased, satisfying that $c(w) = k-1$ and w is reachable from u or v via a path that consists of vertices with the coreness of $k-1$. It infers that w, u, and v are in the same connected subgraph of T_i, i.e., $w \in T_i$. Thus, $T_i \cap T_j \supseteq \{w\} \neq \emptyset$ contradicts to the $(k-1)$-shell partition of $T_i \cap T_j = \emptyset$ by Def. 5. Thus, there exists no new k-core follower in T_j. □

From the above theorem, we can deal with each $(k-1)$-shell component $\mathcal{T}_i$ independently for edge insertions.

5.3 Complete Component Conversion

In this section, we introduce a method of converting all vertices in a $(k-1)$-shell component $\mathcal{T}_i$ to k-core. The key idea is to insert new edges incident to $(k-1)$-collapse vertices $D_{k-1} \subseteq \mathcal{T}_i$. We have a useful theorem for complete k-core conversion as follows.

Theorem 2. *Given a $(k-1)$-shell component $\mathcal{T}_i \subseteq S_{k-1}$, if every $(k-1)$-collapse vertex $u \in \mathcal{T}_i \cap D_{k-1}$ has a new edge (u, v) where $v \in \mathcal{T}_i \cup \Psi_k$, then all vertices of $\mathcal{T}_i$ become k-core followers.*

Proof. Let be $X = \mathcal{T}_i \cup \Psi_k$ and the new edges $\hat{E} = \{(u, v) : u \in \mathcal{T}_i \cap D_{k-1}, v \in X\}$ where $\hat{E} \cap E = \emptyset$. Before insertions, each vertex $u \in \mathcal{T}_i \cap D_{k-1}$ has $|N_X(u)| = k-1$ in original graph $G(V, E)$. After insertions, each vertex $u \in \mathcal{T}_i$ has $|N_X(u)| \geq k$ in new graph $G(V, E \cup \hat{E})$. Thus, $\mathcal{T}_i \subseteq X$ becomes k-core followers.

By Theorem 2, we propose an algorithm of complete conversion to ensure that each vertex $v \in D_{k-1}$ has a new neighbor for edge insertions in the new k-core $\hat{\Psi}_k$.

Algorithm. The algorithm of complete k-core conversion on $\mathcal{T}_i$ is presented in Algorithm 1. The algorithm first identifies the $(k-1)$-collapse vertices of $\mathcal{T}_i$ as $L_0 = \mathcal{T}_i \cap D_{k-1}$. Since one single edge insertion can only convert two vertices from $(k-1)$-collapse, we can give up the conversion in advance if the budget is not enough for $|L_0| > 2b$ (line 2). Next, we consider two vertices $v, u \in L_0$ to add edge (u, v) into C_i and remove them from L_0 (lines 4-5). After that if there is a remaining vertex $v \in L_0$, we add a new edge between v and any $u \in \mathcal{T}_i \cup \Psi_k$ (lines 6-8). The algorithm finishes the complete conversion and returns the edge insertion answer (lines 9-11).

Algorithm 2 FastCM

Input: graph $G(V, E)$, core value k, budget b
Output: A set $\hat{E}$ of newly inserted edges
1: Partition $(k-1)$-shell S_{k-1} with $\mathcal{P}(S_{k-1}) = \{\mathcal{T}_1, \ldots, \mathcal{T}_h\}$;
2: **for** each component $\mathcal{T}_i \in \mathcal{P}(S_{k-1})$ **do**
3: $\quad \{(C_i, \mathcal{F}_i)\} \leftarrow$ Apply the complete conversion on $\mathcal{T}_i$ using Algorithm 1;
4: Sort $\mathcal{T}_i \in \mathcal{P}(S_{k-1})$ in decreasing order of expected gain $\frac{|\mathcal{F}_i|}{|C_i|}$;
5: $\hat{E} \leftarrow \bigcup_{i=1}^{h'} C_i$ such that $\sum_{i=1}^{h'} |C_i| \leq b$;
6: **return** $\hat{E}$;

Example 4. *Continue Example 3, we apply Algorithm 1 on $(k-1)$-shell component $\mathcal{T}_1 = \{v_1, v_2, v_3, v_4, v_5\}$ for complete k-core conversion with $k = 3$. The $(k-1)$-collapse is $L_0 = \{v_1, v_2, v_5\}$. Algorithm 1 first inserts a new edge (v_2, v_5) into C_1. Then, for another vertex v_1, it inserts a new edge (v_1, v_4) as $v_4 \in \mathcal{T}_1 \cup \Psi_3$. Thus, $C_1 = \{(v_2, v_5), (v_1, v_4)\}$.*

5.4 FastCM Algorithm

Integrating with $(k-1)$-collapse D_{k-1} and $(k-1)$-shell partition $\mathcal{P}(S_{k-1})$, we propose our fast core maximization (FastCM) approach using complete k-core conversion, which is outlined in Algorithm 2. The algorithm first identifies $(k-1)$-shell partition and obtains $\mathcal{P}(S_{k-1})$ (line 1). It then applies Algorithm 1 on each component $\mathcal{T}_i \in \mathcal{P}(S_{k-1})$ for complete k-core conversion (lines 2-3). Next, it sorts all components $\mathcal{T}_i$ of $\mathcal{P}(S_{k-1})$ in decreasing order of expected gain $\frac{|\mathcal{F}_i|}{|C_i|}$ and returns a greedy answer of k-core followers by inserting the candidate edges $\bigcup_{i=1}^{h'} C_i$ (lines 4-6).

Theorem 3. *Algorithm 2 takes $O(m + h(b + \log h))$ time in $O(m)$ space.*

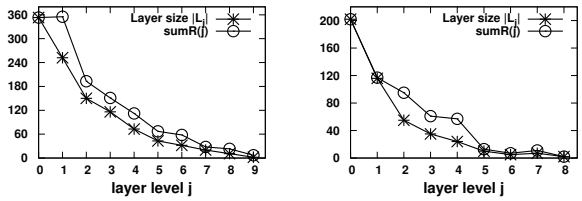
Proof. The core decomposition for calculating all vertices' corenesses takes $O(m)$ time. The $(k-1)$-shell identification and partition also takes $O(m)$ time. The complete k-core conversion for all $(k-1)$-shell components takes $O(bh)$ time. The sorting of h components takes $O(h \log h)$ time. Overall, Algorithm 2 takes $O(m+h(b+\log h))$ time in $O(m)$ space. □

6 FastCM+ USING PARTIAL CONVERSION

In this section, we propose a new edge insertion strategy of partial component conversion in Sections 6.1 and 6.2. Integrating a dynamic programming combination of complete and partial conversions on all independent components, we finally develop a fast core maximization algorithm FastCM+.

6.1 Onion Layers and Partial Conversion

Given a small budget b, a $(k-1)$-shell component $\mathcal{T}_i$ cannot be fully converted into k-core by Algorithm 1 for $b < \frac{|L_0|}{2}$. Thus, we consider a new insertion strategy, which convents *only partial* $(k-1)$-shell vertices of $\mathcal{T}_i$ to k-core followers. According to Theorem 1, we can consider h components one by one and treat each component independently. Without loss of generality, we consider one $(k-1)$-shell component $\mathcal{T}_i \in S_{k-1}$ throughout this section.



(a) Gowalla (b) Stanford

Figure 2: The distribution of requisite degree $\mathsf{sumR}(j)$ and layer size $|L_j|$ w.r.t. the layer level j. Here, $k = 10$.

Onion layers of $(k-1)$-shell component $\mathcal{T}_i$. We begin with a definition of the onion layer. According to the vertex removal order of core decomposition, the vertices are deleted sequentially in a batch way, layer by layer like peeling an onion. Thus, the structure of $(k-1)$-shell component $\mathcal{T}_i$ can be regarded in an organization of onion layers. Specifically, the $(k-1)$-collapse vertices $L_0 = D_{k-1} \cap \mathcal{T}_i \subseteq \mathcal{T}_i$ can be regarded as the *outermost layer* of $\mathcal{T}_i$, which is the first batch of vertices deleted from the k-core of graph G.

Definition 6. ***(Onion Layer)*** *Given $(k-1)$-shell component $\mathcal{T}_i$ and integer $j \geq 1$, the j-th onion layer L_j is the largest vertex set such that $L_j = \{v \in \mathcal{T}_i : |N_X(v)| \leq k-1\}$, where $X = \Psi_{k-1} \setminus \bigcup_{r=0}^{j-1} L_r$ and the 0-th layer $L_0 = D_{k-1} \cap \mathcal{T}_i = \{v \in \mathcal{T}_i : |N_{\Psi_{k-1}}(v)| = k-1\}$.*

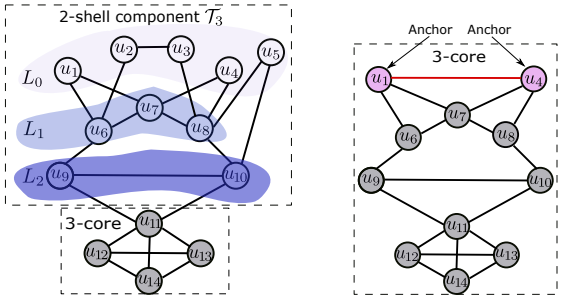
We denote the largest layer number of *innermost layer* in $\mathcal{T}_i$ is $\Phi(\mathcal{T}_i)$. Hence, $\bigcup_{r=0}^{\Phi(\mathcal{T}_i)} L_r = S_{k-1}$. Moreover, for each vertex $u \in \mathcal{T}_i$, we represent its corresponding layer as $l(u)$, i.e., $u \in L_{l(u)}$. Given two vertices v and u with $l(v) > l(u)$, we say that v locates at a higher layer than u, or u locates at a lower layer than v. Continue with Example 3, the 2-shell component $\mathcal{T}_1 = \{v_1, v_2, v_3, v_4, v_5\}$ has two-level onion layers as $L_0 = \{v_1, v_2, v_5\}$ and $L_1 = \{v_3, v_4\}$.

Requisite degree of layer L_j. We observe a useful degree property of $(k-1)$-shell onion layers as follows.

Lemma 1. *Given an onion layer L_j in $\mathcal{T}_i$ for $1 \leq j \leq \Phi(\mathcal{T}_i)$, vertex $v \in L_j$ has $|N_X(v)| \geq k$, where $X = \Psi_k \cup \{u \in \mathcal{T}_i : l(u) \geq j-1\}$.*

Proof. We prove this lemma by contradiction. Assume that there exists a vertex $v \in L_j$ with $|N_X(v)| < k$. Thus, v must be deleted early before removing any vertex at layer L_j by the peeling rule of k-core decomposition. By Def. 6, we have $v \in L_{j'}$ where $\exists 0 \leq j' < j$, which contradicts to $v \in L_j$. □

Lemma 1 shows that we can convert the partial vertices $\{v \in \mathcal{T}_i : l(v) \geq j\}$ to k-core followers by only inserting new edges at the layer L_j. Actually, the complete conversion in Algorithm 1 is to convert the whole vertices in $\mathcal{T}_i$ to followers through edge insertions at the 0-th layer L_0. However, for a vertex u with $l(u) > 0$, the degree gap to k-core requirements may be larger than 1, which is different from the complete conversion case. For example, assume that $k = 3$ and $j = 1$ for graph G in Figure 1, if we discard the vertices $L_0 = \{v_1, v_2, v_5\}$, the remaining vertex $v_3 \in L_1$ has only one neighbor v_4. Thus, the degree gap to k-core for v_3 is two, indicating that two inserted edges associated with v_3 are needed. In the following, we



(a) Graph G_1 (b) The new 3-core Ψ_3.

Figure 3: An example of partial 3-core conversion on G_1 by anchoring two vertices u_1, u_4 in L_0 and $C_3 = \{(u_1, u_4)\}$ for $b = 1$.

give a new definition of *requisite degree* to estimate the required degree for a vertex to become a k-core follower.

Definition 7. ***(Requisite Degree)*** *The requisite degree of vertex $u \in \mathcal{T}_i$, denoted as $R(u)$, is defined as the number of extra new neighbors required for u to become k-core follower at the layer $L_{l(u)}$ above, i.e., $R(u) = k - |N_X(u)|$ where $X = \Psi_k \cup \{v \in \mathcal{T}_i : l(v) \geq l(u)\}$.*

The requisite degree indicates how 'far' the vertices to become k-core followers. If the requisite degrees of all the vertices at the same layer L_j are 0 after a few edge insertions, the vertices of the whole set $\{u \in \mathcal{T}_i : l(u) \geq j\}$ are k-core followers.

Practical feasibility of partial k-core conversion. We show the practical feasibility of partial k-core conversion. This can be validated by the number of required edge insertions w.r.t. the level of onion layers L_j. We use $\mathsf{sumR}(j)$ to denote the total requisite degrees of all vertices in layer L_j, i.e, $\mathsf{sumR}(j) = \sum_{u \in L_j} R(u)$. The larger $\mathsf{sumR}(j)$ is, the more edge insertions are needed. Figure 2 depicts the requisite degree distribution of $\mathsf{sumR}(j)$ w.r.t. layer levels j over two real datasets Gowalla and Stanford as shown in Table 2. We have two useful observations: (1) As the layer level j increases, the requisite degree $\mathsf{sumR}(j)$ and the number of vertices $|L_j|$ decrease greatly. (2) The average requisite degree of vertices $\frac{\mathsf{sumR}(j)}{|L_j|}$ is close to 1 on most cases. Based on these two observations, it is practically feasible to finish the partial conversion successfully at layer L_j for a particular j using a small budget b, even the fail case of complete conversion by Algorithm 1 happens.

6.2 $(k-1)$-Shell Partial Conversion Algorithm

In this section, we propose a $(k-1)$-shell partial conversion algorithm, which converts a part of vertices in $\mathcal{T}_i$ to k-core followers.

Motivations. The key idea of partial conversion is to determine the *lowest* onion layer $j^* \in [0, \Phi(\mathcal{T}_i)]$ such that we can convert the vertices of whole subset $X = \{u \in \mathcal{T}_i : l(u) \geq j^*\}$ to k-core using b budget. The smaller the layer level j^* is, the more the k-core followers are. The best case is when $j^* = 0$, it corresponds to the complete conversion of $X = \{u \in \mathcal{T}_i : l(u) \geq 0\} = \mathcal{T}_i$. One straightforward method of finding the lowest layer is to check the conversion feasibility of L_j, starting from $j = 0$ to $j = \Phi(\mathcal{T}_i)$ layer by layer. Unfortunately, the answer may be not the lowest

layer j^*, even worse no feasible answer may exist. For example, consider a graph G_1 as shown in Figure 3(a), $k = 3$, and $b = 1$. G_1 has three layers: $L_0 = \{u_1, u_2, u_3, u_4, u_5\}, L_1 = \{u_6, u_7, u_8\}, L_2 = \{u_9, u_{10}\}$. The conversion of L_0 needs at least three edge insertions, e.g., (u_1, u_2), (u_3, u_4), and (u_4, u_5), which takes a budget of 3 greater than $b = 1$. Similarly, each conversion of L_1 and L_2 takes a budget of 2, which leads to no feasible answer for $b = 1$. To dismiss the limitation, we propose an improved partial conversion algorithm based on a novel anchoring strategy, which converts lower-layered vertices $v \in L_j$ into followers for a given layer L_{j^*} and $j < j^*$. This can reduce the total requisite degree $\mathsf{sumR}(j^*)$ of L_{j^*} for obtaining more k-core followers using a smaller budget. Continue the above example for $j^* = 1$ and $b = 1$, if we add a new edge between v_1 and v_4 in layer L_0, we can successfully convert the layers L_1 and L_2 into k-core as shown in Figure 3(b).

Anchor. We first define an action called *anchor*. Given a layer L_j, the action of anchor, is to add a few new edges incident to a vertex $u \in Y$ at lower layers such that u becomes k-core follower, where $Y = \bigcup_{r=0}^{j-1} L_r$. In other words, we say that we anchor a vertex u, meaning that we convert vertex u to k-core follower.

Anchor gain. We give a definition of *anchor gain*, to evaluate the importance of anchoring a vertex $u \in Y$ where $Y = \bigcup_{r=0}^{j-1} L_r$.

Definition 8. ***(Anchor Gain)*** *Given $\mathcal{T}_i$ and an integer $j \in [1, \Phi(\mathcal{T}_i)]$, the anchor gain of vertex $u \in \mathcal{T}_i$ with $l(u) < j$ is denoted as* $\mathsf{gain}(u, j) = \mathsf{Benefit}(u, j) - \mathsf{Cost}(u, j)$. *Here, the degree benefit* $\mathsf{Benefit}(u, j) = |\{v \in N(u) \cap L_j : R(v) > 0\}|$ *and the budget cost* $\mathsf{Cost}(u, j) = k - |N_H(u)|$ *where* $H = \Psi_k \cup \{v \in \mathcal{T}_i : l(v) \geq j\}$.

The $\mathsf{gain}(u, j)$ consists of two parts: $\mathsf{Benefit}(u, j)$ and $\mathsf{Cost}(u, j)$. We analyze these two terms in terms of positive and negative aspects. As the anchored u may have edge connections to vertices at layer L_j, thus the requisite degree of each vertex $v \in N(u) \cap L_j$ could decrease by one. Thus, the total requisite degree of L_j decreases by $\mathsf{Benefit}(u, j)$, which is a contribution score. On the other hand, it takes an extra cost for converting u to become k-core follower. Assume that the whole set of $H = \Psi_k \cup \{v \in T_i : l(v) \geq j\}$ becomes k-core, $\mathsf{Cost}(u, j) = k - |N_H(u)|$ indicates how many new edges are needed to insert between u and H such that u becomes k-core follower, which is a budget consumption. For example, consider the vertex u_1 in Figure 3(a) and $j = 1$, $\mathsf{Benefit}(u_1, 1) = |\{u_6, u_7\}| = 2$, $\mathsf{Cost}(u_1, 1) = 3 - |\{u_6, u_7\}| = 1$, and $\mathsf{gain}(u_1, 1) = 2 - 1 = 1$, indicating that it anchors u_1 for bringing two 'useful' edges and using one budget for edge insertion. Correspondingly, the anchor gain of u_2 is $\mathsf{gain}(u_2, 1) = 1 - 2 = -1$. In summary, an integrated function $\mathsf{gain}(u, j)$ measures a trade-off evaluation of anchoring this low-level vertex u to increase k-core followers. We have a useful lemma on anchor gain as follows.

Lemma 2. *Given an existing partial conversion plan $\mathcal{P}$ on $\mathcal{T}_i$ and an integer $j \in [1, \Phi(\mathcal{T}_i)]$, it converts all vertices $\mathcal{F}_i$ to k-core follower using a budget b, where $\mathcal{F}_i \supseteq \bigcup_{r=j}^{\Phi(\mathcal{T}_i)} L_r$. For a vertex $u \in \bigcup_{r=0}^{j-1} L_r$ and $u \notin \mathcal{F}_i$, if $\mathsf{gain}(u, j) \geq 0$, there exists a new plan $\mathcal{P}_u$ based on $\mathcal{P}$ with an additional action of anchoring u, which can achieve more k-core followers $\mathcal{F}_i' = \mathcal{F}_i \cup \{u\}$ using a new budget $b' \leq b$.*

Proof. First, the current k-core in $\mathcal{P}$ is $H' = \Psi_k \cup \mathcal{F}_i$ and $\mathcal{F}_i \supseteq \bigcup_{r=j}^{\Phi(\mathcal{T}_i)} L_r = \{v \in T_i : l(v) \geq j\}$. By Def. 8, $H = \Psi_k \cup \{v \in T_i : l(v) \geq j\} \subseteq H'$ holds. Second, we prove that $|\mathcal{F}_i'| > |\mathcal{F}_i|$. The action of anchoring u is to add $k - |N_{H'}(u)|$ new edges incident to u. As $H' \supseteq H$, the extra budget cost is $b_{extra} = k - |N_{H'}(u)| \leq \mathsf{Cost}(u, j)$ in Def. 8. Thus, we have the new followers $\mathcal{F}_i' = \mathcal{F}_i \cup \{u\}$. Obviously, $|\mathcal{F}_i'| = |\mathcal{F}_i| + 1 > |\mathcal{F}_i|$. Third, we prove that the new budget $b' \leq b$. Due to the newly joining vertex u, it brings additional edges (u, v) for $v \in \mathcal{F}_i$, indicating that the new plan $\mathcal{P}_u$ can insert less edges than original plan $\mathcal{P}$ by saving a budget b_{saving}. We derive b_{saving} as follows. The total increased degree of vertices $v \in \mathcal{F}_i$ is no less than $\mathsf{Benefit}(u, j) + b_{extra}$, where $\mathsf{Benefit}(u, j)$ comes from the original edges incident to u and b_{extra} comes from the new edges to anchor u. As one budget of edge insertion increases two degrees for vertices in $\mathcal{F}_i$ at most. Thus, we obtain that $\mathsf{Benefit}(u, j) + b_{extra} \leq 2b_{saving}$. The new cost is $b' = b - b_{saving} + b_{extra} \leq b - \frac{\mathsf{Benefit}(u,j) + b_{extra}}{2} + b_{extra} = b - \frac{\mathsf{Benefit}(u,j) - b_{extra}}{2} \leq b - \frac{\mathsf{Benefit}(u,j) - \mathsf{Cost}(u,j)}{2} = b - \frac{\mathsf{gain}(u,j)}{2} \leq b$. □.

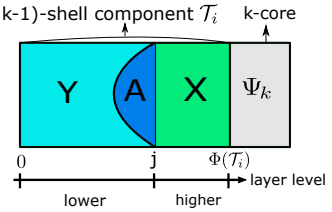


Figure 4: The framework of our partial conversion based on anchoring lower-layered vertices $A \subseteq Y$ in $\mathcal{T}_i$ and converting $A \cup X$ into followers. Here, $Y = \bigcup_{r=0}^{j-1} L_r$ and $X = \bigcup_{r=j}^{\Phi(\mathcal{T}_i)} L_r$.

By Lemma 2, we can anchor vertices with non-negative gains for saving budget and increasing k-core followers, which is presented in the partial k-core conversion on $\mathcal{T}_i$ in Algorithm 3.

Overview. The overview of Algorithm 3 is to find the lowest layer L_j for partial conversion within b budget, by starting from $j = 1$ layer by layer until $j = \Phi(\mathcal{T}_i)$ (lines 4-33). For a given layer L_j, it has three major steps: find all lower-layered vertices A with non-negative anchor gains (lines 5-21), check feasibility of converting $A \cup X$ to k-core followers where $X = \{u \in \mathcal{T}_i : l(u) \geq j\}$ (lines 22-30), and return the answer if the inserted edge size $|C_i| \leq b$ (lines 31-33), otherwise repeating the above process on next layer L_{j+1}. The whole framework of Algorithm 3 is outlined in Figure 4.

Algorithm. Now, we present the details of Algorithm 3. It first identifies all onion layers $\{L_0, \ldots, L_{\Phi(\mathcal{T}_i)}\}$ in $\mathcal{T}_i$ via core decomposition, and computes requisite degree $R(u)$ and layer number $l(u)$ for all vertices $u \in \mathcal{T}_i$ (lines 1-3). Next, it iteratively starts from the low layer $j = 1$ to the highest layer $\Phi(\mathcal{T}_i)$ for finding a feasible plan to convert the anchored vertices A and the vertices $X = \{u \in \mathcal{T}_i : l(u) \geq j\}$ (lines 4-33). Given a layer L_j, it first initializes the variables of edge insertion C_i, followers $\mathcal{F}_i$, anchored vertices A, the requisite degree of layer L_j, and so on (lines 5-10). Note that it partitions T_i into two parts: upper layer $X = \{u \in \mathcal{T}_i : l(u) \geq j\}$ and lower layer $Y = \{u \in \mathcal{T}_i : l(u) \leq j\}$ (lines 7-8). It uses $r(u)$ to record a dynamic requisite degree of vertex $u \in L_j$ based on the dynamic updated set $H = X \cup A \cup \Psi_k$, where $r(u) = R(u)$ in Def. 7 for $A = \emptyset$ (line 9). It also calculates the

Algorithm 3 Partial Component Conversion

Input: a $(k-1)$-shell partition $\mathcal{T}_i$, coreness k, budget b
Output: C_i: the selected edge insertions, $\mathcal{F}_i$: the followers in $\mathcal{T}_i$

1: Identify all onion layers $\{L_0, \ldots, L_{\Phi(\mathcal{T}_i)}\}$ in $\mathcal{T}_i$ by Def. 6;
2: Compute the layer number $l(u)$ for all $u \in \mathcal{T}_i$;
3: Compute the requisite degree $R(u)$ for all $u \in \mathcal{T}_i$ by Def. 7;
4: **for** $j \leftarrow 1$ to $\Phi(\mathcal{T}_i)$ **do**
5: Initialization: $C_i \leftarrow \emptyset, \mathcal{F}_i \leftarrow \emptyset$;
6: Anchored vertex set: $A \leftarrow \emptyset$;
7: Upper layer: $X \leftarrow \{u \in \mathcal{T}_i : l(u) \geq j\}$;
8: Lower layer: $Y \leftarrow \{u \in \mathcal{T}_i : l(u) < j\}$;
9: Initialize a dynamic requisite degree $r(u)$ of each vertex $u \in L_j$ based on $H = X \cup A \cup \Psi_k$, i.e., $r(u) = k - |N_H(u)| = R(u)$;
10: The requisite degree of layer L_j: $\mathsf{sumR}(j) = \sum_{u \in L_j} r(u)$;
11: Calculate $\mathsf{gain}(u, j)$ for each vertex $u \in Y$ by Def. 8;
12: **while** $\exists$ a vertex $u \in Y$ with $\mathsf{gain}(u, j) \geq 0$ **do**
13: $u^* \leftarrow \arg\max_{u \in Y} \mathsf{gain}(u, j)$;
14: $A \leftarrow A \cup \{u^*\}$;
15: $Y \leftarrow Y \setminus \{u^*\}$;
16: $\mathsf{sumR}(j) \leftarrow \mathsf{sumR}(j) - \mathsf{gain}(u^*, j)$;
17: **for** each vertex $v \in N(u^*) \cap L_j$ **do**
18: **if** $r(v) > 0$ **then**
19: $r(v) \leftarrow r(v) - 1$; // Due to the edge (u^*, v) in H.
20: **if** $r(v) = 0$ **then**
21: Update $\mathsf{gain}(w, j)$, where $w \in Y \cap N(v)$;
22: **if** $\mathsf{sumR}(j) \leq 2b$ **then**
23: Let $Q \leftarrow \{v \in H : |N_H(v)| < k\}$ where $H = X \cup A \cup \Psi_k$;
24: **while** $\exists$ two vertices $u, v \in Q$ and $(u, v) \notin E \cup C_i$ **do**
25: $C_i \leftarrow C_i \cup \{(u, v)\}$;
26: Remove u (or v) from Q if its current degree in H is no less than k in H after the insertion of new edges C_i;
27: **while** $\exists$ vertex $u \in Q$ **do**
28: Pick arbitrary vertex $v \in H$ with $(u, v) \notin E \cup C_i$;
29: $C_i \leftarrow C_i \cup \{(u, v)\}$;
30: Remove u from Q if the degree $|N_H(u)| \geq k$ after the insertion of new edges C_i;
31: **if** $|C_i| \leq b$ **then**
32: $\mathcal{F}_i \leftarrow A \cup X$;
33: **return** $\{(C_i, \mathcal{F}_i)\}$;
34: **return** $\emptyset$;

anchor gain $\mathsf{gain}(u, j)$ for each lower-layer vertex $u \in Y$ (line 11). Next, it anchors vertices of Y with non-negative gains and adds them into A by Lemma 2 (lines 12-21). At each iteration, it selects one vertex $u^* \in Y$ with the largest $\mathsf{gain}(u^*, j)$ (line 13), and updates the requisite degree for layer-level $\mathsf{sumR}(j)$ and vertex-level $r(v)$ where $v \in N(u^*) \cap L_j$ (lines 16-21). If $r(v)$ of a vertex v decreases to 0, it needs to update the anchor gain for all vertices $w \in Y \cap N(v)$ to decrease their benefit contributions by Def. 8 (lines 20-21). After obtaining the anchored vertices A, it checks the condition of $\mathsf{sumR}(j) \leq 2b$ (line 22). If $\mathsf{sumR}(j) > 2b$, it takes more than b budget for the conversion, which is not feasible and needs to consider the next layer L_{j+1}; Otherwise, it adds edges into C_i for converting $A \cup X$ to k-core followers (lines 23-30). To find new edges, it first extracts all vertices having degree less than k from H into Q, i.e., $Q = \{v \in H : |N_H(v)| < k\}$ (line 23). It adds a new edge between two vertices in Q and discards them if they admit the condition of k-core followers (lines 23-26). When no edge can be inserted between any two vertices of Q, it adds new edges between $u \in Q$ and $v \in H$ (lines 27-30). Finally, if $|C_i| \leq b$, it returns the answer of C_i and its corresponding followers $\mathcal{F}_i = A \cup X$ (lines 31-33). The algorithm terminates with an empty answer if no feasible solution exists at any layer L_j for $1 \leq j \leq \Phi(\mathcal{T}_i)$ (line 34).

Table 1: The conversion solutions of each $(k-1)$-shell component in a new graph $G_2 = G \cup G_1$, where G and G_1 are shown in Figure 1(a) and Figure 3(a), respectively. Here, $b = 2$.

$(k-1)$-shell components	Selected candidate edges C_i	Followers $\mathcal{F}_i$
$\mathcal{T}_1$	$\{(v_1, v_4), (v_2, v_5)\}$	$\{v_1, v_2, v_3, v_4, v_5\}$
$\mathcal{T}_2$	$\{(v_{10}, v_{11})\}$	$\{v_{10}, v_{11}, v_{14}\}$
$\mathcal{T}_3$	$\{(u_1, u_4)\}$	$\{u_1, u_4, u_6, u_7, u_8, u_9, u_{10}\}$

Example 5. *Consider a graph G_1 in Figure 3(a), $k = 3$, and $b = 1$. The 2-shell component of G_1 denoted as $\mathcal{T}_3$ has three onion layers: $L_0 = \{u_1, u_2, u_3, u_4, u_5\}, L_1 = \{u_6, u_7, u_8\}, L_2 = \{u_9, u_{10}\}$. The requisite degree $R(v) = 1$ for each vertex $v \in L_0 \cup L_1 \cup L_2$. The complete conversion of L_0 needs at least three edge insertions. Thus, it starts the partial conversions from L_1 using Algorithm 3. First, it initializes the upper layer $X = \{u_6, u_7, u_8, u_9, u_{10}\}$, the lower layer $Y = \{u_1, u_2, u_3, u_4, u_5\}$, and the requisite degree $\mathsf{sumR}(1) = 3$. It computes the anchor gain for $u \in Y$ as $\mathsf{gain}(u_1, 1) = \mathsf{gain}(u_4, 1) = 1, \mathsf{gain}(u_5, 1) = 0, \mathsf{gain}(u_2, 1) = \mathsf{gain}(u_3, 1) = -1$ (lines 7-11 of Algo. 3). It first anchors u_4 with the largest non-negative gain and collects u_4 to A by Lemma 2. Then, it updates $\mathsf{sumR}(1) = 2$, the dynamic requisite degree $r(u_7) = 0$, $r(u_8) = 0$, and the anchor gains $\mathsf{gain}(u_1, 1) = 0$, $\mathsf{gain}(u_5, 1) = -1$ (lines 12-21 of Algo. 3). In the next round, it anchors u_1 with the largest gain, $A = \{u_4, u_1\}$, and $\mathsf{sumR}(1) = 1$. It stops finding the vertices to be anchored as all the remaining anchor gains are negative. As $\mathsf{sumR}(1) = 1 \leq 2 \times b = 2$, it assigns $Q = \{u_4, u_1\}$ and inserts an edge (u_4, u_1) into C_3, leading that $|N_H(u_4)| = 3$ and $|N_H(u_1)| = 3$ where $H = A \cup X \cup \Phi_3$ (lines 22-30 of Algo. 3). Thus, Algorithm 3 returns the edge insertions $C_3 = \{(u_4, u_1)\}$ and the followers $\mathcal{F}_3 = \{u_1, u_4, u_6, u_7, u_8, u_9, u_{10}\}$.*

6.3 Dynamic Programming Edge Selections

In this section, we develop a dynamic programming solution to select the $(k-1)$-shell components $\mathcal{P}(S_{k-1}) = \{\mathcal{T}_1, \ldots, \mathcal{T}_h\}$ for edge insertions, to achieve the largest followers for a given budget b.

Based on the insertion strategies by Algorithms 1 and 3, we get the new edges C_i and the corresponding followers $\mathcal{F}_i$ for each $(k-1)$-shell component $\mathcal{T}_i$. Therefore, the objective function is formulated to maximize $\sum_{j=1}^{h} |\mathcal{F}_j| x_j$, where $\sum_{j=1}^{h} |C_j| x_j \leq b$ and $x_j \in \{0, 1\}$ for $j \in [1, h]$, denoted by Eq. (1). For a component $\mathcal{T}_i$, $x_i = 1$ indicates that it selects the C_i edges set for insertions, which costs a budget of $|C_i|$ and achieves a gain of $|\mathcal{F}_i|$ followers; otherwise, it abandons $\mathcal{T}_i$ by inserting no new edges for $x_i = 0$. The problem can be resolved by adopting a similar 0-1 knapsack dynamic programming solution [15]. Note that if the budget b is not used up, we consider to convert other $(k-\lambda)$-shells using the remaining budget in the next section.

Example 6. *Consider a new graph G_2 composed of two graphs: G as shown in Figure 1(a) and G_1 as shown in Figure 3(a), i.e., $G_2 = G \cup G_1$. Here, $k = 3$ and $b = 2$. As discussed in previous examples, G has two 2-shell components $\mathcal{T}_1$ and $\mathcal{T}_2$, and G_1 has one 2-shell component $\mathcal{T}_3$. The new edges C_i and the corresponding followers $\mathcal{T}_i$ for each component are shown in Table 1. We apply our dynamic programming solution on G_2 for 3-core maximization. Finally, it obtains the answer $\hat{E} = C_2 \cup C_3 = \{(v_{10}, v_{11}), (u_1, u_4)\}$, which brings ten new 3-core followers $\mathcal{F} = \mathcal{F}_2 \cup \mathcal{F}_3 = \{v_{10}, v_{11}, v_{14}, u_1, u_4, u_6, u_7, u_8, u_9, u_{10}\}$.*

6.4 Handle $(k-\lambda)$-Shell Conversion

In this section, we discuss how to handle $(k-\lambda)$-shell conversion for $2 \le \lambda \le k$. In practice, this case may happen when $S_{k-1} = \emptyset$ for an extremely large budget b or a large coreness parameter k. Thus, it is necessarily important to generalize the existing techniques to handle coverting the vertices of non-empty $(k-\lambda)$-shell to k-core followers when a budget surplus is available.

We extend the existing $(k-1)$-shell techniques of *complete conversion* and *partial conversion* for $(k-\lambda)$-shell $S_{k-\lambda}$ conversion. The key idea is very similar but has a few differences. One significant difference is that it needs to check the degree of all vertices in $S_{k-\lambda}$ instead of the $(k-\lambda)$-collapse $L_0 \subseteq S_{k-\lambda}$ at outermost layer for conversion only, due to an increased degree gap $\lambda > 1$.

- **Complete $(k-\lambda)$-shell conversion**: It first partitions $G_{S_{k-\lambda}}$ into disjoint components based on graph connectivity. In each component $\mathcal{T}_i$ of $S_{k-\lambda}$, each vertex candidate has the coreness of $k-\lambda$ and the degree gap of no greater than λ to become k-core. Let be the vertex set $X = \{v \in \mathcal{T}_i : |N_{\mathcal{T}_i}(v)| < k\}$. The complete $(k-\lambda)$-shell conversion on $\mathcal{T}_i$ needs to insert at least $\lceil \frac{\sum_{v \in X} k - |N_{\mathcal{T}_i}(v)|}{2} \rceil$ new edges.
- **Partial $(k-\lambda)$-shell conversion**: It identifies the onion layers of $\mathcal{T}_i$. Then, it finds the lowest layer L_{j^*} to convert $\bigcup_{r=j^*}^{\Phi(\mathcal{T}_i)} L_r$ and the anchored vertices to k-core, similar as Algorithm 3. Note that one major change lies on the calculation of anchor gain in Def. 8, where $\mathsf{Benefit}(u, j)$ changes as $\mathsf{Benefit}(u, j) = |\{v \in N(u) \cap X : R(v) > 0\}|$ where $X = \{v \in \mathcal{T}_i : l(v) \ge j\}$.

6.5 FastCM+ Algorithm

Integrating the techniques of complete and partial component conversions and DP-based edge selections, we propose the FastCM+ algorithm outlined in Algorithm 4. The first step is to partition the $(k-1)$-shell S_{k-1} into multiple independent components $\mathcal{P}(S_{k-1})$ (line 2). Then, for each component $\mathcal{T}_i$, we first invoke Algorithm 1 and try to convert the whole vertices to followers (line 3). If the budget is not enough to complete the conversion, we invoke the Algorithm 3 to convert a part of the vertices to followers using a smaller budget (line 6). Next, we apply dynamic programming techniques to tackle the edge selections optimally and add new edges into $\hat{E}$ (lines 7-8). Let be $\lambda = 1$. When there are a remaining budget $b' = b - |\hat{E}| > 0$ and $S_{k-\lambda} = \emptyset$, we iteratively increase λ by one and convert non-empty $(k-\lambda)$-shell into k-core followers as shown in Section 6.4 (lines 9-11). Finally, if the budget is not used up, we iteratively insert new edges (u, v) into $\hat{E}$ in the decreasing order of $|N_{\hat{\Psi}_k}(u)|$ (line 12-14).

Algorithm 4 FastCM+

Input: graph $G(V, E)$, core value k, budget b
Output: a set $\hat{E}$ of newly inserted edges

1: $\hat{E} \leftarrow \emptyset$; $\lambda \leftarrow 1$;
2: Partition $(k-1)$-shell S_{k-1} with $\mathcal{P}(S_{k-1}) = \{\mathcal{T}_1, \ldots, \mathcal{T}_h\}$;
3: **for** each component $\mathcal{T}_i \in \mathcal{P}(S_{k-1})$ **do**
4: $\quad \{(C_i, \mathcal{F}_i)\} \leftarrow$ Apply the full conversion on $\mathcal{T}_i$ by Algo. 1;
5: $\quad$ **if** $C_i = \emptyset$ **then**
6: $\quad\quad \{(C_i, \mathcal{F}_i)\} \leftarrow$ Apply the partial conversion on $\mathcal{T}_i$ using Algorithm 3;
7: Solve the dynamic programming formulation in Eq. (1) and obtain the edge selections $\{x_i : 1 \le i \le h\}$;
8: $\hat{E} \leftarrow \hat{E} \cup \bigcup_{1 \le i \le h, x_i = 1} C_i$; Insert edges $\hat{E}$ into G;
9: **while** the remaining budget $b' = b - |\hat{E}| > 0$ and $S_{k-\lambda} = \emptyset$ **do**
10: $\quad \lambda \leftarrow \lambda + 1$;
11: $\quad$ Convert $(k-\lambda)$-shell to k-core followers by inserting edges into G, following the similar steps at lines 2-8;
12: **if** the remaining budget $b' = b - |\hat{E}| > 0$ **then**
13: $\quad$ Add $\hat{E}$ with new edges (u, v) in the decreasing order of $|N_{\hat{\Psi}_k}(u)|$ where $u \notin \hat{\Psi}_k$, $v \in \hat{\Psi}_k$, and $\hat{\Psi}_k$ is the new k-core of new graph $G(V, E \cup \hat{E})$, until using up all budgets for $|\hat{E}| = b$;
14: **return** $\hat{E}$;

Theorem 4. FastCM+ *in Algorithm 4 for* converting non-empty $(k-1)$-shell partially *takes $O(m + bn)$ time in $O(m + bh)$ space.*

Proof. FastCM+ consists of four search phases. In the first phase, Algorithm 4 partitions $(k-1)$-shell S_{k-1} in $O(m+n)$ time. In the second phase, we consider two cases of k-core conventions. For the complete k-core conversion, for a component $\mathcal{T}_i$, Algorithm 1 takes $O(b)$ time. If Algorithm 1 fails to return a non-empty solution of complete conversion, Algorithm 3 is then invoked for partial conversion. The time complexity for generating all onion layers and calculating all requisite degrees is $O(m)$. For a given j-th level L_j at component $\mathcal{T}_i$, it takes $O(|L_j| + \sum_{v \in L_j} |N(v)|)$ time for calculating gains and anchoring nodes, and $O(b)$ time for finding candidate edges in each layer. Thus, it takes

$$O\Big(\sum_{1 \le i \le h} \sum_{1 \le j \le \Phi(\mathcal{T}_i)} (|L_j| + \sum_{v \in L_j} |N(v)| + b)\Big)$$
$$= O\Big(\sum_{1 \le i \le h, 1 \le j \le \Phi(\mathcal{T}_i)} |L_j| + \sum_{1 \le i \le h, 1 \le j \le \Phi(\mathcal{T}_i)} \sum_{v \in L_j} |N(v)| + bn\Big)$$
$$= O(n + m + bn) = O(m + bn) \text{ time.}$$

In the third phase, the dynamic programming based edge selection takes $O(bh)$ time. In the fourth phase, if a budget b is remaining and $S_{k-1} \neq \emptyset$, it does not use $(k-\lambda)$-shell conversion (lines 9-11). Instead, the remaining budget for edge insertions takes $O(m + b)$ time using the bin sorting (lines 12-13). Overall, as $h \le n$, the time complexity of Algorithm 4 is $O(m + bn + bh) \subseteq O(m + bn)$.

Next, we analyze the space complexity. The candidate edges and dp-based selection takes $O(bh)$ space. The graph G takes $O(m+n) = O(m)$ space. Computing follower gains takes $O(n)$ space for each layer one time. Overall, Algorithm 4 takes $O(bh + m)$ space. □

Table 2: Network statistics. Here, the parameters k_{max} and $|\mathcal{P}(S_{k-1})|$ represent the largest coreness and the component number of $(k-1)$-shell, respectively. '-' denotes that the algorithm cannot finish within 48 hours.

Dataset	Addr.	$\|V\|$	$\|E\|$	k_{max}	$\|\mathcal{P}(S_{k-1})\|$	#Followers				Running time (s)			
						EKC	VEK	FastCM	FastCM+	EKC	VEK	FastCM	FastCM+
Facebook	FB	4,039	88,234	59	25	39	77	77	**198**	0.53	0.28	**0.0007**	0.0024
email-Enron	EN	36,692	183,831	43	97	112	140	140	**241**	3.13	0.66	**0.06**	0.13
Brightkite	BT	58,228	214,078	52	24	93	218	218	**518**	18.8	1.03	**0.112**	0.68
Gowalla	GW	196,591	950,327	51	306	571	571	671	**671**	1,071	15.17	**0.34**	0.47
Twitter	TW	81,306	1,768,149	96	669	712	712	755	**761**	3,923	28.05	**3.01**	3.18
Stanford	ST	281,903	2,312,497	71	3,253	-	508	734	**747**	-	193	**0.82**	0.97
Google	GL	875,713	5,105,039	44	475	-	1,668	2,009	**2,017**	-	922	**2.3**	5.6
Youtube	YT	1,134,890	2,987,624	51	772	734	665	831	**838**	9,472	35.38	**2.31**	2.40
Baidubaike	BD	2,142,101	17,014,946	78	9,852	-	1,035	1,161	**1,178**	-	1,987	**4.59**	5.08
as-Skitter	AS	1,696,415	11,095,298	111	5,182	-	1,618	1,739	**1,809**	-	716	**3.31**	3.69
socfb-konec	KN	59,216,211	92,522,012	16	1,466	-	1,950	488	**5,081**	-	10,413	**106**	276

Note that for a limited budget $b < \frac{|D_{k-1}|}{2}$ in practice, Algorithm 4 cannot completely convert S_{k-1} into k-core, indicating the new $(k-1)$-shell $S_{k-1} \neq \emptyset$ in graph $G(V, E \cup \hat{E})$. Thus, FastCM+ takes $O(m+bn)$ time much faster than $O(bn^2m)$ time by EKC [44] and $O(bmn)$ time by VEK [45]. However, when there is a large remaining budget b and $S_{k-1} = \emptyset$, Algorithm 4 takes extra cost to handle $(k-\lambda)$-shell conversions. Experimental efficiency evaluations in Table 2 validate the superiority of our approach FastCM+ against EKC and VEK on real large graph datasets.

7 EXPERIMENTS

In this section, we conduct experiments to evaluate the effectiveness and efficiency of our proposed algorithms.

Datasets. We use 11 datasets of real-world networks in experiments. Table 2 reports graph statistics of these datasets. All graphs are downloaded from http://networkrepository.com and http://snap.stanford.edu, and treated as undirected graphs.

Compared methods. To evaluate the effectiveness and efficiency of core maximization, we compare four methods as follows.

- EKC [44]: is a greedy algorithm that selects one best edge for edge insertion in each round and ends after b rounds.
- VEK [45]: is a vertex-oriented heuristic method to convert one vertex to a follower in each round.
- FastCM: is our proposed Algorithm 2 using the complete k-core conversion.
- FastCM+: is our proposed Algorithm 4 based on complete and partial k-core conversions and DP-based edge selections.

Beside the above four core maximization approaches using the model of *edge insertions*, we also evaluate another different model for core maximization using *anchored nodes* [21].

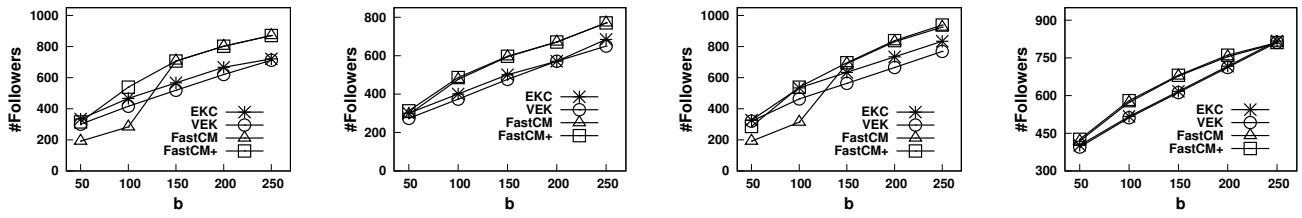
- RCM [21]: is the anchored k-core algorithm for maximizing k-core by fixing b vertices outside of k-core.

Parameters and evaluation metrics. By default, we set parameter $k = 10$ for KN, and $k = 20$ consistently for the remaining graphs. The default parameter budget $b = 200$. We report the number of k-core followers (denoted as #Followers) and the running time (in seconds). Note that we treat the #Followers as *N/A* and the running time as *infinite* if the algorithm runs exceeding 48 hours.

Exp-1: Quality and efficiency evaluations of different algorithms on all datasets. Table 2 reports the effectiveness and efficiency results of different algorithms on all datasets. Our algorithm FastCM+ outperforms all competitors EKC [44], VEK [45], and FastCM on all datasets by achieving the largest number of followers. This reflects the effectiveness of $(k-1)$-shell partition and our edge insertion strategies. Our developed complete/partial conversion allows to consider multiple new edge insertions simultaneously at each time of budget consumption. Note that both VEK and FastCM successfully convert the whole set of $(k-1)$-shell into k-core followers on the datasets Facebook, Email and Brightkite. Our FastCM+ + can achieve even more followers, thanks to the $(k-\lambda)$-shell conversion that convert other $(k-\lambda)$-shells into followers for $\lambda \geq 2$. On the other hand, FastCM uses the shortest running time. Although FastCM+ takes a little more time than FastCM, it achieves much more k-core followers. EKC cannot finish within 48 hours on several large datasets, which performs the worst among all methods. Our proposed methods FastCM+ and FastCM run 1-2 orders of magnitude faster than state-of-the-art VEK. FastCM+ runs at least 34,000**x** faster than EKC and 391**x** faster than VEK on the Baidubaike dataset.

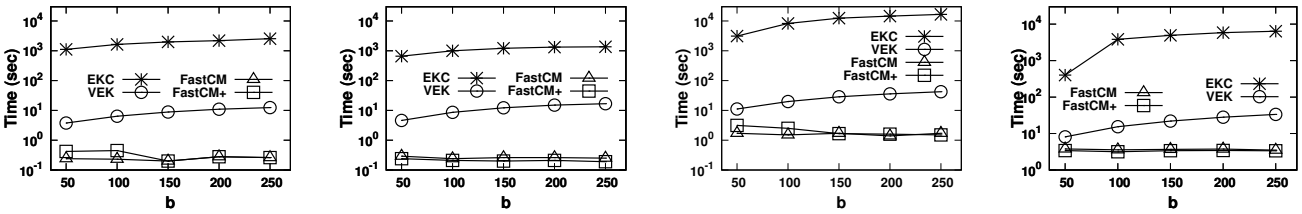
Exp-2: Effectiveness evaluation by varying budget b. Figure 5 reports the number of followers with the increased budget b on four graphs. All methods obtain more followers with an increased budget b. FastCM+ achieves the best performance significantly. FastCM+ can efficiently insert edges to fully convert all the vertices in a component to followers with a sufficient budget. EKC and VEK perform worse than FastCM+ on different parameters b. The comparison between FastCM+ and FastCM in Figures 5(a) and 5(c) shows that our partial conversion strategy greatly improves the quality performance when the budget b is small.

Exp-3: Efficiency evaluation by varying budget b. We evaluate the efficiency of all algorithms in Figure 6. The running time of FastCM+ is stable without significant changes by different parameter b. Our algorithms FastCM and FastCM+ run much faster than EKC and VEK, validating the tighter time complexities of our proposed algorithms. The running time of FastCM+ costs more than FastCM when the budget b is small on Youtube, due to the cost of invoking the partial k-core conversion in Algorithm 3. Note that the running time of FastCM+ decreases on Youtube as b increases. This is because those partial k-core conversions can be changed



(a) Brightkite (b) Gowalla (c) Youtube (d) Twitter

Figure 5: The number of followers varied by budget b.



(a) Brightkite (b) Gowalla (c) Youtube (d) Twitter

Figure 6: The running time of different algorithms varied by budget b.

as the complete conversions for a larger budget b in a faster way. FastCM takes the smallest time cost but may bring fewer followers when the budget b is smaller as shown in Figure 5.

Exp-4: Effectiveness evaluation by varying parameter k. This experiment evaluates the effectiveness impact of parameter k by all the methods on four datasets. Figure 7 shows the results of followers by varying k from 10 to 30. EKC fails to complete the core maximization task with N/A for $k = 10$ on Figures 7(b) and 7(c). With the increase of k, the number of followers shows a decreasing trend. This is because the size of $(k-1)$-shell becomes smaller with a larger k. Our FastCM+ can consistently achieve the best performance, thanks to the extension techniques of handling $(k-\lambda)$-shell conversion for various parameters λ and k.

Exp-5: Efficiency evaluation by varying parameter k. We evaluate the running time of different algorithms on the same setting and datasets as Exp-4. Figure 8 shows the running times of all methods by varying k. Our proposed FastCM and FastCM+ runs much faster than two other competing approaches for different k on most datasets. EKC is the worst in most cases. EKC performs bad especially when k is small. The size of $(k-1)$-shell is large with a small k. Thus, a large number of candidate edges are needed to consider by costing more computation. Overall, our algorithms FastCM and FastCM+ are scalable well with the increased k.

Exp-6: Ablation study. We conduct an ablation study of FastCM+ on two important techniques of *partial conversion* and *dynamic programming based edge selection*. We compare three baseline methods with FastCM+. The first one is FastCM proposed in Section 5. The second method noAnchor-FastCM+ is a variant of FastCM+ equipped with the partial conversion but without anchoring any low-layered vertices. The third method noDP-FastCM+ is a variant of FastCM+ using a greedy strategy for edge selection as FastCM.

Table 3: Ablation study of FastCM+ on Brightkite.

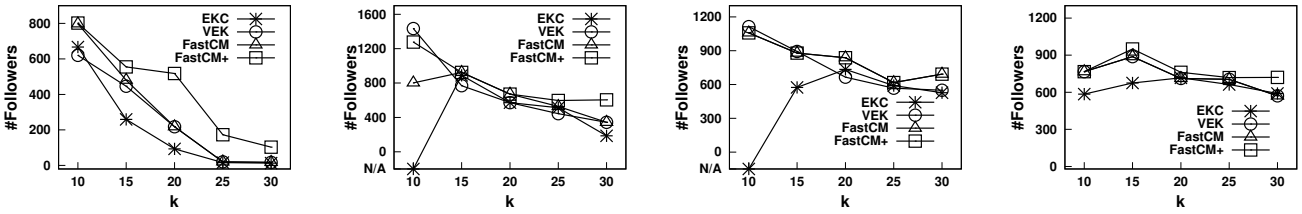
Method	#Followers $\lvert\mathcal{F}\rvert$	$\lvert\Psi_k\rvert$	$\frac{\lvert\mathcal{F}\rvert}{\lvert\Psi_k\rvert}$	Time (s)
FastCM	283	5391	5.25%	0.11
noAnchor-FastCM+	337		6.25%	0.23
noDP-FastCM+	534		9.91%	0.24
FastCM+	539		10%	0.24

Table 3 reports the number of followers, the incremental follower ratio $\frac{|\mathcal{F}|}{|\Psi k|}$, and the running times on Brightkite. Both noDP-FastCM+ and FastCM use the same strategy of greedy edge selection, but noDP-FastCM+ achieves 251 more followers than FastCM, demonstrating the significant effectiveness of partial conversion techniques. noAnchor-FastCM+ and noDP-FastCM+ clearly obtain less followers than FastCM+, reflecting the usefulness of anchoring low-layered vertices in Section 6.2 and dynamic programming based edge selection in Section 6.3. Overall, FastCM+ achieves the largest incremental follower ratio $\frac{|\mathcal{F}|}{|\Psi_k|}$ of 10.0%.

Exp-7: Comparison of different core maximization models. We compare the anchored node model RCM [21] and our edge insertion model FastCM+. We use the same parameter b to anchor b nodes and insert b edges for two models, respectively. Figure 9 reports the number of followers and the running times for $k = 20$ and $b = 200$. The results show that FastCM+ significantly outperforms the RCM algorithm on all datasets, in terms of both quality and efficiency.

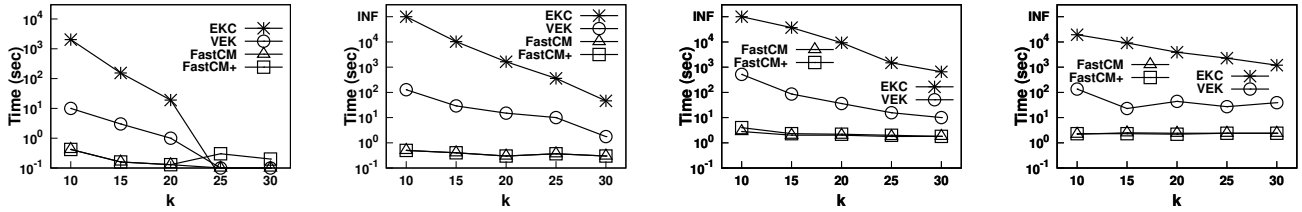
Exp-8: Case study on flight networks. We apply core maximization on a flight network G in Russia.[1] Each vertex represents an airport. An edge (v, u) from airport v to airport u indicates an airline between them. The rationale of adding new airlines to enlarge

[1] https://openflights.org/data.html



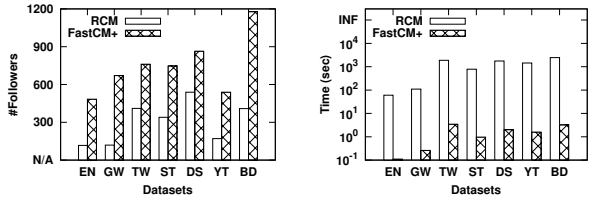
(a) Brightkite (b) Gowalla (c) Youtube (d) Twitter

Figure 7: The number of followers varied by the coreness parameter k.



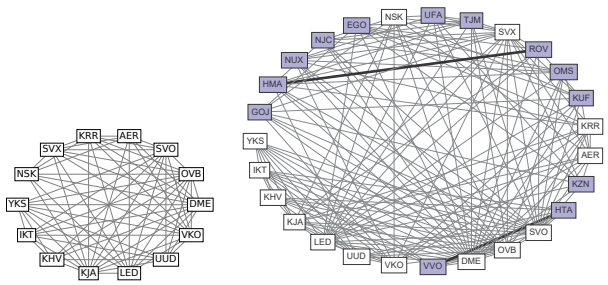
(a) Brightkite (b) Gowalla (c) Youtube (d) Twitter

Figure 8: The running time of different algorithms varied by the coreness parameter k.



(a) The number of followers (b) Running times

Figure 9: Comparison of RCM **[21] and** FastCM+**.**



(a) The original 7-core (b) The updated 7-core

Figure 10: Case study of core maximization on flight networks. Here, $k = 7$ and $b = 2$. Two inserted edges $\hat{E}$ ={(HMA, ROV), (VVO, HTA)} brings 13 k-core followers in blue.

k-core is as follows. It is well known that connectivity plays an important role in an airport's strategic planning [6]. An airport module is a group of airports with strong internal links, but weak connections to the rest of the flight network. The airport module is suitable to be modeled as a connected k-core, where each airport is highly interconnected with at least k other airports but has less than k airlines to another module. Adding new airlines to enlarge k-core certainly improves airports' accessibility and also the flight connectivity of module in a region, which is of interest to carrier operators, airports, and regional governments [30]. Our core maximization aims at providing guidelines for airports to identify which new routes would enhance the connectivity. We apply FastCM+ on G for $k = 7$ and $b = 2$. Figure 10(a) depicts the 7-core of flight network G, where every airport has at least seven airlines with other airports. Here, the abbreviations denote the 3-letter (IATA) code of the airport. Figure 10(b) presents the enlarged 7-core, after inserting two dark black airlines $\hat{E} = \{(\text{HMA, ROV}), (\text{VVO, HTA})\}$ into G. Indeed, FastCM+ brings 13 k-core followers in blue, indicating that 13 new airports join the highly-connected flight network of 7-core by just adding two new airlines.

8 CONCLUSION

In this paper, we study the problem of core maximization, which adds b new edges into a graph to enlarge the k-core. We propose a novel fast algorithm FastCM+ based on several well-designed heuristic strategies including the k-shell partition, the complete and partial $(k - \lambda)$-shell conversions, and also the dynamic programming optimizations. Extensive experiments on eleven large graph datasets validate the efficiency and effectiveness of our proposed algorithm FastCM+ against the state-of-the-art methods of core maximization.

ACKNOWLEDGMENTS

This work was supported by the Natural Science Foundation of China under grants 61772361, 61876128, and HK RGC Grant Nos. 22200320, 12200021.

REFERENCES

[1] J Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. 2006. Large scale networks fingerprinting and visualization using the k-core decomposition. In *NeurIPS*. 41–50.

[2] Francesca Arrigo and Michele Benzi. 2016. Edge Modification Criteria for Enhancing the Communicability of Digraphs. *Siam Journal* 37, 1 (2016), 443–468.

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

[4] Elisabetta Bergamini, Pierluigi Crescenzi, Gianlorenzo D'angelo, Henning Meyerhenke, Lorenzo Severini, and Yllka Velaj. 2018. Improving the betweenness centrality of a node by adding links. *Journal of Experimental Algorithmics (JEA)* 23 (2018), 1–32.

[5] Kshipra Bhawalkar, Jon Kleinberg, Kevin Lewi, Tim Roughgarden, and Aneesh Sharma. 2015. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics* 29, 3 (2015), 1452–1475.

[6] Guillaume Burghouwt. 2016. *Airline network development in Europe and its implications for airport planning*. Routledge.

[7] Chen Chen, Mengqi Zhang, Renjie Sun, Xiaoyang Wang, Weijie Zhu, and Xun Wang. 2021. Locating pivotal connections: The K-Truss minimization and maximization problems. *World Wide Web* (2021), 1–28.

[8] Rajesh Chitnis and Nimrod Talmon. 2018. Can we create large k-cores by adding few edges?. In *International Computer Science Symposium in Russia*. 78–89.

[9] Madelaine Daianu, Neda Jahanshad, Talia M Nir, Arthur W Toga, Clifford R Jack Jr, Michael W Weiner, and Paul M Thompson, for the Alzheimer's Disease Neuroimaging Initiative. 2013. Breakdown of brain connectivity between normal aging and Alzheimer's disease: a structural k-core network analysis. *Brain connectivity* 3, 4 (2013), 407–422.

[10] Palash Dey, Suman Kalyan Maity, Sourav Medya, and Arlei Silva. 2021. Network Robustness via Global k-cores. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. 438–446.

[11] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-core community search over labeled graphs. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2006–2018.

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

[13] Gianlorenzo D'Angelo, Martin Olsen, and Lorenzo Severini. 2019. Coverage centrality maximization in undirected networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 501–508.

[14] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29, 1 (2020), 353–392.

[15] Olivier Goldschmidt, David Nehme, and Gang Yu. 1994. Note: On the set-union knapsack problem. *Naval Research Logistics (NRL)* 41, 6 (1994), 833–842.

[16] Shahrzad Haddadan, Cristina Menghini, Matteo Riondato, and Eli Upfal. 2021. Repbublik: Reducing polarized bubble radius with link insertions. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 139–147.

[17] Qiang-Sheng Hua, Yuliang Shi, Dongxiao Yu, Hai Jin, Jiguo Yu, Zhipen Cai, Xiuzhen Cheng, and Hanhua Chen. 2019. Faster Parallel Core Maintenance Algorithms in Dynamic Graphs. *TPDS* 31, 6 (2019), 1287–1300.

[18] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. 2019. Community search over big graphs. *Synthesis Lectures on Data Management* 14, 6 (2019), 1–206.

[19] Hai Jin, Na Wang, Dongxiao Yu, Qiang-Sheng Hua, Xuanhua Shi, and Xia Xie. 2018. Core maintenance in dynamic graphs: A parallel approach based on matching. *TPDS* 29, 11 (2018), 2416–2428.

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

[22] 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.

[23] Qingyuan Linghu, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2020. Global Reinforcement of Social Networks: The Anchored Coreness Problem. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2211–2226.

[24] Bin Liu and Feiteng Zhang. 2020. Incremental algorithms of the core maintenance problem on edge-weighted graphs. *IEEE Access* 8 (2020), 63872–63884.

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

[26] Qing Liu, Xuliang Zhu, Xin Huang, and Jianliang Xu. 2021. Local algorithms for distance-generalized core decomposition over large dynamic graphs. *Proceedings of the VLDB Endowment* 14, 9 (2021), 1531–1543.

[27] Sourav Medya, Tiyani Ma, Arlei Silva, and Ambuj Singh. 2020. A game theoretic approach for core resilience. In *International Joint Conferences on Artificial Intelligence Organization*. 3473–3479.

[28] 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 (2019), 95–102.

[29] Senni Perumal, Prithwish Basu, and Ziyu Guan. 2014. Minimizing Eccentricity in Composite Networks via Constrained Edge Additions. In *Military Communications Conference*. 1894–1899.

[30] Renato Redondi, Paolo Malighetti, and Stefano Paleari. 2011. New routes and airport connectivity. *Networks and Spatial Economics* 11, 4 (2011), 713–725.

[31] Ahmet Erdem Saríyüce, Buğra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V Çatalyürek. 2013. Streaming algorithms for k-core decomposition. *PVLDB* 6, 6 (2013), 433–444.

[32] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. 2018. Patterns and anomalies in k-cores of real-world graphs with applications. *KAIS* 54, 3 (2018), 677–710.

[33] Longxu Sun, Xin Huang, Ronghua Li, Byron Choi, and Jianliang Xu. 2020. Index-based intimate-core community search in large weighted graphs. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[34] Xin Sun, Xin Huang, Zitan Sun, and Di Jin. 2021. Budget-constrained Truss Maximization over Large Graphs: A Component-based Approach. In *CIKM*. 1754–1763.

[35] Dimitrios Vogiatzis. 2013. Influence Study on Hyper-Graphs.. In *AAAI Fall Symposia*. 24–29.

[36] Kai Wang, Xin Cao, Xuemin Lin, Wenjie Zhang, and Lu Qin. 2018. Efficient computing of radius-bounded k-cores. In *ICDE*. 233–244.

[37] Fan Zhang, Wenjie Zhang, Ying Zhang, Lu Qin, and Xuemin Lin. 2017. OLAK: an efficient algorithm to prevent unraveling in social networks. *PVLDB* 10, 6 (2017), 649–660.

[38] 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*, Vol. 31. 245–251.

[39] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2017. When engagement meets similarity: efficient (k, r)-core computation on social networks. *Proceedings of the VLDB Endowment* 10 (2017), 998–1009.

[40] Yikai Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. 2017. A fast order-based approach for core maintenance. In *ICDE*. 337–348.

[41] Zhiwei Zhang, Xin Huang, Jianliang Xu, Byron Choi, and Zechao Shang. 2019. Keyword-centric community search. In *ICDE*. 422–433.

[42] Feng Zhao and Anthony KH Tung. 2012. Large scale cohesive subgraphs discovery for social network visual analysis. *PVLDB* 6, 2 (2012), 85–96.

[43] Wei Zhou, Hong Huang, Qiang-Sheng Hua, Dongxiao Yu, Hai Jin, and Xiaoming Fu. 2021. Core decomposition and maintenance in weighted graph. *World Wide Web* 24, 2 (2021), 541–561.

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

[45] Zhongxin Zhou, Wenchao Zhang, Fan Zhang, Deming Chu, and Binghao Li. 2021. VEK: a vertex-oriented approach for edge k-core problem. *World Wide Web* (2021), 1–18.

[46] Weijie Zhu, Chen Chen, Xiaoyang Wang, and Xuemin Lin. 2018. K-core minimization: An edge manipulation approach. In *CIKM*. 1667–1670.