Dynamic Seed-GrowthCM: A Dynamic Benefit-Oriented Algorithm for Core Maximization on Large Graphs

Dongyuan Ma[1], Dongxiao He[1], Xin Huang[2*]

[1]College of Intelligence and Computing, Tianjin University
[2]Department of Computer Science, Hong Kong Baptist University
{madongyuan,hedongxiao}@tju.edu.cn, xinhuang@comp.hkbu.edu.hk

Abstract

The k-core has garnered significant attention in recent research as an effective measure of node importance within a graph. A k-core is defined as the maximal induced subgraph where each node has a degree of at least k. This paper addresses the core maximization problem: given a graph G, an integer k, and a budget b, the objective is to insert b new distinct edges into G to maximize the size of its k-core. This problem is theoretically proven to be NP-hard and APX-hard. However, the existing heuristic methods often struggle to achieve a good balance between efficiency and answer quality. In this paper, we propose a novel dynamic approach that, for the first time, uncovers the dynamic changes in node degrees. We introduce a new concept using the contribution of edges across different λ-shell components to the final solution. Based on these findings, we present the Dynamic Seed-GrowthCM method. This method selects the λ-shell component with the largest estimated benefit as the initial seed. In each iteration, depending on complete/partial growth, either a new seed is incorporated into the solution, or an existing seed undergoes growth, becoming a larger seed by adding connected components of the λ-shell component to the solution. Experimental results on ten datasets demonstrate that our algorithm significantly outperforms state-of-the-art methods in terms of solution quality on large graphs, while achieving a high computational efficiency.

1 Introduction

The k-core of a graph is widely applied across multiple disciplines due to its ability to simplify the analysis of complex networks [Batagelj and Zaversnik, 2003]. Defined as the maximal induced subgraph where each vertex maintains at least k neighbors, k-core provides a clear view of the fundamental structure within networks [Seidman, 1983]. This characteristic renders k-core particularly useful in various applications such as social network analysis [Govind and Lal, 2021; Li *et al.*, 2018b], community detection [Malliaros *et al.*, 2016], biological research [Guo *et al.*, 2022], financial network analysis [Batagelj and Zaveršnik, 2011], and transportation systems [Bruckner *et al.*, 2015]. Building on this foundation, the core maximization problem aims to maximize the size of the k-core by inserting a specified number of edges, a challenge that is recognized as NP-hard and APX-hard [Chitnis and Talmon, 2018; Malliaros *et al.*, 2020; Zhou *et al.*, 2019].

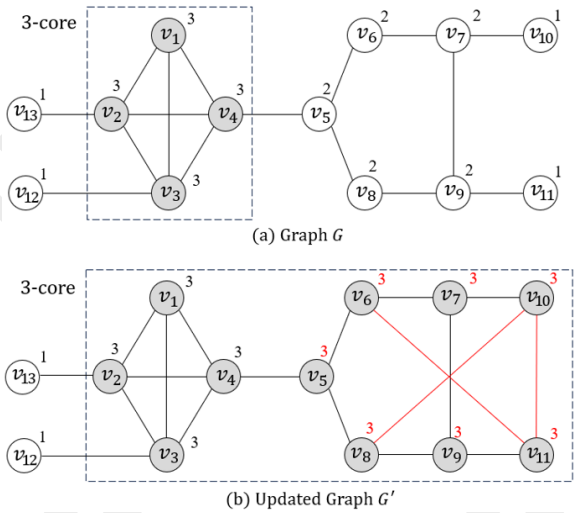


Figure 1: An example of k-core maximization in graph G.

Motivation example. Consider a graph G with 13 vertices, as illustrated in Figure 1(a). The entire graph G forms a 1-core where every vertex has at least one neighbor. The coreness of a vertex, indicated by the number next to its label in Figure 1, refers to the largest k such that the vertex belongs to the k-core of the graph. For instance, the coreness of v_6 is 2. The 3-core includes four vertices: $\{v_1, v_2, v_3, v_4\}$.

Core maximization has numerous practical applications in real-world scenarios. In social network analysis, adding connections can enhance information spread, making marketing campaigns more effective [Li *et al.*, 2018a; Aral and Walker, 2012]. In biology, strengthening protein-protein interaction networks helps discover key functional modules and drug targets [Yue *et al.*, 2020]. In transportation networks, adding routes increases robustness, reduces congestion, and

*Corresponding authors.

improves connectivity [Sarıyüce *et al.*, 2016].

Challenges and Contributions. Core maximization is a challenging task due to its NP-hard and APX-hard nature and high computational complexity, which makes it difficult to solve efficiently [Zhou *et al.*, 2019]. Early methods like EKC [Zhou *et al.*, 2019] and VEK [Zhou *et al.*, 2022] adopt a greedy edge insertion approach, inserting edges one by one to maximize the k-core size. EKC suffers from significant computational complexity, making it impractical for large graphs. VEK attempts to improve efficiency by using a vertex-oriented heuristic, but it still encounters performance bottlenecks. Later, FastCM/FastCM+ [Sun *et al.*, 2022] improved upon these methods by partitioning the graph and adding nodes in groups, which provides greater flexibility. However, all of the above methods directly insert edges in each iteration, overlooking the changes in the degrees of the nodes in the current solution during each iteration, which reduces the number of edges that need to be inserted.

Example-1. Consider Figure 1(a), where $k = 3$ and $b = 3$. FastCM+ transforms the node set $\{v_5, v_6, v_7, v_8, v_9\}$ into a k-core in the first iteration, inserting the edges $\{(v_6, v_9), (v_7, v_8)\}$. However, when v_{10} and v_{11} are considered part of the solution, the degrees of v_7 and v_9 become 3, at which point no edge needs to be inserted that has one endpoint as v_7 or v_9.

These limitations motivate the need for a more adaptive solution. In this paper, we propose Dynamic Seed-GrowthCM, which addresses the shortcomings of these existing approaches by dynamically adjusting the estimated benefit of each λ-shell component before each iteration using two strategies—complete growth and partial growth. Experimental results demonstrate that Dynamic Seed-GrowthCM outperforms state-of-the-art methods on a range of datasets. In summary, we make the main contributions in this paper as follows.

- We propose a novel algorithm, Dynamic Seed-GrowthCM, which dynamically adjusts the estimated benefit of each λ-shell component in each iteration to effectively address the core maximization problem.
- We introduce two growth strategies—complete growth and partial growth—that enable a flexible approach to adding adjacent λ-shells, enhancing the adaptability of the solution.
- We conduct extensive experiments on multiple datasets, demonstrating that Dynamic Seed-GrowthCM significantly outperforms existing state-of-the-art methods in both solution quality and computational efficiency.

2 Related Work

K-Core Decomposition and Maintenance. K-core decomposition is a fundamental technique in network analysis that identifies cohesive subgraphs by iteratively removing nodes with fewer than k connections. Initially proposed by Seidman [Seidman, 1983], and later optimized by Batagelj and Zaveršnik [Batagelj and Zaversnik, 2003], this method efficiently analyzes large-scale networks. Streaming algorithms by Sarıyüce et al. [Saríyüce *et al.*, 2013] further enable k-core decomposition on dynamic graphs, allowing real-time updates. For parallel core maintenance, Hua et al. [Hua *et al.*, 2019] and Jin et al. [Jin *et al.*, 2018] developed faster approaches, improving the efficiency of updating k-core structures as graphs evolve. Zhang et al. [Zhang *et al.*, 2017b] introduced an order-based core maintenance approach, accelerating the process by leveraging node sequence optimizations. Core maintenance on edge-weighted graphs has also been addressed by Liu and Zhang [Liu and Zhang, 2020], and Zhou et al. [Zhou *et al.*, 2021], expanding its application to more complex networks.

To handle large dynamic graphs, Zhang et al. [Weng *et al.*, 2021] developed distributed algorithms that efficiently distribute the workload across multiple processors, ensuring scalability. Additionally, Khaouid et al. [Khaouid *et al.*, 2015] proposed a method for k-core decomposition on large networks using a single PC, while Bonchi et al. [Bonchi *et al.*, 2019] extended the k-core concept to distance-generalized cores. The anchored k-core problem, introduced by Bhawalkar et al. [Bhawalkar *et al.*, 2015], stabilizes core structures by anchoring specific nodes, ensuring resilience even under node removal. A federated approach has also been proposed for secure distributed k-core decomposition [Guo *et al.*, 2024].

Core Minimization and Maximization. Core minimization and maximization are critical approaches in network analysis, focusing on manipulating the k-core structure for various objectives.

In core minimization, the goal is to reduce the k-core size by weakening the connectivity. Zhu et al. [Zhu *et al.*, 2018] proposed edge manipulation techniques to remove edges strategically, reducing the k-core and its influence in the network. Liu et al. [Liu *et al.*, 2021] tackled the anchored k-core budget minimization problem, aiming to minimize the k-core size while maintaining core stability within a defined budget. Zhang et al. [Zhang *et al.*, 2017a] addressed the collapsed k-core problem, focusing on the removal of critical users in social networks whose exit leads to a collapse of the k-core structure, weakening the network significantly.

In contrast, core maximization focuses on enhancing the k-core size. Laishram et al. [Laishram *et al.*, 2020] introduced residual core maximization to increase network stability by expanding residual cores. Zhou et al. [Zhou *et al.*, 2019] used an edge addition approach to achieve a similar goal, while Zhou et al. [Zhou *et al.*, 2022] proposed a vertex-oriented approach for edge k-core problems. Sun et al. [Sun *et al.*, 2022] developed scalable algorithms to efficiently maximize the k-core in large graphs. Medya et al. [Medya *et al.*, 2020] employed a game-theoretic approach to improve core resilience, while Laishram et al. [Laishram *et al.*, 2018] explored ways to measure and enhance the resilience of k-core structures against node and edge removal.

3 Preliminaries

Consider an undirected and unweighted graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Let n be the number of vertices and m be the number of edges, with the assumption that $m \geq n - 1$. For any subgraph S, the neighborhood of a vertex u is represented as $N(u, S)$, and the

degree of u in S, meaning the number of neighbors of u in S, is denoted as $\deg(u, S) = |N(u, S)|$.

Definition 1 (k-core). Given a graph G, a subgraph S is the k-core of G, denoted by $C_k(G)$, if: (i) $\forall u \in S$, $\deg(u, S) \geq k$; and (ii) $\nexists S' \supset S$ that satisfies (i).

The k-core can be obtained by recursively removing every vertex with a degree less than k in the graph, with a time complexity of $O(m)$ [Seidman, 1983]. This process reveals a hierarchical structure, where each node is associated with a λ-shell, representing the collection of nodes with the same coreness. The coreness of a node is determined by the largest k-core in which it exists.

Definition 2 (coreness). Given a graph G, the coreness of a vertex u, denoted by $\text{cn}(u)$, is defined as the largest integer k such that $u \in C_k(G)$ and $u \notin C_{k+1}(G)$.

Definition 3 (λ-shell). Given a graph G, a subgraph S is the λ-shell of G, denoted by $H_\lambda(G)$, if: (i) $\forall u \in S$, $\text{cn}(u) = \lambda$; and (ii) $\nexists S' \supset S$ that satisfies (i).

Based on the above definitions, we can formulate the problem of core maximization as follows.

Problem Statement. Given a graph G, an integer k, and a budget b, the core maximization problem aims to insert b new distinct edges into G to maximize the size of its k-core. The nodes that are converted into the k-core are referred to as k-core followers.

4 Dynamic Seed-GrowthCM

4.1 Method Overview

In this section, we introduce Dynamic Seed-GrowthCM, a novel approach to core maximization that emphasizes dynamic adaptation. It consists of four phases as follows.

- **Phase-I: λ-Shell Partition**. In this phase, the λ-shell is divided into several disjoint components. Each component serves as the smallest unit for conversion within our method, allowing for independent edge insertions aimed at core conversion.
- **Phase-II: Initialization**. In this phase, the estimated benefit of each λ-shell component is calculated and initialized. Estimated benefit measures the gain of converting a λ-shell component into part of the k-core.
- **Phase-III: Iterative Dynamic Growth**. In each iteration, we either select a new seed to add to the current solution or grow an existing seed—through either complete or partial growth. Then, the estimated benefit of all components is updated.
- **Phase-IV: Offline Storage Phase**. Given the predictable growth patterns of graphs, we store strategies offline based on common graph behaviors. When the same graph is encountered again, these pre-determined strategies are quickly retrieved and applied, enhancing efficiency and reducing the need for recalibration. This approach accelerates the core maximization process by instantly implementing effective strategies.

4.2 λ-Shell Partition

In processing large sparse graph data, the effective implementation of core maximization methods is crucial. The budget b for inserting edges is inherently much smaller than the potential edges, quantified as $\left|\frac{|V|\times(|V|-1)}{2} - |E|\right|$, emphasizing the need for an efficient search strategy to handle the vast difference. The hierarchical structure formed by λ-shells is an excellent choice for core maximization. However, since λ-shells are not necessarily connected, we employ λ-Shell Partition to divide them into several connected components, which better facilitates solution generation.

Definition 4 (λ-Shell Partition). The λ-Shell Partition divides $H_\lambda(G)$ into its connected components, denoted as:

$$D(H_\lambda(G)) = \{C_1, C_2, \ldots, C_h\}.$$

where each C_i is a connected component of $H_\lambda(G)$, and satisfies:

$$\bigcup_{i=1}^{h} C_i = H_\lambda(G) \quad \text{and} \quad C_i \cap C_j = \emptyset \quad \text{for all } i \neq j.$$

Theorem 1 (Local Insertion Closure for λ-Shell Components). For any λ_1-shell component C_1 and λ_2-shell component C_2, an edge insertion (u, v) where $u, v \in C_1$ will not change the coreness of any node in C_2.

Proof. Assume that the coreness of $w \in C_2$ has changed. Since no edges have been deleted in the graph, it is obviously true that $\text{cn}(w)$ cannot decrease. If $\text{cn}(w)$ increases to $\lambda_2 + 1$, according to the k-core maintenance rules [Saríyüce *et al.*, 2013], by setting $k = \lambda_2 + 1$, it implies that $\lambda_1 = \lambda_2$ and w is reachable from u or v via a path consisting of vertices with coreness λ_2. This means that $w \in C_1 \cap C_2$. However, this contradicts the definition of λ-shell components, as C_1 and C_2 are distinct, disconnected components. Therefore, the theorem holds.

4.3 Initialization

λ-threshold. We first analyze the structure of the λ-shell component and propose a new concept, the λ-threshold component, to identify the nodes that play an important role in the process of converting the λ-shell component into part of the k-core.

Since the λ-shell component has a local insertion closure, i.e., inserting edges in one component does not affect other components, we focus on considering how to completely convert the nodes in a component to part of the k-core. First, we provide the definition of the dynamic solution for k-core maximization and the dynamic degree of the node, allowing the node's degree to change with the solution, giving the node better adaptability.

Definition 5 (Dynamic Solution) Given a graph G, an integer k, and a budget b, the dynamic solution of the graph, denoted as $V(k, b, i)$, represents the k-core obtained after the i-th iteration.

Definition 6 (Dynamic Degree) Given a graph G, an integer k, and a budget b, the dynamic degree of a node u, denoted as $\deg(u, k, b, i)$, is the number of neighbors of u in the k-core after the $(i-1)$-th iteration, plus the number of nodes in the same component as u. Formally, it is defined as:

$$\deg(u, k, b, i) = |\{v \in N(u) \mid v \in V(k, b, i-1) \cup C_u\}|.$$

Algorithm 1 Initialization

Input: a λ-shell component C_i
Output: the estimated benefit of the $\text{eb}(C_i)$

```
Obtain the λ-threshold component T(C_i, k, b, 0);
for all nodes u ∈ T(C_i, k, b, 0) do
    Calculate deg(u, k, b, 1);
end for
E_need(C_i) = 1/2 Σ_{u∈T(C_i,k,b,j)} (k − deg(u, k, b, 1));
eb(C_i) = |C_i| / E_need(C_i);
Return eb(C_i);
```

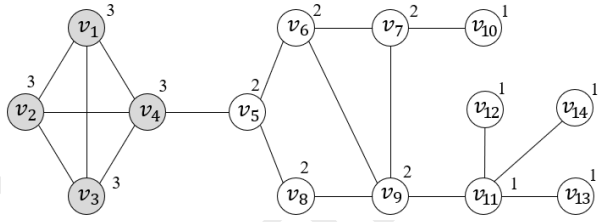


Figure 2: An example of Effective Edges, Semi-Effective Edges, and Ineffective Edges in G. The coreness of each vertex is indicated by a number next to the vertex label.

where $N(u)$ represents the set of neighbors of u in G, and C_u denotes the λ-shell component to which u belongs.

Then, based on the following observation, we define the λ-threshold to identify the nodes that play a key role in the solution generation process.

Given a graph G, a λ-shell component C_i, the nodes in C_i all have coreness λ, but $\deg(u, k, b, i)$ can be larger than λ where $u \in C_i$. These nodes do not become part of the k-core because the number of neighbors satisfying the k-core requirement is less than k.

Example-2. Consider Figure 1(a), where $k = 3$ and $b = 3$, node v_5 has $\deg(v_5, 3, 3, 1) = 3$ at the beginning of the first iteration, but because its neighbors v_6 and v_8 have $\deg(v_6, 3, 3, 1) = 2$ and $\deg(v_8, 3, 3, 1) = 2$, v_5 does not belong to the k-core. After selecting the λ-shell component to be converted into part of the k-core, we must carefully choose which edges to add, rather than adding them randomly.

Definition 7 (λ-threshold component) Given a graph G, an integer k, a budget b, and a λ-shell component C_i, at the j-th iterationits, its λ-threshold component, denoted as $T(C_i, k, b, j)$, is defined as the set of nodes in C_i whose dynamic degree is less than or equal to k, denoted as:

$$T(C_i, k, b, j) = \{u \in C_i \mid \deg(u, k, b, j) \leq k\}.$$

Estimated benefit initialization. In this step, we use the dynamic degree and λ-threshold component to initialize the estimated benefit of each λ-shell component. First, we introduce a method for determining the number of edges needed to convert any λ-threshold component to part of the k-core.

Theorem 2. Given a graph G, an integer k, a budget b, and a λ-shell component C_i, in the j-th iteration, C_i becomes part of a k-core if every node u in $T(C_i, k, b, j)$ is newly connected to other nodes within C_i or $V(k, b, i)$ by $(k - \deg(u, k, b, j))$ edges.

We assume that the conversion of the λ-threshold component C_i into a k-core can be achieved entirely by adding edges

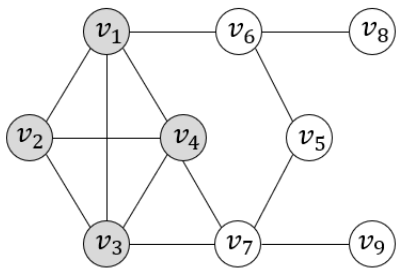


Figure 3: An example to illustrate the advantages of partial growth compared to complete growth.

between nodes with dynamic degrees less than k. The number of edges required to achieve this transformation is given by:

$$E_{\text{need}}(C_i) = \frac{1}{2} \sum_{u \in T(C_i,k,b,j)} (k - \deg(u, k, b, j))$$

Then, we can define the estimated benefit of each λ-shell component.

Definition 8 (Estimated benefit) For a λ-shell component C_i, its estimated benefit, denoted as $\text{eb}(C_i)$, is defined as $\frac{|C_i|}{E_{\text{need}}(C_i)}$, where $E_{\text{need}}(C_i)$ represents the number of edges required to convert C_i into part of the k-core.

4.4 Iterative Dynamic Growth

By Theorem 2, we can deduce that within each λ-shell component, edges can be inserted between the λ-threshold components until the k-core requirement is met. The choice of edges to insert is not unique. Furthermore, we observe that within each λ-shell component, the difference between applying Complete Conversion and Partial Conversion, as proposed in FastCM/FastCM+ [Sun *et al.*, 2022], is minimal. Therefore, this method focuses on the contribution of edges that span across different λ-shell components, and introduces Complete Growth and Partial Growth. We first introduce the definitions of seed and growth.

Definition 9 (Seed) Given a graph G, an integer k, and a budget b, after the i-th iteration,

$$V(k, b, i) = \{S_1, S_2, \ldots, S_n\},$$

where each S_j is a connected component of $V(k, b, i)$, we refer to S_j as a seed.

Definition 10 (Growth) Given a seed S_j, the definition of growth is to add some λ-shell components $C_{\text{growth}}(S_j) = \{C_1, C_2, \ldots, C_m\}$ that are connected to the seed to the current solution, such that the seed grows to $S'_j = S_j \cup C_{\text{growth}}(S_j)$. For complete growth and partial growth, we record S'_j as $\text{CG}(S_j)$ and $\text{PG}(S_j)$, respectively.

Complete Growth. In this section, we present the complete growth, which takes into account the dynamic relationships between different λ-shell components. To begin, we classify the edges connecting λ-shell components into the following categories: effective edges, semi-effective edges, and ineffective edges.

Definition 11 (Effective Edges, Semi-Effective Edges, and Ineffective Edges): When in the i-th iteration, we attempt to convert the λ-shell component C_1 into k-core followers. Suppose C_1 is connected to $V(k, b, i-1)$ through

Algorithm 2 Complete Growth

Input: Graph G, Integer k, Budget b, estimated benefit for all λ-shell components EB $= \{\text{eb}(C_1), \text{eb}(C_2), \ldots, \text{eb}(C_n)\}$
Output: Set of inserted edges E

$i \leftarrow 0$, SeedSet $\leftarrow \emptyset$;
while $b > E_{\text{need}}(V(k,b,i))$ **do**
  $C_j = \arg\max(\text{eb}(C_i)), C_i \in EB$;
  $S_h = \arg\max(\text{eb}(CG(S_i) \setminus S_i)), S_i \in SeedSet$;
  if eb(C_j)>eb(S_h) **then**
    SeedSet $\leftarrow$ SeedSet $\cup\, C_j$ (Add C_j to SeedSet and update it to S_j);
  else
    Completely grow S_h to S'_h;
    SeedSet $\leftarrow$ SeedSet $\setminus\, S_h \cup S'_h$;
  end if
  $i \leftarrow i + 1$;
  for all nodes $u \in$ SeedSet **do**
    Calculate $\deg(u,k,b,i)$;
  end for
end while
Insert edges in SeedSet such that each node's degree achieves k;
Return the inserted edges E;

edge $e(u,v)$, where $u \in C_1$ and $v \in V(k,b,i-1)$. Consider the case where the edge does not exist, and calculate $\deg(u,k,b,i)$ and $\deg(v,k,b,i)$. In this case:

(i) If $\deg(u,k,b,i) < k$ and $\deg(v,k,b,i) < k$, then $e(u,v)$ is an **effective edge**.

(ii) If $\deg(u,k,b,i) \geq k$ and $\deg(v,k,b,i) \geq k$, then $e(u,v)$ is an **ineffective edge**.

(iii) Else, $e(u,v)$ is a **semi-effective edge**.

When both C_u and C_v are in the current solution, compared to the case where the edge does not exist, a valid edge reduces $E_{\text{need}}(C_u)$ and $E_{\text{need}}(C_v)$ by 0.5 each. A semi-valid edge reduces $E_{\text{need}}(C_u)$ or $E_{\text{need}}(C_v)$ by 0.5, while an invalid edge has no effect on $E_{\text{need}}(C_u)$ and $E_{\text{need}}(C_v)$. Here, C_u and C_v represent the λ-shell components to which nodes u and v belong, respectively.

In the i-th iteration, the complete growth of a seed means adding all the λ-shell components connected to that seed into the current solution. Then the dynamic degree of nodes is updated. The growth process stops if the following condition holds: $E_{\text{need}}(V(k,b,i)) = \frac{1}{2}\sum_{u \in V(k,b,i)} \max(0, (k - \deg(u,k,b,i+1))) > b$. To accommodate the dynamic change of node degrees, we perform edge insertion after the iteration is completed.

Example-3. Consider Figure 2, where $k = 3$ and $b = 6$. In the first iteration, the algorithm attempts to convert $\{v_5, v_6, v_7, v_8, v_9\}$ into k-core followers, $e(v_4, v_5)$ is an semi-effective edge. In the second iteration, the algorithm tries to convert $\{v_{11}, v_{12}, v_{13}, v_{14}\}$ into k-core followers, $e(v_9, v_{11})$ is an ineffective edge. In the third iteration, the algorithm attempts to convert $\{v_{10}\}$ into k-core followers, $e(v_7, v_{10})$ is an effective edge.

Partial Growth. Complete growth effectively improves the solution quality of the core maximization. However, the complete growth strategy may be insufficient in certain situations.

Algorithm 3 Partial Growth

Input: Graph G, Integer k, Budget b, estimated benefit for all λ-shell components EB $= \{\text{eb}(C_1), \text{eb}(C_2), \ldots, \text{eb}(C_n)\}$
Output: Set of inserted edges E

$i \leftarrow 0$, SeedSet $\leftarrow \emptyset$;
while $b > E_{\text{need}}(V(k,b,i))$ **do**
  $C_j = \arg\max(\text{eb}(C_i)), C_i \in EB$;
  $S_h = \arg\max(\text{eb}(PG(S_i) \setminus S_i)), S_i \in SeedSet$;
  if eb(C_j)>eb(S_h) **then**
    SeedSet $\leftarrow$ SeedSet $\cup\, C_j$ {(Add C_j to SeedSet and update it to S_j)};
  else
    Partially grow S_h to S'_h;
    SeedSet $\leftarrow$ SeedSet $\setminus\, S_h \cup S'_h$;
  end if
  $i \leftarrow i + 1$;
  for each edge $e = (u,v)$, where $u \in S'_h \setminus S_h$ and $v \notin V(k,b,i)$ **do**
    if edge e is effective **then**
      $E_{\text{need}}(C_v) \leftarrow E_{\text{need}}(C_v) - 1$;
    else if edge e is semi-effective **then**
      $E_{\text{need}}(C_v) \leftarrow E_{\text{need}}(C_v) - 0.5$;
    end if
    Update $\text{eb}(C_v) = \frac{|C_v|}{E_{\text{need}}(C_v)}$;
  end for
end while
Insert edges in all λ-threshold components such that each node's degree achieves k;
Return the inserted edges E;

For instance, the number of edges required to convert all connected components into the k-core might exceed the available budget b, preventing further growth. To address this limitation, we propose a more fine-grained partial growth strategy.

Example-4. Consider Figure 3, where $k = 3$ and $b = 2$. When using complete growth, in the first iteration, $\{v_5, v_6, v_7\}$ is transformed into the k-core, requiring $b = 1$. In the second iteration, the algorithm attempts to grow $\{v_5, v_6, v_7\}$ into $\{v_5, v_6, v_7, v_8, v_9\}$, requiring $b = 2.5$. Since the budget b is not sufficient, the growth fails, and the iteration ends. The final k-core followers are $\{v_5, v_6, v_7\}$, and the inserted edges are $e(v_4, v_6)$ and $e(v_4, v_5)$. As we assume the graph is unweighted, edges like $e(v_5, v_6)$ cannot be reconnected if they already exist. However, this is clearly not the optimal solution, as inserting $e(v_4, v_8)$ and $e(v_5, v_8)$ would result in the k-core followers being $\{v_5, v_6, v_7, v_8\}$.

The above example demonstrates that by refining the growth process and dividing it into more granular steps—where each growth phase converts only one λ-shell component—we can achieve superior results. This is the key idea behind partial conversion. However, to implement partial conversion, a critical issue must be addressed: how to dynamically adjust the estimated benefit of each λ-shell component to ensure that the optimal one is selected for growth.

We derive the following **estimated benefit update rule**: In the i-th iteration, after the algorithm adds the λ-shell component C_i to the k-core followers, all λ-shell components C_j connected to C_i will be updated as follows: for each effective

Dataset	$\|V\|$	$\|E\|$	#Followers							Running Time (s)						
			EKC	VEK	FastCM	FastCM+	DSG CG	DSG PG	DSG PG+	EKC	VEK	FastCM	FastCM+	DSG CG	DSG PG	DSG PG+
Facebook	4,039	88,234	39	77	77	198	333	**373**	**373**	0.53	0.28	**0.003**	0.01	0.023	0.131	0.007
email-Enron	36,692	183,831	112	140	140	241	323	**337**	**337**	3.13	0.66	0.026	0.051	**0.007**	2.325	0.018
Brightkite	58,228	214,078	93	218	218	518	539	**597**	**597**	18.8	1.03	0.044	0.247	**0.006**	1.821	0.023
Gowalla	196,591	950,327	571	571	671	671	788	**879**	**879**	1,071	15.17	0.155	0.177	**0.014**	9.197	0.054
Twitter	81,306	1,768,149	712	712	755	761	1,117	**1,221**	**1,221**	3,923	28.05	1.168	1.033	**0.02**	5.378	0.061
Stanford	281,903	2,312,497	-	508	734	747	929	**948**	**948**	-	193	0.292	0.326	**0.05**	8.223	0.064
Google	875,713	5,105,039	-	1,668	2,009	2,017	2,245	**2,270**	**2,270**	-	922	1.14	3.03	**0.178**	40.093	0.2
Youtube	1,134,890	2,987,624	734	665	831	838	903	**905**	**905**	9,472	35.38	0.941	1.017	**0.023**	45.516	0.223
Baidubaike	2,142,101	17,014,946	-	1,035	1,161	1,178	1,359	**1,359**	**1,359**	-	1,987	2.149	2.291	**0.245**	181.162	0.379
as-Skitter	1,696,415	11,095,928	-	1,618	1,739	1,809	1,856	**1,915**	**1,915**	-	716	2.412	2.578	**0.276**	54.661	0.301

Table 1: Comparison of methods in terms of Graph Statistics, #Followers, and Running Time across different datasets. '-' denotes that the algorithm cannot finish within 48 hours.

edge between C_j and C_i, the $E_{\text{need}}(C_j)$ is reduced by 1, and for each semi-effective edge, the $E_{\text{need}}(C_j)$ is reduced by 0.5. The estimated benefit of C_j is then recalculated based on the updated $E_{\text{need}}(C_j)$ value.

The above update rule is based on the observation that when C_i become k-core followers, all the edges connecting C_i and C_j will still make the dynamic degree of the nodes in C_i change, but since C_i has been added to the solution, the $\text{eb}(C_i)$ change has no effect on the subsequent iteration process, therefore, we reflect this benefit in C_j. In addition, in the i-th iteration, the partial growth of a seed means adding the λ-shell components with largest estimated benefit connected to that seed into the current solution. Then the estimated benefit of the λ-shell component connected to the grown seed will be updated according to the rules presented above.

4.5 Offline Storage

Compared to complete growth, partial growth offers advantages in solution quality but requires more computational time. However, we can still accelerate the process using offline storage techniques. Due to the inherent invariability of the underlying structure of the graph, certain properties remain consistent across the same graph [Malliaros *et al.*, 2020]. In k-core maximization, the relative estimated benefit order of the λ-shell components remains largely consistent, even when faced with different parameters k and b. By storing the relative estimated benefit order, the next time we encounter the same graph, we can directly generate k-core followers based on this order without recalculating the estimated benefit of each λ-shell component.

5 Experiments

In this section, we present the results of evaluating our proposed method on multiple public datasets. We detail the experimental setup, including the parameters used, and the evaluation metrics applied to assess performance. A comprehensive comparison of the results is also provided, showing how our method performs relative to state-of-the-art algorithms.

Datasets. We use 10 real-world network datasets in our experiments, chosen for their diversity and relevance to the task. Table 1 summarizes the key graph statistics for these datasets. All datasets are obtained from two public repositories: Network Repository (http://networkrepository.com) and SNAP (http://snap.stanford.edu), and are treated as undirected graphs to ensure consistency across experiments.

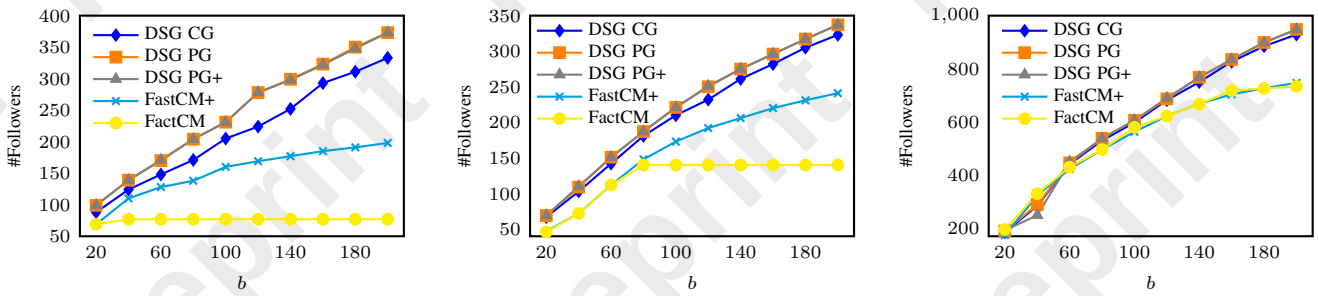
Compared methods. We compare the proposed method against four other methods, described as follows.

- EKC [Zhou *et al.*, 2019]: follows a greedy approach by adding one edge between non-adjacent vertices per iteration to maximize the k-core.
- VEK [Zhou *et al.*, 2022]: adopts a greedy strategy focused on adding one vertex per iteration, optimizing the process through a scoring function and efficient candidate pruning.
- FastCM [Sun *et al.*, 2022]: performs complete conversion and focuses on converting vertices within the $(k-1)$-shell.
- FastCM+ [Sun *et al.*, 2022]: combines complete and partial conversion strategies to convert vertices in the $(k-\lambda)$-shell ($\lambda \geq 1$), using DP-based node selection methods.
- Dynamic Seed-GrowthCM CG: is our proposed algorithm using complete growth, abbreviated as DSG CG.
- Dynamic Seed-GrowthCM PG: is our proposed algorithm using partial growth, abbreviated as DSG PG.
- Dynamic Seed-GrowthCM PG+: is Dynamic Seed-GrowthCM PG combined with an Offline Storage technique, abbreviated as DSG PG+.

Parameters and evaluation metrics. By default, We set the parameter k to 20 and the budget b to 200. We use the number of k-core followers and the algorithm running time as evaluation metrics. Due to the previously demonstrated poor performance of EKC [Zhou *et al.*, 2019] and VEK [Zhou *et al.*, 2022] in terms of both runtime and solution quality, we directly refer to the results reported in [Sun *et al.*, 2022] rather than conducting the experiments again. All experiments were conducted on a machine equipped with an AMD Ryzen 7 5800H CPU (3.2 GHz), 16 GB of RAM, and running Windows 11. Each experiment was repeated ten times, and the average results are reported.

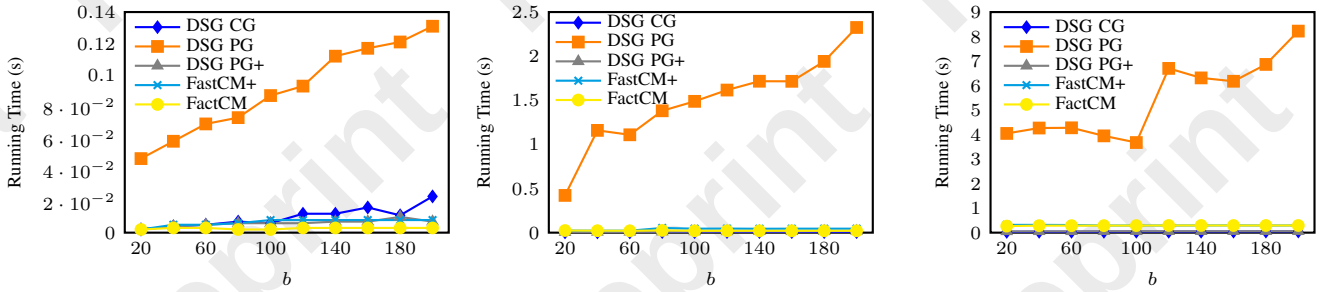
5.1 Effectiveness and Efficiency Evaluations

As shown in Table 1, our proposed algorithms, Dynamic Seed-GrowthCM CG (DSG CG) and Dynamic Seed-GrowthCM PG (DSG PG), consistently outperform all baseline methods across various datasets by achieving a greater



(a) Facebook (b) email-Enron (c) Stanford

Figure 4: The number of followers of different algorithms varied by b.



(a) Facebook (b) email-Enron (c) Stanford

Figure 5: The running time of different algorithms varied by b.

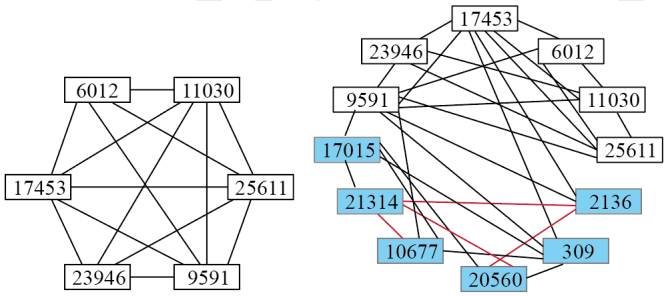
number of k-core followers. The key to this superior performance lies in our utilization of edges spanning across different λ-shell components. On the other hand, it is evident that the method employing complete growth, DSG CG, achieves shorter running times compared to all previously proposed methods, except on the Facebook dataset. While the partial growth method, DSG PG+, incurs slightly longer running times than DSG CG, it maintains the effectiveness of DSG PG and achieves a higher number of k-core followers.

5.2 Evaluation by Varying b

Figure 4 reports that as b increases, the number of k-core followers obtained by all methods increases. Among them, DSG PG achieves the highest number of k-core followers. Additionally, as b increases, the gap between DSG PG/DSG CG and FastCM+/FastCM becomes increasingly larger. This is because the number of λ-shell components requiring conversions increase with b, and the edges spanning different λ-shell components increase accordingly, highlighting the growing advantages of DSG PG/DSG CG. Figure 5 reports the running time of different algorithms as b increases. DSG PG+ maintains a comparable number of k-core followers to DSG PG while significantly reducing its running time, demonstrating the feasibility of offline storage.

5.3 Case Study on Collaboration Network

We applied core maximization to a collaboration network, which is a connected component comprising 14 nodes, extracted from the `ca-GrQc` dataset. The edges in the network represent co-authorship between authors, indicating that they have collaborated on a research paper.



(a)The original 4-core (b)The updated 4-core

Figure 6: Case Study on Collaboration Network. Here, k = 4 and b = 4.By inserting 4 red edges, 6 4-core followers are gained, as indicated by the blue nodes.

Our objective is to strengthen a specific k-core by inserting new edges, thereby increasing interactions among authors. We set $k = 4$ and $b = 4$. Initially, the size of the 4-core is 6. By applying our algorithm, the size of the 4-core increases to 12, as illustrated in Figure 6.

6 Conclusion

In this paper, we propose Dynamic Seed-GrowthCM, which employs complete/partial growth to address k-core maximization. The method accounts for dynamic changes in node degrees and effectively leverages edges across different λ-shell components, leading to improved performance. Experimental results show that our method enhances both efficiency and solution quality. This makes our method particularly suitable for large-scale, real-world networks.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 62276187 and No. 62422210), the National Key Research and Development Program of China (Grant No. 2023YFC3304503), and the Hong Kong RGC Project (Grant No. 12200424).

References

[Aral and Walker, 2012] Sinan Aral and Dylan Walker. Identifying influential and susceptible members of social networks. *Science*, 337(6092):337–341, 2012.

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

[Batagelj and Zaveršnik, 2011] V Batagelj and M Zaveršnik. Fast algorithms for determining core groups in social networks. *Advances in Data Analysis and Classification*, 5(2):129–145, 2011.

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

[Bonchi *et al.*, 2019] Francesco Bonchi, Arijit Khan, and Lorenzo Severini. Distance-generalized core decomposition. In *proceedings of the 2019 international conference on management of data*, pages 1006–1023, 2019.

[Bruckner *et al.*, 2015] Sharon Bruckner, Falk Hüffner, and Christian Komusiewicz. A graph modification approach for finding core-periphery structures in protein interaction networks. *Algorithms for Molecular Biology*, 10:1–13, 2015.

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

[Govind and Lal, 2021] N Govind and Rajendra Prasad Lal. Evaluating user influence in social networks using k-core. In *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2020, Volume 2*, pages 11–18. Springer, 2021.

[Guo *et al.*, 2022] Yuzhi Guo, Jiaxiang Wu, Hehuan Ma, and Junzhou Huang. Self-supervised pre-training for protein embeddings using tertiary structures. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 6801–6809, 2022.

[Guo *et al.*, 2024] Bin Guo, Emil Sekerinski, and Lingyang Chu. Federated k-core decomposition: A secure distributed approach. *arXiv preprint arXiv:2410.02544*, 2024.

[Hua *et al.*, 2019] Qiang-Sheng Hua, Yuliang Shi, Dongxiao Yu, Hai Jin, Jiguo Yu, Zhipen Cai, Xiuzhen Cheng, and Hanhua Chen. Faster parallel core maintenance algorithms in dynamic graphs. *IEEE Transactions on Parallel and Distributed Systems*, 31(6):1287–1300, 2019.

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

[Khaouid *et al.*, 2015] Wissam Khaouid, Marina Barsky, Venkatesh Srinivasan, and Alex Thomo. k-core decomposition of large networks on a single pc. *Proceedings of the VLDB Endowment*, 9(1):13–23, 2015.

[Laishram *et al.*, 2018] Ricky Laishram, Ahmet Erdem Sariyüce, Tina Eliassi-Rad, Ali Pinar, and Sucheta Soundarajan. Measuring and improving the core resilience of networks. In *Proceedings of the 2018 World Wide Web Conference*, pages 609–618, 2018.

[Laishram *et al.*, 2020] Ricky Laishram, Ahmet Erdem Sar, Tina Eliassi-Rad, Ali Pinar, and Sucheta Soundarajan. 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*, pages 325–333. SIAM, 2020.

[Li *et al.*, 2018a] Kan Li, Lin Zhang, and Heyan Huang. Social influence analysis: Models, methods, and evaluation. *Engineering*, 4(1):40–46, 2018.

[Li *et al.*, 2018b] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1852–1872, 2018.

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

[Liu *et al.*, 2021] Kaixin Liu, Sibo Wang, Yong Zhang, and Chunxiao Xing. An efficient algorithm for the anchored k-core budget minimization problem. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1356–1367. IEEE, 2021.

[Malliaros *et al.*, 2016] Fragkiskos D Malliaros, Apostolos N Papadopoulos, and Michalis Vazirgiannis. Core decomposition in graphs: Concepts, algorithms and applications. In *EDBT*, pages 720–721, 2016.

[Malliaros *et al.*, 2020] Fragkiskos D Malliaros, Christos Giatsidis, Apostolos N Papadopoulos, and Michalis Vazirgiannis. The core decomposition of networks: Theory, algorithms and applications. *The VLDB Journal*, 29(1):61–92, 2020.

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

[Saríyüce *et al.*, 2013] Ahmet Erdem Saríyüce, Buğra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V Çatalyürek. Streaming algorithms for k-core decomposition. *Proceedings of the VLDB Endowment*, 6(6):433–444, 2013.

[Sarıyüce *et al.*, 2016] Ahmet Erdem Sarıyüce, Buğra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V Çatalyürek. Incremental k-core decomposition: Algorithms and evaluation. *The VLDB Journal*, 25:425–447, 2016.

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

[Sun *et al.*, 2022] Xin Sun, Xin Huang, and Di Jin. Fast algorithms for core maximization on large graphs. *Proceedings of the VLDB Endowment*, 15(7):1350–1362, 2022.

[Weng *et al.*, 2021] Tongfeng Weng, Xu Zhou, Kenli Li, Peng Peng, and Keqin Li. Efficient distributed approaches to core maintenance on large dynamic graphs. *IEEE Transactions on Parallel and Distributed Systems*, 33(1):129–143, 2021.

[Yue *et al.*, 2020] Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, and Huan Sun. Graph embedding on biomedical networks: Methods, applications and evaluations. *Bioinformatics*, 36(4):1241–1251, 2020.

[Zhang *et al.*, 2017a] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. Finding critical users for social network engagement: The collapsed k-core problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[Zhang *et al.*, 2017b] Yikai Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. A fast order-based approach for core maintenance. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 337–348. IEEE, 2017.

[Zhou *et al.*, 2019] Zhongxin Zhou, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Chen Chen. k-core maximization: An edge addition approach. In *IJCAI*, pages 4867–4873, 2019.

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

[Zhou *et al.*, 2022] Zhongxin Zhou, Wenchao Zhang, Fan Zhang, Deming Chu, and Binghao Li. Vek: A vertex-oriented approach for edge k-core problem. *World Wide Web*, 25(2):723–740, 2022.

[Zhu *et al.*, 2018] Weijie Zhu, Chen Chen, Xiaoyang Wang, and Xuemin Lin. k-core minimization: An edge manipulation approach. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1667–1670, 2018.