Efficient Exact Minimum k-Core Search in Real-World Graphs

Qifan Zhang
Harbin Institute of Technology, Shenzhen
21s151127@stu.hit.edu.cn

Shengxin Liu*
Harbin Institute of Technology, Shenzhen
sxliu@hit.edu.cn

ABSTRACT

The k-core, which refers to the induced subgraph with a minimum degree of at least k, is widely used in cohesive subgraph discovery and has various applications. However, the k-core in real-world graphs tends to be extremely large, which hinders its effectiveness in practical applications. This challenge has motivated researchers to explore a variant of the k-core problem known as the *minimum k-core search problem*. This problem has been proven to be NP-Hard, and most of the existing studies naturally either deal with approximate solutions or suffer from inefficiency in practice. In this paper, we focus on designing efficient exact algorithms for the minimum k-core search problem. In particular, we develop an iterative-based framework that decomposes an instance of the minimum k-core search problem into a list of problem instances on another well-structured graph pattern. Based on this framework, we propose an iterative-based branch-and-bound algorithm, namely IBB, with additional pruning and reduction techniques. We show that, with a n-vertex graph, IBB runs in $c^n n^{O(1)}$ time for some $c < 2$, achieving better theoretical performance than the trivial bound of $2^n n^{O(1)}$. Finally, our experiments on real-world graphs demonstrate that IBB is up to three orders of magnitude faster than the state-of-the-art algorithms on real-world datasets.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**.

KEYWORDS

k-core; cohesive subgraph search; the branch-and-bound algorithm

ACM Reference Format:
Qifan Zhang and Shengxin Liu. 2023. Efficient Exact Minimum k-Core Search in Real-World Graphs. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3583780.3614861

1 INTRODUCTION

Graphs are widely used to model the relationships between entities in a large spectrum of real-world networks including social networks [45, 47, 53], biological networks [1, 2], transaction networks [25, 29], and so on. Finding cohesive subgraphs in a massive real-world graph is one of the central topics in computational social networks where a cohesive subgraph is commonly considered as a community in social networks [9, 10, 12, 18, 21–24, 26, 27, 30, 37, 49, 54]. To measure a cohesive subgraph, a popular cohesive model that has attracted much attention in the literature is the *k-core*, e.g., [4, 46, 52, 56]. Specifically, the k-core in a graph of n vertices is an induced subgraph where each involved vertex is connected to at least k other vertices in this subgraph. Besides the clear definition and powerful expressiveness in the real world, the k-core also has the advantage that it can be obtained by recursively removing the vertex with a degree less than k in linear time.

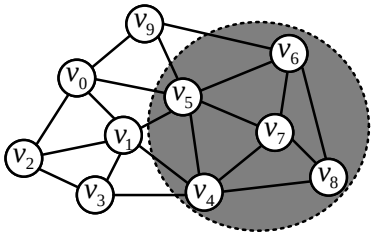


Figure 1: A Motivating Example with $k = 3$

A major obstacle that sometimes makes the k-core incapable of representing a cohesive structure in practice is that the k-core (for a relatively small k) in a real-world network is unreasonably large, as observed in [4, 31, 42, 46]. This phenomenon naturally introduces a stream of studies on a variant of the k-core problem in which the goal is to optimize the size of a "k-core". We call these studies as the *minimum k-core search problem*. More precisely, we want to find an induced subgraph with a minimum degree of at least k while *minimizing* the size of this subgraph. Such a size-constrained cohesive pattern can be considered the most condensed structure when we impose the minimum degree constraint on each vertex. Consider an example in Figure 1 with 10 vertices and $k = 3$. The whole graph is a 3-core and the induced subgraph by the vertex set $\{v_4, v_5, v_6, v_7, v_8\}$ is the minimum 3-core.

The minimum k-core search problem has various applications:
Similar Protein Identification. Studies conducted by [1, 2] have reported that proteins within the same k-core subgraph in protein-protein interaction (PPI) networks exhibit a tendency to possess similar functionalities. However, the empirical analysis of [31] reveals that such homophily property is more applicable to small k-core subgraphs. As a result, the minimum k-core subgraph may be a reasonable approach to identify a group of proteins with a high likelihood of sharing the same functionality.
Talent Recruitment. Suppose there is a programming master club, and the leader wants to assemble a team of at least three members from the club to assist in completing a project. Each member in the club has the same skill level, but due to limited budget, the team size needs to be minimized, and close collaboration

*Corresponding author

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614861

among team members is desired. The minimum k-core model can be utilized to provide valuable references for forming the project team with close-knit interactions.

Some real applications call for the query-based search problem:
Product Recommendation. Many popular online shopping platforms, e.g., Amazon and Taobao, leverage social networks ([53]) to recommend products to users. Specifically, given a social network graph G and a query node q (which represents a user), we can extract a dense subgraph from G that represents a closely connected community to user q. We can then recommend products on this community that have been previously purchased by its members to user q. Considering both development costs and user relevance, the minimum k-core is a suitable model for this application.

Due to the minimum k-core search problem can be reflected in real-world applications, it has attracted research attentions recently [31, 42]. Mikesell and Hicks [42] proposed a branch-and-cut algorithm based on Integer Linear Programming (ILP) for the minimum k-core search problem. In particular, their method utilizes an ILP formulation of the problem and includes some additional valid inequalities to ILP for speeding up. On the other hand, Li et al. [31] devised a progressive search algorithm, called PSA, for the minimum k-core search problem that contains the user-given query vertex. PSA can return an approximate solution where the approximation ratio is bounded by a user-given ratio c. Specifically, PSA performs a branch-and-bound search where a lower bound lb and an upper bound ub of the optimal solution are maintained during the search. Whenever $ub/lb \leq c$, PSA directly returns ub and its corresponding subgraph as output.

However, when we deal with real-world graphs, the minimum k-core search problem faces various challenges.
Challenges and our solutions. One major challenge comes from the hardness of the problem. Peleg, Sau, and Shalom [43] showed that for any integer $k \geq 3$ and any real number $\epsilon > 0$, there is no polynomial-time algorithm for the problem with an approximation ratio of $2^{O(\log^{1-\epsilon} n)}$ unless NP $\subseteq$ DTIME$(2^{O(\log^{1/\epsilon} n)})$.

Thus existing works either focus on the small graphs (around 1000 vertices) as in [42] or deal with approximate solutions as in [31]. Note that, although the above-mentioned PSA algorithm can be used to solve the minimum k-core search problem, it suffers from inefficiency when we set the approximation ratio $c = 1.0$ (which will be discussed later). These methods cannot handle real-world graphs (as demonstrated in our experiment) when we look for the exact solution to the problem. Moreover, most existing works focus on the query-based version of the minimum k-core search problem. However, for applications such as talent recruitment mentioned before, there is a need to consider the minimum k-core search problem (without query vertex).

In our paper, we tackle the minimum k-core search problem on real-world graphs by taking a deep analysis of the k-core pattern. Although we will show that the k-core pattern does not enjoy nice structural properties, we observe a crucial connection between the k-core and another graph pattern where useful properties can be explored. Based on this observation, we propose an iterative-based branch-and-bound algorithm that achieves better time performance in both theory and practice.
Contributions. Our main contributions are listed below:

- In this paper, in both theory and practice, we give a systematic study of the exact solutions for the minimum k-core search problem in general graphs.
- We formally analyze the structural properties of k-core. We then summarize the existing state-of-the-art algorithms of the minimum k-core search problem and propose an enumeration method that runs in $2^n n^{O(1)}$ time. The limitations of these algorithms are also discussed.
- We propose a novel iterative-based branch-and-bound algorithm, namely IBB. Our IBB has the following advantages: (1) IBB is the first exact algorithm that runs in $c^n n^{O(1)}$ time for some $c < 2$ which improves upon the trivial theoretical bound of $2^n n^{O(1)}$; (2) IBB could be easily adapted to solve query-based and top-r problems with high efficiency.
- We conduct extensive experimental studies on various real-world graphs to verify the effectiveness and efficiency of our devised method in terms of running time. The results show that IBB is up to three orders of magnitude faster than the state-of-the-art algorithms on real-world datasets

Roadmap. Section 2 defines the problem and introduces the preprocessing method. Section 3 summarizes state-of-the-art algorithms and presents a novel enumeration method. Section 4 introduces two iterative-based branch-and-bound algorithms. We conduct extensive experiments in Section 5. Section 6 reviews the related work and Section 7 provides the conclusion.

2 PRELIMINARIES

2.1 Notations and Definitions

Let $G = (V, E)$ be an undirected simple graph with $|V| = n$ vertices and $|E| = m$ edges. We denote $G[S]$ of G as the subgraph induced by the set of vertices S of V. We use $\delta_G(v)$ and $\delta_{G[S]}(v)$ to denote the degree of v in G and $G[S]$, respectively. We also denote the set of neighbors for vertex v in G and $G[S]$ by $N_G[v]$ and $N_{G[S]}(v)$. Additionally, we may omit the input graph G in notations if the context is clear, such as using $N(v)$ instead of $N_G(v)$.

In this paper, we are interested in the following cohesive pattern:

Definition 1. *Given any positive integer k or s, an induced subgraph $G[S]$ with the vertex set $S \subseteq V$ is said to be:*

- *a* k-core *if* $\delta_{G[S]}(v) \geq k, \forall v \in S$.
- *an* s-plex *if* $\delta_{G[S]}(v) \geq |S| - s, \forall v \in S$.

In a nutshell, a k-core and an s-plex consider a cohesive subgraph in terms of *connections* and *disconnections*, respectively. We can also notice that a k-core corresponds to a $(|S| - k)$-plex.

We are ready to define the problem we study in this paper:

Problem 1 (The minimum k-core search problem). *Given a graph G and a positive integer k, the minimum k-core search problem aims to find a k-core $G[S]$ with minimum cardinality.*

It is interesting to note that a minimum k-core must be a connected graph. This fact can be verified by seeing that if a minimum k-core is not connected, then we can always extract a connected component (that is clearly a k-core) with a smaller cardinality.
Example. Consider the graph in Figure 1. The whole graph is a 3-core, but the subgraph H induced by the vertex set $\{v_4, v_5, v_6, v_7, v_8\}$ is a minimum 3-core and it is also a 2-plex.

Problem extensions to query-based and top-r. The minimum k-core search problem can be easily extended to the query-based and top-r versions. In particular, the query-based minimum k-core search problem additionally requires a query vertex q and aims to find a minimum k-core containing q. Note that the query-based problem will not always return a solution. For the top-r minimum k-core search problem, the goal is to find r smallest k-cores. Our proposed algorithm can be adapted to these problem extensions which will be discussed later.

Structural challenge for the k-core. All the problems we studied in this paper focus on the pattern of k-core. However, the study on k-core faces challenge due to the following reason.

Definition 2. *An induced subgraph $G[S]$ with a property P is said to be* monotone *(or* anti-monotone*) if every $G[S']$ where $S' \supseteq S$ (or $S' \subseteq S$) also possesses property P.*

Clearly, the s-plex holds the anti-monotone property, i.e., any induced subgraph of an s-plex is also an s-plex. This property provides much more room for devising algorithms for the problem related to s-plex. On the other hand, we have:

Property 1. *The k-core is not monotone or anti-monotone.*

This property can be proved by the illustrative example in Figure 1. The subgraph induced by the vertex set $\{v_0, v_1, v_2, v_3, v_4, v_5\}$ is a 3-core, while the subgraph induced by vertex sets $\{v_0, v_1, v_2, v_3, v_4\}$ and $\{v_0, v_1, v_2, v_3, v_4, v_5, v_9\}$ are not. Property 1 implicitly shows the potential difficulty of the problem. After identifying a set of vertices that can be safely removed without destroying any k-core, it is still not an easy task to efficiently locate the minimum k-core.

Remark that in this paper, we use the term *feasible solution* to represent a subgraph that satisfies the condition of k-core but is not necessarily a solution with the smallest size. Further, we use the term *optimal solution* to represent the subgraph that satisfies the condition of k-core and achieves the smallest size. Besides, we focus on the case when $k \geq 3$. The reason is as follows. For $k = 1$, this problem can be solved trivially, as every edge is an optimal solution. For $k = 2$, the problem corresponds exactly to finding the shortest cycle in the graph (as every subgraph of minimum degree at least 2 contains a cycle) which can be solved in polynomial time.

2.2 Preprocessing Method

We next introduce the preprocessing method used in all the algorithms we studied. Basically, the goal of the preprocessing method is to identify a set of vertices such that these vertices will not be in the optimal solution. Removing these vertices results in a shrank graph which makes an algorithm easier to handle with.

A useful concept is called *coreness*, where a vertex v is said to have coreness of h, i.e., $coreness(v) = h$, if it belongs to a h-core but not to any $(h+1)$-core. Then we have the following property.

Property 2. *A vertex v cannot be in a k-core if $coreness(v) < k$.*

The preprocessing method using Property 2 can be efficiently computed via the core decomposition method in $O(m)$ time [5, 6].

Further, the preprocessing method not only shrinks the input graph but also provides a candidate solution to our problem:

Lemma 2.1. *The graph after preprocessing is either (1) empty (which implies there does not exist any feasible solution) or (2) a feasible solution (but not necessarily minimum).*

Proof. If the remaining graph is empty, by Property 2, we know that all the vertices that are not in the optimal solution are removed which means that there is no feasible solution. For Case (2), Property 2 also implies that each vertex in the remaining graph has coreness at least k. In other words, the remaining graph is a feasible k-core according to the definition. □

3 STATE-OF-THE-ART ALGORITHMS

In this section, we summarize the state-of-the-art algorithms for the minimum k-core search problem. In particular, Section 3.1 introduces an Integer Linear Programming (ILP) based method [42] while Section 3.2 describes the progressive search algorithm (PSA) proposed by [31]. In addition, we propose an enumeration method (ENUM) to solve the problem in Section 3.3. Finally, we discuss the limitations of the state-of-the-art algorithms.

Remark that these algorithms all involve the preprocessing method mentioned in Section 2.2 to remove unqualified vertices and obtain a reduced graph. In the following, we assume these introduced algorithms are based on the reduced graph.

3.1 Integer Linear Programming: ILP [42]

Mikesell and Hicks [42] studied the minimum k-core search problem by using the idea of integer linear programming (ILP). Their method first utilizes a random-vertex-deletion-based heuristic to calculate a feasible solution (where the size is an upper bound ub) to the problem. They then formulated the problem as an ILP:

$$\text{minimize} \sum_{v \in X_v} x_v \tag{1}$$

$$\text{subject to} \sum_{v \in X_v} x_v \geq k+1 \tag{2}$$

$$\sum_{u \in X_v} e_{uv} x_u \geq k x_v \qquad \forall\, v \in X_v, \tag{3}$$

$$x_v \in \{0,1\} \qquad \forall\, v \in X_v, \tag{4}$$

where the binary variable x_v represents whether or not vertex v is in the solution, and e_{uv} is a 0/1 constant such that $e_{uv} = 1$ if edge (u, v) is in E and $e_{uv} = 0$ otherwise. Based on this ILP formulation, they designed a branch-and-cut algorithm that includes some additional valid inequalities to the ILP related to (a) the upper bound ub, (b) the length of the smallest cycle (i.e. girth), and (c) the minimum vertex cover.

3.2 Progressive Search Algorithm: PSA [31]

Li et al. [31] considered the query-based minimum k-core search problem and focused on designing algorithm that returns an approximate solution where the approximation ratio is bounded by an input ratio c. To achieve this, they refine the upper bound s^+ and lower bound s^- of partial solutions iteratively in a search process. The search procedure involves the construction of a search tree where the query vertex q serves as the root, while each node represents a single vertex. The partial solution V_t associated with a node t consists of the vertex from that node and all the vertices from its ancestor nodes. When $s^+/s^- \leq c$, they terminate the algorithm and return the solution set corresponding to the current s^+ as the minimum k-core containing q.

During the search within the PSA algorithm, when a node t containing the vertex u is visited, the search tree expands by adding child nodes to t. Each child node represents a unique neighbor of a vertex in V_t with a vertex ID greater than u. The search operation at node t follows the following two procedures:

Lower bound computation. A size lower bound $s^-(t)$ is computed for the minimum k-core containing the partial solution $V_{t'}$ of each child node t' under node t. This lower bound serves as an approximation of the minimum size. In order to identify the next node to explore, the leaf node with the smallest lower bound $s^-()$ in the ongoing search tree is chosen, indicating the most promising candidate.

Upper bound computation. A Depth-First Search (DFS) is conducted on the partial solution $V_{t'}$ of every child node t' under node t. This DFS employs heuristics to compute a minimal k-core that includes $V_{t'}$. Through this process, the global size upper bound s^+ for the optimally-minimum k-core is updated.

Remarks. PSA can also be applied to the minimum k-core search problem without any query vertex. Specifically speaking, we can choose each vertex in the graph once as the query vertex and set the termination ratio $c = 1.0$. We record the minimum size of the above solution and output the corresponding graph.

3.3 Enumeration Method: ENUM

We develop an enumeration method ENUM that utilizes the common branch-and-bound strategy. In particular, ENUM enumerates the vertex sets with pruning strategies. We initialize the current chosen set as $\emptyset$ and the candidate set as all vertices in the graph. We recursively branch on the current vertex set into a list of different vertex sets by adding vertex from the candidate set to search for the best solution and apply pruning methods to reduce unnecessary enumeration if the current branch cannot produce a better solution than the current best one.

Due to the lack of monotonicity and anti-monotonicity of k-core (see Property 1), the pruning technique is limited where we can only utilize the information of the current best solution. Let $curbest$ and S be the current best solution and the current chosen set of vertices, respectively.

Pruning Rule 1. For any vertex set $S' \supset S$, $G[S']$ is not a minimum k-core if $|S| \geq curbest$.

Pruning Rule 2. For any vertex set $S' \supseteq S$, $G[S']$ is not a minimum k-core if $\delta_{G[S]}(u) < k + |S| - curbest$ for any vertex $u \in S$.

Extension to the query-based search problem. Our enumeration method can be extended to the problem with the query vertex q by adding q to the chosen set before the recursive branching part. The algorithm will return the minimum k-core containing q if such a k-core exists.

3.4 Discussions on State-of-the-art Algorithms

The state-of-the-art algorithms are inefficient due to the following limitations.

One major limitation is the preprocessing method (described in Section 2.2) used for the maximum k-core is not sufficiently effective. As the minimum k-core search problem is NP-hard, the (worst case) time complexity of exact algorithms is exponential to the number of candidate vertices used to conduct searches. Thus the size of the candidate vertex set will hugely affect the time performance in practice. Table 1 shows the reduction effectiveness comparison of state-of-the-art algorithms (that used the preprocessing method in Section 2.2) and our newly proposed algorithm (IBB in Section 4.2). From the table, we can see that even for relatively small graphs with 10,000 to 40,000 vertices, the remaining vertices before searching are still large, e.g., 2,000 to 12,000 vertices. On the other hand, our newly proposed method IBB tries to work with a small set of vertices in order to achieve high efficiency in practice by further applying reduction methods for other well-structured cohesive subgraph model.

Table 1: Reduction method comparison ($k = 18$)

	Remaining vertices before searching	
Dataset	State-of-the-art algorithms	IBB (Section 4.2)
Harvard ($n = 15,126$)	11,286	4,668
Email ($n = 36,692$)	2,078	266
Wiki ($n = 2,394,385$)	24,622	2,452

ILP. The efficiency of ILP highly depends on whether their heuristic can obtain a good upper bound. If not, the computations of girth (in $O(n^3)$ time) and ILP require a huge amount of time. On the other hand, the ILP with a large number of variables will consume massive memory, which will make ILP infeasible.

PSA. PSA can achieve high efficiency when the approximation ratio c is relatively large. As our problem should return an exact solution, we need to set $c = 1.0$. However, due to the stopping criterion of the PSA being $s^+/s^- \leq c$, in most cases, s^- is smaller than s^+. Therefore, even if the value of s^+ is equal to the size of the exact solution, PSA cannot stop because we still have $s^+/s^- > c$. This can result in a significant waste of time. On the other hand, since we are unaware of the exact ratio between s^+ and s^- when obtaining the exact solution, we cannot pre-estimate an appropriate value for c. Besides, as PSA is designed for the query-based problem, when we face the problem without the query vertex, PSA needs to iteratively choose each vertex in the graph as the query vertex once. This is certainly infeasible in practice.

ENUM. ENUM randomly selects one vertex v and generate two branches, one includes v in the solution set and another deletes v. Suppose there are n vertices, the search space will reach $O(2^n)$, which is a huge search space. Moreover, the power of pruning methods is limited due to the structural properties of k-core.

4 OUR PROPOSED ITERATIVE FRAMEWORK-BASED ALGORITHM

In this section we show our proposed algorithms for the minimum k-core search problem in a given graph. We resolve the limitations of state-of-art algorithms discussed in Section 3.4 by transforming our problem into the problem based on s-plex. Specifically, we first present our basic iterative framework in Section 4.1, followed by an optimized iterative framework which has improved time efficiency in Section 4.2.

4.1 Basic Iterative Framework

Before demonstrating our basic iterative framework, we introduce a simple yet important property:

Property 3. *Given an s-plex $G[S]$, if the number of vertices $|S|$ is at least $s + k$, then $G[S]$ is also a k-core.*

Algorithm 1: Basic Iterative Framework

Input: Graph G and the integer k
Output: the minimum k-core
1 $G' \leftarrow$ Preprocess(G, k);
2 **for** $s = 1$ *to* $n' - k$ **do**
3 $\quad y_s^* \leftarrow$ the optimal value of Bounded-Plex(s);
4 $G_{best} \leftarrow$ the graph corresponding to $\min_{1 \le s \le n'-k} y_s^*$;
5 **return** G_{best}

Roughly speaking, the k-core uses the minimum number of neighbors as a measurement while the s-plex utilizes the maximum number of non-neighbors. Then, Property 3 implies that, given a k-core $G[S]$, we can also view $G[S]$ as a corresponding s-plex where s is the maximum number of non-neighbors in $G[S]$. Thus, the minimum k-core can be seen as the minimum s-plex with size at least $s + k$ for a particular s. Our iterative framework is directly based on the above important connection between the minimum k-core search problem and a variant of the maximum s-plex search problem as defined in the following.

Problem 2. *(*Bounded-Plex(s)*) Given a graph G and positive integers s and k, the* Bounded s-Plex Problem (Bounded-Plex(s)) *aims to find one (out of possibly many ones) s-plex $G[S]$ with minimum cardinality such that $|S| \ge s + k$.*

We note that the solution to the minimum s-plex search problem, i.e., the one without the lower bound of the returned size, is trivially the empty set of size 0. We also note that Bounded-Plex(s) may not have a feasible solution, i.e., there is no induced subgraph that satisfies the conditions in the problem statement.

For ease of presentation, we denote by y_s^* the value of the optimal solution to Bounded-Plex(s) for a fixed s, and set $y_s^* \leftarrow \infty$ if Bounded-Plex(s) has no solutions. Let Y_s be the set of selected vertices in the optimal solution for Bounded-Plex(s), and we also have $Y_s = \emptyset$ if $y_s^* \leftarrow \infty$.

Algorithm. Based on Property 3 and Bounded-Plex(s), our proposed iterative framework is shown in Algorithm 1. After the preprocessing step in line 1 (where the details are discussed in Section 2.2), the number of vertices in the graph is reduced, and our algorithm will then work on this shrank graph, denoted by $G' = (V', E')$ with $|V'| = n'$. We assume G' is not empty; otherwise, there is no solution by Lemma 2.1. The algorithm goes in iterations (lines 2-3) where each iteration solves an instance of Bounded-Plex(s) for a specific s (line 3). Finally, the algorithm outputs G_{best} as the final solution in lines 4-5. Note that $\min_{1 \le s \le n'-k} y_s^* \le n' \ne \infty$ since the reduced graph G' is a feasible solution to the minimum k-core search problem (by Lemma 2.1 and the assumption that G' is not empty) which implies that $y_{n'-k}^* = n'$.

Analysis. Let y^* and Y^* be the optimal value and the corresponding selected vertices for the minimum k-core search problem, respectively. The following lemma crucially connects the minimum k-core search problem and Bounded-Plex(s), which also shows the correctness of our iterative framework (Algorithm 1):

Lemma 4.1. $y^* = \min_{1 \le s \le n'-k} y_s^*$.

Proof. First, according to Property 3, it is easy to see that, if $Y_s \ne \emptyset$, each reported s-plex Y_s of Bounded-Plex(s) is a feasible solution to minimum k-core search problem. Thus we have

$$y^* \le \min_{1 \le s \le n'-k} y_s^*.$$

On the other hand, let $s' = y^* - k$. As we must have $y^* > k$ (by the definition of k-core), we get $s' \ge 1$. Then Y^* is a feasible solution of bounded-plex(s') by definition. Then we know $y^* \ge y_{s'}^* \ge \min_{1 \le s \le n'-k} y_s^*$, where the first inequality is due to the fact that $y_{s'}^*$ is the optimal value of Bounded-Plex(s'). □

In summary, we have the following result:

Theorem 4.2. *Our basic iterative framework (Algorithm 1) can correctly solve the minimum k-core search problem by calling an algorithm to solve* Bounded-Plex(s) *exactly $n' - k$ times.*

Theorem 4.2 shows that, with the help of Algorithm 1, one can transform the problem of solving the minimum k-core search problem into the problem of solving Bounded-Plex(s). Note that Bounded-Plex(s) can be solved by a naïve method. We will describe our detailed method after showing an improved framework (compared with Algorithm 1) in the next subsection.

4.2 Optimized Iterative Framework

In this part, we give an optimized iterative framework which reduces the number of iterations in lines 2-3 of Algorithm 1 and thus improves the efficiency in practice. This improvement is due to a deep analysis on the value of y_s^*:

Lemma 4.3. *For all $1 \le s \le n' - k$, if $y_s^* \ne \infty$, we have $y_s^* = s + k$.*

Proof. Consider a fixed s. We prove the lemma by contradiction. First notice that, we have $y_s^* \ge s+k$ if $y_s^* \ne \infty$. We then suppose, to the contrary, that $y_s^* \ne \infty$ and $y_s^* > s+k$. Recall that Y_s corresponds to the set of selected vertices in y_s^*.

We construct Y_s' by arbitrarily removing $y_s^* - (s+k)$ vertices in Y_s. We will show that Y_s' is a feasible solution with size strictly smaller than y_s^* which contradicts the assumption that y_s^* (with Y_s) is optimal. First, it is easy to see that $|Y_s'| = |Y_s| - \left(y_s^* - (s+k)\right) = s+k$. Then, by the anti-monotone property (see Definition 2.1) of s-plex, we know that Y_s' is also an s-plex, i.e., the maximum number of non-neighbors for any vertex in Y_s' is at most s, which fulfills the requirement of s-plex. □

From Lemma 4.3, we obtain the following observation:

Corollary 4.4. *Let $s^* = \arg\min_{1 \le s \le n'-k} y_s^*$. Then we have: (1) $y_s^* = s^* + k$; (2) $y_s^* = \infty$, for all $1 \le s < s^*$.*

In other words, the iteration that solves Bounded-Plex(s^*) with s^* is not only the iteration that yields the optimal solution but also the *first* iteration in Algorithm 1 that has a feasible solution from $s = 1$ to s^* while all the iterations with $s < s^*$ would be infeasible. Based on this idea, we show our optimized iterative framework in Algorithm 2. By combing Lemma 4.1 and Corollary 4.4, we can also conclude that:

Theorem 4.5. *Our optimized iterative framework (Algorithm 2) can correctly solve the minimum k-core search problem by calling an algorithm to solve* Bounded-Plex(s) *at most $n' - k$ times.*

It is obvious that our optimized iterative framework (Algorithm 2) implements fewer iterations of Bounded-Plex(s) than the

Algorithm 2: Optimized Iterative Framework with Branch-and-Bound: IBB

Input: Graph G and the integer k
Output: the minimum k-core
1 $G' \leftarrow$ Preprocess(G, k);
2 $s \leftarrow 0$;
3 **repeat**
4 $s \leftarrow s + 1$;
5 **until** Bounded-Plex(s) (solved by the branch-and-bound method) has a feasible (and also optimum) solution with size exactly $s + k$, i.e., $y_s^* == s + k$;
6 $G_{best} \leftarrow$ the graph corresponding to y_s^*;
7 **return** G_{best}

basic one (Algorithm 1). This property is ensured by Corollary 4.4 where the equality holds (i.e., they have the same number of calls) when $s^* = n' - k$. In practice, the value of s^* is normally small which makes the optimized framework works more efficiently. Moreover, we only need to check whether Bounded-Plex(s) has a feasible solution of size $s + k$ in line 5 of Algorithm 2 instead of solving Bounded-Plex(s) completely as in Algorithm 1 (where the feasibility checking of Bounded-Plex(s) will be discussed later).

Solving bounded-plex(s). Based on the observation from Corollary 4.4, solving Bounded-Plex(s) can be reduced to checking whether there exists a feasible (and the optimal solution) with size exactly $s+k$ (which is also shown in line 5 of Algorithm 2). We next show the method for solving/checking Bounded-Plex(s) which also establishes a critical connection that:

Theorem 4.6. *The minimum k-core search problem can be solved by using the maximum s-plex search problem at most $n' - k$ times.*

Proof. We will show that Bounded-Plex(s) can be computed via any exact algorithm for the maximum s-plex search problem. Then, Theorem 4.6 follows directly from Theorem 4.5.

Given the optimal value, say z, of the maximum s-plex search problem, we compare z with $s + k$. If $z < s + k$, we know there is no feasible solution to Bounded-Plex(s). On the other hand, i.e., $z \geq s + k$, it is easy to see that we can construct a feasible solution of size $s + k$ by arbitrarily removing $z - (s + k)$ vertices in the optimal solution of maximum s-plex search problem since the s-plex has the anti-monotone property (which is similar to the proof of Lemma 4.3). In this latter case, the optimal size of Bounded-Plex(s) is $s + k$. □

Theorem 4.7 ([49]). *The maximum s-plex search problem can be solved in $c^n n^{O(1)}$ time where c is the largest root of the function $x^{s+2} - 2x^{s+1} + 1 = 0$, and $c < 2$ for any $s \geq 1$.*

We next concentrate ourself on the maximum s-plex search problem. Our checking method of Bounded-Plex(s) makes use of the clever branch-and-bound algorithm for the maximum s-plex search problem in [49] coupled with the second-order reduction technique proposed by [55]. This combination not only enjoys the advanced theoretical guarantee (Theorem 4.7) but also achieves high efficiency on real-world graphs. In this checking algorithm, we can get a feasible solution of $s+k$ as long as the current feasible solution is of size at least $s + k$, i.e., we interrupt the checking algorithm immediately once we obtain an s-plex with size at least $s + k$. We also set the lower bound of the optimal solution of Bounded-Plex(s) to be $s+k$, which provides tighter bound and more powerful pruning ability. Finally, combined with Theorems 4.5, 4.6, and 4.7, we then conclude that:

Corollary 4.8. *The minimum k-core search problem can be solved in $c^n n^{O(1)}$ time where c is the largest root of the function $x^{s^*+2} - 2x^{s^*+1} + 1 = 0$, and $c < 2$ for any $s^* \geq 1$.*

Extension to the query-based and top-r search problems. Our iterative framework can be easily applied to query-based and top-r search problems by slightly modifying Algorithm 2. For the query-based search problem, we have the query vertex q as the input. We first check whether q is still in the remaining graph after line 1. If not, we can directly terminate the algorithm because q will not be in any k-core. Otherwise, we continue the algorithm where we enforce q in the solution set when running algorithms for Bounded-Plex(s). For the top-r search problem, when we obtain y_s^* in line 6, we can extract $r^* = \binom{y_s^*}{s+k}$ solutions (remove repetitive ones). We continue our algorithm to obtain more solutions until $r^* \geq r$ to get top-r solutions. Note that, if r is set to be a large value, we may not find r (feasible) solutions given a particular problem instance. In this case, we just return all the feasible solutions that we can find.

5 EXPERIMENTAL STUDIES

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

Algorithms. We compare the following algorithms.

- ILP: The minimum k-core search algorithm based on ILP proposed in [42].
- (adapted) PSA-c: The query-based minimum k-core search algorithm proposed in [31] with approximation ratio c, e.g, PSA-1.1, PSA-1.2. For the minimum k-core search problem without the query vertex, we adapt the algorithm in [31] by enumerating all possible vertices as the query vertex.
- ENUM: our proposed enumeration algorithm which is mentioned in Section 3.3.
- IBB: our proposed optimal iterative-based branch-and-bound algorithm mentioned in Section 4.2.

Datasets. We use 10 real-life datasets in our experiments. Datasets FSU53 and Harvard are downloaded from *Network Repository*[1]. Dataset Skitter is downloaded from *Konect*[2]. Other datasets are downloaded from *SNAP*[3]. Table 2 shows the characterization details of each dataset, where n is the number of vertices, m is the number of edges, d_{avg} is the average degree of the graph and k_{max} is the largest coreness value of the graph. Besides, we set the default value of k as 18 unless otherwise specified.

We vary k in different experiments to compare our algorithms. Besides, we also conduct experiments based on the extension of our algorithm IBB: 1) finding top-r minimum k-cores where $r > 1$; 2) finding the query-based minimum k-core with randomly generated query vertex.

[1] https://networkrepository.com
[2] http://konect.cc/networks
[3] http://snap.stanford.edu

Table 2: Statistic of datasets

Dataset	n	m	d_{avg}	k_{max}
Harvard	15,126	824,617	109	103
UCLA	20,453	747,604	73	66
FSU53	27,737	1,034,802	74.62	81
Email	36,692	126,007	10.02	43
Epinions	75,879	405,740	10.6	67
Gowalla	196,591	950,327	9.67	52
Google	875,713	4,322,051	9.87	44
Youtube	1,134,890	2,987,624	5.26	51
Skitter	1,696,415	11,095,298	13.08	112
Wiki	2,394,385	4,659,565	1.9	131

All algorithms are implemented in C++ with an Intel(R) Xeon(R) Gold 6320R CPU @2.10 GHZ with 128GB memory and an Ubuntu system. For each experiment, we perform 5 repeated runs and show the averages. We set the time limit as 10,000 seconds and terminate the program when it runs out of memory, where mark the running time as inf.

5.1 Evaluation of Effectiveness

Cohesive model comparison. In this experiment, we compare the graph density and clustering coefficient of three models: the k-core [44], the k-truss [15], and the minimum k-core [43]. As depicted in Figures 2 and 3, the minimum k-core exhibits higher density and clustering coefficient compared to the other cohesive models. Note that the parameter k is set at a default value of 18.

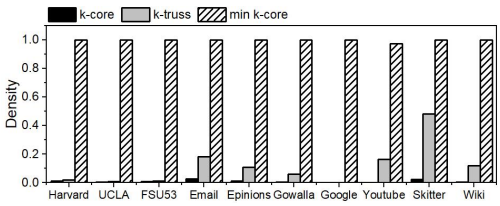


Figure 2: Graph density

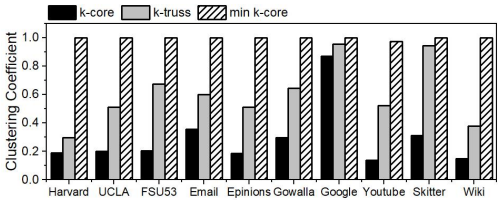
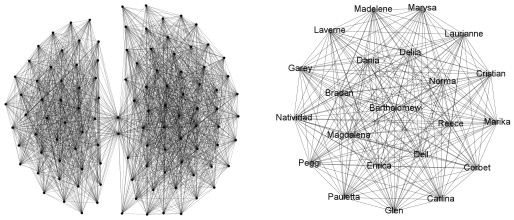


Figure 3: Clustering coefficient

Case study. We construct an email-sending graph based on *Email*. In this graph, each vertex corresponds to an email user and two vertices are connected by an edge if they communicate by email at least 3 times. In Figure 4 (a), we extract the 20-core from *Email*. The 20-core has 108 vertices and 2,110 edges. Besides, the density and the clustering coefficient of the 20-core are 0.365 and 0.77, respectively. In Figure 4(b), we extract the minimum 20-core from *Email*. The minimum 20-core has 22 vertices and 229 edges. Besides, the density and the clustering coefficient of the minimum 20-core are 0.991 and 0.991. From the case study, it is clear to see that the minimum 20-core is useful for finding a small group of users that frequently send emails to each other.



(a) 20-core (b) Minimum 20-core

Figure 4: Case study

5.2 Evaluation of Efficiency

Evaluation against the state-of-the-art algorithms. In this experiment, we show the running times of various exact algorithms, namely ILP, PSA-1.0, ENUM, and IBB, on all datasets. The experimental results are displayed in Figure 5. The value of k is fixed at 18 for this study. We can see that the ENUM algorithm cannot be finished within the time limit (10,000s) on every dataset due to the large candidate set and its inadequate branching strategy. The running times of adapted PSA-1.0 exceed the time constraint since it needs to enumerate all possible vertices as the query vertex. For ILP, obtaining a solution within a reasonable time highly depends on its heuristic's ability to find a good upper bound of the optimum solution; however, in most cases, the heuristic falls short of achieving the optimum solution, and ILP costs significant time and memory with a relative large graph. On the other hand, we observe that our proposed algorithm IBB consistently produces optimum result in an efficient manner on all the datasets. In particular, IBB normally outperforms the state-of-the-art algorithms by at least two orders of magnitude.

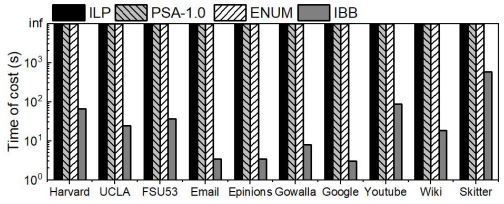
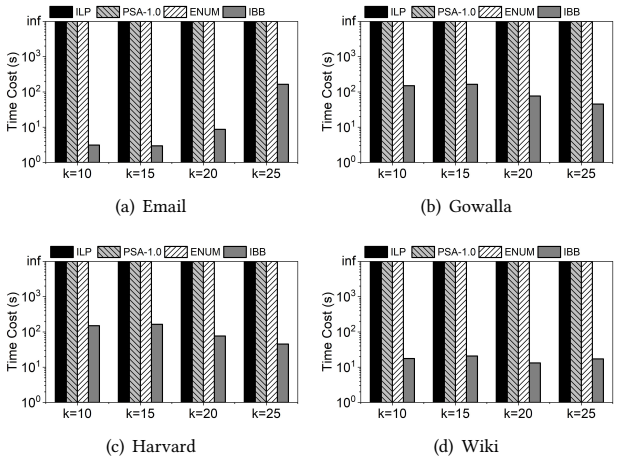


Figure 5: Running time of all datasets

Evaluation of varying k. In this experiment, we show running times of four algorithms, i.e., ILP, PSA-1.0, ENUM, and IBB on graphs *Email*, *Gowalla*, *Harvard*, and *Wiki*, with k varying from 10 to 25. From the results in Figure 6, we can see that 1) ILP, PSA-1.0, and ENUM cannot get results within the specified time limit due to the similar reasons discussed in the last experiments; 2) IBB can

solve the minimum k-core search problem efficiently with different values of k. However, it is difficult to predict the change in running time of IBB when varying k. On one hand, as k increases, the preprocessing stage removes more vertices, resulting in a smaller candidate set. On the other hand, an increase in k also leads to a larger solution set, requiring more rounds of iteration, which can potentially increase the running time of our algorithm.



(a) Email (b) Gowalla

(c) Harvard (d) Wiki

Figure 6: Running time when varying k

Scalability evaluation. We conduct the scalability evaluation on Skitter (with 1 million vertices and 10 million edges) where we perform a random sampling of vertices from 0.2 to 1.0. For each sampled set of vertices, we obtain the corresponding induced subgraph. Figure 7 shows the running time of IBB with different ratios of sampled vertices in the original graph. The result indicates that the increase in running time of IBB is not substantial in relation to the growth of the graph size.

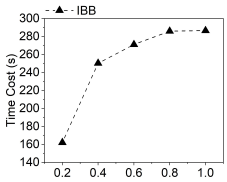
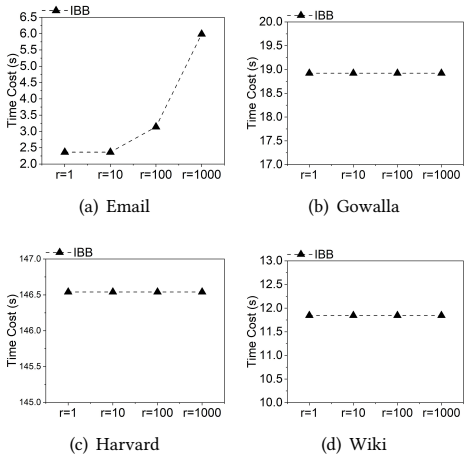


Figure 7: Scalability evaluation

Evaluation on finding top-r minimum k-cores. We also conduct experiments on finding top-r minimum k-cores on different real-world graphs. As mentioned in Section 4.2, our IBB algorithm can be easily adapted to handle the problem of top-r minimum k-cores. In this experiment, we set $k = 10$ and test $r = \{1, 10, 100, 1000\}$. From Figure 8, we can see that, when r increases, the time variation is not significant. This is because, in our iterative-based framework, whenever we obtain a minimum k-core, a large number of minimum k-cores could be found in the current problem instance of bounded s-plex according to Section 4.2. Moreover, as shown in Figure 8 (a), our algorithm needs more time to search more bounded s-plexes in order to generate the required number of minimum k-cores.



(a) Email (b) Gowalla

(c) Harvard (d) Wiki

Figure 8: Running time when varying r

5.3 Evaluations for Query-based Problem

We conduct various experiments for the query-based minimum k-core search problem in which the query vertex is randomly generated. Each experiment generate 5 query vertices and the results use the average. Notice that we use the same set of random query vertices when comparing different algorithms.

Result size and running time comparisons. We first compare the returned result size of different cohesive models (i.e., k-core and k-truss) using the default value of k. Figure 9 demonstrates that the minimum k-core model has the minimum result size. For the running time comparison, we set $k = 18$ for all datasets and the result is shown in Figure 10. We can see that our IBB algorithm can be applied to the query-based problem and runs faster than exact algorithms (e.g., PSA-1.0) for the query-based problem. In particular, PSA-1.0 is not practical for real-world graphs since PSA makes a trade-off between efficiency and result size.

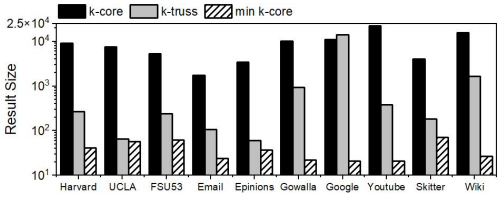


Figure 9: Result size of all datasets (Query-based)

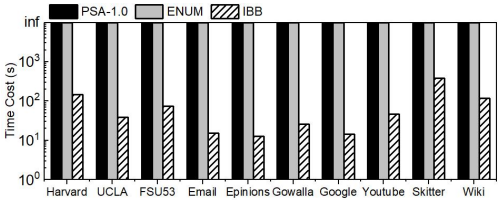
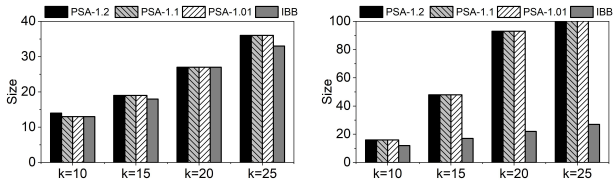


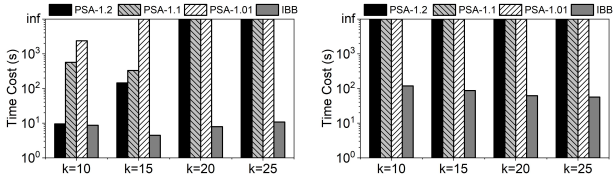
Figure 10: Running time of all datasets (Query-based)

Comparison with PSA. In this part, we instead compare our (query-based) IBB with PSA-1.01, PSA-1.1, and PSA-1.2 for both result size and running time. We use parameters 1.01, 1.1, and 1.2 here since PSA can return a solution close to the optimum solution. We conduct our experiments on two datasets, i.e., *Email* and *FSU53* and vary k from 10 to 25. Note that if PSA cannot obtain the solution within the time limit (10,000s), we use the current best result returned by PSA. Figure 11 shows the result size comparison. We can learn that in some cases, PSA may produce sufficiently good results while IBB can always return the optimum solution. For the time comparison, we can see from Figure 12 that IBB runs much faster than PSA (up to 2,000 times faster) on these two datasets.



(a) Email (b) FSU53

Figure 11: Result size comparison with PSA



(a) Email (b) FSU53

Figure 12: Running time comparison with PSA.

6 RELATED WORKS

In recent years, significant progress has been made in the study of cohesive subgraphs search [19, 20, 30, 41].

Size-bounded cohesive subgraph search. In addition to the studies mentioned in Section 1, there are some other works consider the size-bounded constraint in cohesive subgraph search. Sozio et al. [46] studied the community search problem with a size upper bound and proposed a global heuristic algorithm. Then, Cui et.al [16] proposed the minimum community search problem, i.e., find a minimum subgraph with the largest minimum degree containing a set of query vertices, and showed the NP-hardness of the problem. On the positive side, Barbieri et.al [4] proposed a greedy algorithm for the minimum community search problem. Ma et al. [40] proposed the query-based search algorithm to calculate a k-core with size h and identify the smallest closeness among all k-core subgraphs of size h. Yao and Chang [50] developed exact algorithms for the problem of searching for a connected subgraph containing the query vertex q. The objective is to identify a subgraph with the maximum minimum degree while adhering to the constraints of having at least l and at most h vertices. Liu et al. [35] proposed an algorithm to extract a k-truss subgraph containing the query vertex q with the size of vertex set no more than a user-given upper bound ub. Very recently, Zhang et.al [51] proposed an algorithm to extract a subgraph satisfying constraints which are nearly identical to the one in [50], except for the objective is to maximize the minimum support.

k-core. The k-core model was first proposed in [44], and since then, there have been extensive studies about k-core including the problem of core-decomposition [5, 7, 8, 13, 34] where the objective is to compute the core number for each vertex in the graph. Another related problem is called the collapsed k-core problem in social network engagement [38, 52]. In collapsed k-core, given a graph and integers b and k, we are asked to remove b vertices such that the size of k-core of the remaining graph is minimized. The minimum k-core search problem can be solved exactly by trying all possible values of b if we have an exact algorithm to the collapsed k-core problem at hand. However, prior arts on this topic do not provide any exact method.

s-plex. The maximum s-plex search problem aims to find the largest s-plex [44]. Since the problem has been proven to be NP-complete with any fixed positive integer s [3]. Recent research works focus on developing efficient exact algorithms in the real-world graphs. Xiao et al. [49] proposed a novel branch-and-bound algorithm that achieve better theoretical complexity compared with the classical algorithms. Gao et al. [22] introduced a new branch-and-bound algorithm with powerful graph reduction methods. Then, Jiang et al. [28] proposed novel upper bounds for efficient pruning while Zhou et al. [55] designed efficient pre-processing methods to prune unpromising vertices and edges. Very recently, Chang et.al [11] provided a novel branch-and-bound framework with newly designed reduction/pruning techniques that obtains advanced performance over large sparse graphs.

In addition to the undirected graph, cohesive subgraphs are also expended to other kinds of graphs, e.g., directed graph [14], bipartite graph [17, 33, 39, 48], temporal graph [32, 36], and so on.

7 CONCLUSION

In this paper, we proposed an iterative-based branch-and-bound algorithm IBB for solving the exact problem of minimum k-core search on real-world graphs. The effectiveness and efficiency of IBB are due to an iterative-based framework that decomposes an instance of the minimum k-core search problem into a series of problem instances on the maximum s-plex search problem. We showed that IBB achieves better theoretical time complexity than previous algorithms. Extensive experiments conducted on real-world graphs also demonstrated that IBB runs up to three orders of magnitude faster than state-of-the-art algorithms. Possible future works include developing parallel algorithms for the problem and exploring other minimum cohesive subgraph search problems.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (Grant No. 62102117), by the Shenzhen Science and Technology Program (Grant No. RCBS20210609103900003), and by the Guangdong Basic and Applied Basic Research Foundation (Grant No. 2023A1515011188).

REFERENCES

[1] Md Altaf-Ul-Amine, Kensaku Nishikata, Toshihiro Korna, Teppei Miyasato, Yoko Shinbo, Md Arifuzzaman, Chieko Wada, Maki Maeda, Taku Oshima, Hirotada Mori, et al. 2003. Prediction of protein functions based on k-cores of protein-protein interaction networks and amino acid sequences. *Genome Informatics* 14 (2003), 498–499.

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

[3] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. 2011. Clique Relaxations in Social Network Analysis: The Maximum k-Plex Problem. *Operations Research* 59, 1 (2011), 133–142.

[4] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. *Data Mining and Knowledge Discovery* 29, 5 (2015), 1406–1433.

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

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

[7] Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. 2014. Core decomposition of uncertain graphs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1316–1325.

[8] Francesco Bonchi, Arijit Khan, and Lorenzo Severini. 2019. Distance-generalized core decomposition. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 1006–1023.

[9] Shaowei Cai and Jinkun Lin. 2016. Fast Solving Maximum Weight Clique Problem in Massive Graphs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 568–574.

[10] Lijun Chang and Lu Qin. 2019. Cohesive subgraph computation over large sparse graphs. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2068–2071.

[11] Lijun Chang, Mouyi Xu, and Darren Strash. 2022. Efficient maximum k-plex computation over large sparse graphs. *Proceedings of the VLDB Endowment* 16, 2 (2022), 127–139.

[12] Peilin Chen, Hai Wan, Shaowei Cai, Weilin Luo, and Jia Li. 2019. Combining reinforcement learning and configuration checking for maximum k-plex problem. *CoRR* abs/1906.02578 (2019).

[13] James Cheng, Yiping Ke, Shumo Chu, and M Tamer Özsu. 2011. Efficient core decomposition in massive networks. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 51–62.

[14] Rajesh Chitnis, Fedor V Fomin, and Petr A Golovach. 2016. Parameterized complexity of the anchored k-core problem for directed graphs. *Information and Computation* 247 (2016), 11–22.

[15] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report* 16, 3.1 (2008).

[16] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 991–1002.

[17] Danhao Ding, Hui Li, Zhipeng Huang, and Nikos Mamoulis. 2017. Efficient fault-tolerant group recommendation using alpha-beta-core. In *Proceedings of the ACM on Conference on Information and Knowledge Management (CIKM)*. 2047–2050.

[18] Yi Fan, Nan Li, Chengqian Li, Zongjie Ma, Longin Jan Latecki, and Kaile Su. 2017. Restart and random walk in local search for maximum vertex weight cliques with evaluations in clustering aggregation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 622–630.

[19] 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 (2020), 353–392.

[20] Yixiang Fang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2022. *Cohesive subgraph search over large heterogeneous information networks.* Springer.

[21] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 6 (2020), 854–867.

[22] Jian Gao, Jiejiang Chen, Minghao Yin, Rong Chen, and Yiyuan Wang. 2018. An exact algorithm for maximum k-plexes in massive graphs.. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1449–1455.

[23] Bishwamittra Ghosh, Mohammed Eunus Ali, Farhana M Choudhury, Sajid Hasan Apon, Timos Sellis, and Jianxin Li. 2018. The flexible socio spatial group queries. *Proceedings of the VLDB Endowment* 12, 2 (2018), 99–111.

[24] Emmanuel Hebrard and George Katsirelos. 2018. Conflict directed clause learning for maximum weighted clique problem. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1316–1323.

[25] William Jack, Adam Ray, and Tavneet Suri. 2013. Transaction networks: Evidence from mobile money in Kenya. *American Economic Review* 103, 3 (2013), 356–361.

[26] Xun Jian, Yue Wang, and Lei Chen. 2020. Effective and efficient relational community detection and search in large dynamic heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1723–1736.

[27] Hua Jiang, Chu-Min Li, Yanli Liu, and Felip Manyà. 2018. A two-stage MaxSAT reasoning approach for the maximum weight clique problem. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 1338–1346.

[28] Hua Jiang, Dongming Zhu, Zhichao Xie, Shaowen Yao, and Zhang-Hua Fu. 2021. A new upper bound based on vertex partitioning for the Maximum k-plex Problem.. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1689–1696.

[29] Franziskos Kyriakopoulos, Stefan Thurner, Claus Puhr, and Stefan W Schmitz. 2009. Network and eigenvalue analysis of financial transaction networks. *The European Physical Journal B* 71 (2009), 523–531.

[30] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. 2010. A survey of algorithms for dense subgraph discovery. *Managing and Mining Graph Data* (2010), 303–336.

[31] Conggai Li, Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2020. Efficient progressive minimum k-core search. *Proceedings of the VLDB Endowment* 13, 3 (2020), 362–375.

[32] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent community search in temporal networks. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 797–808.

[33] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2020. Efficient (α, β)-core computation in bipartite graphs. *The VLDB Journal* 29, 5 (2020), 1075–1099.

[34] Boge Liu, Fan Zhang, Chen Zhang, Wenjie Zhang, and Xuemin Lin. 2019. Corecube: Core decomposition in multilayer graphs. In *Proceedings of the Web Information Systems Engineering (WISE)*. 694–710.

[35] Boge Liu, Fan Zhang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2021. Efficient community search with size constraint. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 97–108.

[36] Jingxin Liu, Chang Xu, Chang Yin, Weiqiang Wu, and You Song. 2020. k-core based temporal graph convolutional network for dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering* 34, 8 (2020), 3841–3853.

[37] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based community search over large directed graphs. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 2183–2197.

[38] Junjie Luo, Hendrik Molter, and Ondrej Suchý. 2018. A parameterized complexity view on collapsing k-cores. In *Proceedings of the International Symposium on Parameterized and Exact Computation (IPEC)*. 7:1–7:14.

[39] Wensheng Luo, Kenli Li, Xu Zhou, Yunjun Gao, and Keqin Li. 2022. Maximum biplex search over bipartite graphs. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 898–910.

[40] Yu-Liang Ma, Ye Yuan, Fei-Da Zhu, Guo-Ren Wang, Jing Xiao, and Jian-Zong Wang. 2019. Who should be invited to my party: A size-constrained k-core problem in social networks. *Journal of Computer Science and Technology* 34 (2019), 170–184.

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

[42] Derek Mikesell and Illya V Hicks. 2022. Minimum k-cores and the k-core polytope. *Networks* 80, 1 (2022), 93–108.

[43] David Peleg, Ignasi Sau, and Mordechai Shalom. 2013. On approximating the d-girth of a graph. *Discrete Applied Mathematics* 161, 16-17 (2013), 2587–2596.

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

[45] Jieying She, Yongxin Tong, Lei Chen, and Tianshu Song. 2017. Feedback-aware social event-participant arrangement. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 851–865.

[46] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. 939–948.

[47] Panagiotis Symeonidis, Eleftherios Tiakas, and Yannis Manolopoulos. 2011. Product recommendation and rating prediction based on multi-modal social networks. In *Proceedings of the ACM conference on Recommender systems (RecSys)*. 61–68.

[48] Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, Lu Qin, and Yuting Zhang. 2021. Efficient and effective community search on large-scale bipartite graphs. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 85–96.

[49] Mingyu Xiao, Weibo Lin, Yuanshun Dai, and Yifeng Zeng. 2017. A fast algorithm to compute maximum k-plexes in social network analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 919–925.

[50] Kai Yao and Lijun Chang. 2021. Efficient size-bounded community search over large networks. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1441–1453.

[51] Fan Zhang, Haicheng Guo, Dian Ouyang, Shiyu Yang, Xuemin Lin, and Zhihong Tian. 2023. Size-constrained community search on large networks: An effective and efficient solution. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[52] 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 *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 245–251.

[53] Hanpeng Zhang, Zhaohua Wang, Shengjun Chen, and Chengqi Guo. 2019. Product recommendation in online social networking communities: An empirical study of antecedents and a mediator. *Information & Management* 56, 2 (2019), 185–195.

[54] Ying Zhang, Lu Qin, Fan Zhang, and Wenjie Zhang. 2019. Hierarchical decomposition of big graphs. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. IEEE, 2064–2067.

[55] Yi Zhou, Shan Hu, Mingyu Xiao, and Zhang-Hua Fu. 2021. Improving maximum k-plex solver via second-order reduction and graph color bounding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 12453–12460.

[56] Zhongxin Zhou, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Chen Chen. 2019. K-core maximization: An edge addition approach. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 4867–4873.