Efficient size-prescribed k-core search

Yiping Liu[1]
yliu823@aucklanduni.ac.nz

Bo Yan[2]
yanbo@bit.edu.cn

Bo Zhao[2]
xbackturbo@163.com

Hongyi Su[2]
henrysu@bit.edu.cn

Yang Chen[1]
yang.chen@auckland.ac.nz

Michael Witbrock[1]
m.witbrock@auckland.ac.nz

1 *School of computer science,Univsersity of Auckland, Auckland, New Zealand*
2 *School of Computer Science, Beijing Institute of Technology, BeiJing, China*

***Abstract*—k-core is a subgraph where every node has at least k neighbors within the subgraph. The k-core subgraphs has been employed in large platforms like Network Repository to comprehend the underlying structures and dynamics of the network. Existing studies have primarily focused on finding k-core groups without considering their size, despite the relevance of solution sizes in many real-world scenarios. This paper addresses this gap by introducing the size-prescribed k-core search (SPCS) problem, where the goal is to find a subgraph of a specified size that has the highest possible core number. We propose two algorithms, namely the *TSizeKcore-BU* and the *TSizeKcore-TD*, to identify cohesive subgraphs that satisfy both the k-core requirement and the size constraint. Our experimental results demonstrate the superiority of our approach in terms of solution quality and efficiency. The *TSizeKcore-BU* algorithm proves to be highly efficient in finding size-prescribed k-core subgraphs on large datasets, making it a favorable choice for such scenarios. On the other hand, the *TSizeKcore-TD* algorithm is better suited for small datasets where running time is less critical.**

***Index Terms*—Social network, k-core, community detection, subgraph search, prescribed size.**

I. Introduction

Graphs are widely used to model relationships between individuals with wide applications in various domains, such as social science, biology, information technology, and physics. Within graph analysis, a fundamental challenge lies in the discovery of k-cores, which are subgraphs where every node has at least k neighbors within the subgraph. These k-core groups play a significant role in comprehending the underlying structures and dynamics of the network [11].

Despite the extensive research on k-core group detection, existing studies primarily focus on finding k-core groups without considering their size, neglecting an important aspect of network analysis. However, in many real-world scenarios, researchers and practitioners are interested in identifying k-core groups with a specific and prescribed size.

In this paper, we focus on addressing the size-prescribed k-core search problem: Given a graph $G = (V, E)$ and a positive integer $t \leqslant |V|$, find a size-t subgraph with highest core number, where the core number of a subgraph H as the highest k such that H is a k-core subgraph.

The size-prescribed k-core search problem poses unique challenges and calls for innovative methodologies and algorithms. Existing approaches for k-core group detection do not directly tackle this specific problem, as they do not consider the size constraint during the search process [3]–[5], [10]. Furthermore, the existing branch-and-cut method, which addresses bounded size subgraphs, is less time-efficient and only suitable for small desired sizes [13]. And the existing method [6] relax the k-core constraint with a closeness requirement. Consequently, there is a gap in the literature.

We list two applications of the SPCS problem. *Event Organization.* Online event recommendation platforms e.g. Meetup, Groupon, Eventbrite commonly employ the social group information to suggest relevant events to users [15], [16]. When organizing events, size requirements are often specified. Therefore, the goal of platforms is to identify tightly-knit groups of a specific size that can effectively engage participants. This problem can be formulated as the SPCS problem, where the graph represents individuals and their friendships. *Protein Analysis.* In protein-protein interaction (PPI) networks, where each node represents a protein and each edge represents a protein-protein connection, the proteins within the same k-core subgraph often share similar functionalities [1], [8], [9], [12]. However, it has been observed that the homophily property does not hold for large k-core subgraphs [5]. Consequently, there is a need to control the size of the founded k-core subgraphs. The problem of searching for protein clusters of a given size and with the highest core number in the PPI can be modeled as the SPCS problem.

Our Contributions. The main contributions of this paper can be summarized as follows:

1) We formalize the size-prescribed k-core search problem. This problem is proven to be NP-hard, W[1]-hard and hard-to-approximate. See Sec.III.
2) We introduce a novel algorithm called the TSizeKCore algorithm to address the size-prescribed k-core search problem. The algorithm incorporates two strategies: a

ASONAM '23, November 6-9, 2023, Kusadasi, Turkey
© 2023 Association for Computing Machinery.
ACM ISBN 979-8-4007-0409-3/23/11 $15.00
https://doi.org/10.1145/3625007.3627316

top-down strategy and a bottom-up strategy. The top-down strategy starts by locating a large k-core within the graph. Then a size refinement process is applied to reduce the size of the k-core until it reaches the desired size. On the other hand, the bottom-up strategy begins by identifying a small k-core and gradually expands it by iteratively adding nodes until the desired size is achieved. See Sec.IV.

3) We validate the effectiveness and efficiency of the proposed algorithms through extensive experiments. In the majority of cases, our algorithm yields optimal solutions.

II. Related work

The line of work that related to k-core subgraph search can be classified into three categories according to the desired sizes of k-core: maximal k-core subgraph search (e.g., [7], [14]), minimum k-core subgraph search (e.g., [2]–[5]) and bouned-size k-core subgraph search (e.g., [6], [13]). The primary technique to solve the bouned-size k-core subgraph search is called branch-and-bound, which is based on the enumeration of all feasible solutions. However, it should be noted that these exact methods, owing to the fact of numerous candidates, exhibit lower time efficiency and are only applicable to small desired sizes. To the best of our knowledge, there currently does not exist an efficient algorithm to address the bounded-size k-core search problem when the desired size is large. This highlights the need for novel methodologies and algorithms.

III. The size-prescribed k-core search Problem

Consider a simple connected graph $G = (V, E)$, where V denotes the set of nodes and $E \subseteq V^2$ denotes the set of edges. Write $n = |V|$ and $m = |E|$ as the number of nodes and edges in G respectively. For a subset $S \subseteq V$, write $G[S]$ as the subgraph induced by S in G. Given a subgraph H of G, write $H \cup S$, $H - S$ as the subgraph induced by $V_H \cup S$ and $V_H - S$ in G respectively. For a node $v \in V$, let $N(v)$ be the nodes that links to v in the graph G.

k-core subgraph [11]. Given an integer $k \in [0, n]$, a subgraph H is a k-core subgraph of G if every node in H has at least k neighbors within H.

If H is a k-core subgraph, then it is also an i-core subgraph for all $i = 1, 2, \cdots, k-1$.

Core number. The core number of a subgraph H is the highest k such that H is a k-core subgraph.

Higher core number indicating greater cohesion among graph members. When the core number of H is $|V_H| - 1$, then H is a clique. Conversely, when the core number of H is 0, then H is disconnected.

The size-prescribed k-core search (SPCS) Problem. Given a graph $G = (V, E)$ and a positive integer $t \leqslant |V|$, the SPCS problem searches for a size-t subgraph with highest core number.

It is easy to see that the q-clique problem is a special case of the SPCS problem and thus we have the following results.

Theorem 1. *The SPCS problem is (a) $W[1]$-hard; (b) hard-to-approximate within $n^{1-\epsilon}$ for any $\epsilon > 0$.* □

IV. The TSizeKCore Algorithm

The SPCS problem involves two objectives: maximizing the core number and outputting a subgraph of size t. Simultaneously satisfying both objectives is computationally challenging, as discussed in Section III. To address this challenge, it is crucial to establish a priority between these two objectives. Given the potentially large number of t-size subgraphs, we firstly maximize the core number and then refine the size of the subgraph to satisfy size requirement. We provide a step-by-step description of Alg.1 below:

1) Calculate an upper bound $\overline{k}$ on the core number of all size-t subgraphs.
2) Search for a set of k-core subgraphs as candidates.
3) Starting from a k-core subgraph, the algorithm adds or removes nodes in an attempt to find a size-t k-core subgraph.
4) Repeat step 2) and step 3) until it either finds a size-t k-core subgraph or failure. If a failure is occurred, decrease k by 1 and go to step 2).

In a connected graph, finding a size-t 1-core subgraph can be achieved by a breadth-first search (BFS) which ensures Alg.1 will output a feasible solution. Notably, Alg.1 employs two strategies: the top-down strategy and the bottom-up strategy. The top-down strategy begins with a larger subgraph and iteratively removes nodes until the target size is achieved (Section IV-A). Conversely, the bottom-up strategy starts with a smaller subgraph and iteratively adds nodes until the target size is reached (Section IV-B).

Algorithm 1: TSizeKCore algorithm

```
   Input: Graph $G = (V, E)$, integer $t \in [1, n]$
   Output: A core subgraph of size $t$
 1 Run the algorithm in [7]
 2 Let $\overline{k}$ be the largest core number such that a maximal
     $k$-core with size $\geqslant t$ exists
 3 Initialize $k \leftarrow \overline{k}$
 4 while $k > 1$ do
 5     Run $GetKcore(G, k, t)$ and get $\mathcal{G}$
 6     for each $k$-core subgraph $H \in \mathcal{G}$ do
 7         if $|V_H| = t$ then
 8             return $H$
 9         else
10             $H' \leftarrow$ SizeRefinement(H,t,k)
11             if $|V_{H'}| = t$ then
12                 return $H'$
13     $k \leftarrow k - 1$
14 return an arbitrary connected subgraph of size $t$
```

A. The Top-down strategy

The top-down strategy consists of two parts: identify large k-core subgraphs and refine large k-core subgraphs. Initially, the algorithm utilizes the *GetKcore-TD* subprocedure (Line 5) to obtain a set of k-core subgraphs that are larger than the desired size t. Subsequently, the strategy employs the *SizeRefinement-TD* function (Line 10) to remove nodes from the given subgraph H until the resulting subgraph is of size t. By starting with a larger subgraph and iteratively removing nodes, the strategy aims to find a subgraph that satisfies both the core number and size requirements.

To identify large k-core subgraphs, the maximal k-core subgraphs naturally serve as large k-core subgraph candidates.

Maximal k-core subgraph. A k-core subgraph H is a maximal k-core subgraph if for any subset $S \subseteq V - V_H$, $H \cup S$ is not a k-core subgraph. Maximal k-core subgraphs can be enumerated in $O(m)$ [7].

GetKcore-TD subprocedure: Given a graph G, a core number k, and a size requirement t, the subprocedure outputs all maximal k-core subgraphs that are larger than t. We directly employ the enumeration algorithm in [7].

Once a large k-core subgraph is identified, the Top-down strategy begins to reduce its size to achieve the desired size t while maintaining its core number.

SizeRefinement-TD subprocedure takes as input a graph H, a core number k, and a size requirement t, and outputs either a t-size k-core subgraph or failure. The procedure iteratively selects a node from H. For each selected node v, the subprocedure checks whether there exists a k-core subgraph of size larger than t in $H - v$. If such a subgraph exists, then H is updated with the k-core subgraph found in $H - v$. The subprocedure returns H if it is of size t. The subprocedure continues this process until H reaches the desired size t or all nodes have been checked. When each node has been checked by the procedure, the resulting subgraph H is a minimal k-core subgraph. This means that no subset of nodes can be further removed from H while still maintaining the k-core property. Notice that the order of node removal is determined randomly and thus might result in different outputs.

Algorithm 2: SizeRefinement-TD

```
Input: $k$-core graph $H$, integers $k, t < n$
Output: A $k$-core subgraph of size $t$ or failure
for each node $v$ of $H$ do
    $H' \leftarrow H$
    Remove $v$ and its incident edges from $H'$
    $\mathbb{C} \leftarrow$ all maximal $k$-core subgraphs of $H'$ that has
      at least $t$ nodes
    if $\exists C \in \mathbb{C}$ such that $|C| = t$ then
        return $C$
    if $\exists C \in \mathbb{C}$ such that $|C| > t$ then
        $H \leftarrow H'$
return failure
```

B. The Bottom-up strategy

The bottom-up strategy is an alternative strategy of the top-down strategy. The strategy consists of two parts: identify large k-core subgraphs and refine large k-core subgraphs. In this strategy, the *GetKcore-BU* subprocedure (Line 5) is called to output a set of k-core subgraphs that are smaller than t. The *SizeRefinement-BU* subprocedure (Line 10) is then used to add nodes from the input graph into the given subgraph H, until the resulting subgraph is of size t. Similarly, the minimal k-core subgraphs serve as small k-core subgraph candidates.

Algorithm 3: GetKcore-BU

```
Input: Graph $G = (V, E)$, integers $k, t < n$
Output: A set of $k$-core subgraph of size smaller or
        equal to $t$
Initialize an empty set $\mathbb{S}$
$\mathbb{C} \leftarrow$ all maximal $k$-core subgraphs of $G$
for each $H \in \mathcal{G}$ do
    if $|V_H| \leqslant t$ then
        add the subgraph $H$ into $\mathbb{S}$
    if $|V_H| > t$ then
        let $H'$ be the residual subgraph of $H$ after
          randomly removing $|V_H| - t$ nodes
        add all maximal $k$-core subgraphs of $H'$ into $\mathbb{S}$
return $\mathbb{S}$
```

Algorithm 4: SizeRefinement-BU

```
Input: Graph $G = (V, E)$, $k$-core subgraph
       $H = (V_H, E_H)$ of $G$, integers $k, t < n$
Output: A $k$-core subgraph of size $t$ or failure
while $|V_H| < t$ and $\exists v \in V \backslash V_H$ such that
  $N(v) \cap V_H \geqslant k$ do
    add $v$ and its incident edges with nodes of $V_H$ into
      $H$
if $|H| = t$ then
    return $H$
return failure
```

Minimal k-core subgraph. A k-core subgraph H is a minimal k-core subgraph if for any subset $S \subseteq V_H$, $H - S$ is not a k-core subgraph.

There can exist an exponential number of minimal k-core subgraphs making it NP-hard to enumerate them. For example, consider a clique graph with n nodes. For any k in the range $[1, n-1]$, the number of minimal k-core subgraphs is $\binom{n}{k+1}$, which grows exponentially with n.

The *GetKcore-BU* subprocedure aims to provide k-core subgraphs with sizes smaller than t. The subprocedure generates subgraphs that are smaller than t but may not be minimal. Such subgraphs can then be further refined through the *SizeRefinement-BU* procedure.

GetKcore-BU subprocedure takes as input a graph G, a core number k, and a size requirement t, and returns a set of randomly selected k-core subgraphs that have a size smaller than t. This subprocedure operates by first selecting a subgraph of size t at random from the input graph G. Then, all maximal k-core subgraphs within the selected subgraph are identified and returned. It is important to note that the output of this subprocedure is a result of a random process. More details on this approach can be found in Alg.3.

Once a small k-core subgraph is identified, the Bottom-up strategy begins to add nodes into it to achieve the desired size t while maintaining its core number.

SizeRefinement-BU Suppose H is a k-core subgraph of size smaller than t. To find t nodes that induce a k-core subgraph, the subprocedure iteratively adds nodes that has at least k neighbors from G into H. If no such node is found, the subprocedure stops and declare a failure. By the definition of k-core, it is clear that the addition does not deteriorate the core number of H. More details can be found in Alg.4.

V. Experiments

Algorithms:

S-greedy algorithm, originally proposed by Barbieri et al. [3], is a heuristic algorithm designed to solve the minimum k-core search problem. In our work, we adapt this algorithm to target a prescribed-size subgraph, addressing the size-prescribed k-core search problem.

TSizeKcore-TD. The proposed TSizeKCore algorithm with top-down strategy is refereed to as *TSizeKcore-TD*.

TSizeKcore-BU. The proposed TSizeKCore algorithm with bottom-up strategy is refereed to as *TSizeKcore-BU*.

Critical. We also include a naive strategy, the *Critical* strategy, as a benchmark. This strategy operates similarly to the top-down strategy. The distinction lies in that it allows repeated selection of nodes and directly checks whether the resulting subgraph H is a k-core. It terminates when either a size-t k-core subgraph or a critical k-core is achieved. A critical k-core subgraph is a k-core subgraph where for each $v \in V_H$, $H - \{v\}$ is not a k-core.

We calculate an upper bound $\overline{k}$ on the optimal core number (see line 3 in Alg.1). The closer the core number of the outputted subgraph is to the upper bound, the better the solution is considered to be. In particular, if the outputted subgraph is a $\overline{k}$-core, then it is an optimal solution.

Datasets. We conducted experiments on ten real-world datasets to evaluate the performance of our algorithm. These datasets are collected from SNAP and KONECT.

TABLE I: Dataset statistics

Dataset	nodes	edges	$avg.deg.$	$max.deg.$
Arenas	1,133	5,451	9.62	71
Friend	1,858	12,534	13.49	272
Yeast	1,870	2,277	2.43	56
P2P8	6,301	20,777	6.59	97
Lastfm	7,624	27,806	7.29	216
Hepph	34,546	421,578	24.4	846
Enron	36,692	183,831	10.02	1,383
GPlus	107,614	13,673,453	254	20
DBLP	310,297	1,186,302	4.64	340
Youtube	1,134,890	2,987,624	5.26	28,754

Settings. All programs are implemented in PYTHON 3.7 and conducted on a machine with Intel Xeon(R) 2.10GHz CPU. Each result averages 200 outputs.

Source Code: https://github.com/turbojob/wrapper.

Effectiveness experiment. We compare the core numbers of the outputted subgraphs for different desired sizes t on seven datasets. For large datasets, we conduct solely the *TSizeKcore-BU* algorithm due to the high time complexity of other algorithms. To facilitate comparison on various datasets, we normalize the desired size by the input graph size.

As shown in Fig.1, the *TSizeKcore-TD* algorithm consistently outperforms all other algorithms in all cases. Notably, in most cases, the core numbers outputted by the *TSizeKcore-TD* algorithm meet the upper bound $\overline{k}$, indicating that these outputs are optimal solutions. We also observe a similar performance of the *Critical* algorithm and the *TSizeKcore-BU* algorithm, despite the *TSizeKcore-BU* algorithm having lower time complexity. Furthermore, both algorithms tend to output optimal solutions as the desired size increases, which can be explained by the fact that it becomes less likely for important nodes to be removed, leading to more optimal solutions. It is remarkable to observe that the *TSizeKcore-BU* algorithm consistently outputs near optimal solutions on three large datasets, even when the desired sizes are small and the optimal core number is large. This indicates the effectiveness and accuracy of the algorithm in finding optimal or near-optimal solutions in these scenarios.

Efficiency experiment. We focus on the running time comparison of the mentioned algorithms on seven datasets. The running time of other algorithms exceeds 24 hours on three large datasets. We further analyze the average running time of the *TSizeKcore-BU* algorithm on the *Gplus*, *DBLP*, and *Youtube* datasets to provide additional insights.

The experimental results are presented in Fig.2. As shown in the first plot of Fig.2, the *TSizeKcore-BU* algorithm exhibits the lowest running time among all algorithms. Specifically, its average running time remains below 300 seconds on all datasets. On the other hand, the *S-greedy* and *Critical* algorithms require longer running times compared to the *TSizeKcore-TD* algorithm. For the *Youtube* dataset, which contains 1 million nodes and 3 million edges, the running time of the *TSizeKcore-BU* algorithm remains below 20,000 seconds. These results further confirm the efficiency of the *TSizeKcore-BU* algorithm in handling large-scale datasets. The spikes might result from the large difference between target size and the sizes of existing maximal subgraphs.

In general, the *S-greedy* algorithm exhibits the worst performance in terms of both core number and running time. The *Critical* algorithm shows similar core number to the *TSizeKcore-BU* algorithm but has a higher running time. On the other hand, the *TSizeKcore-TD* algorithm demonstrates the best performance in terms of core number output, but it requires much running time.

VI. Summary

This paper studies the SPCS Problem which is proven to be computationally hard. We propose two algorithms, namely the *TSizeKcore-BU* and the *TSizeKcore-TD*, which have been shown to provide optimal solutions on most of real-world graphs. In particular, the *TSizeKcore-TD* algorithm is effective

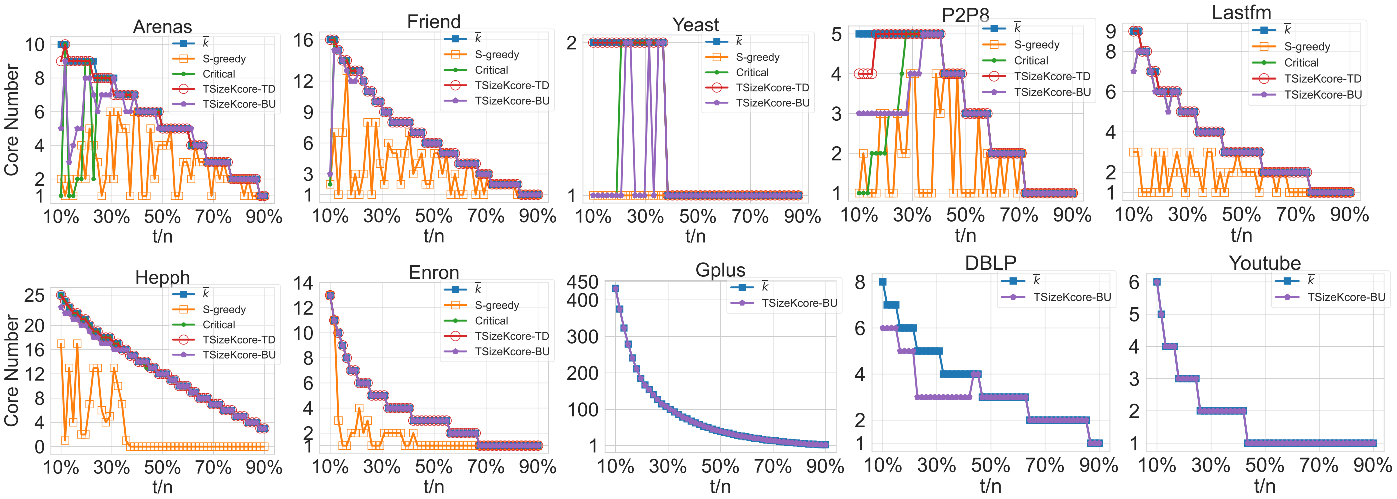

Fig. 1: Effectiveness. The experiment compares the core number of subgraphs outputted by different algorithms with varying t.

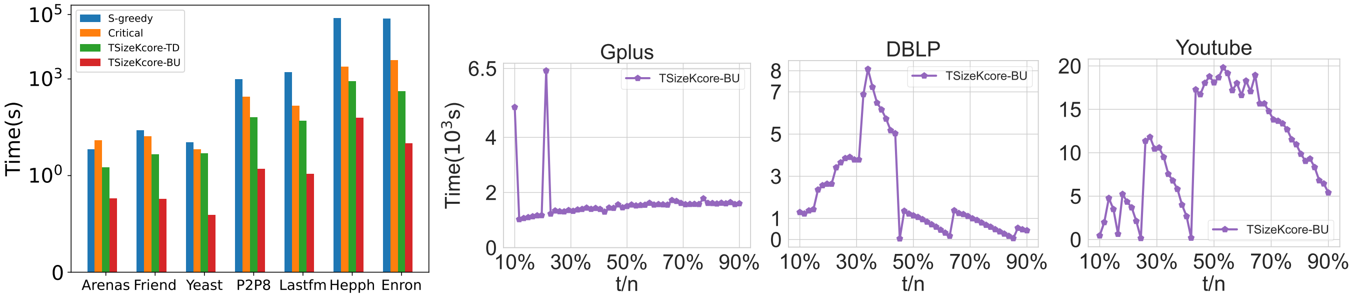

Fig. 2: The average running time of different algorithms. We further list the running time of *TSizeKcore-BU* on large datasets. The running time of the *S-greedy*, *Critical*, and *TSizeKcore-TD* algorithms is not listed due to their excessive running time.

for small datasets where running time is not critical, while the *TSizeKcore-BU* algorithm is efficient on large datasets.

Acknowledgement This work is supported by China Scholarship Council (CSC) Grant #201906030067.

References

[1] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. 2005. Large scale networks fingerprinting and visualization using the k-core decomposition. In Advances in Neural Information Processing Systems 18, 2005.

[2] Omid Amini, Ignasi Sau, and Saket Saurabh. 2012. Parameterized complexity of finding small degree-constrained subgraphs. Journal of Discrete Algorithms 10 (2012), 70–83.

[3] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. Data mining and knowledge discovery 29 (2015), 1406–1433.

[4] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data. 991–1002.

[5] 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 (2020).

[6] Yu-Liang Ma, Ye Yuan, Fei-Da Zhu, Guo-Ren Wang, Jing Xiao, and Jian-Zong Wang. 2019a. 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.

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

[8] Aua Md., H. Mori, S. Kanaya, K. Nishikata, T. Korna, T. Miyasato, Y. Shinbo, A. Md., C. Wada, and M. Maeda. 2011. Prediction of Protein Functions Based on K-Cores of Protein-Protein Interaction Networks and Amino Acid Sequences. Genome Informatics 14 (2011)

[9] Mikail Rubinov and Olaf Sporns. 2010. Complex network measures of brain connectivity: uses and interpretations. NEUROIMAGE 3 (2010).

[10] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. KDD, 939-948.

[11] Douglas Brent West et al. 2001. Introduction to graph theory. Vol. 2. Prentice hall Upper Saddle River.

[12] Stefan Wuchty and Eivind Almaas. 2005. Peeling the yeast protein network. Proteomics 5, 2 (2005), 444–449.

[13] K. Yao and L. Chang. 2021. Efficient size-bounded community search over large networks. In Very Large Data Bases.

[14] Batagelj V, Zaversnik M. An o (m) algorithm for cores decomposition of networks[J]. arXiv preprint cs/0310049, 2003.

[15] Luo X, Andrews M, Song Y, et al. Group-buying deal popularity[J]. Journal of Marketing, 2014, 78(2): 20-33.

[16] Boutsis I, Karanikolaou S, Kalogeraki V. Personalized event recommendations using social networks[C]//2015 16th IEEE International Conference on Mobile Data Management. IEEE, 2015, 1: 84-93.

[17] Downey R G, Fellows M R. Fixed-parameter tractability and completeness II: On completeness for W [1][J]. Theoretical Computer Science, 1995, 141(1-2): 109-131.

[18] Hastad J. Clique is hard to approximate within n/sup 1-/spl epsiv[C]//Proceedings of 37th Conference on Foundations of Computer Science. IEEE, 1996: 627-636.