Data Science and Engineering (2018) 3:293–306
https://doi.org/10.1007/s41019-018-0078-0

K-Connected Cores Computation in Large Dual Networks

Lizhen Cui[1] · Lingxi Yue[1] · Dong Wen[2] · Lu Qin[2]

Received: 31 July 2018 / Accepted: 20 October 2018 / Published online: 12 November 2018

Abstract
Computing *k-core*s is a fundamental and important graph problem, which can be applied in many areas, such as community detection, network visualization, and network topology analysis. Due to the complex relationship between different entities, dual graph widely exists in the applications. A dual graph contains a physical graph and a conceptual graph, both of which have the same vertex set. Given that there exist no previous studies on the *k-core* in dual graphs, we formulate a *k-connected core* (*k-CCO*) model in dual graphs. A *k-CCO* is a *k-core* in the conceptual graph, and also connected in the physical graph. Given a dual graph and an integer k, we propose a polynomial time algorithm for computing all *k-CCO*s. We also propose three algorithms for computing all *maximum-connected core*s (*MCCO*), which are the existing *k-CCO*s such that a $(k+1)$-*CCO* does not exist. We further study a subgraph search problem, which is computing a *k-CCO* that contains a set of query vertices. We propose an index-based approach to efficiently answer the query for any given parameter k. We conduct extensive experiments on six real-world datasets and four synthetic datasets. The experimental results demonstrate the effectiveness and efficiency of our proposed algorithms.

Keywords Dual graph · k-core · Connectivity · Search

1 Introduction

Graph model has been used to represent the relationship of entities in many real-world applications, such as social networks, web graphs, collaboration networks, and biological networks. Given a graph $G(V, E)$, vertices in V represent the interested entities and edges in E represent the relationship between entities. Significant research efforts have been devoted toward many fundamental problems in managing and analyzing graph data. Among them, cohesive subgraph detection has been extensively studied recently [5, 9, 13, 17, 31].

Given a graph G and an integer k, a *k-core* of G is a maximal-connected subgraph in which each vertex has degree at least k [29]. The problem of computing *k-core*s draws a lot of attention [7, 17, 28, 32] due to the elegant structural properties of *k-core* [29] and the linear time solution [3]. It can be applied in many areas including but not limited to community detection [11], dense subgraph discovery [2, 6], graph visualization [1], and system analysis [10].

Motivations In many real-world applications, a single simple graph is hard to express the complex relationship between entities. [33] models a dual graph containing two complementary graphs with the same vertex set, one of which represents the physical interaction between vertices, and the other represents the conceptual interaction. They study the problem of computing the subgraph, namely *DCS*, which is the densest in the conceptual graph and also connected in the physical graph. However, computing the *DCS* in dual graphs is NP-hard. Even though an approximate solution is proposed and a relatively poorer result quality is endured in [33], the time consuming for this problem is still large and not scalable to big graphs. Additionally, they do not restrict the connectivity of *DCS* in the conceptual graph.

✉ Lizhen Cui
clz@sdu.edu.cn; zhengyongqing@dareway.com.cn

Lingxi Yue
yuelingxi@mail.sdu.edu.cn

Dong Wen
dong.wen@uts.edu.au

Lu Qin
lu.qin@uts.edu.au

[1] School of Software Engineering, Shandong University, Jian, China

[2] Centre for Artificial Intelligence, University of Technology Sydney, Sydney, Australia

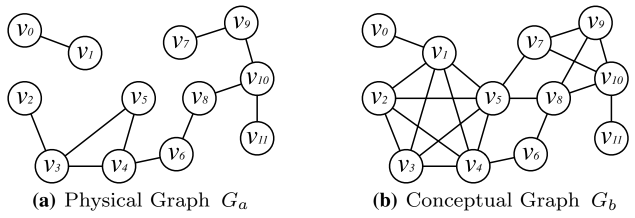


Fig. 1 An example of dual graphs $G(V,E_a,E_b)$

The result subgraph is probably disconnected and obviously not cohesive.

Given that there exists no any research on the *k-core* computation in dual graphs, in this paper, we adopt the classic *k-core* definition to model a *k-**C**onnected **CO**re* (*k-CCO*) in dual graphs. Given a dual graph and an integer k, a *k-CCO* is a dual subgraph g satisfying the following three conditions: (1) the minimum degree of g is not less than k in the conceptual graph; (2) g is connected in the conceptual graph; and (3) g is connected in the physical graph.

An example of a dual graph is given in Fig. 1. Figure 1a shows a physical graph, and Fig. 1b shows a conceptual graph. Given an integer $k = 3$, there exists only one 3-*CCO*, that is the induced dual subgraph of $\{v_2, v_3, v_4, v_5\}$. The minimum degree is not < 3, and the subgraph is connected in both two graphs.

Our *k-CCO* model restricts the connectivity for both two graphs and guarantees the cohesiveness of the result graph by given integer parameter k. Based on this model, we formulate two global detection problems. Given a dual graph G and an integer k, the first problem is computing all *k-CCO*s in G. It offers a flexible selection for the degree constraint k and returns a subjective result for users. Similar to the *DCS* problem in [33], we also study a parameter-free problem, that is computing the ***M**aximum*-**C***onnected* ***CO**res* (*MCCO*s) in a given dual graph G. Here, a *MCCO* is a *k-CCO* in G such that there does not exist any $(k + 1)$ -*CCO* in G. The 3-*CCO* in the dual graph G in Fig. 1 is also a *MCCO*, since there does not exist any 4-*CCO* in G.

We further study a subgraph search problem for the purpose of personalized query. Specifically, given an integer k and a set of query vertices, we aim to compute a *k-CCO* containing these vertices. In Fig. 1, given an integer $k = 3$ and a set of vertices $\{v_2, v_5\}$, the 3-*CCO* containing $\{v_2, v_5\}$ is the induced dual subgraph of $\{v_2, v_3, v_4, v_5\}$.

Applications Computing *k-CCO*s and *MCCO*s can be applied in many areas. For example, to mine a research group, the researchers in the group should be connected in their collaboration network (physical graph), in which each edge represents two researchers have co-authored a paper. Simultaneously, each researcher should have enough neighbors in a similarity network (conceptual graph), in which each edge represents two researchers have similar research interests. In social networks, each user may have many interest labels, such as soccer, basketball, cartoon. A conceptual graph can be built by computing the interest similarity between any two users. A physical graph can be built by checking whether any two users follow each other. A social community should be connected in the physical graph, and each user in the group should have enough neighbors with similar interest.

Challenges It is nontrivial to compute all *k-CCO*s. A *k-core* in conceptual graph may be disconnected in the physical graph, and a connected component in the physical graph may conversely violate the degree constraint and connectivity constraint in the conceptual graph. For the problem of computing the *MCCO*s, let k_{max} be the maximum k in a dual graph such that a *k-CCO* exists. Given the solution for computing *k-CCO*s, the *MCCO*s can be obtained if k_{max} is known. Therefore, a main challenge in computing the *MCCO*s is computing k_{max}. About the *k-CCO* search problem, a straightforward idea is invoking the *k-CCO* detection procedure as a preprocessing step and returning the one that contains all query vertices. It is costly to start by processing the whole graph, given that the size of the result subgraph is normally quite small.

Our Approaches and Contributions We propose a polynomial algorithm to compute all *k-CCO*s in dual graphs. It performs by recursively removing the vertex which violates the *k-CCO* definition. For the problem of computing *MCCO*s, we first follow the similar idea in computing all *k-CCO*s, and give a bottom-up solution. More specifically, we compute the *MCCO*s by iteratively removing all unsatisfied vertices. We also propose a top-down algorithm, which selects k_{max} following a top-down strategy and returns the *k-CCO*s if exist. To further improve the algorithmic efficiency, we propose a binary search algorithm for computing all *MCCO*s. We propose an index-based solution to compute the *k-CCO* containing the query vertices. Based on the proposed index, we well bounded the index size and process the query in the complexity only related to the result subgraph. The experimental results show the excellent performance of our optimized algorithms. More details are given in Sect. 6. We summarize the main contributions in this paper as follows.

- *A k-connected core model in dual graphs* We design a k-connected core model, which inherits the properties of classic *k-core* model in dual graphs. To the best of our knowledge, this is the first work that studies the *k-core* concept in dual graphs.
- *A polynomial time algorithm for computing all k-connected cores.* Given a dual graph G and an integer k, we propose a polynomial peeling-style algorithm, named

KCCO, to compute all *k-CCO*s in G. We prove the time complexity of KCCO is $O(h \times m)$. Here, m is the number of edges in the conceptual graph, and h is a value theoretically roughly bounded by but practically much less than the number of vertices in G.

- *Three algorithms for computing the maximum-connected cores* We give a bottom-up and a top-down algorithms for the *MCCO* computation. An optimized binary search algorithm is finally proposed to achieve significant speedup.
- *An index-based solution for searching the k-connected core* We design an elegant index structure to compute a *k-CCO* containing a set of query vertices. Based on the proposed index structure, we give an efficient query-processing algorithm and a polynomial time index construction algorithm. The size of index is bounded by $O(n)$ and the time complexity of index-based query algorithm is $O(m_b)$. Here, n is the number of vertices in G and m_b is the number of physical edges in the result subgraph.
- *Extensive performance studies* We conduct extensive performance studies on four synthetic graphs and six real large graphs. We also present a case study. The results demonstrate the effectiveness and efficiency of our proposed model and algorithms.

Organization The rest of this paper is organized as follows. Section 2 introduces preliminary concept and defines the problem. Section 3 proposes an algorithm for computing all *k-CCO*s. Section 4 studies the problem of computing all *MCCO*s. Section 5 studies the *k-CCO* search problem and proposes an index-based solution. Section 6 evaluates our proposed algorithms in extensive experiments. Section 7 introduces the related works, and Sect. 8 concludes the paper.

A short version of this paper is published in [34]. The current version extends the original paper by studying the *k-CCO* search problem, which is computing a *k-CCO* that contains a set of query vertices. We propose an index-based solution and conduct extensive experiments to show the high efficiency of our method.

2 Preliminaries

Cores in Simple Graphs Before studying the dual graphs, we briefly introduce several definitions and recall the problem of *k-core* computation in simple graphs. Let $G(V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Given a vertex u in G, we use $N_G(u)$ to denote the neighbor set of u in G, i.e., $N_G(u) = \{v \in V | (u, v) \in E\}$. The degree of a vertex u in G is denoted by $deg_G(u)$, i.e., $deg_G(u) = |N_G(u)|$. Given a vertex set S, the induced subgraph of S in G is denoted by $G[S]$, i.e., $G[s] = (S, \{(u, v) \in E | u \in S \wedge v \in S\})$. The formal definition of *k-core* is given below.

Definition 1 *(K-Core)* A *k-core* of graph $G(V, E)$ is a maximal-connected subgraph in which each vertex has degree at least k. [29]

Definition 2 *(Core Number)* The core number of a vertex u in G, denoted by $core(u)$, is the maximal number of k such that u is contained in a *k-core*.

Definition 3 *(Degeneracy)* The degeneracy of a graph G, denoted by $\mathcal{D}(G)$, is the maximal number of k such that a *k-core* exists, i.e., $\mathcal{D}(G) = \max_{u \in V} core(u)$.

We denote the *k-core* containing a given vertex u by $G_k(u)$, and have the following lemma.

Lemma 1 $\forall 1 \le k < \mathcal{D}(G), G_{k+1}(u) \subseteq G_k(u)$.

Let $V_k(u)$ be the set of vertices in which each vertex v can be reached from u via a path that every vertex w in the path satisfies $core(w) \ge k$. Following lemma holds:

Lemma 2 $G_k(u) = G[V_k(u)]$

Algorithm 1 Core-Decomposition[3]

```
Input: A graph $G(V, E)$
Output: The core numbers of all vertices in $G$
1: $G'(V', E') \leftarrow G(V, E)$;
2: while $V' \neq \emptyset$ do
3:     $k \leftarrow \min_{u \in V'} deg_{G'}(u)$;
4:     while $\exists u \in V', deg_{G'}(u) < k + 1$ do
5:         $core(u) \leftarrow k$;
6:         remove $u$ and its incident edges from $G'$;
7: return $core(u)$ for all $u \in V$;
```

Given the core numbers of all vertices, all *k-core*s can be easily found based on Lemma 2. The algorithm for computing all core numbers [3] is given in Algorithm 1 . It performs by iteratively removing the vertex with minimum degree and its incident edges. The time complexity of Algorithm 1 is $O(m)$.

Cores in Dual Graphs In this paper, we focus on an undirected dual graph $G(V, E_a, E_b)$, where E_a and E_b represent the edge sets in physical graph G_a and conceptual graph G_b, respectively. The example of the dual graph is shown in Fig. 1. Based on the aforementioned classic *k-core* concept, we define the *k-Connected* ***CO****re* (*k-CCO*) in dual graphs.

Definition 4 Given a dual graph $G(V, E_a, E_b)$, a dual subgraph $G[C]$ is a k-connected core (k-CCO) if: (1) $G_a[C]$ is connected; (2) $G_b[C]$ is connected; (3) $\forall u \in C, deg_{G_b[C]}(u) \geq k$; and 4) $G[C]$ is maximal.

Note that in the existing work [33] for computing the densest connected subgraph in dual graphs, only the connectivity in physical graph is required. This condition is insufficient to support the cohesiveness of result subgraphs, since the subgraph may be disconnected in the conceptual graph. To conquer this drawback, our *k-CCO* definition guarantees the connectivity for both physical and conceptual graphs. Based on Definition 4, we further define the ***Maximum-Connected COre*** (*MCCO*) below.

Definition 5 Given a dual graph $G(V, E_a, E_b)$, a dual subgraph $G[C]$ is a maximum-connected core (*MCCO*) if $G[C]$ is a *k-CCO*, and $(k+1)$-*CCO* does not exist.

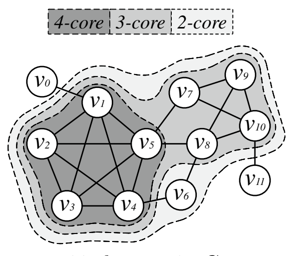
Definition 6 *(Maximum CCO Number)* Given a dual graph G, the maximum *CCO* number of G, denoted by $k_{max}(G)$, is the maximum value of k such that a *k-CCO* exists.

Based on Definitions 4 and 5, we formally define the two problems studied in this paper as follows.

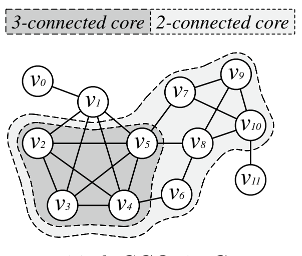
Problem 1 Given a dual graph $G(V, E_a, E_b)$ and an integer k, find all *k-CCO*s in G.

Problem 2 Given a dual graph $G(V, E_a, E_b)$, find all *MCCO*s in G.

Example 1 We give an example of *k-CCO* and *MCCO*. The *k-CCO*s of the dual graph G in Fig. 1 are presented in Fig. 2 (b). The *k-cores* of G_b are also reported as comparisons in Fig. 2a. There does not exist a 4-*CCO* in G, and the *MCCO* of G is the 3-*CCO* containing v_2, v_3, v_4, and v_5. The degeneracy of G_b is 4, and the 4-*core* is the induced subgraph of v_1, v_2, v_3, v_4, and v_5. It is not a 4-*CCO* in G, since v_1 does not connect to other vertices in G_a.



(a) *k-cores* in G_b



(b) *k-CCO*s in G_b

Fig. 2 An example of *k-CCO*s

3 Computing K-Connected Cores

Given an integer k, we study the problem of computing all *k-CCO*s in this section. We first give several lemmas about *k-CCO* based on Definition 4. Given a dual graph $G(V, E_a, E_b)$ and a *k-CCO* $G[C] \subset G$, following lemmas hold.

Lemma 3 *There exists a k-core $G_b[S]$ in G_b, such that $C \subset S$.*

Lemma 4 *There exists a connected component $G_a[H]$ in G_a, such that $C \subset H$.*

Based on Lemmas 3 and 4, we propose a peeling algorithm for computing all *k-CCO*s. The pseudocode is given in Algorithm 2.

Algorithm 2 Computing **K**-**C**onnected **CO**res (KCCO)

```
Input: A graph $G(V, E_a, E_b)$, and a parameter $k$
Output: The set $\mathbb{C}$ containing all $k$-$CCO$s in $G$
1: $\mathbb{C} \leftarrow \emptyset$;
2: for each connected component $G_a[C]$ in $G_a$ do
3:     if $\forall u \in C, deg_{G_b[C]}(u) \geq k$ and $G_b[C]$ is connected then
4:         $\mathbb{C} \leftarrow \mathbb{C} \cup G_b[C]$;
5:     else
6:         while $\exists u \in C, deg_{G_b[C]}(u) < k$ do
7:             $C \leftarrow C - \{u\}$;
8:         for each connected component $G_b[H]$ in $G_b[C]$ do
9:             $\mathbb{C} \leftarrow \mathbb{C} \cup$ KCCO($G_b[H], k$);
10: return $\mathbb{C}$;
```

The algorithm performs by recursively removing the vertex that does not satisfy the degree constraint and the connectivity constraint in Definition 4. We compute all connected components of G_a in line 2. Lemma 4 guarantees that we will not lose any *k-CCO* in this step. We add $G_b[C]$ to the result set if $G_b[C]$ is connected and satisfies the degree constraint (line 3–4). Otherwise, the algorithm from line 6 to line 8 computes a *k-core* $G_b[H]$ of $G_b[C]$. All vertices that violate the degree constraint in $G_b[C]$ are iteratively removed from C; and for each connected component $G_b[H]$, we recursively invoke KCCO to find *k-CCO*s in $G_b[H]$ (line 8–9). The correctness of this step is guaranteed by Lemma 3.

The process of Algorithm 2 can be represented by a DFS tree as depicted in Fig. 3. Each node in the tree demonstrates an input dual graph G for the invocation of

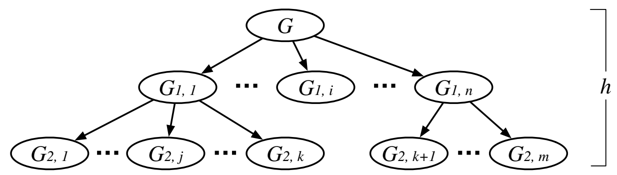


Fig. 3 DFS Tree

KCCO. Let h be the height of the tree. The time complexity of *Algorithm* 2 is given as follows.

Theorem 1 *Given a graph* $G(V, E_a, E_b)$ *and an integer* k, *the time complexity of Algorithm 2 is* $O(h|E_b|)$.

Proof Obtaining all connected components in line 2 of Algorithm 2 costs $O(|E_a|)$ time. Checking the degree constraint and connectivity of G_b in line 3 costs $O(|E_b|)$ time. From line 6 to line 8, Algorithm 2 also costs $O(|E_b|)$ time to remove all vertices whose degree is less than k and compute the connected components $G_b[H]$. Normally, we have $|E_a| < |E_b|$, and the time cost for each node in the DFS tree is $O(|E_b|)$, where E_b is the edge set of input conceptual graph.

Let $\mathbb{G}_l$ be the set of all input graphs on height l of DFS tree, where the height of a node is the distance from root to that node. We can find that there does not exist any vertex or edge overlap between different connected components in line 8. Given the tree height h, we have $\forall 0 \leq l \leq h, \sum_{G'(V', E'_a, E'_b) \in \mathbb{G}_l} |E'_b| \leq |E_b|$, where $|E_b|$ is the number of edges in the original conceptual graph. Therefore, the total time complexity of Algorithm 2 is $O(h|E_b|)$. □

Discussion The time complexity of Algorithm 2 is the product of two parts:

- The first part is the tree height h. Note that in the DFS tree, the size of input graph in each node must be less than that in its parent node. Therefore, h is roughly bounded by $|V|$. However, h is much smaller than $|V|$ in practice. In our experiments, h is not larger than 5 on all datasets.
- The second part is the graph size $|E_b|$. Given that vertices violating the degree constraint are removed in line 7 of Algorithm 2, the graph size becomes small when the tree height increases. The practical performance of Algorithm 2 is given in Sect. 6.

4 Computing Maximal-Connected Cores

We study the problem of computing all *MCCO*s in this section. A straightforward bottom-up solution BU-MCCO is first given in Sect. 4.1. Then, we propose a top-down solution TD-MCCO in Sect. 4.2. To further improve the algorithmic efficiency, we give a binary search algorithm, namely BIN-MCCO, in Sect. 4.3. Our experiments demonstrate BIN-MCCO outperforms TD-MCCO and BU-MCCO. The details are given in Sect. 6.

Algorithm 3 BU-MCCO

```
Input: A graph $G(V, E_a, E_b)$
Output: The set containing all $MCCO$s in $G$
1: $\mathbb{C} \leftarrow \emptyset$;
2: $\mathbb{T} \leftarrow$ KCCO$(G, 1)$;
3: $k \leftarrow 1$;
4: while $\mathbb{T} \neq \emptyset$ do
5:     $\mathbb{C} \leftarrow \mathbb{T}$;
6:     $k \leftarrow k + 1$;
7:     $\mathbb{T} \leftarrow \emptyset$;
8:     for each $G[C] \in \mathbb{C}$ do
9:         $\mathbb{T} \leftarrow \mathbb{T} \cup$ KCCO$(G[C], k)$;
10: return $\mathbb{C}$;
```

4.1 A Bottom-Up Approach

We give a straightforward algorithm for computing all *MCCO*s in this section. Similar to the concept of *k-core*, a nest property of *k-CCO* can be also easily obtained according to Definition 4.

Lemma 5 *Given an integer* $1 < k \leq k_{max}$ *and a k-CCO C, there exists a* $(k-1)$*-CCO* C' *such that* $C' \supseteq C$.

Inspired by the lemma above, we propose a bottom-up algorithm, namely BU-MCCO. More specifically, we iteratively compute the *k-CCO*s based on computed $(k-1)$-*CCO*s when increasing k. The detailed pseudocode is given in Algorithm 3. We first compute all 1-*CCO*s in G (line 2). Then we iteratively increase k (line 6), and compute *k-CCO*s in $\mathbb{C}$, where $\mathbb{C}$ is the set of the previous computed $(k-1)$-*CCO*s (line 9). The algorithm terminates once no any *k-CCO* is found. The time complexity of Algorithm 3 is given as follows.

Theorem 2 *Given an input dual graph* $G(V, E_a, E_b)$, *the time complexity of Algorithm 3 is* $O(\mathcal{D}(G_b) \times h|E_b|)$.

Proof KCCO costs $O(h|E_b|)$ time in line 2. The number of iterations in line 4 is at most $\mathcal{D}(G)$. Since there does not exist any overlap between any two components in $\mathbb{C}$ (line 8), the time complexity from line 4 to line 9 is $O(\mathcal{D}(G_b) \times h|E_b|)$. The total time complexity of Algorithm 4 is obtained. □

4.2 A Top-Down Approach

A bottom-up solution is given in the previous section. Given that k_{max} may be very large, the time consuming in BU-MCCO may be very large. To handle this problem, we propose a top-down algorithm, namely TD-MCCO, in this section.

Given a dual graph G, computing all *MCCO*s is equivalent to computing all $k_{max}(G)$-*CCO*s. We adopt a top-down strategy to select the k_{max}. An upper bound for $k_{max}(G)$ can be easily obtained according to Lemma 3:

Lemma 6 *Given a dual graph* $G(V, E_a, E_b)$, $k_{max}(G) \leq \mathcal{D}(G_b)$.

```
Algorithm 4 TD-MCCO
Input: A graph $G(V, E_a, E_b)$
Output: The set containing all $MCCO$s in $G$
1: $\mathbb{C} \leftarrow \emptyset$;
2: Core-Decomposition($G_b$);
3: $k_{max} \leftarrow \max_{u \in V}(core(u))$;
4: while $\mathbb{C} = \emptyset$ and $k_{max} > 0$ do
5:     $C \leftarrow \{u \in V | core(u) \geq k_{max}\}$
6:     $\mathbb{C} \leftarrow$ KCCO($G[C], k_{max}$);
7:     $k_{max} \leftarrow k_{max} - 1$;
8: return $\mathbb{C}$;
```

Based on Lemma 6, we propose the algorithm TD-MCCO in Algorithm 4. Core-Decomposition is invoked in line 2, and we initialize k_{max} by the graph degeneracy in line 3. For each k_{max}, the vertex set of k-*core*s of G_b is obtained in line 5 based on Lemma 3. KCCO is invoked to compute all k_{max} -*CCO*s in $G[C]$ (line 6). We terminate the algorithm if any k_{max}-*CCO* is found.

Theorem 3 *Given an input dual graph* $G(V, E_a, E_b)$, *the time complexity of Algorithm 4 is* $O(\mathcal{D}(G_b) \times h|E_b|)$.

Proof The proof is similar to that for Theorem 2 and is omitted here. □

4.3 Binary Searching *MCCOs*

We propose BU-MCCO and TD-MCCO in Sects. 4.1 and 4.2, respectively. Even though they can successfully compute all *MCCO*s in the given dual graph G, both of them endure $\mathcal{D}(G_b)$ times of KCCO invocation in the time complexity. To conquer this drawback, we propose a binary search algorithm, namely BIN-MCCO, in this section. Similar to the conventional binary search, we maintain a lower bound $\underline{k}$ and an upper bound $\overline{k}$ of k, and attempt to find all k-*CCO*s in each iteration, where $k = \lfloor(\overline{k} + \underline{k})/2\rfloor$. If no any k-*CCO* is found, we know there does not exist any k'-*CCO* for $k < k' < \overline{k}$ according to Lemma 5. In this case, we assign the upper bound by k, and continue the search. Otherwise, we assign the lower bound by k. The procedure terminates once we find all k-*CCO*s and $(k+1)$-*CCO* does not exist. The initial lower bound for k is assigned by 1, and the upper bound is assigned by $\mathcal{D}(G_b)$ based on Lemma 6. The detailed pseudocode of BIN-MCCO is given in Algorithm 5.

```
Algorithm 5 BIN-MCCO
Input: A graph $G(V, E_a, E_b)$
Output: The set containing all $MCCO$s in $G$
1: Core-Decomposition($G_b$);
2: $d \leftarrow \max_{u \in V}(core(u)) + 1$;
3: $\mathbb{C} \leftarrow$ KCCO($G, 1$);
4: return BIN-Search($\mathbb{C}, 1, d$);

Procedure BIN-Search($\mathbb{C}, \underline{k}, \overline{k}$)
5: if $\overline{k} - \underline{k} \leq 1$ then
6:     return $\mathbb{C}$;
7: $k \leftarrow \underline{k} + (\overline{k} - \underline{k})/2$;
8: $\mathbb{T} \leftarrow \emptyset$;
9: for each $G[C] \in \mathbb{C}$ do
10:     $S \leftarrow \{u \in C | core(u) \geq k\}$;
11:     $\mathbb{T} \leftarrow \mathbb{T} \cup$ KCCO($G[S], k$);
12: if $\mathbb{T} = \emptyset$ then
13:     return BIN-Search($\mathbb{C}, \underline{k}, k$);
14: else
15:     return BIN-Search($\mathbb{T}, k, \overline{k}$);
```

The core numbers for each vertex in G_b are first computed in line 1. In line 2, we set the upper bound d by $\mathcal{D}(G) + 1$. This guarantees no any d-*CCO* exists. The subroutine BIN-Search is invoked recursively to find the *MCCO*s (line 4). The first parameter of BIN-Search is the set of all $\underline{k}$-*CCO*s in G. Recall that a k-*CCO* must be contained in a $(k-1)$-*CCO* according to Lemma 5. An optimization here is that we maintain all $\underline{k}$-*CCO*s in BIN-Search. Instead of computing k-*CCO*s in the original graph, we compute k-*CCO*s in a smaller graph induced by $\underline{k}$-*CCO*s (line 9), and never lose any result. For each induced subgraph of $\underline{k}$-*CCO* (line 9), we prune all vertices whose core number is less than k in line 10 based on Lemma 3. Then KCCO is invoked to compute all k-*CCO*s. If there does not exist any k-*CCO*s (line 12), we decrease $\overline{k}$ to k and continue searching (line 13). Otherwise, we increase $\underline{k}$ to k, and change the first parameter to the set of all k-*CCO*s (line 15).

Theorem 4 *Given an input dual graph* $G(V, E_a, E_b)$, *the time complexity of Algorithm 5 is* $O(\log \mathcal{D}(G_b) \times h|E_b|)$.

Proof Given that there does not exist any between different $G[C]$s (line 9), the time complexity of line 9 to line 11 is $O(h|E_b|)$. Given the upper bound $\mathcal{D}(G_b)$, the total invocation of BIN-Search is bounded by $O(\log \mathcal{D}(G_b))$, and the total time complexity of Algorithm 5 is $O(\log \mathcal{D}(G_b) \times h|E_b|)$. □

5 Efficient K-Connected Cores Search

In this section, we study the k-connected core search problem. That is computing a k-connected core containing a set of given vertices. Note that for the ease of presentation, we mainly study the problem for only one query vertex, while we will show that the solution can be naturally extended to handle the case of over one query vertices. The problem is formally defined as follows.

Problem 3 Given a dual network $G(V, E_a, E_b)$, an integer k and an vertex $u \in V$, computing a *k-CCO* in G that contains u.

5.1 Online Search Algorithm

To solve the Problem 3, we can naturally extend the Algorithm 2 and get an online search algorithm, namely CCO-Online. Specifically, we compute all *k-CCO*s in the dual graph and return the one that contains the query vertex. Note that we can directly return an empty set if the query vertex is removed due to the degree constraint (line 7 of Algorithm 2), as there is no *k-CCO* containing the query vertex. The search algorithm has the same time complexity as Algorithm 2. The detailed pseudocode of CCO-Online is given in Algorithm 6.

For the query with several vertices, we only need to check whether there is a *k-CCO* containing all query vertices.

Algorithm 6 Online Query Processing Algorithm (CCO-Online)

```
Input: A graph $G(V, E_a, E_b), k, v$
Output: $k$-$CCO$ containing $v$
1: $\mathbb{C}$ = KCCO($\mathbb{C}, k$);
2: if $v \notin \mathbb{C}$ then
3:     return $\emptyset$;
4: return $\mathbb{C}$;
```

5.2 Index-Based Search Algorithm

Even though the online algorithm successfully computes the *k-CCO* containing the query vertex, there are several drawbacks. First, the algorithm is inefficient. The result *k-CCO* is normally much smaller than the original graph, while the online algorithm still need to scan the whole graph at least to get the result. Additionally, in certain special case, we cannot find the result *k-CCO* for a query vertex, and the algorithm may be still costly in computing the *k-CCO*s. To address these drawbacks, we propose an index-based search algorithm in this section.

5.2.1 The Index Structure

Before giving the detailed index structure, we first give the following observation based on Lemma 5.

Observation 1 *Given a dual network $G(V, E_a, E_b)$, and a vertex $v \in V$, if $G[C]$ is a k-CCO and $v \in C$, there exists a $(k-1)$-CCO $C' \supseteq C$ such that $v \in C'$, where $1 < k \leq k_{max}$.*

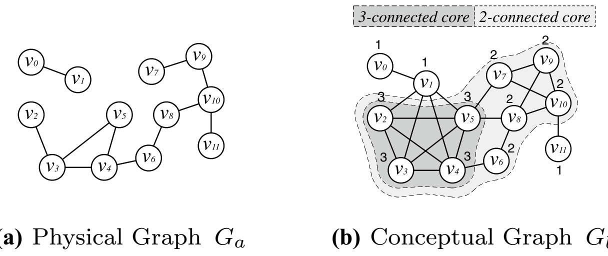


Fig. 4 An example of *CCO-Index*

According to the above observation, we save the maximum k for each vertex u that will be in a *k-CCO*. We call such a value the *connected-core number* and it is formally defined as follows.

Definition 7 *(Connected Core Number)* Given a dual graph $G(V, E_a, E_b)$ and a vertex $v \in V$, The *connected-core number* of v, denoted by $ccn(v)$, is the maximal number of k such that v is contained in the *k-CCO*.

We have the following lemma based on Definition 7.

Lemma 7 *Given a dual graph $G(V, E_a, E_b)$ and a vertex $v \in V$, there exists a k-CCO $G[S]$, where $1 \leq k \leq ccn(v)$ and $S \subseteq V$, such that $v \in S$.*

We compute the *connected-core number* for all vertices as the proposed index, named ***C**onnected **CO**re-Index* (*CCO-Index*). An example is given as follows.

Example 2 We give an example of *CCO-Index* in Fig. 4. The *connected-core number* is shown over each vertex. The *MCCO* of *G* is 3-*CCO* containing v_2, v_3, v_4 and v_5, and the *connected-core number*s of these nodes are 3. For nodes v_6, v_7, v_8, v_9 and v_{10}, they are not contained in 3-*CCO* and the maximum *k-CCO* containing them is 2-*CCO*, thus the *connected-core number*s of them are 2. Similarly, the *connected-core number*s of v_0, v_1 and v_{11} are 1.

Theorem 5 *Given a dual graph $G(V, E_a, E_b)$, the space complexity of CCO-Index is $O(V)$.*

Proof The proof is omitted here. □

5.2.2 The Query-Processing Algorithm

Based on *CCO-Index*, we propose a query-processing algorithm. The idea of this algorithm is based on the following theorem:

Theorem 6 *Given a dual network $G(V, E_a, E_b)$, a vertex $u \in V$ and an integer $1 \leq k \leq ccn(u)$, $G[A_k(u)]$ is a k-CCO,* *where $A_k(u)$ is the set of vertices in which each vertex v can be reached from u via a path in G_b that every vertex w in the path satisfies $ccn(w) \geq k$.*

Proof Firstly, we proof $G_b[A_k(u)]$ is a *k-core*. We define the degree of vertex $v' \in A_k(u)$ in $G_b[A_k(u)]$, which denoted by $deg_{G_b[A_k(u)]}(v') = k'$ as the minimum degree in $G_b[A_k(u)]$. Suppose that $G_b[A_k(u)]$ is not a *k-core*, we have $k < k'$, and $G_b[A_k(u)]$ is a *k'-CCO*. Obviously, $G_a[A_k(u)]$ is connected and $G[A_k(u)]$ is maximal. Therefore, $G[A_k(u)]$ is a *k'-CCO*, where $k' < k$.

Given that $1 \leq k \leq ccn(u)$, based on Lemma 7, there exists $S \subseteq V$ such that $G[S]$ is a *k-CCO* and $u \in S$. In addition, according to Lemma 5, $S \supseteq A_k(u)$. Thus, $\forall v \in S, deg_S(v) \geq k$. Obviously, $v' \notin S$ because $deg_{G_b[A_k(u)]}(v') = k' < k$. Thus, the maximal-connected core containing v' is *k'-CCO*. However, this contradicts the precondition that $ccn(v') \geq k$. Therefore, $G_b[A_k(u)]$ is a *k-core*. Obviously, $G_b[A_k(u)]$ is connected and for any $\forall v \in A_k(u)$, $deg_{G_b[A_k(u)]}(v) \geq k$.

Given the definition of $A_k(u)$, it is easy to know $G_a[A_k(u)]$ is connected.

$G[A_k(u)]$ is maximal, because once we add a vertex v into $G[A_k(u)]$, either $G_b[A_k(u) \cup \{v\}]$ is disconnected, or $ccn(V) < k$, i.e., there is no *k-CCO* containing v. Therefore, $G[A_k(u)]$ is a maximal *k-CCO*. According to Definition 4, we have that $G[A_k(u)]$ is a *k-CCO*. □

Based on the Theorem 6, we propose an algorithm for *k-CCO* search. The pseudocode is given in Algorithm 7.

In line 2, the algorithm firstly checks whether $ccn(v)$ lower than k. If it is, the algorithm terminates. Otherwise, in line 6, the algorithm recursively invokes a depth-first search subroutine CCO-DFS to find the vertex set $\mathbb{C}$. Each vertex v in $\mathbb{C}$ can be reached from u via a path in G_b, and every vertex w in the path satisfies $ccn(w) \geq k$ (line 8–9). The first parameter of CCO-DFS is the set of vertices in the *k-CCO* containing v. We terminate the algorithm if all vertices in *k-CCO* are found. The correctness of CCO-Query can be guaranteed by Theorem 6.

Algorithm 7 Query Processing Algorithm (CCO-Query)

```
Input: A graph $G(V, E_a, E_b), k, v$
Output: $k$-$CCO$ containing $v$
1: $\mathbb{C} \leftarrow \emptyset$;
2: if $ccn(v) < k$ then
3:     return $\mathbb{C}$;
4: else
5:     $\mathbb{C} \leftarrow$ CCO-DFS($\mathbb{C}, v, k$);
6: return $\mathbb{C}$;

Procedure CCO-DFS($\mathbb{C}, v, k$)
7: $\mathbb{C} \leftarrow \{u\} \cup \mathbb{C}$;
8: if $\exists u \in N_{G_a}(v), ccn(u) \geq k$ then
9:     $\mathbb{C} \leftarrow$ CCO-DFS($\mathbb{C}, u, k$);
10: return $\mathbb{C}$;
```

Example 3 We give an example of CCO-Query. Consider the graph G in Fig. 2. Given $k = 3$ and the query vertex v_3, the *CCO-Index* of G is shown in Fig. 4. CCO-Query depth-first searches the physical graph with v_3 as the source. $\{v_2, v_4\}$ are the vertices whose *connected-core number* is greater than or equal to 3 in $N_{G_b}(v_3)$. Suppose CCO-Query first visits v_2, and v_5 is the vertex which *connected-core number* is greater than or equal to 3 in $N_{G_b}(v_2)$. CCO-Query expands to v_5 and finds no vertex to be expanded. CCO-Query next visits v_4, and the only qualified neighbor v_5 has been visited. Then the depth-first search terminates. Finally, we get the result vertex set $\{v_2, v_3, v_4, v_5\}$.

Theorem 7 *Given a graph $G(V, E_a, E_b)$, a vertex $v \in V$ and an integer k, the time complexity of Algorithm 7 is $O(m_b)$, where m_b is the number of physical edges in the result subgraph.*

Proof The theorem is obvious and the proof is omitted here. □

5.3 The Index Construction Algorithm

In order to construct *CCO-Index*, we propose a peeling algorithm as follows. The detailed pseudocode is given in Algorithm 8. The idea of Algorithm 8 is similar to that of Algorithm 3. If $G_b[C]$ is connected and satisfies the degree constraint (line 3–4), we iteratively increase k, and recursively invoke Algorithm 8 to label the node v in C (line 4). Otherwise, the algorithm from line 5 to line 6 computes a *k-core* $G_b[H]$ of $G_b[C]$. All vertices that violate the degree constraint in $G_b[C]$ are iteratively removed from C and labeled as $ccn(v) = k - 1$(line 7). For each connected component $G_b[H]$, we recursively invoke Algorithm 8 to label nodes in $G_b[H]$ (line 9). The algorithm terminates once all nodes are labeled.

Algorithm 8 The Index Construction Algorithm (CCO-Construct)

```
Input: A graph $G(V, E_a, E_b), k$
Output: The $CCO$-$Index$ $\mathbb{N}$
1: $\mathbb{N} \leftarrow \emptyset$;
2: for each connected component $G_a[C]$ in $G_a$ do
3:     if $\forall u \in C, deg_{G_b[C]}(u) \geq k$ and $G_b[C]$ is connected then
4:         CCO-Construct($G_b[H], k+1$);
5:     else
6:         while $\exists u \in C, deg_{G_b[C]}(u) < k$ do
7:             $C \leftarrow C - \{u\}$;
8:             $ccn(u) = k - 1$;
9:             $\mathbb{N} \leftarrow ccn(u)$;
10:        for each connected component $G_b[H]$ in $G_b[C]$ do
11:            CCO-Construct($G_b[H], k$);
12: return $\mathbb{N}$;
```

The time complexity of Algorithm 8 is given as follows.

Theorem 8 *Given an input dual graph $G(V, E_a, E_b)$. Let h be the height of the DFS tree of Algorithm 8 and $\mathcal{D}$ be the degeneracy of G_b, the time complexity of Algorithm 8 is $O(\mathcal{D}(G_b) \times h|E_b|)$.*

Proof The proof is similar to that for Algorithm 3 and is omitted here. □

6 Experiments

We conduct extensive experiments to evaluate the performance of our proposed solutions. We obtain the code for DCS from the author as a comparison. All other algorithms are implemented in C++. All the experiments are conducted on a Windows Server operating system running on a machine with an Intel Xeon 2.0 GHz CPU, 32 GB 1333 MHz DDR3-RAM. The time cost for algorithms is measured as the amount of wall-clock time elapsed during the program execution.

Real-World Datasets We evaluate the algorithms on six real graphs. The detailed statistics of these graphs are summarized in Table 1. $\overline{d}_b$ is the average degree in the conceptual graph.

We adopt a similar idea in [33] to construct the dual graphs. DBLP[30] is constructed based on the computer science bibliography *DBLP*. We select several conferences and journals in database research area. The vertices represent the authors of the published papers. An edge exists if two authors have a common paper in the physical graph, and edges in the conceptual graph are constructed by measuring the similarity between the abstracts of papers published by any two authors. Hep-TH[18] is a theory collaboration network in high energy physics area. The construction for Hep-THis same as that for DBLP.

Epinions[23] and CiaoDVD[1] are recommendation networks. Each vertex represents a user. A physical edge exists if a user expresses a positive trust statement on the other user. To construct the conceptual graph, we calculate the correlation coefficient[22] of the common ratings between users, and connected two users by a conceptual edge if their coefficient value is larger than a threshold.

Brightkite [8] and Gowalla [8] are geosocial networks. Each vertex represents a user. The physical edges represent the friend relationship between users, and the conceptual edges are constructed based on the Euclidean distance between the locations of users.

Synthetic Datasets We adopt the same method in [33] to generate several synthetic graphs. In specific, we use the graph generator *GTgraph*[2] to construct both physical graphs and conceptual graphs. The statistics of generated graphs are summarized in Table 2.

Table 1 Statistics of real-world datasets

Datasets	$\|V\|$	$\|E_a\|$	$\|E_b\|$	$\|E_b\|/\|E_a\|$	$\overline{d}_b$
DBLP	40,490	203,670	400,448	1.97	9.89
Hep-TH	29,381	352,807	886,791	2.51	30.18
Epinions	49,290	487,002	729,403	1.50	14.80
CiaoDVD	14,811	40,133	124,533	3.10	8.41
Brightkite	58,228	214,078	602,836	2.82	10.35
Gowalla	196,591	950,327	1,458,456	1.53	7.42

Table 2 Statistics of synthetic datasets

	GT1	GT2	GT3	GT4
$\|V\|$	1×2^{20}	2×2^{20}	4×2^{20}	6×2^{20}
$\|E_a\|, \|E_b\|$	1×10^7	2×10^7	4×10^7	8×10^7

Algorithms The experiment involved 1 algorithm for computing all *k-CCO*s, 3 algorithms for computing *MCCO*s, 2 algorithms for searching *k-CCO* and 1 algorithm for index construction. Algorithms that appeared in the experiments are summarized as follows:

- KCCO: The algorithm that computes all *k-CCO*s.
- BU-MCCO: The bottom-up algorithm that computes *MCCO*s.
- TD-MCCO: The top-down algorithm that computes *MCCO*s.
- BIN-MCCO: The algorithm that binary-searches *MCCO* s.
- CCO-Query: The optimized algorithm that searches *k-CCO*s.
- CCO-Online: The naive algorithm that searches *k-CCO*s.
- CCO-Construct: The algorithm that constructs our proposed index *CCO-Index* in Sect. 5.

6.1 Performance Studies of the Algorithm for Computing all *k-CCO*s

Eval-I: Evaluating the algorithm for computing all k-CCO s. The time consuming for algorithm KCCO on six real-world graphs is reported in Fig. 5. For each dataset, we select $20\% \times k_{max}$, $40\% \times k_{max}$, $60\% \times k_{max}$, $80\% \times k_{max}$ and k_{max} as the input integer k, and present a line chart. We can find that the time cost of KCCO decreases when increasing k. This is

[1] https://www.librec.net/datasets.html.

[2] http://www.cse.psu.edu/~kxm85/software/GTgraph/.

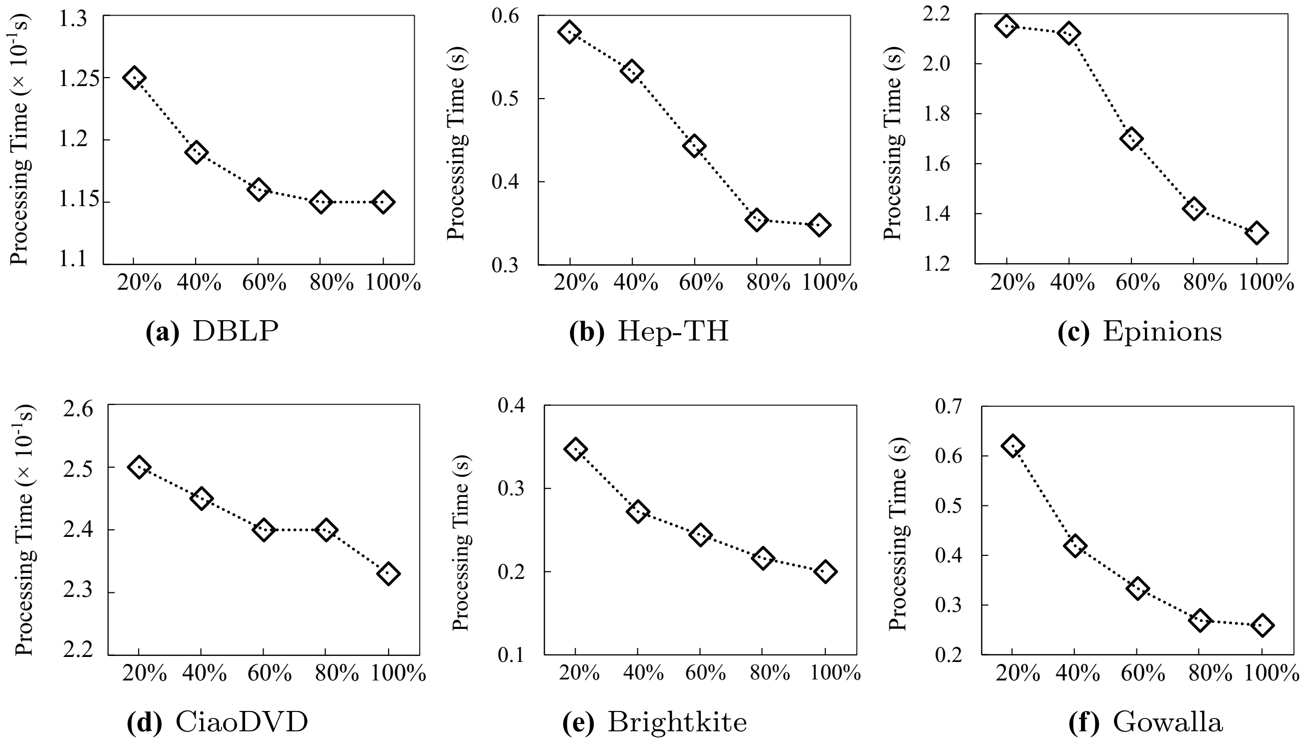


Fig. 5 Computing *k-CCO*s in real-world graphs

mainly because a large number of vertices are removed when the degree constraint k is large, and the result subgraph is small.

6.2 Performance Studies of the Algorithm for Computing *MCCO*s

Eval-II: Evaluating the algorithms on real-world graphs The time consuming for algorithms BU-MCCO, TD-MCCO and BIN-MCCO on six real-world graphs is reported in Fig. 6a. Given that there exists no previous work on this problem, we give the time cost for computing *DCS* [33], namely DCS, as a comparison in the figure. Note that the time cost for DCS is not given in some datasets, since the procedure cannot terminate in 4 hours.

As we can see from the figure, BIN-MCCO is the fastest algorithm. It costs about 13 s in Gowallaand $<$ 4 s in all other datasets. TD-MCCO is the second fastest algorithm in all datasets, while BU-MCCO is slightly slower than TD-MCCO. For example, in Brightkite, TD-MCCO and BU-MCCO cost about 77 s and 113 s, respectively. BIN-MCCO costs about 2 s, which is almost two orders of magnitude faster than TD-MCCO and BU-MCCO. As a comparison, DCS costs over 3000 s and 750 s in DBLP and CiaoDVD, respectively, while BIN-MCCO costs only about 1.3 s and 0.7 s, respectively, in those two datasets. The result demonstrates the high efficiency of BIN-MCCO.

Eval-III: Evaluating the algorithms on synthetic graphs The running time for computing *MCCO*s in synthetic graphs is given in Fig. 6b. BIN-MCCO is the fastest algorithm on all graph size. BU-MCCO has a slower increasing rate than TD-MCCO, and is even faster than TD-MCCO finally. This is mainly because the gap between k_{max} and $\mathcal{D}$ is large given a big graph size.

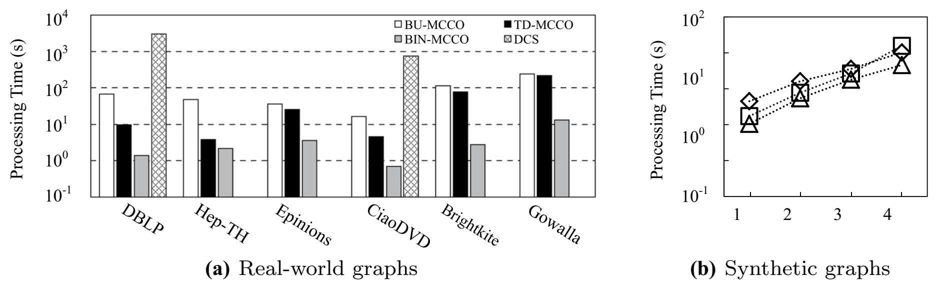


Fig. 6 Computing *MCCO*s in real-world graphs and synthetic graphs

6.3 Scalability Testing of the Algorithm for Computing *MCCOs*

We test the scalability of our proposed algorithms in this section. For each real-world dual graph, we randomly sample physical edges, conceptual edges and vertices, respectively, from 20 to 100%. When sampling physical edges, we get the incident vertices of the edges as the vertex set, and preserve the induced subgraph of this vertex set in the conceptual graph. The sampling strategy for conceptual edges is same as that for physical edges. When sampling vertices, we get the induced dual subgraph of the sampled vertices. Due to the space limitation, we only report the charts for DBLP, Epinions and Brightkite, while the results in other datasets show the similar trends.

Eval-IV: Sampling physical edges The running time of BU-MCCO, TD-MCCO and BIN-MCCO is reported in Fig. 7a–c when sampling physical edges. We can see that BIN-MCCO is the fastest, and the time cost of all algorithms performs a slightly downward trend in all datasets. This is mainly due to the speedup of performing KCCO. In specific, when the physical edge size is large, a *k-core* in the conceptual graph is more likely to be connected in the physical graph, which means the depth of the invocation tree depicted in Fig. 3 is small.

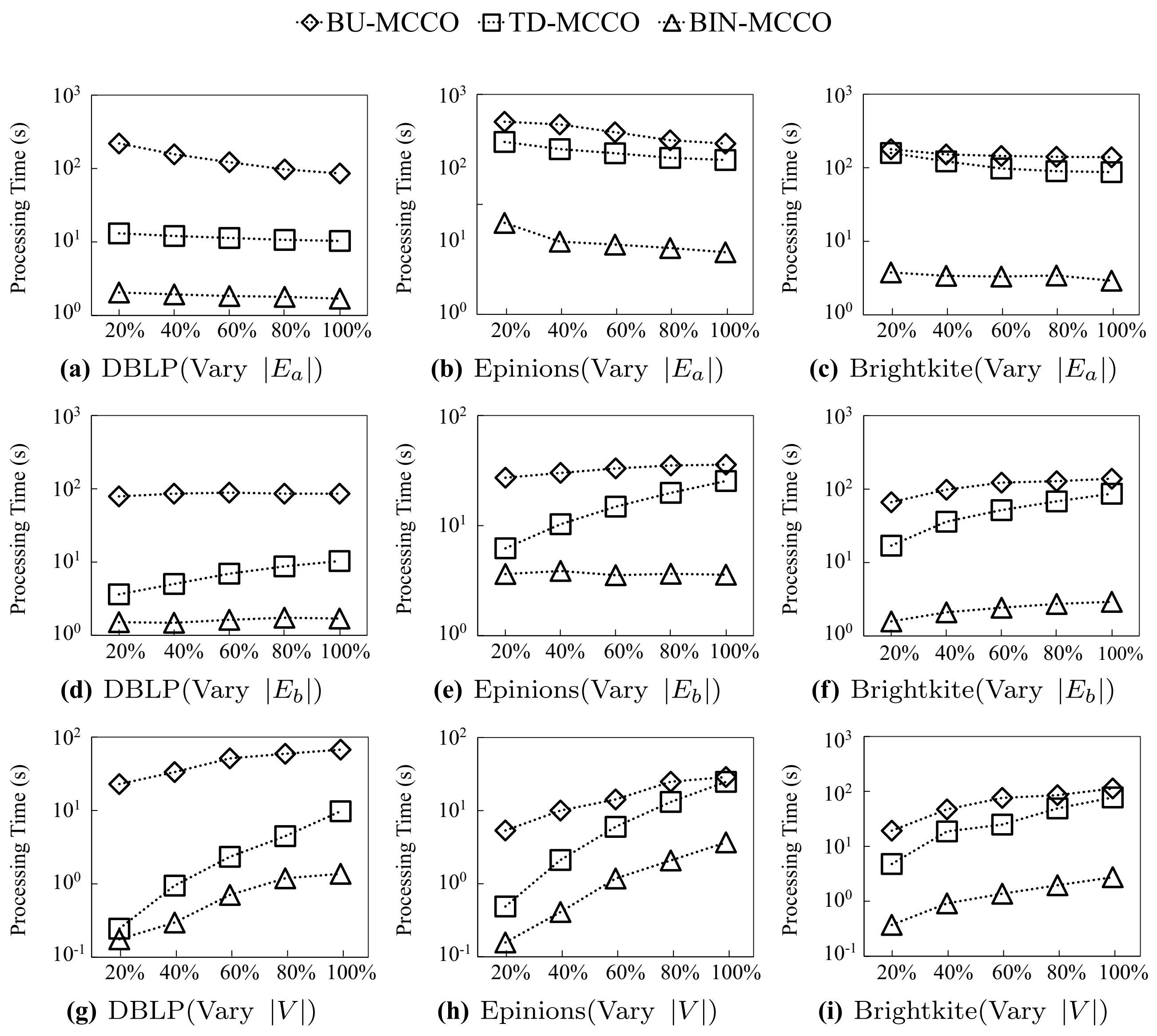


Fig. 7 Scalability testing

Eval-V: Sampling conceptual edges The running time of BU-MCCO, TD-MCCO and BIN-MCCO is reported in Fig. 7d–f when sampling conceptual edges. BIN-MCCO is the fastest algorithm, and the lines for BIN-MCCO in all datasets are stable. TD-MCCO is the second fastest algorithm. The time cost of TD-MCCO presents a relatively obvious increase from 20 to 100% in all datasets, and the gap between TD-MCCO and BU-MCCO decreases when edge size increases. This is mainly because the graph degeneracy $\mathcal{D}$ of G_b increases when increasing $|E_b|$, and the gap between $\mathcal{D}$ and k_{max} increases. Therefore, more iterations in TD-MCCO are performed, and the efficiency of TD-MCCO declines.

Eval-VI: Sampling vertices The running time of BU-MCCO, TD-MCCO and BIN-MCCO is reported in Fig. 7g–i when sampling vertices. We can see that BIN-MCCO is still the fastest in all scenarios. The chart for BIN-MCCO presents a slight increase when increasing the vertex size. TD-MCCO is faster than BU-MCCO, and in some datasets, the gap between them decreases when increasing vertex size. For example, in Epinions, TD-MCCO costs about 0.5 s on 20% and reaches about 25 s on 100%. By contrast, BU-MCCO costs about 5.3 s on 20% and reaches about 29 s on 100%. The main reason is similar to that in sampling conceptual edges. From the three scalability experiments, we can see that high efficiency and stability of BIN-MCCO. The top-down solution TD-MCCO is the second fastest, while the efficiency of TD-MCCO highly depends on the graph structure, and the gap between k_{max} and $\mathcal{D}$. The bottom-up solution BU-MCCO is the slowest but performs more stable than TD-MCCO.

6.4 Performance Studies of the Search Algorithm

Eval-VII: Query processing (Vary k) We vary k from 5 to 25 and evaluate the efficiency of our two proposed subgraph search algorithms (CCO-Online, CCO-Query). For each dataset, we select 5, 10, 15, 20 and 25 as the input integer k, and present a line chart. To assess query performance, we randomly generate 1000 queries in each dataset (each query contains one query vertex) and compute the average query time. Due to the space limitation, we only report the charts for three real-world graphs (DBLP, Hep-TH, and Brightkite) and three synthetic graphs, while the results in other datasets show the similar trends.

The results are reported in Fig. 8. We can see that CCO-Query is around two orders of magnitude faster than CCO-Online. When k increases, the change for CCO-Online is not obvious in all datasets, while the processing time of CCO-Query performs a downward trend in all graphs. This is because the running time of CCO-Query is only related to the size of result subgraph, which becomes small when k is large.

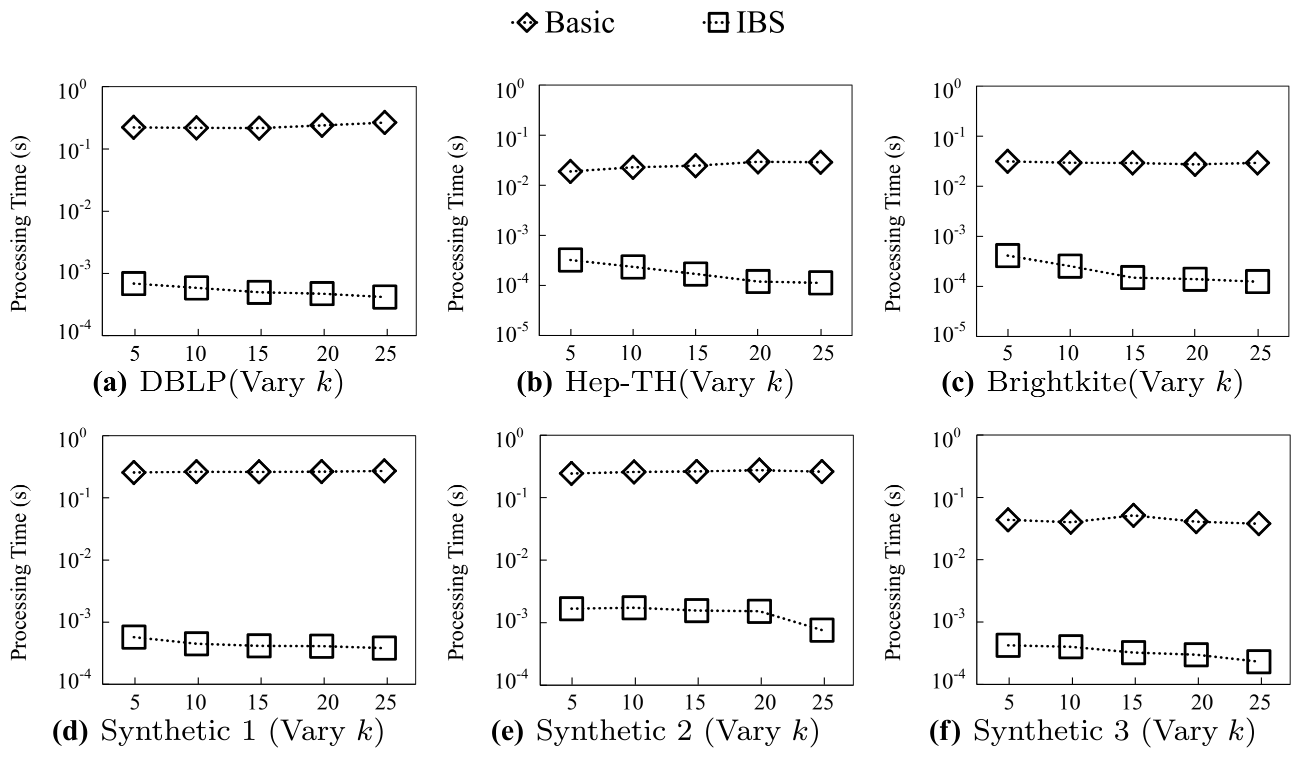


Fig. 8 Query Processing (Vary k)

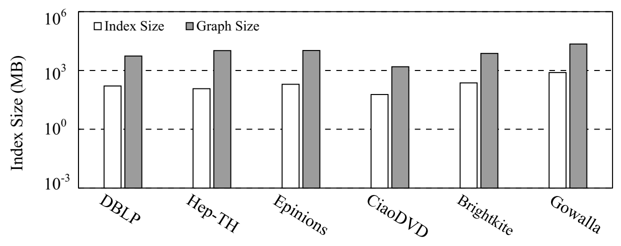


Fig. 9 Index size

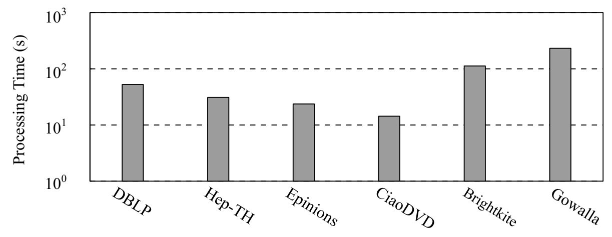


Fig. 10 Index time

Eval-VIII: Index size We report the size of the our proposed index for all datasets. The size of original graph is also given as a comparison. The results are depicted in Fig. 9. Over all the datasets, the sizes of *CCO-Index* is equal to the size of the vertices and much less than the size of the original graph. For example, the index size of *Gowalla* is about 0.8 GB, while the size of the whole graph is > 22 GB.

Eval-IX: Index construction We report the running time of index construction algorithm for all datasets. The result is shown in Fig. 10. CCO-Construct takes about 231 s in the *Gowalla* dataset, and takes about only 14 s in *CiaoDVD* dataset.

6.5 Effectiveness Evaluation

Eval-X: Case study in Gowalla We conduct a case study to present the effectiveness of our solution. Due to the space limitation, we select a subgraph of Gowalla, and compute the *MCCO* in this subgraph. The result is reported in Fig. 11b. As a comparison, we also give the result of *DCS* in the same subgraph in Fig. 11a. We can see that there exist several vertices whose degree less than three in the *DCS*. This demonstrates the approximate solution for *DCS* may generate a result with a sparse subgraph. By contrast, the degree of each vertex is not less than k_{max} in Fig. 11b, and the result of *MCCO* is cohesive.

7 Related Works

Computing k-core k-core is first introduced in [3, 29] proposes a linear time solution for core decomposition. *k-core* in directed graph and weighted graphs is studied in [14, 15], respectively. [7] proposes a partition-based external memory algorithm for computing *k-cores*. [17, 32] apply a semi-external model and further speed up the core decomposition algorithm for big graphs. [25] gives a distributed algorithm for core decomposition. Given that real-world graphs are highly dynamic, core number maintenance is studied in [20, 28]. Locally estimating core number is studied in [26].

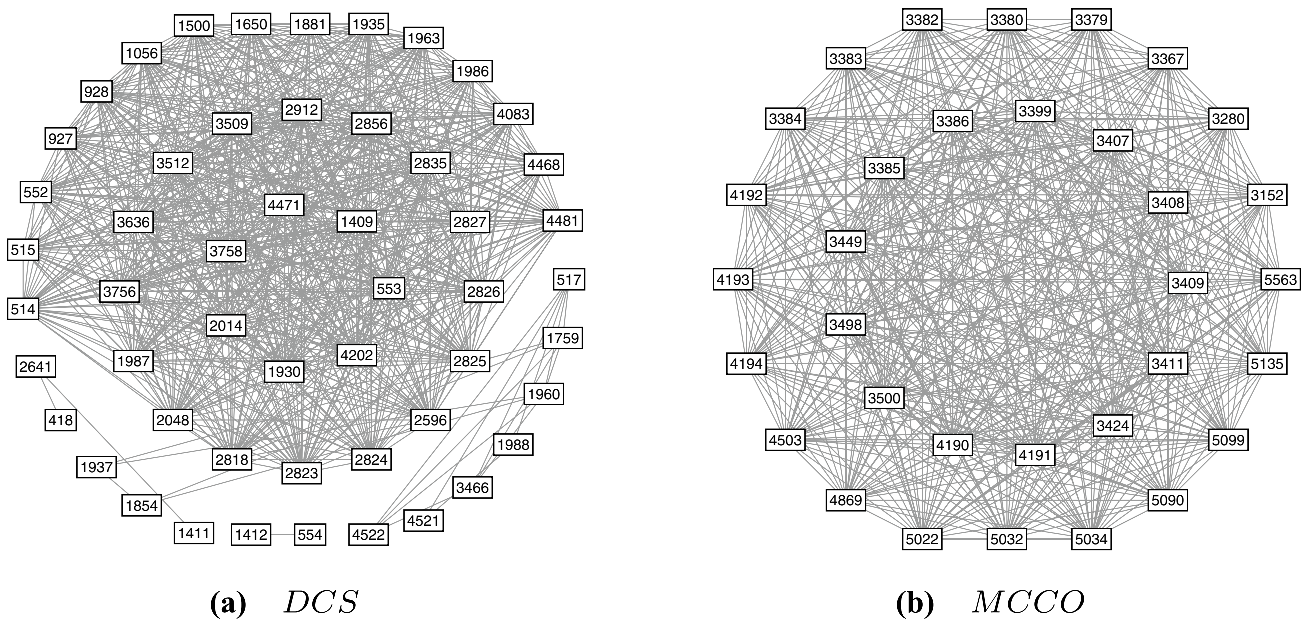


Fig. 11 The *DCS* and *MCCO* in a conceptual subgraph of Gowalla

Several work studies *k-core* in different graph models, such as uncertain graphs [4], random graphs [16, 21, 24, 27], and attribute graphs [12]. [11, 19] use *k-core* to detect communities in the graph.

Cohesive subgraph detection in dual networks [33] studies the cohesive subgraph problem in dual networks. An approximate algorithm is proposed for computing the densest connected subgraph in the input dual graph.

8 Conclusion

Computing *k-core*s is a fundamental and important graph problem. In this paper, we define the k-connected core in dual graphs. A subgraph g is a k-connected core if the minimum degree of g is at least k in the conceptual graph, and g is connected in both conceptual graph and physical graph. We propose a polynomial time algorithm for computing all k-connected cores in the dual graph. We also propose three algorithms for computing all maximum-connected cores, which are the maximum k-connected cores such that a $(k + 1)$ -connected core does not exist. Given a set of query vertices, we study the *k-CCO* search problem and propose an index-based solution. We do extensive experiments to demonstrate the effectiveness and efficiency of our propose algorithms.

Acknowledgements The work is supported by the National Key R&D Program (Nos. 2017YFB1400102, 2016YFB1000602), NSFC (No. 61572295), and SDNSF (No. ZR2017ZB0420).

References

1. Alvarez-Hamelin JI, Dall'Asta L, Barrat A, Vespignani A (2006) Large scale networks fingerprinting and visualization using the k-core decomposition. In: NIPS, pp 41–50
2. Asahiro Y, Iwama K, Tamaki H, Tokuyama T (2000) Greedily finding a dense subgraph. J Algorithms 34(2):203–221
3. Batagelj V, Zaversnik M (2003) An o(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049
4. Bonchi F, Gullo F, Kaltenbrunner A, Volkovich Y (2014) Core decomposition of uncertain graphs. In: KDD, pp 1316–1325
5. Chang L, Yu JX, Qin L, Lin X, Liu C, Liang W (2013) Efficiently computing k-edge connected components via graph decomposition. In: SIGMOD, pp 205–216
6. Charikar M (2000) Greedy approximation algorithms for finding dense components in a graph. In: APPROX, pp 84–95
7. Cheng J, Ke Y, Chu S, Tamer Özsu M (2011) Efficient core decomposition in massive networks. In: ICDE, pp 51–62
8. Cho E, Myers SA, Leskovec J (2011) Friendship and mobility: user movement in location-based social networks. In: KDD, pp 1082–1090
9. Conte A, Firmani D, Mordente C, Patrignani M, Torlone R (2017) Fast enumeration of large k-plexes. In: KDD, pp 115–124
10. da Fontoura Costa L, Oliveira ON Jr, Travieso G, Rodrigues FA, Villas Boas PR, Antiqueira L, Viana MP, Correa Rocha LE (2011) Analyzing and modeling real-world phenomena with complex networks: a survey of applications. Adv Phys 60(3):329–412
11. Cui W, Xiao Y, Wang H, Wang W (2014) Local search of communities in large graphs. In: SIGMOD, pp 991–1002
12. Fang Y, Cheng R, Luo S, Hu J (2016) Effective community search for large attributed graphs. PVLDB 9(12):1233–1244
13. Gallo G, Grigoriadis MD, Tarjan RE (1989) A fast parametric maximum flow algorithm and applications. SIAM J Comput 18(1):30–55
14. Giatsidis C, Thilikos DM, Vazirgiannis M (2011) D-cores: measuring collaboration of directed graphs based on degeneracy. In: ICDM, pp 201–210
15. Giatsidis C, Thilikos DM, Vazirgiannis M (2011) Evaluating cooperation in communities with the k-core structure. In: ASONAM, pp 87–93
16. Janson S, Luczak MJ (2007) A simple solution to the k-core problem. Random Struct Algorithms 30(1–2):50–62
17. Khaouid W, Barsky M, Srinivasan V, Thomo A (2015) K-core decomposition of large networks on a single pc. PVLDB 9(1):13–23
18. Leskovec J, Kleinberg J, Faloutsos C (2007) Graph evolution: densification and shrinking diameters. TKDD 1(1):2
19. Li R-H, Qin L, Yu JX, Mao R (2015) Influential community search in large networks. PVLDB 8(5):509–520
20. Li R-H, Yu JX, Mao R (2014) Efficient core maintenance in large dynamic graphs. TKDE 26(10):2453–2465
21. Łuczak T (1991) Size and connectivity of the k-core of a random graph. Discrete Math 91(1):61–68
22. Ma H, Zhou D, Liu C, Lyu MR, King I (2011) Recommender systems with social regularization. In: Proceedings of the fourth ACM international conference on Web search and data mining, pp 287–296
23. Massa P, Avesani P (2007) Trust-aware recommender systems. In: RecSys, pp 17–24
24. Molloy M (2005) Cores in random hypergraphs and boolean formulas. Random Struct Algorithms 27(1):124–135
25. Montresor A, De Pellegrini F, Miorandi D (2013) Distributed k-core decomposition. TPDS 24(2):288–300
26. OBrien MP, Sullivan BD (2014) Locally estimating core numbers. In: ICDM, pp 460–469
27. Pittel B, Spencer J, Wormald N (1996) Sudden emergence of a giantk-core in a random graph. J Combin Theory Ser B 67(1):111–151
28. Saríyüce AE, Gedik B, Jacques-Silva G, Wu K-L, Çatalyürek ÜV (2013) Streaming algorithms for k-core decomposition. PVLDB 6(6):433–444
29. Seidman SB (1983) Network structure and minimum degree. Soc Netw 5(3):269–287
30. Tang J, Zhang J, Yao L, Li J, Zhang L, Su Z (2008) Arnetminer: extraction and mining of academic social networks. In: KDD, pp 990–998
31. Wang J, Cheng J (2012) Truss decomposition in massive networks. PVLDB 5(9):812–823
32. Wen D, Qin L, Zhang Y, Lin X, Yu JX (2016) I/o efficient core graph decomposition at web scale. In: ICDE, pp 133–144. IEEE
33. Wu Y, Jin R, Zhu X, Zhang X (2015) Finding dense and connected subgraphs in dual networks. In: ICDE, pp 915–926
34. Yue L, Wen D, Cui L, Qin L, Zheng Y (2018) K-connected cores computation in large dual networks. In: Pei J, Manolopoulos Y, Sadiq S, Li J (eds) Database systems for advanced applications. Springer, Cham, pp 169–186