



# K-Connected Cores Computation in Large Dual Networks

Lingxi Yue<sup>1</sup>, Dong Wen<sup>2</sup>, Lizhen Cui<sup>1</sup>(✉), Lu Qin<sup>2</sup>, and Yongqing Zheng<sup>1</sup>

<sup>1</sup> School of Software Engineering, Shandong University, Jinan, China  
yuelingxi@mail.sdu.edu.cn, clz@sdu.edu.cn, zhengyongqing@dareway.com.cn

<sup>2</sup> Centre for Artificial Intelligence, University of Technology Sydney,  
Ultimo, Australia

dong.wen@student.uts.edu.au, lu.qin@uts.edu.au

**Abstract.** Computing *k*-cores 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*-CCOs. We also propose three algorithms for computing all *maximum-connected cores* (MCCO), which are the existing *k*-CCOs such that a  $(k + 1)$ -CCO does not exist. We conduct extensive experiments on six real-world datasets and several synthetic datasets. The experimental results demonstrate the effectiveness and efficiency of our proposed algorithms.

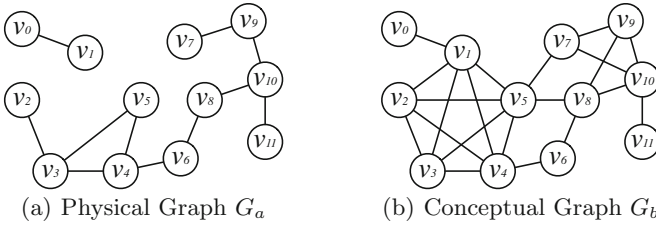
## 1 Introduction

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 towards 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*-cores 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. 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-Connected CORE* (*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: (i) the minimum degree of *g* is not less than *k* in the conceptual graph; (ii) *g* is connected in the conceptual graph; and (iii) *g* is connected in the physical graph.



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

An example of a dual graph is given in Fig. 1. Figure 1(a) is a physical graph, and Fig. 1(b) is 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 less than 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 problems in this paper. Given a dual graph *G* and an integer *k*, the first problem is computing all *k-CCOs* 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 *Maximum-Connected COREs* (*MCCOs*) 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*.

**Applications.** Computing  $k$ -CCOs and MCCOs 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  $k$ -CCOs. 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 MCCOs, let  $k_{max}$  be the maximum  $k$  in a dual graph such that a  $k$ -CCO exists. Given the solution for computing  $k$ -CCOs, the MCCOs can be obtained if  $k_{max}$  is known. Therefore, a main challenge in computing the MCCOs is computing  $k_{max}$ .

**Our Approaches and Contributions.** We propose a polynomial algorithm to compute all  $k$ -CCOs in dual graphs. It performs by recursively removing the vertex which violates the  $k$ -CCO definition. For the problem of computing MCCOs, we first follow the similar idea in computing all  $k$ -CCOs, and give a bottom-up solution. More specifically, we compute the MCCOs 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  $k$ -CCOs if exist. To further improve the algorithmic efficiency, we propose a binary search algorithm for computing all MCCOs. The experimental results show the excellent performance of our optimized algorithm. More details can be found in Sect. 5. 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  $k$ -CCOs 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.
- *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-CCOs*. Section 4 studies the problem of computing all *MCCOs*. Section 5 evaluates our proposed algorithms in extensive experiments. Section 6 introduces the related works, and Sect. 7 concludes the paper.

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

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  $core(w) \geq k$ . 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:      $\text{core}(u) \leftarrow k$ ;
6:     remove  $u$  and its incident edges from  $G'$ ;
7: return  $\text{core}(u)$  for all  $u \in V$ ;

```

---

Given the core numbers of all vertices, all  $k$ -cores 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 can be found in Fig. 1. Based on the aforementioned classic  $k$ -core concept, we define the  $k$ -*Connected COre* ( $k$ -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.

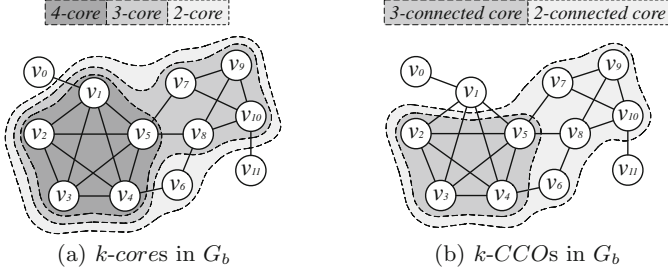
**Definition 5.** Given a dual graph  $G(V, E_a, E_b)$ , a dual subgraph  $G[C]$  is a maximum-connected core (MCCO) if  $G[C]$  is a  $k$ -CCO, and  $(k + 1)$ -CCO does not exist.

**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  $k$ -CCOs in  $G$ .

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



**Fig. 2.** An example of  $k$ -CCOs

*Example 1.* We give an example of  $k$ -CCO and  $MCCO$ . The  $k$ -CCOs 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. 2(a). 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  $G_a$ .

### 3 Computing K-Connected Cores

Given an integer  $k$ , we study the problem of computing all  $k$ -CCOs 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  $k$ -CCOs. The pseudocode is given in Algorithm 2.

---

#### Algorithm 2. Computing K-Connected COres (KCCO)

---

**Input:** A graph  $G(V, E_a, E_b)$ , and a parameter  $k$

**Output:** The set  $\mathbb{C}$  containing all  $k$ -CCOs 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:        $\mathbb{C} \leftarrow \mathbb{C} \cup \text{KCCO}(G_b[H], k)$ ;
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  $k$ -CCOs in  $G_b[H]$  (line 8–9). The correctness of this step is guaranteed by Lemma 3.

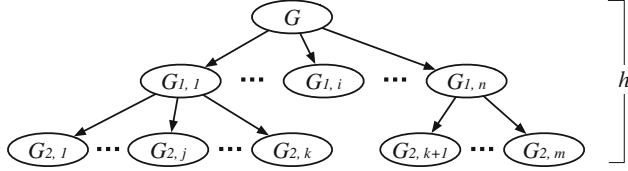


Fig. 3. DFS tree

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 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 can be found in Sect. 5.

## 4 Computing Maximal-Connected Cores

We study the problem of computing all *MCCOs* in this section. A straightforward bottom-up solution BU-MCCO is first given in Subsect. 4.1. Then, we propose a top-down solution TD-MCCO in Subsect. 4.2. To further improve the algorithmic efficiency, we give a binary search algorithm, namely BIN-MCCO, in Subsect. 4.3. Our experiments demonstrate BIN-MCCO outperforms TD-MCCO and BU-MCCO. The details can be found in Sect. 5.

### 4.1 A Bottom-Up Approach

We give a straightforward algorithm for computing all *MCCOs* 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-CCOs* based on computed  $(k - 1)$ -CCOs when increasing  $k$ . The detailed pseudocode is given in Algorithm 3. We first compute all 1-CCOs in  $G$  (line 2). Then we iteratively increase  $k$  (line 6), and compute *k-CCOs* in  $\mathbb{C}$ , where  $\mathbb{C}$  is the set of the previous computed  $(k - 1)$ -CCOs (line 9). The algorithm terminates once no any *k-CCO* is found. The time complexity of Algorithm 3 is given as follows.

---

#### Algorithm 3. BU-MCCO

---

**Input:** A graph  $G(V, E_a, E_b)$

**Output:** The set containing all *MCCOs* in  $G$

---

```

1:  $\mathbb{C} \leftarrow \emptyset$ ;
2:  $\mathbb{T} \leftarrow \text{KCCO}(G, 1)$ ;
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:      $\mathbb{T} \leftarrow \mathbb{T} \cup \text{KCCO}(G[C], k)$ ;
10: return  $\mathbb{C}$ ;
```

---



**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 MCCOs is equivalent to computing all  $k_{max}(G)$ -CCOs. 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  $G(V, E_a, E_b)$ ,  $k_{max}(G) \leq \mathcal{D}(G_b)$ .*

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  $k$ -cores of  $G_b$  is obtained in line 5 based on Lemma 3. KCCO is invoked to compute all  $k_{max}$ -CCOs 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.

---

### Algorithm 4. TD-MCCO

---

**Input:** A graph  $G(V, E_a, E_b)$

**Output:** The set containing all MCCOs in  $G$

---

```

1:  $\mathbb{C} \leftarrow \emptyset$ ;
2: Core-Decomposition( $G_b$ );
3:  $k_{max} \leftarrow \max_{u \in V}(\text{core}(u))$ ;
4: while  $\mathbb{C} = \emptyset$  and  $k_{max} > 0$  do
5:    $C \leftarrow \{u \in V | \text{core}(u) \geq k_{max}\}$ 
6:    $\mathbb{C} \leftarrow \text{KCCO}(G[C], k_{max})$ ;
7:    $k_{max} \leftarrow k_{max} - 1$ ;
8: return  $\mathbb{C}$ ;
```

---

### 4.3 Binary Searching *MCCOs*

We propose BU-MCCO and TD-MCCO in Subsects. 4.1 and 4.2 respectively. Even though they can successfully compute all *MCCOs* 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  $\bar{k}$  of  $k$ , and attempt to find all  $k$ -CCOs in each iteration, where  $k = \lfloor (\bar{k} + \underline{k})/2 \rfloor$ . If no any  $k$ -CCO is found, we know there does not exist any  $k'$ -CCO for  $k < k' < \bar{k}$  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  $k$ -CCOs 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.

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 *MCCOs* (line 4). The first parameter of BIN-Search is the set of all  $\underline{k}$ -CCOs 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  $\underline{k}$ -CCOs in BIN-Search. Instead of computing  $k$ -CCOs in the original graph, we compute  $k$ -CCOs in a smaller graph induced by  $\underline{k}$ -CCOs (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  $k$ -CCOs. If there does not

---

#### Algorithm 5. BIN-MCCO

---

**Input:** A graph  $G(V, E_a, E_b)$

**Output:** The set containing all *MCCOs* in  $G$

- 1: Core-Decomposition( $G_b$ );
- 2:  $d \leftarrow \max_{u \in V}(\text{core}(u)) + 1$ ;
- 3:  $\mathbb{C} \leftarrow \text{KCCO}(G, 1)$ ;
- 4: **return** BIN-Search( $\mathbb{C}, 1, d$ );

**Procedure** BIN-Search( $\mathbb{C}, \underline{k}, \bar{k}$ )

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

exist any  $k$ -CCOs (line 12), we decrease  $\bar{k}$  to  $k$  and continue searching (line 13). Otherwise, we increase  $\underline{k}$  to  $k$ , and change the first parameter to the set of all  $k$ -CCOs (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 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.

**Table 1.** Statistics of real-world datasets

Datasets	$ V $	$ E_a $	$ E_b $	$ E_b / E_a $	$\bar{d}_b$
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

**Real-World Datasets.** We evaluate the algorithms on six real graphs. The detailed statistics of these graphs are summarized in Table 1.  $\bar{d}_b$  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-TH is same as that for DBLP.

Epinions [23] and CiaoDVD<sup>1</sup> 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.

**Table 2.** Statistics of synthetic datasets

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

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

## 5.1 Performance Studies on Real-World Datasets

**Eval-I: Evaluating the Algorithm for Computing All  $k$ -CCOs.** The time-consuming for algorithm KCCO on six real-world graphs is reported in Fig. 4. 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 mainly because a large number of vertices are removed when the degree constraint  $k$  is large, and the result subgraph is small.

**Eval-II: Evaluating the Algorithms for Computing the MCCOs.** The time-consuming for algorithms BU-MCCO, TD-MCCO and BIN-MCCO on six real-world graphs is reported in Fig. 5(a). 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 h.

As we can see from the figure, BIN-MCCO is the fastest algorithm. It costs about 13 s in Gowalla and less than 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

<sup>1</sup> <https://www.librec.net/datasets.html>.

<sup>2</sup> <http://www.cse.psu.edu/~kxm85/software/GTgraph/>.

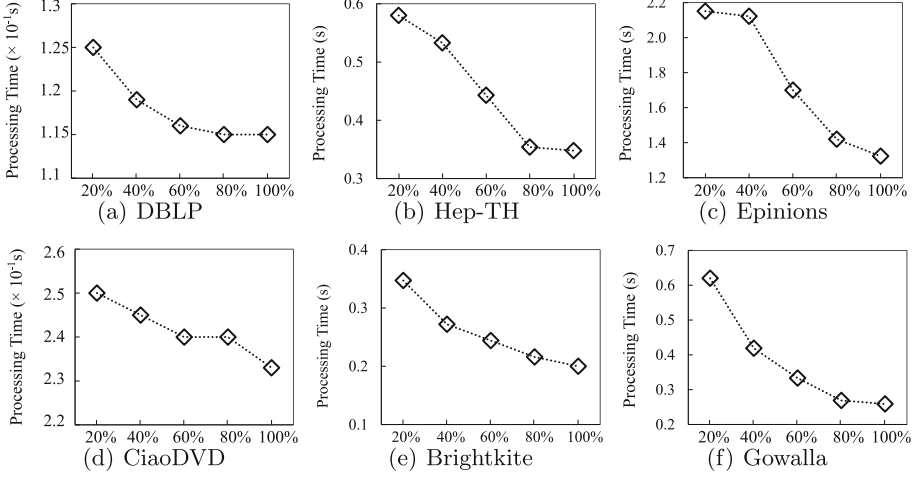


Fig. 4. Computing  $k$ -CCOs in real-world graphs

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.

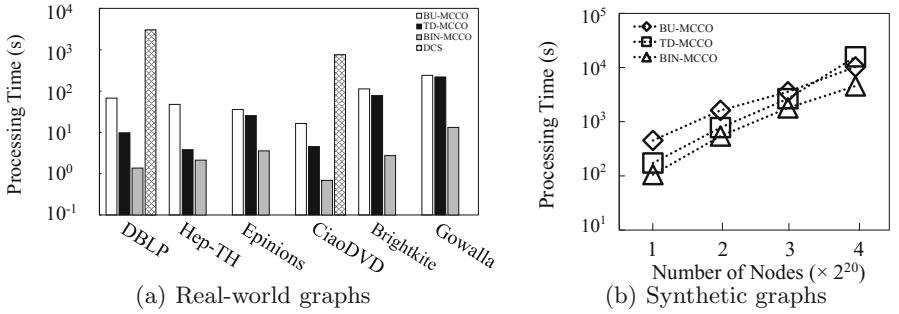


Fig. 5. Computing  $MCCOs$  in real-world graphs and synthetic graphs

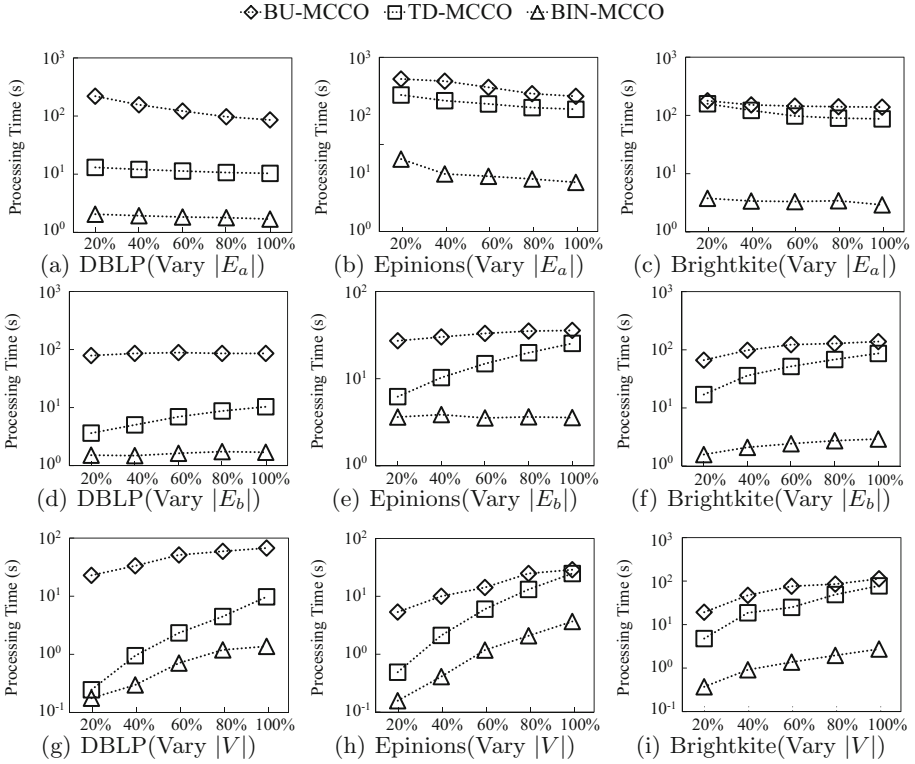
## 5.2 Scalability Testing

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-III: Sampling Physical Edges.** The running time of our proposed algorithms is reported in Fig. 6(a), (b) and (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.

**Eval-IV: Sampling Conceptual Edges.** The running time of our proposed algorithms is reported in Fig. 6(d), (e) and (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.



**Fig. 6.** Scalability testing



in Fig. 7(a). 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. 7(b), and the result of *MCCO* is cohesive.

## 6 Related Works

**Computing  $k$ -core.**  $k$ -core is first introduced in [29]. [3] 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]. 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.

## 7 Conclusion

Computing  $k$ -cores 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. We do extensive experiments to demonstrate the effectiveness and efficiency of our propose algorithms.

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

## References

1. Alvarez-Hamelin, J.I., Dall'Asta, L., Barrat, A., Vespignani, A.: Large scale networks fingerprinting and visualization using the  $k$ -core decomposition. In: NIPS, pp. 41–50 (2006)
2. Asahiro, Y., Iwama, K., Tamaki, H., Tokuyama, T.: Greedily finding a dense subgraph. J. Algorithms **34**(2), 203–221 (2000)



3. Batagelj, V., Zaversnik, M.: An  $o(m)$  algorithm for cores decomposition of networks. arXiv preprint [arXiv:cs/0310049](https://arxiv.org/abs/cs/0310049) (2003)
4. Bonchi, F., Gullo, F., Kaltenbrunner, A., Volkovich, Y.: Core decomposition of uncertain graphs. In: KDD, pp. 1316–1325 (2014)
5. Chang, L., Yu, J.X., Qin, L., Lin, X., Liu, C., Liang, W.: Efficiently computing  $k$ -edge connected components via graph decomposition. In: SIGMOD, pp. 205–216 (2013)
6. Charikar, M.: Greedy approximation algorithms for finding dense components in a graph. In: Jansen, K., Khuller, S. (eds.) APPROX 2000. LNCS, vol. 1913, pp. 84–95. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-44436-X\\_10](https://doi.org/10.1007/3-540-44436-X_10)
7. Cheng, J., Ke, Y., Chu, S., Özsu, M.T.: Efficient core decomposition in massive networks. In: ICDE, pp. 51–62 (2011)
8. Cho, E., Myers, S.A., Leskovec, J.: Friendship and mobility: user movement in location-based social networks. In: KDD, pp. 1082–1090 (2011)
9. Conte, A., Firmani, D., Mordente, C., Patrignani, M., Torlone, R.: Fast enumeration of large  $k$ -plexes. In: KDD, pp. 115–124 (2017)
10. da Fontoura Costa, L., Oliveira Jr., O.N., Travieso, G., Rodrigues, F.A., Boas, P.R.V., Antiqueira, L., Viana, M.P., Rocha, L.E.C.: Analyzing and modeling real-world phenomena with complex networks: a survey of applications. *Adv. Phys.* **60**(3), 329–412 (2011)
11. Cui, W., Xiao, Y., Wang, H., Wang, W.: Local search of communities in large graphs. In: SIGMOD, pp. 991–1002 (2014)
12. Fang, Y., Cheng, R., Luo, S., Hu, J.: Effective community search for large attributed graphs. *PVLDB* **9**(12), 1233–1244 (2016)
13. Gallo, G., Grigoriadis, M.D., Tarjan, R.E.: A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.* **18**(1), 30–55 (1989)
14. Giatsidis, C., Thilikos, D.M., Vazirgiannis, M.: D-cores: measuring collaboration of directed graphs based on degeneracy. In: ICDM, pp. 201–210 (2011)
15. Giatsidis, C., Thilikos, D.M., Vazirgiannis, M.: Evaluating cooperation in communities with the  $k$ -core structure. In: ASONAM, pp. 87–93 (2011)
16. Janson, S., Luczak, M.J.: A simple solution to the  $k$ -core problem. *Random Struct. Algorithms* **30**(1–2), 50–62 (2007)
17. Khaouid, W., Barsky, M., Srinivasan, V., Thomo, A.:  $K$ -core decomposition of large networks on a single PC. *PVLDB* **9**(1), 13–23 (2015)
18. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graph evolution: densification and shrinking diameters. *TKDD* **1**(1), 2 (2007)
19. Li, R.-H., Qin, L., Yu, J.X., Mao, R.: Influential community search in large networks. *PVLDB* **8**(5), 509–520 (2015)
20. Li, R.-H., Yu, J.X., Mao, R.: Efficient core maintenance in large dynamic graphs. *TKDE* **26**(10), 2453–2465 (2014)
21. Luczak, T.: Size and connectivity of the  $k$ -core of a random graph. *Discrete Math.* **91**(1), 61–68 (1991)
22. Ma, H., Zhou, D., Liu, C., Lyu, M.R., King, I.: Recommender systems with social regularization. In: Proceedings of the fourth ACM International Conference on Web Search and Data Mining, pp. 287–296 (2011)
23. Massa, P., Avesani, P.: Trust-aware recommender systems. In: RecSys, pp. 17–24 (2007)
24. Molloy, M.: Cores in random hypergraphs and Boolean formulas. *Random Struct. Algorithms* **27**(1), 124–135 (2005)
25. Montresor, A., De Pellegrini, F., Miorandi, D.: Distributed  $k$ -core decomposition. *TPDS* **24**(2), 288–300 (2013)

26. OBrien, M.P., Sullivan, B.D.: Locally estimating core numbers. In: ICDM, pp. 460–469 (2014)
27. Pittel, B., Spencer, J., Wormald, N.: Sudden emergence of a giantk-core in a random graph. *J. Comb. Theory Ser. B* **67**(1), 111–151 (1996)
28. Sarıyüce, A.E., Gedik, B., Jacques-Silva, G., Wu, K.-L., Çatalyürek, Ü.V.: Streaming algorithms for k-core decomposition. *PVLDB* **6**(6), 433–444 (2013)
29. Seidman, S.B.: Network structure and minimum degree. *Social Netw.* **5**(3), 269–287 (1983)
30. Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., Su, Z.: Arnetminer: extraction and mining of academic social networks. In: KDD, pp. 990–998 (2008)
31. Wang, J., Cheng, J.: Truss decomposition in massive networks. *PVLDB* **5**(9), 812–823 (2012)
32. Wen, D., Qin, L., Zhang, Y., Lin, X., Yu, J.X.: I/o efficient core graph decomposition at web scale. In: ICDE, pp. 133–144. IEEE (2016)
33. Wu, Y., Jin, R., Zhu, X., Zhang, X.: Finding dense and connected subgraphs in dual networks. In: ICDE, pp. 915–926 (2015)