Search of *Center-Core* Community in Large Graphs

Linlin Ding, Yu Xie, Xiaohuan Shan, and Baoyan Song(✉)

School of Information, Liaoning University, Shenyang 110036, Liaoning, China
bysong@lnu.edu.cn

Abstract. Community search plays an important role in complex network analysis. It aims to find a densely connected subgraph containing the query node in a graph. However, the most existing community search methods do not consider the influence of nodes and can not perfectly support the search in large graphs, making them have limitations in practical applications. In this paper, we introduce a community model called *center-core* community based on k-core decomposition, which can both capture the influence of nodes and guarantee the cohesiveness of community. Then we devise a *center-core* community graph index (CCG-Index), and online search algorithms (SingleQuery and MultiQuery) which support efficient search of *center-core* community in optimal time. Extensive experiments on four real-world large networks demonstrate the efficiency and effectiveness of our methods.

1 Introduction

Many real-world complex networks, such as the Internet, social networks, and biological neural networks, contain community structures. Because community structures can highly reflect the correlation of the complex networks, finding community structures of real-life networks is an important problem. Community search aims to find the most likely community structure that a node belongs to. Usually, a good community structure is described by the closeness of the nodes and defined as a densely connected subgraph in most previous studies. However, in many application domains, they need stricter requirements which consider the potential influence (or importance) of nodes and ask for the influence of nodes within a community is not lower than query node. For example, in military intelligence analysis domain, consider an ordinary soldier getting the confidential information in an army interaction network. He will try his best to pass the information to the soldiers with higher influence who usually take important positions as soon as possible in the detachment containing him. Finding a core subgraph made up of these people helps to analyze military intelligence. More, finding a subgraph containing the terrorists with important roles in communication network and finding a subgraph containing the researchers with higher influence and in the same research filed in research collaborator network are other examples of our applications.

The most existing methods only find the well connected community containing query node based on various dense subgraph structures, such as degree [1–3], distance [4,5], triangle [6,7], etc. Among them, a k-core is the largest subgraph of graph G such that each node in it has at least a degree k. However, the existing community search methods based on k-core have limitations applying to the above alike applications since they do not consider the influence of nodes, such as global search (GS) proposed in [8], an algorithm based on CoreStruct named GrCon [9], a local search (LS) proposed in [10], etc.

In order to solve the above problems, we use k-core as a qualifying dense structure for modeling a densely connected community. To further consider the node influence, we use coreness to evaluate the node influence by k-core decomposition algorithm [11]. Coreness of a node is the highest order of a core that contains it. The node influence increases as coreness increases. Thus we propose a new community model called *center-core* community based on k-core decomposition. It requires the community is a connected component of the maximal k-core containing the query node q and the coreness of nodes in community is no less than q. In addition, for the nodes whose corenesses are equal to q in the graph, the community needn't contain those nodes which are only reached from q via nodes with larger corenesses. The intuition behind this definition can be easily understood through the example in the army interaction network mentioned above. Assume Fig. 1 is the army interaction network, then the coreness of v_2, v_9, v_{10} is 2 respectively, the coreness of v_3 is 3 and the coreness of the rest nodes is 4 respectively. So the influence of all nodes in Fig. 1 is no less than v_2. Certainly, $A \cup B$ can be seen as a community for v_2. But in fact, its not good to make too many soldiers know the information according to the intelligence confidentiality requirements. For instance, though the influence of v_9 is equal to v_2, v_9 is insignificant to v_2 because the information has already pass to v_7 with higher influence before reaching v_9. Thus all the nodes in B are insignificant to v_2 without the link between v_7 and v_9 due to the connectivity requirement. So A is the *center-core* community containing v_2 by our definition.

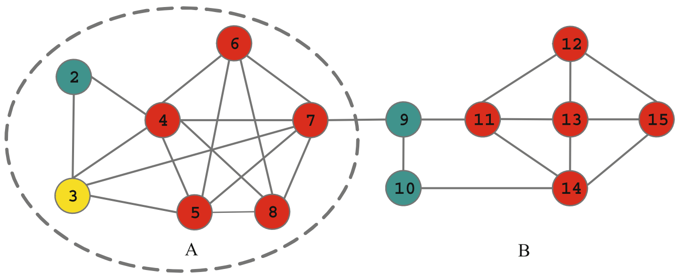

Fig. 1. $A \cup B$ is a connected 2-core of v_2; A is the *center-core* community of v_2

In this paper, we study the problem of *center-core* community search in large and dynamic graphs. First, we introduce a community model called *center-core* community based on the concept of k-core decomposition. This model can reflect the cohesiveness of the community, and also as much as possible contain the important nodes to query nodes by considering the potential influence of nodes and be more meaningful in practical applications. Then, we devise a linear-space index structure, *center-core* community graph index, called CCG-Index according to the characteristics of *center-core* community. The index only takes up a linear space and can be efficiently constructed. We also propose online search algorithms based on the index to find the *center-core* community for given query node(s) in large networks. Finally, we conduct extensive experiments on four real-world large networks, and the results demonstrate the efficiency and effectiveness of the proposed methods.

In summary, our contributions are as follows:

- We propose a community search model called *center-core* community based on k-core decomposition and motivate the problem of finding *center-core* community containing the given query nodes;
- We design a space-efficient index structure called CCG-Index which can well preserve the information about *center-core* community;
- We propose a method to effectively search *center-core* community in large graphs based on CCG-Index;
- Extensive experiments over four real-world graphs demonstrate the efficiency and effectiveness of the proposed algorithms.

2 Related Work

2.1 Community Search

Sozio and Gionis study the community search problem (CSP) based on minimum degree in social networks that aims to find the maximal connected k-core with largest k containing the query nodes [8]. Cui et al. [10] propose a more efficient local search algorithm for a query node. Barbieri et al. [9] propose a very efficient community search method based on index, and further propose the minimum community search problem (MINCCSP) to reduce the community size. They show MIN-CSP problem is NP problem, and get the approximate solution of the problem by a heuristic algorithm. All the mentioned methods are static algorithm and do not consider the influence of nodes. Except being based on minimum degree, Cui et al. [2] proposed a quasi-clique model to study the overlap community search problem. Huang et al. [12,13] study the CSP based on a k-truss community model. In addition, the computational complexity of those methods is higher than the methods based on minimum degree.

2.2 *K*-core Decomposition

Seidman introduces the concept of k-core for measuring the group cohesion in a network. The cohesion of the k-core increases as k increases [1]. Recently, the

k-core decomposition in graph has been successfully used in identifying the influential spreader in complex network [14,15]. Batagelj and Zaversnik [11] propose a O(n+m) algorithm for k-core decomposition in general graphs. This algorithm may be inefficient for the disk-resident graphs. Cheng et al. [16] propose an efficient k-core decomposition algorithm for the disk-resident graphs. Their algorithm works in a top-to-down manner that calculates the k-cores from higher order to lower order. To make the k-core decomposition more scalable, Montresor et al. propose a distributed algorithm for k-core decomposition by exploiting the locality property of k-core. Li et al. [17] propose an efficient core maintenance algorithm in dynamic graphs.

3 Problem Statement

Let $G(V, E)$ denote an undirected graph with node set V and edge set E. For any subset $H \subseteq V$, the subgraph induced by H, denoted as $G[H]$, is the graph whose node set is H and whose edge set is $(H \times H) \cap E$. Table 1 summarizes the notations used in this paper.

Table 1. Notations

Notation	Description
$G = (V, E), \lvert V\rvert = n, \lvert E\rvert = m$	An undirected graph G and n, m is the number of nodes and edges respectively
$G(H)$	The subgraph induced by H
C_k	The node set of the k-core
$d(v, G)$	The degree of v in G
$\mu(G)$	$\mu(G) = min_{v \in G} d(v, G)$
S_k	$S_k = C_k / C_{k+1}$
$c(v)$	The coreness of v
$c(Q)$	$c(Q) = min_{q \in Q} c(q)$
$Con_{k\text{-}core}(v/Q)$	The k-core connected components containing node v or node set Q
$Con_{k\text{-}shell}(v/Q)$	The k-shell connected components containing node v or node set Q
$indexV(v)$	The index vertex containing node v
X_v	$X_v = \lvert u : u \in N(v), c(u) \geq c(v)\rvert$
$N(v)$	The set of neighbors of node v

Definition 1 (k-*core*). *Given a graph* $G = (V, E)$, *a* k-*core of* G *is a maximal subgraph of* G *such that the degree of its each node is at least* k, *that is,* $d(v, G[C_k]) \geq k, v \in C_k$.

Denote the node set of a core as C_k to represent and identify a k-core. It is easy to see that the order of a core corresponds to the minimum degree of a node in that core i.e., $\mu(C_k) \geq k$. As shown in Fig. 2, the entire graph is 1-core i.e. $C_1 = \{v_1, v_2, v_3, ..., v_{20}\}$, $\mu(G[C_1]) = 1, C_2 = \{v_2, ..., v_{15}, v_{17}, v_{18}, v_{19}, v_{20}\}$, $\mu(G[C_2]) = 2, C_3 = \{v_3, ..., v_8, v_{11}, ..., v_{15}\}$, $\mu(G[C_3]) = 3$, $C_4 = \{v_4, v_5, v_6, v_7, v_8\}$, $\mu(G[C_4]) = 4$.

Property 1. Each k-core with different k value is unique and may not be connected.

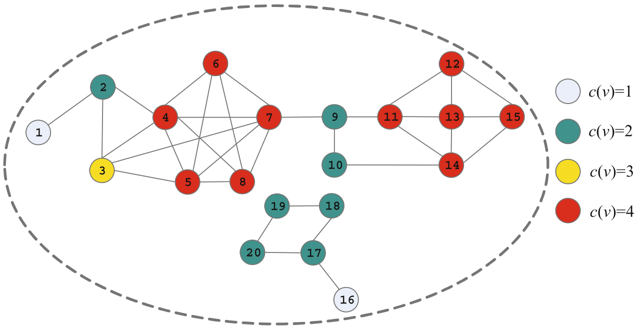

Fig. 2. An example graph G

As shown in Fig. 2, the k-core where the k is from value 1 to 4 are unique. Among them, 1-core, 2-core, 3-core have many connected components which are not connected.

Property 2. The cores are nested. The k-core with smaller k contains the k-core with larger k i.e., $C_j \subseteq C_i, i < j$.

As shown in Fig. 2, $C_4 \subseteq C_3 \subseteq C_2 \subseteq C_1 = V$.

Definition 2 (k*-shell*). *A k-shell is the induced subgraph by the set of nodes that only belongs to the k-core but not to the $(k+1)$-core i.e., the subgraph induced by the set of all nodes whose coreness is k.*

Denote the node set of a shell as S_k to represent and identify a k-shell, then $S_k = C_k/C_{k+1}$. Specifically, $S_{k_{max}} = C_{k_{max}}$, where the k_{max} is the highest order in all cores. As shown in Fig. 2:

$S_1 = C_1/C_2 = \{v_1, v_2, v_3, ..., v_{20}\}/\{v_2, ..., v_{15}, v_{17}, ..., v_{20}\} = \{v_1, v_{16}\}$.
$S_2 = C_2/C_3 = \{v_2, ..., v_{15}, v_{17}, ..., v_{20}\}/\{v_3, ..., v_8, v_{11}, ..., v_{15}\} = \{v_2, v_9, v_{10}, v_{17}, ..., v_{20}\}$.
$S_3 = C_3/C_4 = \{v_3, ..., v_8, v_{11}, ..., v_{15}\}/\{v_4, ..., v_8\} = \{v_3, v_{11}, ..., v_{15}\}$.
$S_4 = S_{kmax} = C_4/C_4 = \{v_4, v_5, v_6, v_7, v_8\}$.

Definition 3 (*coreness*). *The coreness of a node $v \in V$ is the highest order of a core that contains v, that is, the nodes with coreness k belong to k-core, but not belong to $(k+1)$-core.*

Denote the coreness of a node v as then $c(v)$, then $c(v) = max\{k|v \in C_k, k \in [0, 1, ..., k_{max}]\}$. In Fig. 2, take node v_6 as an example. v_6 belongs to C_1, C_2, C_3 and C_4 respectively, the highest order of the core containing v_6 is 4, so $C(v_6) = 4$.

Definition 4 (*connected k-core*). *A connected k-core is one of the connected components of the subgraph induced by k-core, denoting as $Con_{k-core}(v/Q)$.*

As shown in Fig. 2, 2-core is the induced graph of C_2, which has two connected components. Each one is a connected 2-core, $\{v_2, v_3, ..., v_{15}\}, \{v_{17}, v_{18}, v_{19}, v_{20}\}$. $Con_{k\text{-}core}(v)$ stands for the connected k-core of v, i.e. $Con_{2-core}(v_2) = \{v_2, v_3, ..., v_{15}\}$. According to Property 2, node v may belong to many connected k-core, i.e. v_2 belongs to one connected 2-core, and also belongs to one connected 1-core.

Definition 5 (*connected k-shell*). *A connected k-shell is one of the connected components of the subgraph induced by k-shell, denoting as $Con_{k-core}(v/Q)$.*

As shown in Fig. 2, the 2-shell that is the induce subgraph of S_2 has three connected components:$\{v_2\}$, $\{v_9, v_{10}\}$, $\{v_{17}, v_{18}, v_{19}, v_{20}\}$. According to Definition 2, the connected k-shell of v is unique, $Con_{shell}(v)$, i.e. $Con_{shell}(v_2) = \{v_2\}$.

Definition 6 (*center-core*). *Given a graph $G = (V, E)$ and a query node set $Q = \{q_1, q_2, ..., q_r\}, |Q| = r, r \geq 1, Q \subseteq V$. Set the k value of the maximum connected k-core containing Q as c. H is a center-core, if H satisfies the following conditions:*

1. *H is a connected c-core containing Q with the minimum degree c, that is, $H \subseteq Con_{c-core}(Q), \mu(H) = c$;*
2. *$\forall v \in H, c(v) \geq c(q)$;*
3. *The connected k-shell of node w, which coreness is equal to c in H, contains any query q or connects with any two query nodes, that is, $\exists q \in Q, q \in Con_{c-shell}(w)$ or $\exists q_1 \in Q, q_2 \in Q, q_1$ reaches q_2 with nodes in $Con_{c-shell}(w)$.*

Condition 1 ensures that *center-core* community containing q is densely connected since it has the largest minimum degree based on the concept of k-core. Condition 2 makes sure the influence of each node in *center-core* community is no less than q. And condition 3 ensures the nodes whose influence are equal to q needn't contact with q via nodes whose influence are larger than q. Because those nodes are certainly in the same connected shell with q, so that they can reach each other without the other nodes that have different coreness.

Problem Definition (*center-core* community search). Given a graph $G = (V, E)$ and a query node set $Q = \{q_1, q_2, ..., q_r\}, |Q| = r, r \geq 1, Q \subseteq V$, find a node set $R \subseteq V$ where the subgraph induced by R, $G[R]$, is the *center-core* community containing Q.

Example 1. In Fig. 2, suppose the query node $q = v_2$, then by the problem definition, the subgraph induced by node set $\{v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ is the *center-core* community containing v_2.

4 Querying *Center-core* Community

4.1 The Novel CCG-Index

4.1.1 CCG-Index Structure

We propose the CCG-Index structure, which can reflect the hierarchical structure of the graph. It's a hierarchical structure according to coreness. As shown in Fig. 3, the top level is the first level (level = 1). The number of level increases by the level increasing, where the level number corresponds to coreness. Each index item vertex (we call the node element in index as index item vertex to distinguish from the nodes in graph) in index is a connected k-shell with different k. The nodes in the same connected k-shell must be connected, so it can compress storage space since the edges between those nodes needn't be preserved. Besides, each vertex keeps the children and parents information for query and update later. Let the vertices in S_k point to the connected vertices in S_{k+1}, and each directed edge can both indicate the hierarchical and connected relationship between vertices. The level k index item vertex is the parent index item vertex of the connected $k+1$ level, that is, the level $k+1$ index item vertex is the child index item vertex of the connected k level. We can directly output the *center-core* community containing a query node according to the direction relationship in index. It can avoid repetitively visiting nodes if we find the nodes higher than query node from top to bottom level by level. Figure 4 is the CCG-Index of G in Fig. 2. The index vertices are A, B, C, D, E, F, G, H.

We can see that the index is composed of connected components of all S_k. If query node is v_2, which coreness is 2. We first find the index item vertex $indexV(v_2) = D = \{v_2\}$, then find the nodes that have the same coreness and connected directly, and then find nodes with higher coreness level by level. As shown in Fig. 4, first find the index item vertex $B = \{v_3\}$ in the next level, then continue find the vertex $C = \{v_4, v_5, v_6, v_7, v_8\}$ then continue find the vertex

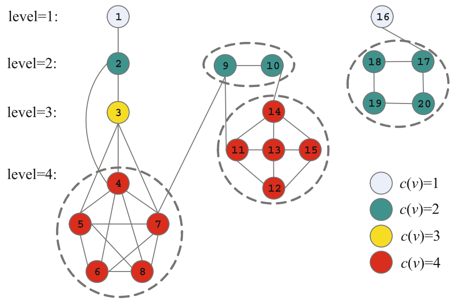

Fig. 3. The hierarchical division graph of G

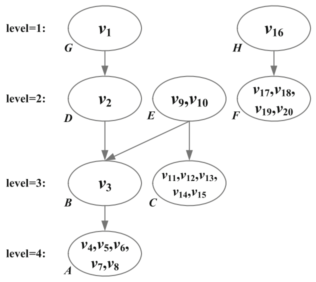

Fig. 4. The CCG-index of G

in the next level. So the induced subgraph of all the nods in these vertexes $\{D, B, A\}$ is the *center-core* community containing v_2 : $\{v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ as shown in the subgraph A in Fig. 1.

4.1.2 CCG-Index Construction

The construct course of CCG-Index is shown in Algorithm 1. First, calculate the coreness of each node by k-core decomposition algorithm in [11]. Second, compute each connected component in each connected k_{max}-shell where k is the largest coreness in graph and initialize it to a single index item vertex. The coreness of nodes in the bottom level index vertex is the maximum without any child vertex. Each vertex in S_k can be created by seeking the neighbors of each node in S_k. Next, recursively find and let the vertices in S_k point to the connected vertices in S_{k+1} from $k = k_{max-1}$ to $k = 1$ according to the neighbors of nodes in S_k. That is to say, the S_k vertex is the parent vertex of S_{k+1} connected with itself, and the S_{k+1} vertex is the child vertex of S_k connected with itself, $k = k_{max}$-1, ..., 1.

Algorithm 1. Construct CCG-Index

```
Input: $G = (V, E)$
Output: The CCG-Index
 1: Compute the $k$-core decomposition for $G$ and keep the core index;
 2: $k_{max} = max\{c(v)|v \in V\}$;
 3: CCG-Index=$\emptyset$;
 4: Create a vertex for each connected $k_{max}$-shell in CCG-Index;
 5: for $k = k_{max} - 1$ to 1
 6:     for all nodes $v \in S_k$
 7:             Create a vertex containing $v$ if $v$ is not visited in CCG-Index;
 8:         for $w \in N(v)$
 9:             if $c(w) == c(v)$ then
10:                 Merge $indexV(v)$ and $indexV(w)$;
11:             if $c(w) > c(v)$ then
12:                 Let $indexV(v)$ points to the vertex in $S_{k+1}$ which is connected to
    $indexV(w)$;
13: return The CCG-Index;
```

Example 2. Take the CCG-Index in Fig. 4 as an example to illustrate the construction course in Fig. 5. First, compute its k-core decomposition (line 1) and save the coreness of each node as shown in Fig. 3. Second, compute $k_{max} = 4$ (line 2), and there is only one connected 4-shell $\{v_4, v_5, v_6, v_7, v_8\}$ since the nodes in S_4 are adjacent to each other, so initialize it to a single index item vertex $indexV(v_4) = \{v_4, v_5, v_6, v_7, v_8\}$ (line 4) as shown in Fig. 5(a). Then recursively process each node in S_k for every k ($k = 3, 2, 1$ process each node in) (lines 5–12). For $S_3 = \{v_3, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}\}$, We process v_3: initialize $indexV(v_3) = \{v_3\}$ (line 7); let $B = indexV(v_3)$ point to $indexV(v_4)$ in S_4 because $v_4 \in N(v_3)$ and the coreness of v_4 is larger than v_3 (lines 11–12). Process v_{11}: initialize $indexV(v_{11}) = C = \{v_{11}\}$; add neighbors of v_{11} in S_3 into $C = indexV(v_{11})$ so $indexV(v_{11}) = \{v_{11}, v_{12}, v_{13}, v_{14}\}$. Because no neighbors of $v_{11}, v_{12}, v_{13}, v_{14}$ have larger coreness respectively, further process v_{15}: $indexV(v_{15}) = \{v_{15}\}$. Now, v_{15} is adjacent to v_{12}, so merge $indexV(v_{15})$ and $indexV(v_{12})$ and the vertex becomes $C = \{v_{11}, v_{12}, v_{13}, v_{14}, v_{15}\}$ (lines 9–10) as shown in Fig. 5(b). The operations for S_3 are completed. We can finally get the CCG-Index of G as shown in Fig. 5(c) and (d) by recursively processing S_2 and S_1.

In Algorithm 1, the calculation of k-core decomposition require $O(n + m)$ time [11]. The main cycle in lines 5–12 which process each nodes need $O(n' + m')$ time, n' and m' is the number of vertexes and edges in index respectively, and $n' \ll n, m' \ll m$. The other operation can be processed in constant time, so the time complexity of Algorithm 1 is $O(n + m)$. In addition, it only needs $O(n)$ space since each node is only storaged once in the index.

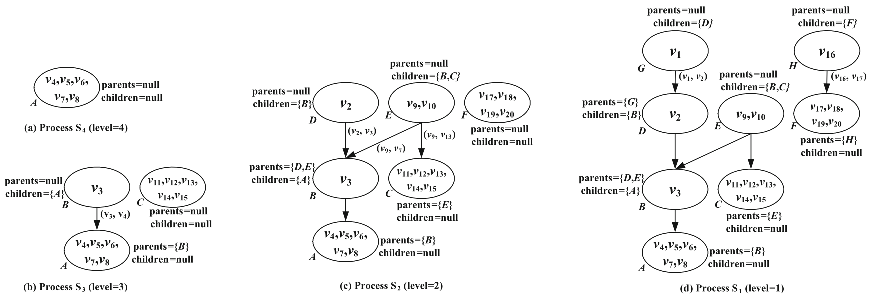

Fig. 5. The construction of CCG-index of G

4.2 *Center-core* Community Query Processing

4.2.1 Single Query Node Processing

For single query node, the *center-core* community query processing is straightfoward based on the CCG-Index. Treat the index structure as a tree, then the larger index level, the larger number of coreness of index vertex. The nodes in the subtree taking $indexV(q)$ as root is the result of *center-core* community search for q. Because the $indexV(q)$ includes all the nodes whose coreness equals q and in the same connected k-shell with q. The coreness of the nodes in its children vertices in larger levels is all larger than q. Thus such subtree is the *center-core* community for q. The query algorithm is shown in Algorithm 2.

Algorithm 2. SingleQuery

```
Input: The CCG-Index, a given node $q$
Output: The center-core containing $q$
1: $R = indexV(q)$;
2: if children($indexV(q)$)==null then
3:     return $R$;
4: Push all children of $indexV(q)$ into the stack $s$;
5: while $s$ is not empty do
6:     Pop the vertex in top of $s$;
       add the nodes in it into $R$;
       push its children into $s$;
7: return $R$;
```

Example 3. Consider Fig. 2, given a query node $q = \{v_2\}$, we query the *center-core* community R containing q by Algorithm 2. By retrieving the CCG-Index as shown in Fig. 4, first initialize $R = indexV(v_2) = D = \{v_2\}$ (line 1). We use a stack s to process this traversal of children of each vertex that be visited (lines 3–6). Because D only has one child vertex B, push B to stack s, and then pull, $R = \{v_2, v_3\}$. Continue the iterated operation, then push vertex A, which is the child of vertex B, and then pull, $R = \{v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$, utill the stack s is null. The query course is over, without any node to be visited. So the search result is $R = \{v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$, the subgraph induced by R is the subgraph A in Fig. 1.

Algorithm 2 just needs to traverse the vertices related to query node in CCG-Index, so its time complexity is no more than O(n'). Due to this algorithm need not to consider the branches connected to the connected k-shell without query node, it narrows the search scope and accelerates the retrieval speed.

4.2.2 Multiple Query Nodes Processing

In CCG-Index, for the multiple query nodes processing, that is the query set $Q = \{q_1, q_2, ..., q_r\}$, we find a substree which root node is the least common ancestors (LCA) of index vertex of all the query nodes. The subtree is the *center-core* dense subgraph containing Q. Because the LCA of quey nodes is the nodes with maximum coreness making the nodes connected of Q, where the minimum coreness of subgraph can be maximized. Specially, if two query nodes with the same coreness locating in different index item vertices and they are connected, the union of these two index item vertices is the root node.

Algorithm 3. MultiQuery

```
Input: The CCG-Index; a set of query nodes $Q = \{q_1, q_2, ..., q_r\}$
Output: The center-core containing $Q$
1: $R = \emptyset$;
2: $Root = indexV(q_1)$;
3: for $i$=2 to $r$
4:     for $v \in Root$;
5:         $Root = findLCA(v, indexV(q_1))$;
6:         $R = R \cup subtree(Root)$;
7:     if $(Root == \emptyset)$ then break;
8: return $R$;
```

Example 4. As shown in Fig. 2, suppose the query set is $Q = \{v_3, v_6, v_{11}\}$. According to Algorithm 3, search the CCG-Index in Fig. 4. First, initialize $R = \emptyset$ (line 1); $Root = indexV(v_3) = B$ (line 2); $indexV(v_6) = A$, find the LCA of B and A, $C = min\{c(B), c(A)\} = min\{3, 4\} = 4$. The LCA of forth level of B and A is B, so R=$\{substree(B)\}$. Then, $indexV(v_{11} = C)$, also continue to find the LCA of B and C, that is E, so the final R=$\{subtree(B), subtree(e)\} = \{B, A, C, E\} = \{v_3, v_4, ..., v_{15}\}$ (lines 3–8).

Algorithm 3 shows the multiple query nodes *center-core* dense subgraph algorithm. We use a traversing algorithm to traverse the index vertices corresponding to R from top to bottom with the time complex $O(n'')$. n'' is the number of nodes in R. So, the time complex of Algorithm 3 is $O((|Q| - 1)n'')$. Because $n'' \ll n$, the time complex is still small in actual applications.

5 Experiments

We conduct extensive experiments on four real-world large networks to evaluate the efficiency and effectiveness of the proposed algorithms in this paper. We implemented all of the algorithms in Java and ran the experiments on a PC with Intel quad core at 3.2 GHz, 8G memory. The experimental datasets are from four real-world networks named Twitter, DBLP, Amazon and Youtube. Twitter is a social network, where each node represents a user and each edge represents the friendships of users. DBLP is an author collaboration network, where each node represents an author and each edge represents a coauthor relationship. Amazon is a e-commerce network, where each node stands for a product and each edge stands for purchasing this product. Youtube is a user-to-user link network. The statistics of these graphs are reported in Table 2, containing the number of nodes n, the number of edges m and the maximum coreness h gaining from k-core decomposition.

For index construction, we compare the index construction time and index size of CCG-Index (Algorithm 1) and CoreStruct [9] respectively. For query processing, we implement three algorithms, CCG (*center-core* community search, Algorithm 2 and Algorithm 3), GS [8], GrCon [9] and compare their execution time with different query nodes on different datasets.

Table 2. Datasets

Dataset	n	m	h	Description
Twitter	81,306	1,768,149	38	Social network
DBLP	317,080	1,049,866	40	Collaboration network
Amazon	410,236	3,356,824	41	Product network
Youtube	1,134,890	2,987,624	51	Social network

5.1 Index Construction

Figure 6 shows the index construction time on different datasets of CCG-Index and CoreStruct. The time of CCG-Index is slightly less than the CoreStruct. They are almost the same because they all need to traverse the each node and its incident edges in G. However, the construction time of CoreStruct depends on the maximum coreness of different k-core which needs to traverse a part of nodes repeatedly. So, the larger of value h, the more time of construction index.

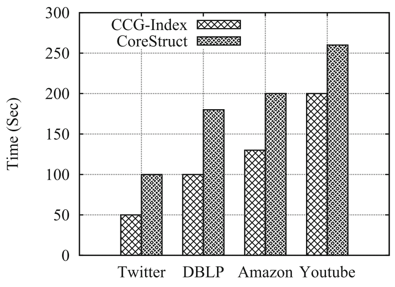

Fig. 6. Index construct time

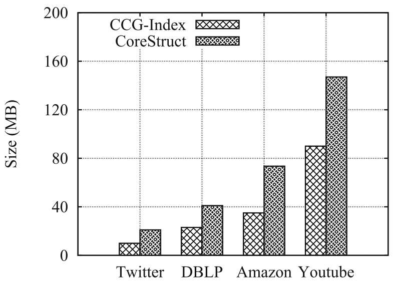

Fig. 7. Index storage space

Figure 7 is the index size in different datasets. It can be seen that the storage space of CCG-Index is obviously lower than CoreStruct because there is nested feature of k-core. CoreStruct repeatedly stores many nodes. However, CCG-Index only stores the node information once by fully using the structure feature of *center-core*.

5.2 Query Processing

The query node set Q is designed by randomly choosing 1, 4, 8, 16 and 32 nodes from four real datasets respectively. So, for single query processing, $|Q| = 1$, CCG is the algorithm SingleQuery, Algorithm 2. For the multiple nodes query processing, $|Q| > 1$, CCQ is the algorithm MultiQuery, Algorithm 3.

Figure 8 shows the performance of query processing of three algorithms in four real datasets. In each dataset, with the increasing of $|Q|$, the query time of CCG is much better than the other two algorithms. Because the more nodes of

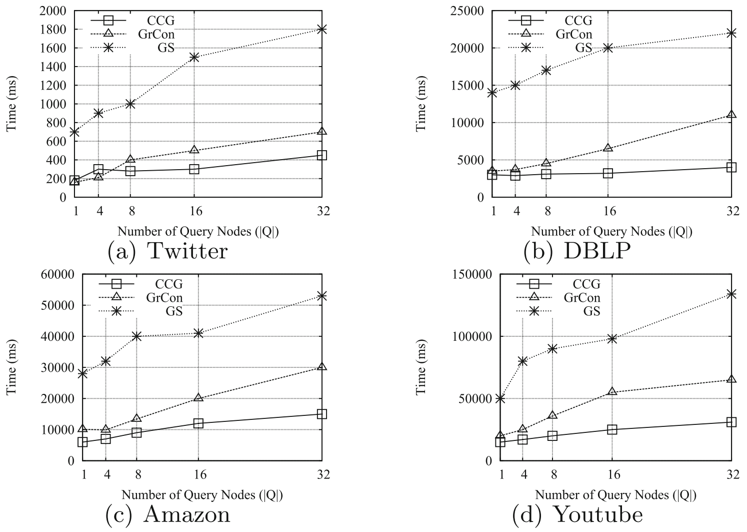

Fig. 8. Query processing time

query processing, the more nodes to be traversed, causing more time. The query time of CCG algorithm is mainly about the index vertices, which is much less than the whole nodes.

6 Conclusions

In this paper, we mainly study such meaningful community search for a query node, called *center-core* community search. To effectively reduce query time to apply to large and dynamic graphs, we further propose the index called CCG-Index to preserve information about the level classification of node influence and connected relationship of nodes. Thus we can quickly query *center-core* community by retrieving index. Extensive experiments over four real-world graphs demonstrate the efficiency and effectiveness of the proposed algorithms.

Acknowledgement. This work is supported by National Natural Science Foundation of China (No. 61472169, 61502215), Science Research Normal Fund of Liaoning Province Education Department (No. L2015193), Doctoral Scientific Research Start Foundation of Liaoning Province (No. 201501127), National Key Research and Development Program of China (No. 2016YFC0801406).

References

1. Seidman, S.B.: Network structure and minimum degree. Soc. Netw. **5**(3), 269–287 (1983)
2. Cui, W., Xiao, Y., Wang, H., Lu, Y., Wang, W.: Online search of overlapping communities. In: ACM SIGMOD International Conference on Management of Data, pp. 277–288 (2013)

3. Tsourakakis, C., Bonchi, F., Gionis, A., Gullo, F., Tsiarli, M.: Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 104–112 (2013)
4. Bta, A., Krsz, M.: A high resolution clique-based overlapping community detection algorithm for small-world networks. Informatica **39**(2), 177–187 (2015)
5. Koujaku, S., Takigawa, I., Kudo, M., Imai, H.: Dense core model for cohesive subgraph discovery. Soc. Netw. **44**, 143–152 (2016)
6. Wang, N., Zhang, J., Tan, K.L., Tung, A.K.H.: On triangulation-based dense neighborhood graph discovery. Proc. VLDB Endow. **4**(2), 58–68 (2010)
7. Li, R.H., Yu, J.X.: Triangle minimization in large networks. Knowl. Inf. Syst. **45**, 617–643 (2015)
8. Sozio, M., Gionis, A.: The community-search problem and how to plan a successful cocktail party. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, D.C., USA, pp. 939–948, July 2010
9. Barbieri, N., Bonchi, F., Galimberti, E., Gullo, F.: Efficient and effective community search. Data Min. Knowl. Discov. **29**(5), 1406–1433 (2015)
10. Cui, W., Xiao, Y., Wang, H., Wang, W.: Local search of communities in large graphs. In: ACM SIGMOD International Conference on Management of Data, pp. 991–1002 (2014)
11. Batagelj, V., Zaversnik, M.: Fast algorithms for determining (generalized) core groups in social networks. Adv. Data Anal. Classif. **5**(2), 129–145 (2011)
12. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: ACM SIGMOD International Conference on Management of Data, pp. 1311–1322 (2014)
13. Huang, X., Lakshmanan, L.V.S., Yu, J.X., Cheng, H.: Approximate closest community search in networks. Proc. VLDB Endow. **9**(4), 276–287 (2015)
14. Miorandi, D., Pellegrini, F.D.: K-shell decomposition for dynamic complex networks. In: Proceedings of the International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, pp. 488–496 (2010)
15. Zhao, Q., Lu, H., Gan, Z., Ma, X.: A K-shell decomposition based algorithm for influence maximization. In: Cimiano, P., Frasincar, F., Houben, G.-J., Schwabe, D. (eds.) ICWE 2015. LNCS, vol. 9114, pp. 269–283. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19890-3_18
16. Cheng, J., Ke, Y., Chu, S., Ozsu, M.T.: Efficient core decomposition in massive networks. In: IEEE International Conference on Data Engineering, pp. 51–62 (2011)
17. Li, R.H., Yu, J.X., Mao, R.: Efficient core maintenance in large dynamic graphs. IEEE Trans. Knowl. Data Eng. **26**(10), 2453–2465 (2014)