Applied Mathematics & Information Sciences
An International Journal

A new algorithm to determine minimally k-edge-connected graphs with odd k

Yunming Ye[1] *and Yueping Li*[2,1,*]

[1]Shenzhen Key Laboratory of Internet Information Collaboration, Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen, 518055, China
[2]Shenzhen Polytechnic, Shenzhen, 518055, China

Received: 11 Sep. 2012, Revised: 27 Nov. 2012, Accepted: 7 Dec. 2012
Published online: 1 May 2013

Abstract: A necessary and sufficient condition for a graph to be minimally k-edge-connected where k is odd is presented. Based on this result, a divide-and-conquer algorithm to determine minimally k-edge-connected graphs is developed. Experiments are performed to check minimally 3-edge-connectivity. The results show that our algorithm is much more effective than the previously known best algorithm.

Keywords: Edge connectivity, minimally edge-connected, divide-and-conquer, time complexity.

1. Introduction

Edge connectivity is a classical property of graph which has been extensively studied since 1970s [1,2]. It has well-known applications in various areas such as network survivability and VLSI design. In addition, 3-edge-connectivity can be applied in physics, quantum chemistry [3] and computational biology [4] . Recently, there are many works related with k-edge connectivity such as k-edge-connectivity augmentation [5,6], k-edge-connected subgraph detection [7] and approximation [8].

By far, there are few results on minimally k-edge-con-nectivity [9,10]. Furthermore, there is no any algorithm developed specially for detecting minimally k-edge-con-nectivity. Several algorithms are proposed for checking k-edge-con-nectivity [11–14]. However, minimally k-edge-con-nectivity plays an important role in construction of reliable network of least cost.

In this paper, we present a necessary and sufficient condition for a graph to be minimally k-edge-connected where k is odd. Based on this result, we develop a divide-and-conquer algorithm to determine minimally 3-edge-connected graphs. The experimental results show that our algorithm runs much faster than the best algorithm known by far. This paper is organized as follows: In Section 2, basic definitions are given. In Section 3, we characterize the structure of minimally k-edge-connected graphs. In Section 4, our algorithm is proposed. The experimental results are shown in Section 5. Conclusions and future work are discussed in Section 6.

2. Basic definitions

All graphs under consideration are undirected, finite and loopless. Multiple edges are allowed. Let G=(V,E) be a graph consists of a nonempty set $V(G)$ of vertices and a set $E(G)$ of edges. An edge cutset is a set of edges whose removal leaves a disconnected graph. If it consists of exactly k edges, then we call it a k-edge cutset. A k-edge cutset is *trivial* if the k edges of it are just the associate edges of a vertex whose degree is k. The edge-connectivity $\lambda(G)$ of a graph G is defined to be the minimum size of edge cutsets. A graph G is said to be minimally k-edge-connected if $\lambda(G) = k$ but $\lambda(G-e) = k-1$ for every edge e of G. Loops have no effect on edge connectivity, so they cannot appear in minimally k-edge-connected graphs. Denote the set of edges between vertex sets A and B by $\Gamma(A,B)$. For the terminology and notation not defined in this paper, the readers are referred to [15].

[*] Corresponding author e-mail: leeyueping@gmail.com

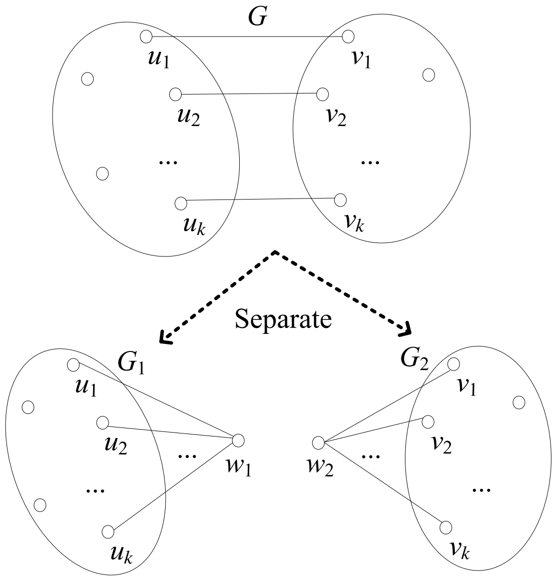

Figure 1 Separating operation.

We introduce the "*separating*" operation on minimally k-edge-connected graph through a non-trivial k-edge cutset which is shown in Fig. 1.

Let G be a minimally k-edge-connected graph and k is odd. If no special mention, k refers a positive odd integer in the rest of this paper. Suppose EC is a non-trivial k-edge cutset and C_1,C_2 are the two components separated by EC. Suppose EC=$\{u_1v_1, u_2v_2, \dots, u_kv_k\}$ such that $u_1, u_2, \dots, u_k \in V(C_1)$ and $v_1, v_2, \dots, v_k \in V(C_2)$. Note that u_iv_i might be the same as u_jv_j for $i \neq j$; that is, parallel edges are allowed in EC. The separating operation is as following: Let w_1 be a new vertex and G_1 be a graph such that $V(G_1) = V(C_1)\cup\{w_1\}$ and $E(G_1) = E(C_1)\cup\{u_1w_1, u_2w_1, \dots, u_kw_1\}$. Let w_2 be a new vertex and G_2 be a graph such that $V(G_2) = V(C_2)\cup\{w_2\}$ and $E(G_2) = E(C_2)\cup\{v_1w_2, v_2w_2, \dots, v_kw_2\}$. If $u_i = u_j$ for $i \neq j$, then u_iw_1 and u_jw_1 compose a multiple edge. Similarly, if $v_i = v_j$ for $i \neq j$, then v_iw_2 and v_jw_2 form a multiple edge.

3. Main results

Theorem 1. *Let G be a minimally k-edge-connected graph for odd k and Cut_1=$\{e_1, e_2, \dots, e_k\}$ be a k-edge cutset. Let C_1 and C_2 be the two components of $G-Cut_1$. For any other k-edge cutset Cut_2 =$\{f_1, f_2, \dots, f_k\}$, if $f_j \in E(C_i)$ for some j and i where $1 \le j \le k$, $1 \le i \le 2$, then we have $Cut_2 \subseteq E(C_i) \cup Cut_1$.*

Proof. Suppose it does not hold. Let D_1 and D_2 be the two components of $G-Cut_2$. Based on the intersect part of Cut_1 and Cut_2, there are two cases.

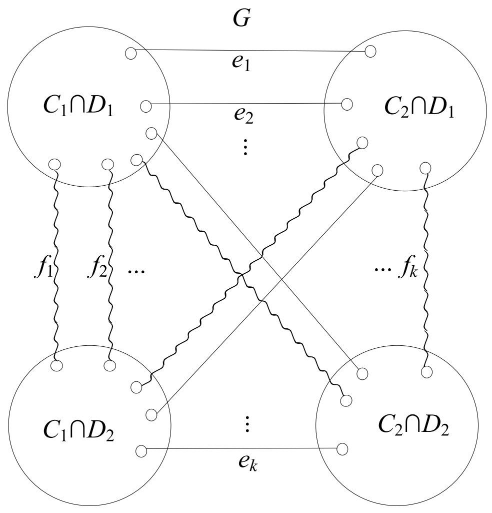

Figure 2 An example of Cut_1 and Cut_2.

Case 1: $Cut_1 \cap Cut_2 = \emptyset$. By the assumption, we have $Cut_2 \cap E(C_1) \neq \emptyset$ and $Cut_2 \cap E(C_2) \neq \emptyset$. Without loss of generality, suppose $|Cut_2 \cap E(C_1)| \le |Cut_2 \cap E(C_2)|$. Since $|Cut_2| = k$ and k is odd, $|Cut_2 \cap E(C_1)| \le (k-1)/2$. Without loss of generality, assume that $|E(D_1) \cap Cut_1| \le |E(D_2) \cap Cut_1|$. Since $|Cut_1| = k$ and k is odd, $|E(D_1) \cap Cut_1| \le (k-1)/2$. We have $|Cut_2 \cap E(C_1)| + |E(D_1) \cap Cut_1| \le k-1$, then $(Cut_2 \cap E(C_1)) \cup (E(D_1) \cap Cut_1)$ composes an edge cutset whose size is smaller than k. Contradict!

Case 2: $Cut_1 \cap Cut_2 \neq \emptyset$. We divide this case into the following two sub-cases.

Case 2.1: $|Cut_1 \cap D_1| = |Cut_1 \cap D_2|$. Then, we investigate the intersection part $Cut_1 \cap Cut_2$. It is clear that $Cut_1 \cap Cut_2 = \Gamma(C_1 \cap D_1, C_2 \cap D_2) \cup \Gamma(C_2 \cap D_1, C_1 \cap D_2)$. Therefore, we conclude $|\Gamma(C_1 \cap D_1, C_2 \cap D_2)| \neq |\Gamma(C_2 \cap D_1, C_1 \cap D_2)|$, since k is odd and $|Cut_1 \cap D_1| = |Cut_1 \cap D_2|$. Without loss of generality, we suppose $|\Gamma(C_1 \cap D_1, C_2 \cap D_2)| < |\Gamma(C_2 \cap D_1, C_1 \cap D_2)|$. If $|Cut_2 \cap C_1| \le |Cut_2 \cap C_2|$, then $(Cut_2 \cap C_1) \cup \Gamma(C_1 \cap D_1, C_2 \cap D_2) \cup (Cut_1 \cap D_1)$ is an edge cutest whose size is smaller than k. A contradiction. On the other hand, if $|Cut_2 \cap C_1| > |Cut_2 \cap C_2|$, then $(Cut_2 \cap C_2) \cup \Gamma(C_1 \cap D_1, C_2 \cap D_2) \cup (Cut_1 \cap D_2)$ is also an edge cutest whose size is smaller than k. It is also a contradiction.

Case 2.2: $|Cut_1 \cap D_1| \neq |Cut_1 \cap D_2|$. Without loss of generality, we suppose $|Cut_1 \cap D_1| < |Cut_1 \cap D_2|$. If $|Cut_2 \cap C_1| + |\Gamma(C_1 \cap D_1, C_2 \cap D_2)| \le |Cut_2 \cap C_2| + |\Gamma(C_1 \cap D_1, C_2 \cap D_1)|$, then $(Cut_2 \cap C_1) \cup \Gamma(C_1 \cap D_1, C_2 \cap D_2) \cup (Cut_1 \cap D_1)$ is an edge cutest contains less than k edges. A contradiction. If $|Cut_2 \cap C_1| + |\Gamma(C_1 \cap D_1, C_2 \cap D_2)| > |Cut_2 \cap C_2| + |\Gamma(C_1 \cap D_1, C_2 \cap D_1)|$, then we have

$(Cut_2 \cap C_2) \cup \Gamma(C_1 \cap D_1, C_2 \cap D_2) \cup (Cut_1 \cap D_1)$ is an edge cutest contains less than k edges. It also finds a contradiction. Finally, we conclude that our theorem holds.

Theorem 2. *Let G be a minimally k-edge-connected graph for odd k. Let EC be a non-trivial k-edge cutset and G_1, G_2 be the two graphs obtained by separating G through EC. Then G_1 and G_2 are minimally k-edge-connected.*

Proof. Suppose it does not hold and G_1 is not minimally k-edge-connected. Let C_1,C_2 be the two components of $G-EC$, let $EC = \{u_1v_1, u_2v_2, ..., u_kv_k\}$ such that u_1, u_2, $\ldots$, $u_k \in V(C_1)$ and v_1, v_2, $\ldots$, $v_k \in V(C_2)$. Suppose $\delta(w_1)$=$\{u_1w_1,\ u_2w_1,\ \ldots,\ u_kw_1\}$. Then, we have $V(G_1) = V(C_1) \cup \{w_1\}$ and $E(G_1) = E(C_1) \cup \delta(w_1)$.

Case 1: if G_1 is not k-edge-connected, then there is an edge cutset whose size is smaller than k, denoted by EC_1, in G_1. If $EC_1 \cap \delta(w_1) = \emptyset$, EC_1 is also an edge cutset in G. Contradict!

If $EC_1 \cap \delta(w_1) \neq \emptyset$, suppose $EC_1 \cap \delta(w_1) = \{u_{i_1}w_1$, $u_{i_2}w_1$, ..., $u_{i_n}w_1\}$,$n < k$. Let D_1,D_2 be the two components of $G - EC_1$. Without loss of generality, suppose D_1 contains $u_{i_1}, u_{i_2}, ..., u_{i_n}$. D_1 is separated from G by edge cutset $(EC_1 \backslash \delta(w_1)) \cup EC$. Since u_j,$v_j \notin D_1$ for $j \notin \{i_1, i_2, ..., i_n\}$, the edge (u_j,v_j) can be removed in the edgecut. Then, edge set $(EC_1 \backslash \delta(w_1)) \cup \{u_{i_1}v_{i_1}$, $u_{i_2}v_{i_2}$, ..., $u_{i_n}v_{i_n}\}$ separates $V(D_1)$ and $V(G) \backslash V(D_1)$ in G. So it is an edge cutset whose size is smaller than k in G. Contradict!

Case 2: G_1 is k-edge-connected. There is an edge $f \in E(G_1)$ such that $\lambda(G_1) = \lambda(G_1 - f)$. Since G is minimally k-edge-connected, there is a k-edge cutset, denoted by EC_f, containg f in G. Theorem 1 guarantees that $EC_f \subset E(C_1) \cup EC$. If $EC_f \cap EC = \emptyset$, then EC_f is also a k-edge cutset in G_1. Hence $\lambda(G_1 - f) = k-1 < \lambda(G_1)$. Contradict!

If $EC_f \cap EC \neq \emptyset$, suppose $EC_f \cap EC = \{u_{i_1}v_{i_1}$, $u_{i_2}v_{i_2}$, ..., $u_{i_m}v_{i_m}\}$,$m < k$. Let C_1,C_2 be the two components of $G - EC_f$. Without loss of generality, suppose C_1 contains u_{i_1}, u_{i_2}, ..., u_{i_m}. $V(C_1)$ is separated from $V(G)$ by $(EC_f \backslash EC) \cup$ $\{u_{i_1}v_{i_1}$, $u_{i_2}v_{i_2}$, ..., $u_{i_m}v_{i_m}\}$. After the separating operation through EC, the edges of EC are deleted while the edges $u_{i_1}w_1$, $u_{i_2}w_1$, ..., $u_{i_m}w_1$ and the vertex w_1 are added. So $V(C_1)$ is separated from $V(G_1)$ by $(EC_f \backslash EC) \cup \delta(w_1)$. Since the edge (w_1,u_j) for $j \notin \{i_1, i_2, ..., i_m\}$ is not incident with any vertex of C_1, it can be eliminated from the edge cutset. Then, edge set $(EC_f \backslash EC) \cup \{u_{i_1}w_1$, $u_{i_2}w_1$, ..., $u_{i_m}w_1\}$ separates $V(C_1)$ and $V(G_1) \backslash V(C_1)$. Hence there is a k-edge cutset containing f in G_1 contradicting with $\lambda(G_1) = \lambda(G_1 - f)$.

Theorem 3. *Let G_1, G_2 be two minimally k-edge-connected graphs for odd k such that $V(G_1) \cap V(G_2) = \emptyset$. Let $w_1 \in V(G_1)$ and $w_2 \in V(G_2)$ such that $degree(w_1) = degree(w_2) = k$. Suppose the edges adjacent with w_1 are $u_1w_1, u_2w_1, \ldots, u_kw_1$ and the edges adjacent with w_2 are $v_1w_2, v_2w_2, \ldots, v_kw_2$ (parallel edges are allowed). Construct graph $G = (V,E)$ be the graph such that $V(G) = V(G_1) \cup V(G_2) - \{w_1, w_2\}$ and $E(G) = E(G_1) \cup E(G_2) - \{u_1w_1,\ u_2w_1,\ \ldots,\ u_kw_1\} - \{v_1w_2,\ v_2w_2,\ \ldots,\ v_kw_2\} + \{u_1v_1,\ u_2v_2,\ \ldots,\ u_kv_k\}$. Then G is minimally k-edge-con-nected.*

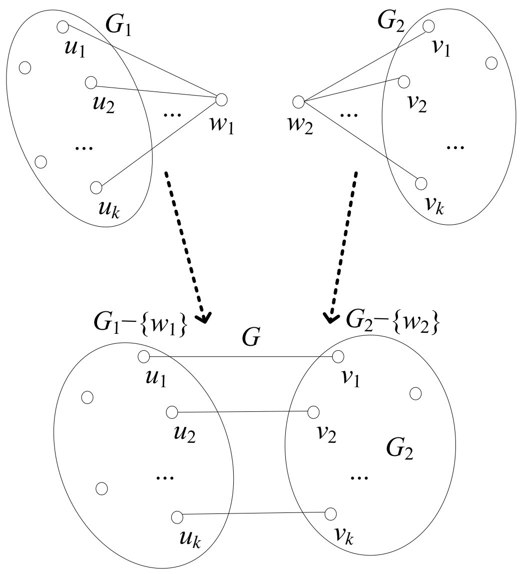

Figure 3 An example of construction.

Proof. Suppose it does not hold and G is not minimally k-edge-connected. Denote $\{u_1v_1, u_2v_2, ..., u_kv_k\}$ by EC and denote $\{u_1w_1, u_2w_1, \ldots, u_kw_1\}$ by $\delta(w_1)$.

Case 1: if G is not k-edge-connected, then there is an edge cutset whose size is smaller than k, denoted by EC'. Without loss of generality, suppose $|EC' \cap E(G_1)| \leq |EC' \cap E(G_2)|$. Since $|EC'| < k$ and k is odd, we have $|EC' \cap E(G_1)| \leq (k-1)/2$. Let C_1,C_2 be the two components of $G - EC'$. Without loss of generality, suppose $|E(C_1) \cap EC| \leq |E(C_2) \cap EC|$. Since $|EC| = k$ and k is odd, we have $|E(C_1) \cap EC| \leq (k-1)/2$. Thus, $(EC' \cap E(G_1)) \cup (E(C_1) \cap EC)$ is an edge cutset whose size is smaller than k. Furthermore, $(EC' \cap E(G_1)) \cup \delta(w_1)$ is an edge cutset in G_1. Suppose $E(C_1) \cap EC$ =$\{u_{i_1}v_{i_1}$, $u_{i_2}v_{i_2}$, ..., $u_{i_n}v_{i_n}\}$, $n \leq (k-1)/2$. The edge (u_j,w_1), for $j \notin \{i_1, i_2, ..., i_n\}$ is not incident with any vertex of C_1, so it can be removed from the edge cutset. Hence, $(EC' \cap E(G_1)) \cup$ $\{u_{i_1}w_1$, $u_{i_2}w_1$, ..., $u_{i_n}w_1\}$ is an edge cutset in G_1. Its size is smaller than k. Contradict!

Case 2: G is k-edge-connected. There is an edge $f \in E(G)$ could be deleted (i.e. $\lambda(G) = \lambda(G-f)$). We have $f \notin EC$; otherwise, $EC - f$ is a $(k-1)$-edge cutset in G. Without loss of generality, suppose $f \in E(G_1)$. Since G_1 is minimally k-edge-connected, there is a k-edge cutset, denoted by EC_f, containing f in G_1. If $EC_f \cap \delta(w_1) = \emptyset$, then $EC_f - f$ is also a $(k-1)$-edge cutset in $G-f$. Thus, $\lambda(G-f) = k-1 < \lambda(G-f)$. Contradict!

If $EC_f \cap \delta(w_1) \neq \emptyset$, suppose $EC_f \cap \delta(w_1)$=$\{u_{i_1}w_1, u_{i_2}w_1, ..., u_{i_m}w_1\}$, $m \leq k-1$. Let D_1,D_2 be the two components of $G - EC_f$. Without loss of generality, suppose D_1 contains u_{i_1}, u_{i_2}, ..., u_{i_m}. $V(D_1)$ is separated from $V(G_1)$ by $(EC_f \backslash \delta(w_1)) \cup$ $\{u_{i_1}w_1, u_{i_2}w_1, ..., u_{i_m}w_1\}$. Thus, $V(D_1)$ is separated from $V(G)$ by $(EC_f \backslash \delta(w_1)) \cup EC$. Since the edge (u_j, v_j) for $j \notin \{i_1, i_2, ..., i_m\}$ is not incident with any vertex of D_1, it can be omitted from the edge cutset. $(EC_f \backslash \delta(w_1)) \cup$ $\{u_{i_1}v_{i_1}, u_{i_2}v_{i_2}, ..., u_{i_m}w_{i_m}\}$ separates $V(D_1)$ and $V(G) \backslash V(D_1)$. Hence there is a k-edge cutset containing f in G contradicting with $\lambda(G) = \lambda(G-f)$.

By Theorem 2 and Theorem 3, we have the following corollary.

Corollary 1. *Let G be a simple graph and EC be a non-trivial k-edge cutset of G where k is odd. Let G_1, G_2 be the two graphs obtained by separating G through EC. Then, G is minimally k-edge-connected if and only if G_1 and G_2 are minimally k-edge-connected.*

4. Algorithm description

Designing efficient algorithms for determining the connectivity of graphs has been a subject of great interest during the last two decades. However few of them are devised for testing the minimality to certain edge-connected graphs. Current algorithms for this purpose are developed basing on the definition of “minimality”; that is, every edge is examined whether it can be deleted. We denote this algorithm by “Edge Enumeration Algorithm (EEA)”. It is clear that the time complexity of EEA depends on the procedure to find k-edge cutsets. Furthermore, there is no special technique improved to the effectiveness of these algorithms by far.

Edge Enumeration Algorithm

```
1 Initialize all the edges to be “unchecked”
2 FOR (an unchecked edge e and e is not incident a vertex whose degree is 3) DO
3 {
4     Delete e from G; //just need to disable it
5     Find a $(k-1)$-edge cutset of G, denoted by EdgeCutset;
6     IF EdgeCutset is empty
7     THEN G is not minimally, halt;
8     ELSE {
9         Mark e and the edges in EdgeCutset “checked”;
10        Add e from G; //enable it
11    }
12 } //FOR
13 G is minimally 3-edge-connected.
```

Based on Corollary 1, we can determine whether graph G is minimally k-edge-connected by separating G into two subgraphs through a k-edge cutset. Therefore, a divide-and-conquer algorithm is developed as follows.

Our Algorithm

```
1 Initialize all the edges to be ”unchecked”
2 IF IS-M3EC(G) THEN graph G is minimally k-edge-connected;
3 ELSE graph G is not minimally k-edge-connected.
```

FUCNTION IS-M3EC(G) :boolean

```
1 FOR (an unchecked edge e which is not incident a vertex whose degree is 3) DO
2 { Delete e from G; //just need to disable it
3 Find a $(k-1)$-edge cutset of G, denoted by EdgeCutset;
4 IF EdgeCutset is empty
5     THEN G is not minimally, return FALSE;
6     ELSE { separate G into $G_1$ and $G_2$;
              //Operate on adjacency lists
7             return IS-M3EC($G_1$) && IS-M3EC($G_2$); }
8} //FOR
```

5. Time complexity and experimental results

The separating operation just needs to alter six values in adjacency lists and add six entries (edges) to them, which is $O(1)$ time. Furthermore, the sum of unchecked edges without separating operation is the same as that using it. Since the graph is separated iteratively in our divide-and-conquer strategy, for each unchecked edge the time to search k-edge cutsets is less than Edge Enumeration Algorithm (EEA). In the worst case, the time complexities of these two algorithms are almost the same. However, in the best case, the time complexity of our divide-and-conquer algorithm is a logarithmic of EEA’s running time, in theory. That is, suppose the time complexity of search k-edge cutsets is $O(\varphi(m,n))$, EEA runs in $O(m \times \varphi(m,n))$ time while our algorithm costs $O(\log m \times \varphi(m,n))$ time in the best case. Thus, we perform experiments to evaluate the running time of these two algorithms in average case.

For comparison, we assign EEA and our algorithm to employ the same algorithm of finding a k-edge cutset. Since there are many studies on searching 2-edge cutset, we choose $k = 3$. The best algorithm for searching 2-edge cutset is proposed by Tsin [13]. Thus, we implement this algorithm in our experiments.

All the experiments were conducted on a PC with an AMD Athlon Processor 3600+ and a 1.5 GB memory. The programs are written in C++. We generated the graphs of which the numbers of vertices are in the range of 250 $\sim$ 3000. For each vertex count, we generated 1000 graphs which are minimally 3-edge-connected graphs. The average numbers of edges and the average running time are shown in Table 1. The average numbers of edges are round to integer. The time unit is millisecond.

The results show that our algorithm is much faster than EEA. Our algorithm costs only 1 percent running

Table 1 Running time on minimally 3-edge-connected graphs.

Nodes	Edges	Our algorithm	EEA
250	456	31	1359
500	917	62	6031
750	1357	125	15516
1000	1828	187	25031
1250	2271	359	40063
1500	2731	516	61766
1750	3177	624	86391
2000	3640	672	112906
2250	4079	1047	141500
2500	4520	1197	173765
2750	5008	1172	220500
3000	5437	1218	259591

time of EEA in average. We next present some insight statistical values to reveal the effectiveness of our algorithm. As mentioned above, we delete one edge temporarily at first. Secondly, we search a 2-edge cutset. Then, we separate the graph into two subgraphs, and check the minimally 3-edge connectivity in these two subgraphs. In addition, we employ Tsin's algorithm [13] to find a 2-edge cutset which costs linear time of the input graph's size. Hence, the computing time of our algorithm is determined by the sum of the size of all the processed subgraphs. We take one graph with 250 vertices as an example. In each separate step, we record the size of the subgraphs, shown in Table 2. In this graph, the sum of the size is 1939. In EEA, we perform Tsin's algorithm 381 times. Thus, the sum of the size is $381 \times 250 = 95250$. It is clear that the proportion of the sum of the size $\frac{1939}{95250} \approx 0.020$ coheres to the proportion of the running time $\frac{31}{1359} \approx 0.023$.

Table 2 Separate sequence of one graph with 250 nodes.

Separate Step	0	1	2	3	4	5	6
$Subgraph_1$	250	55	38	4	30	15	10
$Subgraph_2$	0	195	18	35	6	16	7
Separate Step	7	8	9	10	11	12	13
$Subgraph_1$	7	192	46	24	21	136	111
$Subgraph_2$	4	4	147	23	4	12	26
Separate Step	14	15	16	17	18	19	20
$Subgraph_1$	69	24	43	38	30	12	6
$Subgraph_2$	43	46	4	6	9	19	4
Separate Step	21	22	23	24	25	26	
$Subgraph_1$	16	8	4	22	22	19	
$Subgraph_2$	28	9	6	7	5	4	

By the above analysis, we can obtain the improvement of running time from the total size of the processed subgraphs. This improvement is independent with the algorithm which is employed to search the 2-edge cutset, since our algorithm reduces the size of the subgraphs which are the input of the algorithm searching 2-edge cutset. For better evaluation, we compute these total size for all test graphs. The average total size of our algorithm and EEA are shown in Table 3. The numbers of separating operation are round to integer.

Table 3 Total size comparison

Number of Vertices	Total Size of Our Algorithm	Total Size of EEA
250	1941	95250
500	4082	313500
750	7179	768000
1000	10097	1286000
1250	14464	2041250
1500	17834	2629500
1750	22396	3494750
2000	27226	4406000
2250	33503	5618250
2500	41619	6767500
2750	39498	8544250
3000	42259	9918000

Table 4 Number of separating operations

Number of Vertices	Number of Separating Operations
250	27
500	51
750	80
1000	105
1250	135
1500	169
1750	185
2000	222
2250	252
2500	279
2750	306
3000	330

In addition, we also present the average numbers of separating operation performed in our algorithms by Table 4. The values show that the number of separating operations is about 10 percent of the number of vertices. Since the worst case of separating operation is to divide one subgraph of which the number of vertices is 2, the number of separating operations equals one half of the number of vertices of the original graph in this case. The comparison of the average of our algorithm and the worst case is illustrated by Fig. 5.

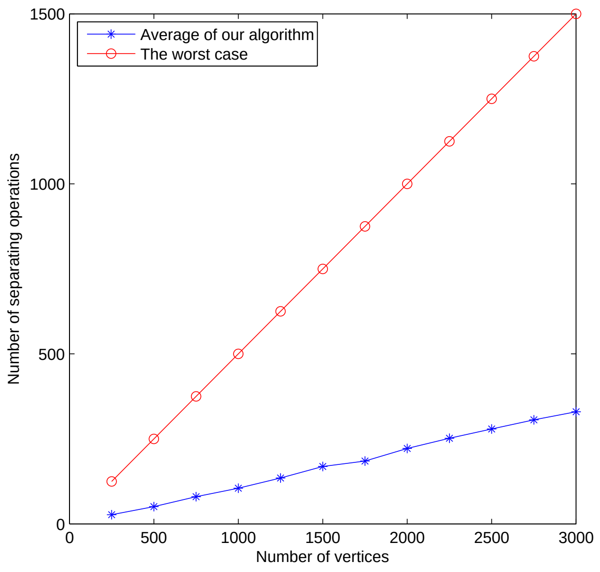

Figure 4 Numbers of separating operations of our algorithm.

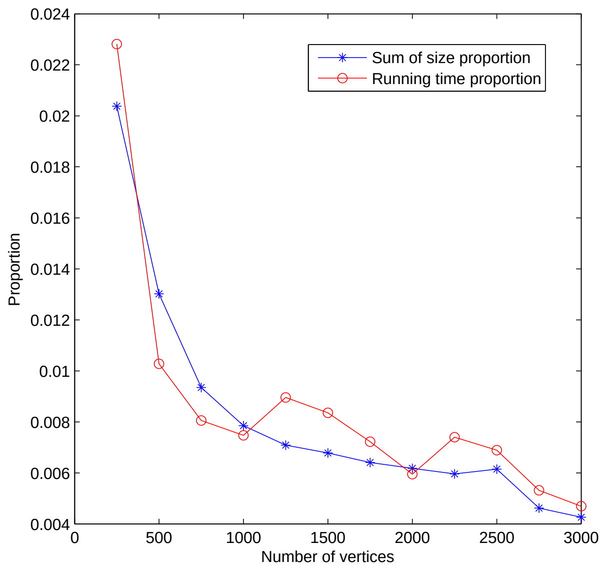

Figure 5 Running time proportion and total size proportion.

By the results, we conclude that our algorithm significantly reduces the total size which is the input scale. In addition, we also compute the growing proportion of the total size of our algorithm over EEA. We compare this total size proportion with the running time proportion. Two plots are illustrated in Fig. 5. It appears that the trends of these two plots are approximately the same. Therefore, the improvement of running time is mainly by the reduction of total size which is achieved by separating operation.

6. Conclusions and future works

We present a necessary and sufficient condition of one graph to be minimally k-edge-connected with odd k. Based on this result, divide-and-conquer algorithms for determining minimally k-edge-connected can be developed. We propose a new algorithm for checking minimality of 3-edge-connected graph. The experimental results show that our algorithm is much more effective than the current one developed by the definition of minimality.

It is clear that a parallel algorithm can be developed based on our result. The results of our experiments have shown that separating operation significantly reduces the total size of 2-edge cutset searching algorithm. In addition, the subgraphs after separating have no common used data structure. Thus, a parallel version of our algorithm can be implemented. Therefore, the running time can be reduced further.

Corollary 1 characterizes the structure of minimally k-edge-connected graphs for odd k. These graphs can be separated recursively until the resulting graphs are irreducible. A graph is irreducible if every edge lies in a trivial k-edge cutset. Furthermore, if we remove the k-degree vertices, the resulting graphs are singletons. So some techniques applied in k regular graphs might be used to the irreducible graphs.

Acknowledgement

This research was supported in part by NSFC under Grants No.61100190 and No.61272538 , National Key Technology R&D Program of MOST China under Grant No. 2012-BAK17B08, Shenzhen Strategic Emerging Industries Program under Grants No. ZDSY20120613125016389 and No.JCYJ20120613135329670, and Natural Scientific Research Innovation Foundation in HIT under Grant No. HIT. NSFIR.2010128.

References

[1] J.E. Hopcroft and R.E. Tarjan, Dividing a graph into triconnected components. SIAM J. Comput. **2**, 135-158 (1973).

[2] H. Nagamochi and T. Ibaraki, A linear time algorithm for computing 3-edge connected components in a multigraph. Japan J. Indust. Appl. Math. **8**, 163-180 (1992).

[3] J. Negele and H. Orland, Quantum Many-Particle Systems, Addison Wesley (1988).

[4] N. Nakanishi, Graph Theory and Feynman Integrals, Gordon and Bridge Science Publishers (1971).

[5] T. Watanabe and M. Yamakado, A linear time algorithm for smallest augmentation to3-edge-connect a graph. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, **76**, 518-531 (1993).

[6] T.S. Hsu and V. Ramachandran, A linear time algorithm for triconnectivity augmentation. Proceeding of the 32nd Annual Symposium of Foundations of Computer Science (FOCS), 548-559 (1991).

[7] S. Taoka, T. Watanabe and K. Onaga, A linear time algorithm for computing all 3-edge-connected components of a multigraph. IEICE Trans. Fundamentals E, **75**, 410-424 (1992).

[8] H.N. Gabow, M. X. Goemans, E. Tardos and D. P. Williamson, Approximating the smallest k-edge connected spanning subgraph by LP-rounding, Networks **53**, 345-357 (2009).

[9] M.C. Cai, The Number of Vertices of Degree k in a Minimally k-Edge-Connected Graph. Journal of Combinatorial Theory Series B, **58**, 225-239 (1993).

[10] Q. Liu, Y. Hong and Z. Zhang, Minimally 3-restricted edge connected graphs. Discrete Applied Mathematics **157**, 685-690 (2009).

[11] O. Goldreich and D. Ron, Property testing in bounded degree graphs. Algorithmica, **32**, 302-343 (2002).

[12] Y.H. Tsin, A simple 3-edge-connected component algorithm. Theory Comput. Syst. **40**, 125-142 (2007).

[13] Y.H. Tsin, Yet another optimal algorithm for 3-edge-connectivity. Journal of Discrete Algorithms **7**, 130-146 (2009).

[14] A.M. Saifullah and A. Ungor, A simple algorithm for triconnectivity of a multigraph. Proceedings of the Fifteenth Australasian Symposium on Computing: The Australasian Theory, 53-62 (2009).

[15] J.A. Bondy and U.S.R. Murty, Graph Theory with Application, Macmillan (1976).

Yunming Ye received the Ph.D. degree in Computer Science from Shanghai Jiao Tong University. He is now a professor in the Shenzhen Graduate School, Harbin Institute of Technology. His research interests include data mining, text mining, and ensemble learning algorithms.

Yueping Li received his PhD in Computer Science from Sun Yat-sen University in 2008. Currently, he is a post doctor in Shenzhen Graduate School, Harbin Institute of Technology. His research interests involve web mining, graph algorithm and optimization.