A polynomial algorithm for cyclic edge connectivity of cubic graphs

Dingjun Lou Lihua Teng Xiangjun Wu

Department of Computer Science
Zhongshan University, Guangzhou 510275
People's Republic of China

Abstract

In this paper, we develop a polynomial time algorithm to find out all the minimum cyclic edge cutsets of a 3-regular graph, and therefore to determine the cyclic edge connectivity of a cubic graph. The algorithm is recursive, with complexity bounded by $O(n^3 \log_2 n)$. The algorithm shows that the number of minimum cyclic edge cutsets of a 3-regular graph G is polynomial in $\nu(G)$ and that the minimum cyclic edge cutsets can be found in polynomial time, and so the cyclic edge connectivity of G can be calculated.

0. Introduction

For a connected graph G, a vertex set S is said to be a vertex cutset of G, if G-S is not connected. The connectivity $\kappa(G)$ of G is the minimim cardinality of all the vertex cutsets of G. Similarly, an edge cutset E of G is an edge set such that G-E is not connected, and the edge connectivity $\lambda(G)$ of G is the minimum cardinality of all the edge cutsets of G. Here we are going to discuss another type of connectivity of a graph, the cyclic edge connectivity, which is defined below.

For a connected graph G, a cyclic edge cutset is an edge cutset whose deletion disconnects the graph and such that two of the components created each must contain at least one cycle. The cyclic edge connectivity $c\lambda(G)$ is the minimum cardinality of all the cyclic edge cutsets of G. If no cyclic edge cutset exists, we set $c\lambda(G)=0$. In this paper, we consider only simple, undirected graphs. All terminology and notation not defined in the paper can be found in [2].

The concept of cyclic edge connectivity was introduced by Tait[10] and studied, in particular, by Plummer[8], for planar graphs, with a slight difference: if no cyclic edge cutset exists, they set $c\lambda(G)=\infty$. In references [4], [5] and [7], the relation between cyclic edge connectivity and n-extendability of graphs is studied. In a paper of Peroche[9], several sorts of connectivity, including cyclic edge connectivity, and their relations are studied. The following upper bound for the cyclic edge connectivity of a graph G is given there.

Theorem 1: If G=(V, E) is a simple graph with |V|=n, then $c\lambda(G) \leq 3(n-3)$, for $n \geq 6$, and the bound is sharp. Equality holds when $G=K_n$.

How to compute the cyclic edge connectivity of an arbitrary graph has not been studied in the literature as far as we know. Even for cubic graphs, the distribution of minimum cyclic edge cutsets is unknown. If there are polynomially many such cutsets in a cubic graph, is it possible to find them in polynomial time? This paper presents a recursive algorithm to find all the cyclic edge cutsets of an input cubic graph, with time complexity bounded by $O(n^3 \log_2 n)$.

In this paper, we are going to develop a polynomial time algorithm that computes the cyclic edge connectivity of a cubic graph. We use the concept of removing an edge from a 3-connected graphs. This was introduced and studied by Barnette and Grünbaum[1]. The distribution of removable edges in 3-connected graphs was studied by Holton, Jackson, Saito and Wormald[3].

In the first and second sections, we introduce a necessary and sufficient condition for a cubic graph to have a cyclic edge cutset. Then the concept of removing an edge from a 3-connected graph is presented, and how the removed edge is used in the algorithm to help compute the cyclic edge connectivity is discussed. In the third section, an algorithm that returns all the minimum cyclic edge cutsets of a given cubic graph is described. In the fourth section, we give an example of applying the algorithm. We find all the minimum cyclic edge cutsets of the Petersen graph, and show that the cyclic edge connectivity of the Petersen graph is 5. In the last section, the time complexity of the recursive program is analysed.

1. Preliminaries

Firstly, we give a necessary and sufficient condition for a cubic graph to have a cyclic edge cutset.

Theorem 2: Let G be a 3-regular graph of order υ, let g be the girth of G. Then G has a cyclic edge cutset if and only if $\upsilon > 2g-2$.

Proof.

If G has a cyclic edge cutset S, then let C be one of the cycles in G-S of length c. By the definition of cyclic edge cutset, G-V(C) must contain at least one cycle. Then, we have

$$3(\upsilon - c) - c > 2(\upsilon - c - 1)$$
$$\upsilon > 2c - 2 \geq 2g - 2.$$

Conversely, if $\upsilon > 2g-2$, let C be a minimum cycle in G, then

$$3(\upsilon - g) - g > 2(\upsilon - g - 1).$$

hence G-V(C) must contain at least one cycle and (V(C), V(G)\V(C)) is a cyclic edge cutset of G. □

Now, we consider the cyclic edge connectivity of a cubic graph with vertex connectivity of 1 or 2.

Lemma 3: Let G be a 3-regular graph.

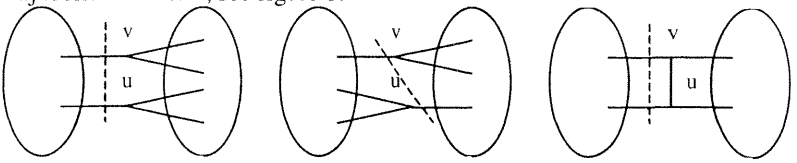
(1) If $\kappa(G)=1$ and $c\lambda(G)>0$, then $c\lambda(G)=1$;

(2) If κ(G)=2 and $c\lambda(G)>0$, then $c\lambda(G)=2$.

Proof.

If κ(G)=1 and $c\lambda(G)>0$, let v be a cut vertex of G. Then one of the components of G-v must be connected to v by only one edge e. So {e} is a cyclic edge cutset, since 3n-1>2(n-1), where n is the cardinality of either component of G-e, then either component of G-e has a cycle. Hence $c\lambda(G)=1$.

If κ(G)=2 and $c\lambda(G)>0$, let {u, v} be a vertex cut of G. Then there are three cases for the edges adjacent to u and v, see figure 1.



case 1 case 2 case 3

Figure 1

In each of the cases, there is at least one edge cutset S of cardinality 2. And such a cutset is a cyclic edge cutset, since 3n-2>2(n-1), where n is the cardinality of either component of G-S, then either component of G-S has a cycle. And since $c\lambda(G)\geq\lambda(G)\geq\kappa(G)=2$, we have $c\lambda(G)=2$.□

We now consider the concept of removing an edge from a 3-connected graph, as introduced and studied by Barnette and Grünbaum[1].

Let G be a 3-connected graph and e be an edge of G. We consider the following operation:

(1) Delete e from G to get G-e;

(2) If some endvertices of e have degree two in G-e, then suppress them;

(3) If multiple edges occur after (2), then replace them by single edges to make the graph simple.

The resulting graph is denoted by G⊖e and G⊖e is said to be obtained by removing e from G.

2. Description of the algorithm

The algorithm we are going to introduce is a recursive algorithm, it returns all the minimum cyclic edge cutsets of the input 3-regular graph G. In our algorithm, we make use of the linear algorithm[5] to divide a graph into 3-connected components, denoted by "triconnect".

Firstly, our algorithm checks if the input graph G is K_4. If it is, then the empty set will be returned, since K_4 has no cyclic edge cutsets. If κ(G)=1 or κ(G)=2, by lemma3, the cyclic edge connectivity is 1 or 2 respectively, and all the cyclic edge cutsets can be found by checking all the minimum vertex cutsets. If G is 3-connected and G is not K_4, then an edge e of G will be removed from G so that G⊖e is still 3-regular. If the resulting graph G⊖e is 3-connected, then the program will be recursively called with the input

graph G⊖e instead of G. After all the minimum cyclic edge cutsets of G⊖e have been returned, we compare them with the minimum cyclic edge cutsets of G. Then we find out and return all the minimum cyclic edge cutsets of G at the end of the algorithm. If after removing e from G, the resulting graph G⊖e has κ=2, then in the next recursive step, all the minimum cyclic edge cutsets of G⊖e will be returned, and then we can get those of G.

The main problem remaining is the relation between the minimum cyclic edge cutsets of G⊖e and those of G. We discuss this in three cases. Let S be a cyclic edge cutset of G, e be the removed edge in the algorithm, and S' be a cyclic edge cutset of G⊖e.

Case1:

If e is contained in the only cycle C in one of the components of G-S, say A, then (G⊖e)-S has a component which does not contain any cycle, hence the cyclic edge cutset S of G is not a cyclic edge cutset of G⊖e. If S is minimum, we have to consider it after the recursive calling on G⊖e. This case is illustrated in Figure 2.

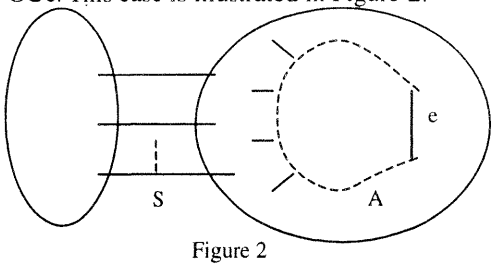


Figure 2

If C is a cycle as above, then (V(C), V(G)\V(C)) is the minimum cutset which satisfies that C is the only cycle in A, since G is 3-regular, and |(V(C), V(G)\V(C))|=|V(C)|. If there is more than one such cutset, the ones with cardinality of the length of the minimum cycles are those we need, others will be of larger cardinality. So, we can find all such cutsets by finding all the minimum cycles that contain e. To achieve this, we use a width-first search of the graph, starting from the two endpoints of e simultaneously, until an edge connecting the two sub-trees occurs, and each such edge together with the paths in the sub-trees forms a minimum cycle containing e.

If a cycle appears before the edge connecting two sub-trees occurs, then min|V(C)|>g, where g is the girth of G, and g is equal to or larger than $c\lambda(G)$, so no such cutsets are minimum cyclic edge cutsets, and we discard them.

Case2:

If for any minimum cyclic edge cutset S' of G⊖e, S' ∪ {e} is a cyclic edge cutset of G satisfying that the components of G⊖e-S' and G-(S' ∪ {e}) are the same, then the cyclic edge connectivity of G may be larger than G⊖e by 1 if in Case1 it does not find smaller cutsets. This case is illustrated in Figure 3. We determine whether the two endpoints of e lie in different components of G⊖e-S' by finding a path P connecting the

two endpoints, then see if P passes through an odd number of edges in S'.

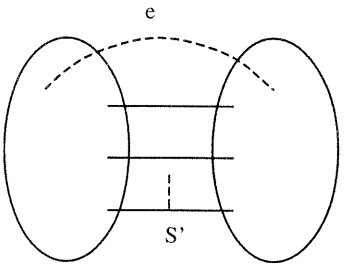


Figure 3

Case3:

Let e=xy. If there are $e_1, e_2 \in S'$ such that e_1 is obtained by suppressing x during the operation of removing e from G, and e_2 is obtained by suppressing y, then S' corresponds to two cyclic edge cutsets of G, S_1 and S_2, where e belongs to the two components A', B' of $G\ominus e$-S' respectively. This case is illustrated in Figure 4.

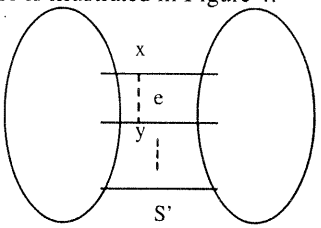


Figure 4

In the following section, we will describe the algorithm in more detail using pseudo-PASCAL code.

3. The algorithm for cyclic edge connectivity of cubic graphs

Input: a 3-regular graph G
output: all the minimum cyclic edge cutsets of G

```
Cyclic_connectivity(G):
0)  if G=K4 then return null;
1)  if triconnect(G)=false  {G is not 3-connected}
2)   then
3)     if κ(G)=1 then
4)        if υ>2*g(G)-2 then      {O(υ²), but this may be run only once during the
     algorithm}
5)        begin
6)          get all the cut vertices of G;  {See[6]}
7)          S:={Si | Si is a cut-edge derived from a cut vertex in the last step };
8)          return S;
```

```
9)        endthen
10)      else return null
11)    else      {κ(G)=2}
12)        if  υ>2g(G)-2 then    {O(υ²), but this may be run only once during the
  algorithm}
13)       begin
14)         divide G into 3-connected components;
15)         get all the vertex cutsets of order 2;     {See[6]}
16)          S:={Si | Si is an edge cutset of order 2 derived from a vertex cutset in the
  last step };
17)         return S;
18)       endthen
19)      else return null;
20) choose an edge e=xy of G;   {G is 3-connected, if we reach here}
21) find out all the minimum cycles Ci that contain e, let c:=|Ci|, and C:={(V(Ci),
  V(G)\V( Ci ) )}; { Case 1, see function Min_cycle(x,y), which carries out the task
  in detail and is described after the main procedure }
22) G':=GΘe;
23) S:=Cyclic_connectivity(G'), and s:=|Si|, ∀Si ∈S   {the algorithm is recursively
  called}
24) if S=Φ then
25)  if υ>2c-2 then return C
26)   else return null;
27) flag:=true;
28) for each Si∈S do
29)  begin
30)   find out an xy-path P in G, which is not the edge e;
31)   if |P∩Si| is odd
32)    then
33)      flag:=flag∧true;  {x and y lies in different components of GΘe -Si}
34)    else begin   {|P∩Si| is even}
35)         flag:=flag∧false;
36)          mark Si as "remain";
37)            endelse
38)  endfor;
39) if flag=true    { Case 2 }
40)  then if c=s then return C    { smaller cyclic edge cutsets are found in Case 1 }
41)  else begin        {c>s, if we reach here}
42)       S:={Si∪{e}| Si∈S };
43)       if c=s+1 then return C∪S ;
44)       else return S;
45)     endelse;
46) S:={Si | Si∈S and Si is marked as "remain"}
47) For each Si∈S do
48)  If there are Si1, Si2 such that Si1, Si2 are obtained from Si as in Case3
```

```
49)    then S:=S-{S_i}+{ S_i1, S_i2};    { Case 3 }
50) if c=s
51)  then return C∪S
52)  else return S;    {c>s, if we reach here}
53) end-of-program.
```

In step 20 of the main procedure, we must maintain the 3-regularity of G⊖e, so that the recursive calling of the algorithm can be continued. To achieve this, we choose an edge e such that neither of its endpoints is contained in a triangle. If the first edge we consider has one endpoint in a triangle, we can choose an edge in this triangle, and this edge must satisfy the requirement, or else G is not 3-connected. This operation of choosing an edge e to remove can be done in constant steps.

The following Function is to find out all the minimum cycles that contain the edge e=xy. This is carried out by simutaneously growing width-first search trees from x and y, one level during each loop, see Figure 5 and Figure 6. The vertices in the left tree are marked as "left" while the ones in the right tree are marked as "right". If minimum cycles are found, the both trees do not need to grow any further, see Figure 5 and Figure 6. The algorithm is described in pseudo-PASCAL as below.

```
Function Min_cycle(x,y):
 0)  Begin
 1)  mark x as "left";
 2)  mark y as "right";
 3)  push x to queue1;
 4)  push y to queue2;
 5)  C:=null;  {C is the set of the minimum cycles}
 6)  i:=0;    {i is the level of the width-first search "tree"}
 7)  even:=false;  {true if minimum cycles of even length are found}
 8)  odd:=false;  {true if minimum cycles of odd  length are found}
 9)  While not(even) and not(odd) do
 10)  Begin
 11)   for j=1 to 2^i do
 12)    begin
 13)     r_1:=dequeue(queue1);
 14)     for each vertex v adjacent to r_1 do
 15)     if v is not marked
 16)      then mark v as "left" and enqueue(queue1, v);
 17)      else if (v is marked "left") and (v is not the father of r_1)
 18)           then
 19)             return null   {no such cyclic edge cutsets are minimum}
 20)           else begin   {see Figure 5, v is marked "right"}
 21)              even:=true;
 22)              C:=C∪{the cycle containing (r_1,v)};
 23)             endelse
 24)    endfor
 25)   if not(even)
```

```
26)      then for j=1 to 2^i do
27)           begin
28)            r2:=dequeue(queue2);
29)            for each vertex v adjacent to r2 do
30)            if v is not marked
31)              then mark v as "right" and enqueue(queue2, v);
32)              else if (v is marked "right") and (v is not the father of r2)
33)                   then return null  {no such cyclic edge cutsets are minimum}
34)                   else begin    {see Figure 6, v is marked "left"}
35)                        odd:=true;
36)                        C:=C ∪ {the cycle containing (r2,v)};
37)                   endelse
38)      endfor;
39)     i:=i+1;
40)   endwhile
41) return C;
42) End-of-function
```

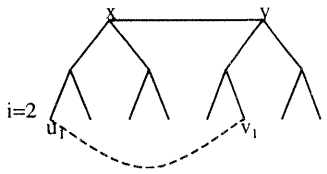


Figure 5

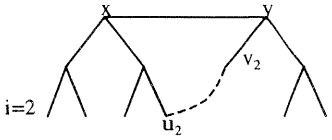


Figure 6

Theorem 4: The algorithm Min_cycle(x,y) returns all the minimum cycles that contain edge (x,y).

Proof.

The algorithm Min_cycle(x,y) proceeds by growing two width-first search trees simultaneously. Since the trees are contructed by simultaneously applying width-first search of a graph, when an edge e=uv in E(G)\E(T) occurs, its two ends must lie in the same level (from left tree to right) or lie in two successive levels (from right tree to left). Otherwise, it joins two vertices in the same tree (left or right).

If u and v are in the same level, e will be found while extending the next level in the left tree, and all such edges can be found after extending this level in the left tree. So if an even cycle is found, we don't need to extend the right tree any more, see line 24). Each of these edges corresponds to an even minimum cycle containing edge (x,y).

If u and v are in successive levels, u in the left tree and level(u)>level(v), then e will be found when extending the (u)th level in the right tree and all such edges can be found after extending this level in the right tree. Each of these edges corresponds to an odd minimum cycle containing edge (x,y).

If we find an edge joining two vertices in the same tree (left or right) before we find an edge joining two vertices in different trees, then we find a cycle shorter than any cycle containing e=xy. When the input graph G has cyclic edge cutsets, i.e. $v > 2g-2$ by Theorem 1, the edges incident with a shortest cycle form a cyclic edge cutset, hence

$c\lambda(G) \leq g$ and the edges incident with a minimum cycle containing e=xy do not form a minimum cyclic edge cutset. That is why in line 17)-19) and line 32)-33) of Function Min_cycle(x, y), we return null instead of going further to find cycles containing e=xy. □

Theorem 5: The algorithm Cyclic_connectivity(G) returns all the minimum cyclic edge cutsets of G.

Proof.

The algorithm proceeds by comparing the minimum cyclic edge cutsets of G⊖e and those of G.

Each time the program is recursively called, the input graph is smaller than the original one by two vertices. If the recursive calling continues, it will certainly stop when the input graph is 2-connected or it has shrunk to K_4. So the algorithm will necessarily terminate with any input 3-regular graph.

The relation between the minimum cyclic edge cutsets of G⊖e and those of G is described in the second section.

In Case1, let C be a minimum cycle containing the removed edge, then S=(V(C),V(G)\V(C)) is a minimum edge cutset of Case 1, i.e. C is the only cycle in one component A (actually A=C) of G-S. If there are other edges in A, the component A will be a forest, in which the trees have their roots in C, and the edge cutset (V(A), V(G)\V(A)) must contain more edges than |V(C)|. So it is sufficient to find all the minimum cycles containing the removed edge to form the component A, which is described in the Function Min_cycle(x, y).

In Case2 and Case3, some steps are taken to convert certain cyclic edge cutsets of G⊖e into those of G. Others are cutsets both of G⊖e and G, and they will remain as they are returned. Finally, we should compare the edge cutsets in Case1 with those derived from G⊖e, so that the program returns the minimum among them, this is done in lines 40), 43), 50) in the main procedure. □

4. Cyclic edge cutsets of the Petersen graph found by applying the algorithm

Here we apply the algorithm provided above to the well-known 3-connected, 3-regular graph, the Petersen graph. We get 6 minimum cyclic edge cutsets. The first two edge cutsets are found by adding (v_1,v_6) to the minimum cyclic edge cutsets of G⊖ (v_1,v_6), which is Case 2 of the algorithm; and the other four cutsets are found by finding 4 minimum cycles containing (v_1,v_6), which is Case 1 of the algorithm. We will run through the algorithm step by step.

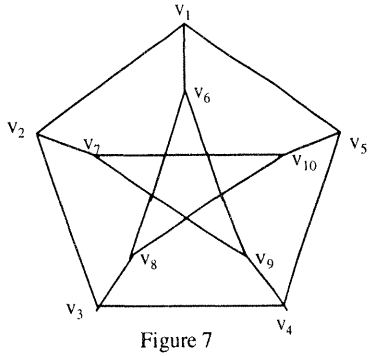


Figure 7

1) We choose (v_1,v_6) as the edge to remove, then G⊖e will look as the graph in Figure 8;

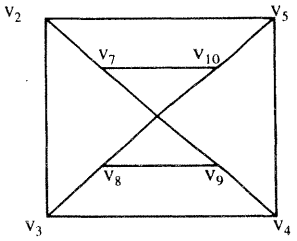


Figure 8

2) Since the graph is still 3-connected, so we continue to choose an edge (v_2,v_5) to remove, and G⊖e will be $K_{3,3}$, which is shown as below, removing (v_7,v_{10}) will result in K_4 which consists of v_8, v_9, v_3 and v_4.

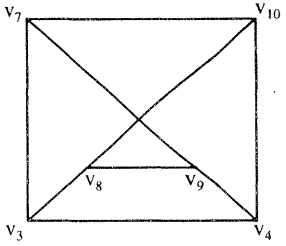


Figure 9

3) When the graph is K_4, the algorithm returns an empty set, since K_4 has no cyclic edge cutsets. In the graph in Figure 9, there are four minimum cycles contain (v_7,v_{10}), they are $v_7v_3v_4v_{10}$, $v_7v_3v_8v_{10}$, $v_7v_9v_4v_{10}$ and $v_7v_9v_8v_{10}$. In line 21 in the algorithm, c is 4, and in line 23, S is an empty set. Since $v>2c-2$ does not hold, in line 25 the algorithm returns an empty set to the caller.

4) In the graph in Figure 8, there are two minimum cycles containing (v_2,v_5), they are

$v_2v_7v_{10}v_5$ and $v_2v_3v_4v_5$. This time S is still an empty set, and υ>2c-2 holds, in line 25 the algorithm returns the set C= { $((v_2,v_3), (v_7,v_9), (v_{10},v_8), (v_5,v_4))$, $((v_2,v_7), (v_3,v_8), (v_4,v_9), (v_5,v_{10}))$ }.

5) In the original graph, there are four minimum cycles containing (v_1,v_6), they are $v_1v_2v_3v_8v_6$, $v_1v_2v_7v_9v_6$, $v_1v_5v_{10}v_8v_6$ and $v_1v_5v_4v_9v_6$. Figure 10 shows how the minimum cycles are found by applying Function Min_cycle. In line 21, c=5, C={ $((v_1,v_5), (v_2,v_7), (v_3,v_4), (v_8,v_{10}), (v_6,v_9))$, $((v_1,v_5), (v_2,v_3), (v_7,v_{10}), (v_9,v_4), (v_6,v_8))$, $((v_1,v_2), (v_5,v_4), (v_{10},v_7), (v_8,v_3), (v_6,v_9))$, $((v_1,v_2), (v_5,v_{10}), (v_4,v_3), (v_9,v_7), (v_6,v_8))$ }. In line 23, s=4, S={ $((v_2,v_3), (v_7,v_9), (v_{10},v_8), (v_5,v_4))$, $((v_2,v_7), (v_3,v_8), (v_4,v_9), (v_5,v_{10}))$ }.

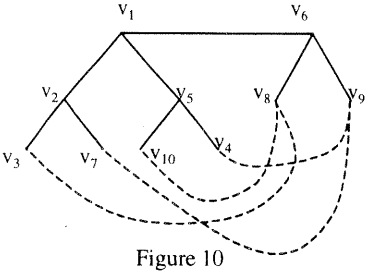


Figure 10

6) Since G-{$S_i \cup (v_1,v_6)$} and GΘ (v_1,v_6)-S_i are the same for all S_i in S, in line 39, the flag is true. And line 41 is carried out. Then since c=s+1, C$\cup$S is returned as the set of all the minimum cyclic edge cutsets of the Petersen graph. These are shown in Figure 11, the first two cutsets are in S, the remaining are in C.

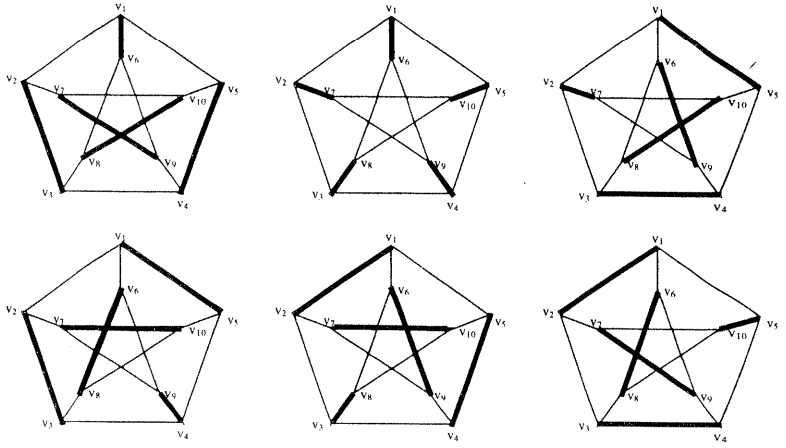


Figure 11

5. The time complexity the algorithm

Lemma 6: The girth g of a 3-regular graph G is bounded by $2*\log_2 n$, where n is the

number of vertices of G.

Proof.

According to the Function Min_cycle(x,y), let c be the length of the minimum cycles containing edge (x,y). If c is even, see Figure 5, then

$$2(2^{\frac{c}{2}} - 1) \leq n$$

$$g \leq c \leq 2\log_2(\frac{n}{2} + 1).$$

If c is odd, see Figure 6, we get

$$(2^{\lfloor \frac{c}{2} \rfloor + 1} - 1) + (2^{\lfloor \frac{c}{2} \rfloor} - 1) \leq n$$

$$g \leq c \leq 2\log_2(\frac{n+2}{3}) + 1$$

Hence the lemma follows.□

Lemma 7: The number *cn* of minimum cycles found in the Function Min_cycle(x,y) is bounded by $2^{\lfloor \frac{c}{2} \rfloor}$, where c is the length of the cycles.

Proof.

In the Function Min_cycle(x,y), each leaf vertex of the width-first search tree can stretch out at most two edges to the other tree. If c is even, see Figure 5, then

$$cn \leq 2 \times 2^{\frac{c}{2}-1} = 2^{\lfloor \frac{c}{2} \rfloor}.$$

If c is odd, see Figure 6, we have

$$cn \leq 2 \times 2^{\lfloor \frac{c}{2} \rfloor - 1} = 2^{\lfloor \frac{c}{2} \rfloor}.$$

Hence the lemma follows. □

Only when c is equal to the girth of G, can these cycles each be incident to a minimum cyclic edge cutset of G, hence the number of minimum cyclic edge cutsets found in Case1 is bounded by υ, by lemma 6 and lemma 7.

Theorem 8: In the worst situation, the time complexity of the algorithm Cyclic_connectivity(G) is bounded by $O(n^3 \cdot \log_2 n)$.

Proof.

The recursive program will stop recursive calling when either G is K_4 or G is 1 or 2-connected. Suppose G has n vertices and, when the program stops recursively calling, G' has r vertices. In the worst situation, each time when the program returns to the upper level, it returns S∪C, that is all the minimum cycles found are finally each incident to a minimum cyclic edge cutset of G.

During each recursive step, the time needed by the algorithm is linear to the sum of edges of all the minimum cyclic edge cutsets. By Lemmas 6 and 7, the total time is

$$O(r \cdot 2\log_2 r + (r+(r+2)) \cdot 2\log_2(r+2) + ... + (r+(r+2)+...+n) \cdot 2\log_2 n),$$

where (r+(r+2)+...+i) is the possible number of minimum cyclic edge cutsets of the graph with i vertices, this is calculated by recursively adding the possible number of minimum cyclic edge cutsets of G⊖e and the number of those found in Case1; $2\log_2(i)$ is

the upper bound of the girth of the graph with i vertices. The product of these two terms is an upper bound of the sum of edges of all the minimum cyclic edge cutsets.

From the formula above, we have

$$O(\sum_{\substack{i=0 \\ i \text{ is even}}}^{n-r} [(r+i)\frac{(n-r-i+2)}{2}] \cdot 2 \cdot \log_2 n)$$

$$\leq O(\frac{n-r+2}{2} \cdot 2 \cdot \log_2 n \cdot \sum_{\substack{i=0 \\ i \text{ is even}}}^{n-r} (r+i))$$

$$\leq O(\frac{n-r+2}{2} \cdot 2 \cdot \log_2 n \cdot \frac{(n+2) \cdot n}{4})$$

To be simple, the time complexity is bounded by $O(n^3 \cdot \log_2 n)$. □

Remark: The time complexity as analysed here provides only a very rough upper bound, which can be reduced by refined analysis. The problem is the difficulty of counting the minimum cyclic edge cutsets of a graph.

6. References

[1] D. W. Barnette and B. Grünbaum, On Steinitz's theorem concerning convex 3-polytopes and on some properties of planar graphs. Many Facets of Graph Theory, Lecture Notes in Mathematics, Vol. 110. Springer-Verlag, New York (1966) 27-40.

[2] J. A. Bondy and U. S. R. Murty, Graph Theory with Applications, Macmillan Press, London (1976).

[3] D. A. Holton, B. Jachson, A. Saito and N. C. Wormald, Removable Edges in 3-connected Graphs, Journal of Graph Theory, John Wiley & Sons Inc., Vol. 14, No. 4, (1990) 465-473.

[4] D. A. Holton, Dingjun Lou and M. D. Plummer, On the 2-extendability of Planar Graphs, Discrete Math., North-Holland, 96(1991), 81-99.

[5] D. A. Holton and M. D. Plummer, Matching Extension and Connectivity in graphs II, Proc. Sixth International Conference on the Theory and Applications of Graphs (Kalamazoo, 1988), John Wiley & Sons, New York, 1988, 651-665.

[6] J. E. Hopcroft and R. E. Tarjan, Dividing a graph into triconnected components, SIAM J. Computing 2:3 (1973).

[7] Dingjun Lou and D. A. Holton, Lower Bound of Cyclic Edge Connectivity for n-extendability of Regular Graphs, Discrete Math., North-Holland, 112(1993),139-150.

[8] B. Peroche, On Several Sorts of Connectivity, Discrete Math., North-Holland, 46 (1983) 267-277.

[9] M. D. Plummer, On the cyclic connectivity of planar graphs, in: Y. Alavi, D. R. Lick and A. T. White, eds., Graph Theory and Applications (Springer-Verlag, Berlin, 1972) 235-242.

[10] P. G. Tait, Remarks on the colouring of maps, Proc. Roy. Soc. Edingburg 10 (1880) 501-503.

(Received 1/9/2000)