The Ninth International Symposium on Operations Research and Its Applications (ISORA'10)
Chengdu-Jiuzhaigou, China, August 19–23, 2010

A Linear Time Algorithm for Edge Searching on 3-Cycle-Disjoint Graphs

Boting Yang Runtao Zhang Yi Cao

Department of Computer Science, University of Regina, Regina, Canada

Abstract The edge searching problem is to find the minimum number of searchers to capture an intruder that is hiding on vertices or edges of a graph. A 3-cycle-disjoint graph is a connected graph in which no pair of cycles share a vertex and every cycle has at most three vertices with degree more than two. In this paper, we consider the edge searching problem on 3-cycle-disjoint graphs. We propose a linear time algorithm to compute the edge search number and the optimal edge search strategy of a 3-cycle-disjoint graph.

Keywords graph searching; edge searching; cycle-disjoint graphs

1 Introduction

The edge searching problem is to find the minimum number of searchers to capture an intruder that is hiding on vertices or edges of a graph. There are other searching models besides edge searching, but in this paper we only consider the edge searching problem. For this reason, we will use "search" instead of "edge search" for simplicity.

Let G be a graph with no loops or multiple edges. Initially, all vertices and edges of G are *contaminated*, which means an intruder can hide on any vertices or anywhere along edges. The intruder is invisible to all searchers and can slide along a path that contains no searcher at a great speed at any time. There are three *actions* for searchers: (1) place a searcher on a vertex; (2) remove a searcher from a vertex; and (3) slide a searcher along an edge from one endpoint to the other. An edge uv in G can be *cleared* in one of the following two ways by a sliding action: (1) two searchers are located on vertex u, and one of them slides along uv from u to v; or (2) a searcher is located on vertex u, where all edges incident with u, other than uv, are already cleared, and the searcher slides from u to v. A *search strategy* is a sequence of actions designed so that the final action leaves all edges of G cleared. The minimum number of searchers required to clear G is called the *search number* of G, denoted by $s(G)$.

Megiddo et al. [5] showed that determining the search number of a graph is NP-hard. The search number is closely related to the vertex separation. Ellis et al. [3] proposed an algorithm to compute the vertex separation of a tree in $O(n)$ time. Ellis and Markov [4] gave an $O(n \log n)$ time algorithm to compute the vertex separation and the corresponding optimal layout of a unicyclic graph.

Bodlaender and Kloks [1] gave a polynomial time algorithm for computing the path-width of a graph with constant treewidth. Since the search number of a graph equals the

pathwidth of its 2-expansion [3], we know that the search number of a graph with constant treewidth is polynomial time computable. However, the exponent in the running time of this algorithm is very large. Even for a graph with treewidth two, it takes at least $\Omega(n^{11})$ time. Yang et al. [6] improved Ellis and Markov's algorithm [4] from $O(n\log n)$ to $O(n)$ for computing the vertex separation and the optimal layout of a unicyclic graph. They showed how to compute the search number of a k-ary cycle-disjoint graph. They also investigated approximation algorithms for edge searching on cycle-disjoint graphs.

All graphs in this paper are finite with no loops or multiple edges. A graph G is called a *cycle-disjoint graph (CDG)* if it is connected and no pair of cycles in G share a vertex. If every cycle of a CDG G has at most three vertices with degree more than two, then we call G a *3-cycle-disjoint graph (3CDG)*. If a vertex or an edge is on a cycle of G, it is called a *cycle vertex* or a *cycle edge* respectively.

The motivation of this paper is to find an efficient algorithm for computing the search number of a graph with treewidth two. We have successfully found an $O(n)$ time algorithm for a unicyclic graph [6]. We now try to extend this algorithm to CDGs. However, we will see that the necessary structural information of CDGs is much more complicated than that of unicyclic graphs. In this paper, we extend the labeling method used in [3] and propose a linear time algorithm to compute the search number and the optimal search strategy of a 3-cycle-disjoint graph.

2 Structures of cycle-disjoint graphs

For two graphs G and H, the *union* of G and H, denoted by $G\cup H$, is the graph with vertex set $V(G)\cup V(H)$ and edge set $E(G)\cup E(H)$. If H is a small graph that consists of only one or two edges, we may use $G\cup E(H)$ to represent $G\cup H$.

A *rooted CDG* is a connected CDG with one vertex designated as the root of the graph. Let $G[r]$ be a rooted CDG with root r. We first define that each vertex of $G[r]$ except r is a descendant of r. For any edge uv that is not on a cycle, the graph $G[r]-uv$ has two connected components. If $u=r$ or u and r are in the same component, then we say that v is a *child* of u, and each vertex that is in the same component as v is called a *descendant* of u. Note that there is no parent-child relationship among vertices in the same cycle. If v is the child of u, then we orient this edge with the direction from u to v and the oriented edge is denoted by (u,v). For any cycle $v_1v_2\ldots v_kv_1$ in $G[r]$, if v_1 has an incoming edge (u,v_1) in $G[r]$ or $v_1=r$, then v_1 is referred to as the *entrance-vertex* of the cycle. For any vertex v of $G[r]$, the subgraph induced by v and all its descendant vertices is called the *vertex-branch* of v, denoted by $G[v]$. $G[r]$ can be considered as a vertex-branch of r. For any directed edge (u,v) of $G[r]$, let G_v be the connected component of $G[r]-(u,v)$ that contains v. The graph $G_v\cup\{(u,v)\}$ is called the *edge-branch* of the directed edge (u,v), denoted by $G[uv]$. We also say that $G[uv]$ is an edge-branch of u. Edge-branch is only defined for non-cycle edges since only these edges are oriented.

In our labeling process, we assign proper labels to all vertices, non-cycle edges and cycles of a rooted CDG $G[r]$. The label of a vertex v, non-cycle edge e or cycle C in $G[r]$ records the necessary structural information of $G[v]$, $G[e]$ or $G[C]$, respectively. Intuitively, a label is a sequence of elements $(s_1^{t_1},s_2^{t_2},\ldots s_m^{t_m})$, where each element $s_i^{t_i}$ consists of a positive integer s_i and a superscript t_i, where $0\le t_i\le 5$. Let G_1 be $G[v]$ (similarly, $G[e]$ or $G[C]$) and s_1 be the search number of G_1. If G_1 has two edge disjoint subgraphs

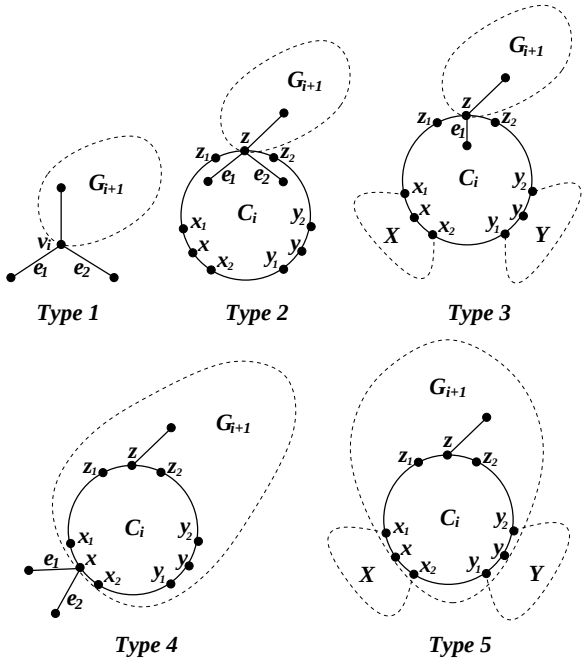

Figure 1: five typical critical structures of G_i

(they may share a vertex) such that each of them has search number $s(G_1)$, then we say G_1 is *critical* and G_1 must be one of the five typical structures illustrated in Figure 1 and t_1 indicates which structure it is. If G_1 is not critical, then we say G_1 is *non-critical* and $t_1 = 0$. When G_1 is critical, according to its type of structure, we can obtain a corresponding reduced graph G_2 by deleting some vertices from G_1. s_2 is the search number of G_2 and t_2 indicates the structure of G_2. Continue this procedure until the reduced graph is non-critical or empty. The precise definition of a label is given as follows.

Definition 1. Let $G[r]$ be a rooted 3CDG, the label of a vertex v (resp. non-cycle edge e or cycle C) in $G[r]$, denoted by $L(v)$ (resp. $L(e)$ or $L(C)$), is defined as a sequence of elements $(s_1^{t_1}, s_2^{t_2}, \ldots s_m^{t_m})$. Each element $s_i^{t_i}$ consists of a positive integer s_i and a superscript t_i, where $0 \leq t_i \leq 5$. If $t_i = 0$, we call $s_i^{t_i}$ a non-critical element; otherwise, we call it a s_i-critical element of type-t_i. The value of $s_i^{t_i}$, denoted by $|s_i^{t_i}|$, is the positive integer s_i. The value of $L(v)$ (resp. $L(e)$ or $L(C)$), denoted by $|L(v)|$ (resp. $|L(e)|$ or $|L(C)|$), is the value of its first element s_1. $L(v)$ (resp. $L(e)$ or $L(C)$) satisfies the following conditions.

1. $s_1 > s_2 > \ldots > s_m$ and only the last element $s_m^{t_m}$ can be non-critical.
2. G_1 is $G[v]$ (resp. $G[e]$ or $G[C]$), and for $2 \leq i \leq m$, G_i is defined as a graph obtained from G_{i-1} according to t_{i-1}, see condition 3 for details.
3. If $m > 1$, then for $i = 1, 2, \ldots, m-1$, we have $s_i = s(G_i)$, $t_i > 0$ and
   (a) if $t_i = 1$, there exists a non-cycle vertex v_i in G_i and v_i has two outgoing edges e_1 and e_2 such that $s(G[e_1]) = s(G[e_2]) = s_i$. G_{i+1} is defined as the graph obtained by deleting all the vertices of $G[v_i]$ except v_i from G_i.

(b) if $2 \le t_i \le 5$, there exists a cycle $C_i = zz_1x_1xx_2y_1yy_2z_2z$ (see Figure 1) in G_i. Let z be the entrance-vertex of C_i, let X be $G[x] \cup \{xx_1, xx_2\}$, Y be $G[y] \cup \{yy_1, yy_2\}$, and Z be $G[z] \cup \{zz_1, zz_2\}$. Assume $s(X) \ge s(Y)$, then we have

- if $t_i = 2$, z has two outgoing edges e_1 and e_2 such that $s(G[e_1]) = s(G[e_2]) = s_i$. G_{i+1} is defined as the graph obtained by deleting all vertices of $G[C_i]$ except z from G_i.
- if $t_i = 3$, $s(X \cup Y \cup \{x_2y_1\}) = s_i$ and z has one outgoing edges e_1 such that $s(G[e_1]) = s_i$. G_{i+1} is defined as the graph obtained by deleting all vertices of $G[C_i]$ except z from G_i.
- if $t_i = 4$, x has two outgoing edges e_1 and e_2 such that $s(G[e_1]) = s(G[e_2]) = s_i$. G_{i+1} is defined as the graph obtained by deleting all the vertices of $G[x]$ except x from G_i.
- if $t_i = 5$, $s(X) = s(Y) = s_i$. G_{i+1} is defined as the graph obtained by deleting all vertices of $G[x]$ except x and all the vertices of $G[y]$ except y from G_i.

4. $s(G_m) = s_m$, $0 \le t_m \le 3$ and

(a) if $t_m = 1$, L must be the label of a vertex v, and v has two outgoing edges e_1 and e_2 such that $s(G[e_1]) = s(G[e_2]) = s_m$.

(b) if $2 \le t_m \le 3$, L must be the label of a cycle $C = zz_1x_1xx_2y_1yy_2z_2z$ (see Figure 1). Let z be the entrance-vertex of C, let X be $G[x] \cup \{xx_1, xx_2\}$, Y be $G[y] \cup \{yy_1, yy_2\}$, and Z be $G[z] \cup \{zz_1, zz_2\}$. Assume $s(X) \ge s(Y)$, then we have

- if $t_i = 2$, z has two outgoing edges e_1 and e_2 such that $s(G[e_1]) = s(G[e_2]) = s_m$.
- if $t_i = 3$, $s(X \cup Y \cup \{x_2y_1\}) = s_m$ and z has one outgoing edges e_1 such that $s(G[e_1]) = s_m$.

For the first element $s_1^{t_1}$ of $L(v)$ (resp. $L(e)$ or $L(C)$), if $t_1 > 0$, $G[v]$ (resp. $G[e]$ or $G[C]$) is said to be s_1-*critical* of *type*-t_1, and the corresponding vertex v_1 or cycle C_1 is called the s_1-*critical vertex* or s_1-*critical cycle* in $G[v]$ (resp. $G[e]$ or $G[C]$).

3 Linear time algorithm

Let S be a monotonic search strategy for a graph G and v be a vertex in G. During the procedure of performing S on G, if a searcher is placed on v and this searcher is never removed from v until G is cleared, then we say that S *ends at* v; if a searcher is placed on v in the first action of S and this searcher will never been removed from v until all the edges incident with v are cleared, then we say that S *starts from* v.

For simplicity, we will use a normalized 3CDG as the input of our algorithm. For each cycle C in a 3CDG, let x, y and z be the three vertices with degree more than 2. Note that there may not be three vertices each of which has degree more than 2 and in this case, we will choose the degree-two cycle vertex. Recall that there are at least 3 vertices in each cycle since we require that all the graphs in this paper are finite without loops and multiple edges. Replace each of $x \sim y$, $y \sim z$ and $z \sim x$ by a path of length three such that $C = zz_1x_1xx_2y_1yy_2z_2z$ (see Figure 2). This procedure takes $O(n)$ time. Notice that the search number of the normalized 3CDG equals the search number of the original graph.

The following algorithm SEARCHNUMBER-3CDG computes the labels of vertices, non-cycle edges and cycles in a rooted 3CDG $G[r]$ by the labeling method. We can construct the corresponding optimal search strategy based on these labels.

Algorithm SEARCHNUMBER-3CDG($G[r]$)
Input: A rooted 3CDG $G[r]$.
Output: Labels of all vertices, cycles and non-cycle edges.

1. Assign label (0^0) to each leaf (except r if r is also a leaf), (1^0) to each pendant edge and (2^0) to each pendant cycle in $G[r]$.
2. If r is labeled, then return labels of all vertices, cycles and non-cycle edges in $G[r]$.
3. - For each vertex v whose all out-going edges have been labeled, compute the label $L(v)$.

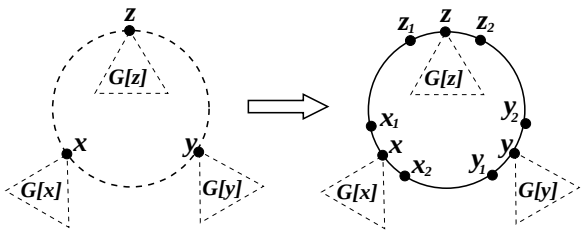

Figure 2: normalization of 3CDG

- For each cycle C in which all the vertices with degree more than two have been labeled, compute the label $L(C)$.
- For each non-cycle edge (u,v), if v is on a labeled cycle or if v is a labeled non-cycle vertex, then compute the label $L(uv)$.
- Go to Step 2.

We now consider how to compute the label of a vertex v when the labels of all its outgoing edges are known. Suppose v has d children, $v_1,\ldots,v_d$. For $1 \le i \le d$, let $L(vv_i)$ be the label of (v,v_i) that contains m_i elements. For $1 \le j \le m_i$, $s_{j,i}^{t_{j,i}}$ is the j-th element in $L(vv_i)$. Here we use additional subscripts in s_j and t_j to indicate whose label it belongs to, i.e., $L(vv_i) = (s_{1,i}^{t_{1,i}}, s_{2,i}^{t_{2,i}}, \ldots s_{m_i,i}^{t_{m_i,i}})$. Then we have a graph $G_{m_i,i}$ that is defined in Definition 1, where $G[vv_i]$ and $G_{m_i,i}$ correspond to G_1 and G_{m_i} respectively. Let G_0 be the union of all $G_{m_i,i}$, for $1 \le i \le d$. Let L_0 be a multiset that contains all non-critical elements of each $L(vv_i)$, $1 \le i \le d$. We can compute the label of v in $G_0[v]$, denoted by L_v, from L_0 (we omit the details due to the space limit). After obtaining the label L_v of v in $G_0[v]$, we merge L_v with all critical elements of $L(vv_i)$, for $1 \le i \le d$, which can be done by function MERGELABEL-VERTEX. The output of this function is the label of v in $G[r]$.

Function MERGELABEL-VERTEX$(L_1, L_2, \ldots, L_d, L_v)$
Input: $L_1, L_2, \ldots, L_d$ and L_v, where L_i is the label of the i-th outgoing edge of v, for $1 \le i \le d$, and L_v is the label of v in $G_0[v]$ that contains only one element.
Output: The label of v in $G[r]$.

1. Set $\alpha \leftarrow$ the only element of L_v.
2. Let w be the value of the largest repeated critical elements in the labels $L_1, L_2, \ldots, L_d$.
   /* Two critical elements are repeated if they have the same value */
3. **if** $|\alpha| < w+1$, **then** $\alpha = (w+1)^0$.
4. Let L be a sequence containing all the critical elements of $L_1, L_2, \ldots, L_d$ with value larger than or equal to $|\alpha|$.
5. Set $h \leftarrow$ the value of the last element in L;
   if $h > |\alpha|$ **then return** $L(v) \circ (\alpha)$;
   else update L by deleting its last element;
   Set $\alpha \leftarrow (|\alpha|+1)^0$;
   Go to Step 5.

We now consider how to compute the label of a cycle C when we know the labels of all the three cycle vertices with degree more than 2.

Let $C = zz_1x_1xx_2y_1yy_2z_2z$ be a cycle in a rooted 3CDG $G[r]$ and z be the entrance-vertex of C. Suppose $L(x)$, $L(y)$ and $L(z)$ are the labels of x, y and z respectively. Let X

be $G[x] \cup \{xx_1, xx_2\}$, Y be $G[y] \cup \{yy_1, yy_2\}$, and Z be $G[z] \cup \{zz_1, zz_2\}$. By using function MERGELABEL-VERTEX to merge $L(x)$ (resp. $L(y)$ or $L(z)$) with (1^1), we can obtain the label of x (resp. y or z) in $X[x]$ (resp. $Y[y]$ or $Z[z]$), denoted by L_x (resp. L_y or L_z). $L_x = (s_{1,x}^{t_{1,x}}, s_{2,x}^{t_{2,x}}, \ldots s_{m_x,x}^{t_{m_x,x}})$. Then we have a graph $G_{m_x,x}$ that is defined in Definition 1, where $X[x]$ and $G_{m_x,x}$ correspond to G_1 and G_{m_x} respectively. Similarly, we have $G_{m_y,y}$ and $G_{m_z,z}$. Let $G_0 = G_{m_x,x} \cup G_{m_y,y} \cup G_{m_z,z} \cup C$. We can compute the label of C in $G_0[z]$, denoted by L_C, from $s_{m_x,x}^{t_{m_x,x}}, s_{m_y,y}^{t_{m_y,y}}, s_{m_z,z}^{t_{m_z,z}}$ (we omit the details due to the space limit). After obtaining the label L_C of C in $G_0[z]$, we will use the following function MERGELABEL-CYCLE to merge L_C with the three labels $L_x - a_X$, $L_y - a_Y$ and $L_z - a_Z$. The output of this function is the label of C in $G[r]$.

Function MERGELABEL-CYCLE(L_x, L_y, L_z, L_C)
Input: L_x, L_y, L_z and L_C, where L_x (resp. L_y or L_z) is the label of x (resp. y or z) in $X[x]$ (resp. $Y[y]$ or $Z[z]$) without the last element and L_C is the label of C in $G_0[z]$.
Output: The label of C in $G[r]$.

1. Set $\alpha \leftarrow$ the first element of L_C, $q \leftarrow |L_C|$.
2. Let w be the value of the largest repeated critical element of the input labels L_x, L_y and L_z. /* Two critical elements are repeated if they have the same value */
3. **if** $|\alpha| < w+1$, **then** $\alpha = (w+1)^0$.
4. Let L be a sequence containing all the critical elements of L_x, L_y and L_z with value larger than or equal to $|\alpha|$.
5. Set $h \leftarrow$ the value of the last element in L;
   if $h > |\alpha|$ **then if** $|\alpha| = q$ **then return** $L(v) \circ L_C$;
   else return $L(v) \circ (\alpha)$;
   else update L by deleting its last element;
   Set $\alpha \leftarrow (|\alpha|+1)^0$;
   Go to Step 5.

Let $G[r]$ be a rooted 3CDG and (u,v) be a non-cycle edge in $G[r]$. If v is on a labeled cycle or v is a labeled non-cycle vertex, then function EDGELABEL computes the label of (u,v) in $G[r]$, denoted by $L(uv)$.

Function EDGELABEL($G[uv]$)
Input: The label of v, denoted by L.
Output: The label of the edge (u,v).

1. Let p be the last element of L.
2. **if** p is not critical, **then return** L.
3. **if** p is critical with value larger than 1, **then return** $L \circ (1^0)$.
4. **if** p is critical with value 1,
   then q is the smallest positive integer such that no element in L has value q;
   Update L by deleting all the elements with value less than q;
   return $L \circ (q^0)$.

4 Correctness and time complexity

Lemma 1. *Function* MERGELABEL-VERTEX *outputs the label of a vertex in* $G[r]$.

Based on Lemma 1 and the discussion in Section 3, we have the following lemmas.

Lemma 2. *Function* MERGELABEL-CYCLE *outputs the label of a cycle in* $G[r]$.

Lemma 3. *Function* EDGELABEL *outputs the label of a non-cycle edge in* $G[r]$.

From Lemmas 1, 2 and 3, SEARCHNUMBER-3CDG can compute the labels of each vertex, cycle and non-cycle edge. In the rest of this section we will analyze the time complexity of this algorithm.

We introduce a data structure used in [3] that compresses the label representation. For a sub-list of value consecutive critical elements in a label, we use an interval to represent them. For example, the label $(9^{t_1}, 8^{t_2}, 7^{t_3}, 6^{t_4}, 5^{t_5}, 3^{t_6}, 2^{t_7}, 1^0)$ is represented as $((9,5),(3,2),1^0)$. Note that the non-critical element is not put into any interval. The benefit of this representation is to improve the label merging operation. For example, if we want to merge label $(9^{t_1}, 8^{t_2}, 7^{t_3}, 6^{t_4}, 5^{t_5}, 3^{t_6}, 2^{t_7}, 1^0)$ with (5^0), we can obtain the result (10^0) in one step by using this compressed representation.

Lemma 4. *The time complexity of function* EDGELABEL *is* $O(1)$ *with the compressed label representation.*

Lemma 5. *The time complexity of function* MERGELABEL-VERTEX *is* $O(|L_2|+d)$*, where* L_2 *is the second largest label among* $L_1, L_2, \ldots, L_d$*.*

Proof. Lines 1 and 3 take constant time. Line 5 also takes $O(1)$ time by using the same technique as in function EDGELABEL. We now consider the time complexity of Lines 2 and 4.

For $1 \leq i \leq d$, suppose that $|L_1| \geq |L_2| \geq |L_i|$ for $3 \leq i \leq d$. In order to achieve $O(|L_2|+d)$ time for MERGELABEL-VERTEX, we first merge part of L_1 with all the other labels and then merge the result with the rest of L_1. Replace Line 2 by the following fragment.

2.1 Set $w \leftarrow 0$, and remove elements with value less than or equal to $|L_2|$ from L_1 and put them into Y.

2.2 **for** $i = 2$ to d, **do**
  for $j = 1$ to m_i, **do**
  /* L_i contains m_i critical elements and let s_j be the j-th largest one in L_i. */
    if no element in Y has value $|s_j|$,
      then put s_j into Y;
      else if $|s_j| > w$, **then** $w = |s_j|$;
        break the inner **for** loop;

2.3 Delete all elements with value less than or equal to w from Y.

2.4 Represent Y by the compressed form.

Line 2.1 takes $O(|L_2|)$ time. In Line 2.2, each time when we check an element, we either add it into Y or finish checking the label that contains it. The size of Y is at most $|L_2|$ and there are d labels. Thus, Line 2.2 takes $O(|L_2|+d)$ time and Lines 2.3 and 2.4 take $O(|L_2|)$ time. Hence, the time complexity of Line 2 is $O(|L_2|+d)$.

After the execution of this fragment, w is the value of the largest repeated critical elements in $L_1, L_2, \ldots, L_d$ and Y contains all the critical elements with value less than or equal to $|L_2|$ and larger than w (if there is no such element, Y is empty). Consider the following two cases regarding the value of α after Line 3.

CASE 1. $|\alpha| > |L_2|+1$. Then critical elements with value larger than or equal to $|\alpha|$ can only appear in L_1. Let L be L_1 and Line 4 takes $O(1)$ time.

CASE 2: $|\alpha| \leq |L_2|+1$. Let $L = L_1 \circ Y$ and delete all elements in L with value smaller than $|\alpha|$. Line 4 takes $O(|L_2|)$ time.

Therefore, the time complexity of MERGELABEL-VERTEX is $O(|L_2|+d)$. □

Similarly to Lemma 5, we have the following lemma.

Lemma 6. *The time complexity of function* MERGELABEL-CYCLE *is $O(\beta)$, where β is the value of the second largest label among L_x, L_y and L_z.*

Lemma 7. *If a function $f(n)$ is defined on the positive integers by the recurrence equation*

$$f(n) = \begin{cases} c, & n = 1,2, \\ f(m_1) + c, & k = 1, n \geq 3, \\ \max_M \{\sum_{i=1}^{k} f(m_i) + c(\lceil \log m_2 \rceil + k)\}, & k \geq 2, n \geq 3, \end{cases}$$

where $M = \{(m_1, m_2, \ldots, m_k) : m_1 \geq m_2 \geq \cdots m_k \geq 1, \text{and } \sum_{i=1}^{k} m_i = n-2\}$, and $c \geq 1$ is a constant, then $f(n)$ is $O(n)$.

In algorithm SEARCHNUMBER-3CDG, we use $f(G[v])$ to denote the time used to compute the label of vertex v and use $f(G[C])$ to denote the time used to compute the label of cycle C. Then we have

$$f(G[v]) = f(G[vv_1]) + f(G[vv_2]) + \cdots + f(G[vv_d]) + O(s(G[vv_2]) + d),$$

where v has d edge-branches and $G[vv_2]$ is the second largest edge-branch according to their search numbers. We also have

$$f(G[C]) = f(G[x]) + f(G[y]) + f(G[z]) + O(s(G^*)),$$

where x, y and z are the three vertices on C with degree more than 2, and G^* is one of $G[x]$, $G[y]$ and $G[z]$ that has the second largest search number.

Theorem 8 follows from Lemma 7.

Theorem 8. *For a rooted 3CDG $G[r]$, algorithm* SEARCHNUMBER-3CDG *computes the labels of all vertices, cycles and non-cycle edges in O(n) time.*

Finally, we can construct an optimal search strategy in linear time as well. We omit the details due to the space limit.

Theorem 9. *Let $G[r]$ be a rooted 3CDG. If the labels of all vertices, cycles and non-cycle edges are known, then we can construct an optimal search strategy for G in O(n) time.*

References

[1] H. Bodlaender, T. Kloks, Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs. *J. Algorithms* 21 (1996) 358–402.

[2] H. Bodlaender, F. Fomin, Approximation of pathwidth of outerplanar graphs. *J. Algorithms* 43 (2002) 190–200.

[3] J. Ellis, I. Sudborough and J. Turner, The vertex separation and search number of a graph, *Information and Computation* 113 (1994) 50–79.

[4] J. Ellis and M. Markov, Computing the vertex separation of unicyclic graphs, *Information and Computation* 192 (2004) 123–161.

[5] N. Megiddo, S. L. Hakimi, M. Garey, D. Johnson and C. H. Papadimitriou, The complexity of searching a graph, *Journal of ACM* 35 (1988) 18–44.

[6] B. Yang, R. Zhang and Y. Cao, Searching cycle-disjoint graphs, Proceedings of the 1st International Conference on Combinatorial Optimization and Applications (COCOA'07), Lecture Notes in Computer Science, Vol. 4616, Springer-Verlag, Berlin, pp.32–43, 2007.