SRC TR 87-130

Efficient Enumeration of Maximal and Maximum Independent Sets of an Interval Graph and a Circular-Arc Graph

by

S. Masuda, K. Nakajima,
T. Kashiwabara, and T. Fujisawa

Efficient Enumeration of Maximal and Maximum Independent Sets of an Interval Graph and a Circular-Arc Graph*

by

Sumio MASUDA

Electrical Engineering Department and Systems Research Center
University of Maryland
College Park, Maryland 20742
on leave from
Department of Information and Computer Sciences
Osaka University
Toyonaka, Osaka 560, Japan

Kazuo NAKAJIMA

Electrical Engineering Department,
Institute for Advanced Computer Studies,
and Systems Research Center
University of Maryland
College Park, Maryland 20742

Toshinobu KASHIWABARA and Toshio FUJISAWA

Department of Information and Computer Sciences
Osaka University
Toyonaka, Osaka 560, Japan

* This work was supported in part by National Science Foundation grants MIP-84-51510 and CDR-85-00108, and in part by a grant from AT&T Information Systems.

Efficient Enumeration of Maximal and Maximum Independent Sets of an Interval Graph and a Circular-Arc Graph

by

Sumio Masuda, Kazuo Nakajima, Toshinobu Kashiwabara and Toshio Fujisawa

Abstract

We present efficient algorithms for generating all maximal and all maximum independent sets of an interval graph and a circular-arc graph. When an interval graph is given in the form of a family of n intervals, the first and second algorithms produce all maximal and all maximum independent sets, respectively, in $O(n \cdot \log n + (\textit{the size of an output}))$ time. When a circular-arc graph is given in the form of a family of n arcs on a circle, the third algorithm generates all maximal independent sets in $O(n \cdot \log n + (\textit{the size of an output}))$ time. In the same situation, the fourth algorithm enumerates all maximum independent sets in $O(n^2 + (\textit{the size of an output}))$ time. The first three algorithms are optimal to within a constant factor.

Key words. circular-arc graph, enumeration, graph algorithm, independent set, interval graph, time complexity

1. Introduction

Let $G = (V, E)$ be a graph. Two distinct vertices u and v in V are said to be *adjacent* to each other if $(u, v) \in E$; otherwise they are said to be *independent* from each other. A subset X of V is called an *independent set* of G if any two vertices in X are independent. An independent set X of G is called a *maximal independent set* (abbreviated to an *MLIS*) if no proper superset of X is an independent set of G, and it is called a *maximum independent set* (abbreviated to an *MMIS*) if its cardinality is the largest among all independent sets of G. It is well known that the problem of finding an MMIS is NP-hard for general graphs [1,6].

Let F be a finite family of nonempty sets. We call a graph $G = (V, E)$ an *intersection graph* for F and F an *intersection model* of G if there is a one-to-one correspondence between V and F such that two vertices in V are adjacent to each other if and only if the corresponding sets in F have a nonempty intersection. If F is a family of intervals on the real line, G is called an *interval graph.* If F is a family of arcs on a circle, G is called a *circular-arc graph.*

Interval graphs are not only interesting in their own right in graph theory but also useful in many practical applications [12,19,22,23]. Therefore, various algorithms for interval graphs have been developed [3,5,8,9,11,13,15-17]. Furthermore, as a generalization of interval graphs, circular-arc graphs have widely been studied in recent years [2,7,10,13,14,16,18,21].

In this paper, we consider the problems of generating all MLIS's and all MMIS's of an interval graph and of a circular-arc graph. In the past, research efforts have been directed mainly at the problem of finding a "single" MMIS for these classes of graphs. Gavril [8] proposed the first polynomial time algorithm for finding an MMIS of an interval graph. He also developed an $O(n^4)$ time algorithm for finding an MMIS of a circular-arc graph [10], where n is the number of vertices in a given graph. Later, Gupta, Lee and Leung [13] developed an $O(n \cdot \log n)$ time algorithm for an interval graph and an $O(n^2)$ time algorithm for a circular-arc graph. They also showed that their algorithm for an interval graph is optimal to within a constant factor. Recently, Asano, Asano and Imai [3] presented an $O(n \cdot \log n)$ time algorithm for finding a maximum weight independent set of an interval graph when each vertex is assigned a real number as its weight. More recently, Masuda and Nakajima [18] developed an $O(n \cdot \log n)$ time algorithm for finding an MMIS of a circular-arc graph. Since every interval graph is a circular-arc graph, their algorithm is also optimal to within a constant factor. All of the above algorithms except Gavril's interval graph algorithm assume that an input graph is given by its corresponding intersection model.

On the other hand, Leung [16] considered the problems of generating all MLIS's of an interval graph and of a circular-arc graph. Based on the Gupta-Lee-Leung algorithms [13], he obtained an $O(n^2+\beta_I)$ time algorithm for an interval graph and an $O(n^2+\beta_C)$ time algorithm for a circular-arc graph, where β_I and β_C denote the total sum of the cardinalities of the MLIS's of a given interval graph and a given circular-arc graph, respectively.

In this paper, we present faster algorithms for finding all MLIS's of an interval graph and of a circular-arc graph. We also consider the problems of generating all MMIS's of these classes of graphs. More specifically, we first give an $O(n \cdot \log n+\beta_I)$ time algorithm for finding all MLIS's of an interval graph, where β_I is as defined above. Next, we develop an $O(n \cdot \log n+\delta_I)$ time algorithm for generating all MMIS's of an interval graph, where δ_I denotes the total sum of the cardinalities of the MMIS's of a given interval graph. We then show an $O(n \cdot \log n+\beta_C)$ time algorithm for finding all MLIS's of a circular-arc graph, where β_C is as defined above. Finally, we present an $O(n^2+\delta_C)$ time algorithm for generating all MMIS's of a circular-arc graph, where δ_C denotes the total sum of the cardinalities of the MMIS's of a given circular-arc graph. The first three algorithms are shown to be optimal to within a constant factor.

2. Definitions for Interval Graphs

Let $F = \{I_1, I_2, \ldots, I_n\}$ be a family of closed intervals on the real line $\mathbf{R}$. Each interval $I_j \in F$ is specified by its left and right endpoints and such endpoints are assigned real numbers, called *coordinates*. The endpoints are located on $\mathbf{R}$ in ascending order of their coordinates from left to right. For each interval I_j, let l_j (resp., r_j) denote the coordinate of its left (resp., right) endpoint, that is, $I_j = [l_j, r_j]$. Without loss of generality, we assume that all endpoints of the intervals in F are distinct. For two distinct intervals I_j and I_k in F, we say that they intersect with each other if there exists a real number m such that $l_j < m < r_j$ and $l_k < m < r_k$. The interval graph for F, which we denote by $G_I(F)$, is defined as follows :

$G_I(F) = (V, E)$, where

$V = \{v_1, v_2, \ldots, v_n\}$, and

$E = \{ (v_i, v_j) \mid I_i \text{ and } I_j \text{ intersect with each other}\}$.

A family of n intervals F is said to be *canonical* if the coordinates of the endpoints of the intervals in F are distinct integers between 1 and $2n$. For example, Fig. 1(a) depicts a canonical family of intervals, where the intervals are drawn as nonoverlapping horizontal line segments for clarity. On the other hand, the family of intervals shown in Fig. 1(b) is not canonical. Note, however, that these two families correspond to the same interval graph which is shown in Fig. 1(c). When a noncanonical family of n intervals F is given, using a regular sorting algorithm [1], one can construct in $O(n \cdot \log n)$ time a canonical family of intervals F' such that $G_I(F) = G_I(F')$.

Two distinct intervals I_j and I_k in F are said to be *independent* from each other if they do not intersect with each other. A subfamily X of F is called an *independent interval family* (abbreviated to an *IIF*) if any two intervals in X are independent from each other. An IIF, X of F is called a *maximal independent interval family* (abbreviated to an *MLIF*) if no proper superfamily of X is an IIF of F, and it is called a *maximum independent interval family* (abbreviated to an *MMIF*) if its cardinality is the largest among all IIF's of F. For example, $\{I_1, I_3, I_6\}$, $\{I_1, I_3, I_7\}$ and $\{I_1, I_4, I_7\}$ are the MMIF's of the family of intervals of Fig. 1(a), and the other MLIF's are $\{I_1, I_5\}$, $\{I_2, I_6\}$ and $\{I_2, I_7\}$.

It is clear from the definition of $G_I(F)$ that two intervals I_j and I_k in F are independent if and only if the corresponding vertices in $G_I(F)$ are independent. This implies that the MLIF's (resp., MMIF's) of F and the MLIS's (resp., MMIS's) of $G_I(F)$ are in one-to-one correspondence. Therefore, it suffices to find all MLIF's (resp., MMIF's) of F in order to generate all MLIS's (resp., MMIS's) of $G_I(F)$. In Sections 3 and 4, we will present optimal algorithms for generating all MLIF's and all MMIF's, respectively, of F.

3. Enumeration of Maximal Independent Sets of an Interval Graph

Let $F = \{I_1, I_2, ..., I_n\}$ be a canonical family of intervals. For each interval $I_j \in F$, let L_j denote $\{I_k \in F \mid r_k < l_j\}$. Furthermore, we define XL_j as the set of all MLIF's of L_j and YL_j as $\{X \cup \{I_j\} \mid X \in XL_j\}$. In particular, if $L_j = \phi$, then $XL_j = \{\phi\}$ and $YL_j = \{\ \{I_j\}\ \}$. Consider the family of intervals of Fig. 1(a) again, where for example, $L_3 = \{I_1\}$ and $L_7 = \{I_1, I_2, I_3, I_4\}$; and since $XL_7 = \{\ \{I_1, I_3\}, \{I_1, I_4\}, \{I_2\}\ \}$, $YL_7 = \{\ \{I_1, I_3, I_7\}, \{I_1, I_4, I_7\}, \{I_2, I_7\}\ \}$.

Let $END(F)$ denote $\{I_j \in F \mid$ there is no interval $I_k \in F$ such that $r_j < l_k\}$. We begin with the following theorem.

Theorem 1. The set of all MLIF's of F is $\displaystyle\bigcup_{I_j \in END(F)} YL_j$.

Proof. ("$\subseteq$" *part.*) Let $X = \{I_{j_1}, I_{j_2}, ..., I_{j_k}\}$ with $l_{j_1} < r_{j_1} < l_{j_2} < r_{j_2} < \cdots < l_{j_k} < r_{j_k}$ be an MLIF of F. The maximality of X implies that $I_{j_k} \in END(F)$. It also implies that $X - \{I_{j_k}\} \in XL_{j_k}$ if $k \neq 1$ and that $XL_{j_k} = \{\phi\}$ if $k = 1$. Thus, $X \in YL_{j_k}$ in any case, and hence $X \in \displaystyle\bigcup_{I_j \in END(F)} YL_j$.

("$\supseteq$" *part.*) Let I_j be an interval in $END(F)$ and X be an arbitrary IIF in YL_j. By definition, $X - \{I_j\} \in XL_j$. Assume that X is not an MLIF of F. Then, there exists an interval $I_k \in F$ such that $X \cup \{I_k\}$ is an IIF of F. Since $I_j \in END(F)$ and I_k is independent from I_j, $I_k \in L_j$. Therefore, $X - \{I_j\} \cup \{I_k\}$ is an IIF of L_j, which contradicts the fact that $X - \{I_j\}$ is an MLIF of L_j. Thus, X is an MLIF of F. □

For $i = 1,2,...,2n$, we call the point with coordinate i as point i. We define PRE_i as $\{I_j \in F \mid r_j < i$ and there exists no interval $I_q \in F$ such that $r_j < l_q < r_q < i\}$ for $i = 1,2,...,2n,2n+1$. It should be noted that $PRE_1 = \phi$ and $PRE_{2n+1} = END(F)$. The following lemma is useful in determining XL_j's efficiently.

Lemma 1. For $j = 1,2,...,n$, $XL_j = \bigcup \{YL_k \mid I_k \in PRE_{l_j}\}$.

Proof. For each integer j such that $L_j = \phi$, the above equality trivially holds. We show below that the same equality also holds for any integer j such that $L_j \neq \phi$.

("$\subseteq$" *part.*) Let $X = \{I_{j_1}, I_{j_2}, ..., I_{j_p}\}$ with $l_{j_1} < l_{j_2} < \cdots < l_{j_p}$ be an MLIF of L_j. Because of the maximality of X, $I_{j_p} \in PRE_{l_j}$. Furthermore, $X - \{I_{j_p}\} \in XL_{j_p}$, and thus $X \in YL_{j_p}$. Therefore, $X \in \bigcup \{YL_k \mid I_k \in PRE_{l_j}\}$, and hence $XL_j \subseteq \bigcup \{YL_k \mid I_k \in PRE_{l_j}\}$.

("$\supseteq$" *part.*) Let I_k be an interval in PRE_{l_j} and X be an IIF in YL_k. By definition, $X - \{I_k\} \in XL_k$. Assume that $X \notin XL_j$. Then, there exists an interval $I_q \in L_j$ such that $X \cup \{I_q\}$ is an IIF of L_j. Since $I_k \in PRE_{l_j}$, $r_q < l_k$, and hence $I_q \in L_k$. Thus, $X - \{I_k\} \cup \{I_q\}$ is an IIF of L_k, which contradicts the fact that $X - \{I_k\} \in XL_k$. Therefore, $X \in XL_j$, and hence $XL_j \supseteq \bigcup \{YL_k \mid I_k \in PRE_{l_j}\}$. □

Let i be an integer such that $1 \leq i \leq 2n$ and I_j be the interval in F which has point i as one of its endpoints. If point i is the left endpoint of I_j, $PRE_{i+1} = PRE_i$ (see Fig. 2(a)); otherwise, $PRE_{i+1} = (PRE_i - PRE_{l_j}) \cup \{I_j\}$ (see Fig. 2(b)). Based on this relationship, one can successively determine PRE_i for $i = 1,2,...,2n,2n+1$. The detailed operations for this are described in

Algorithm 1 which is shown below.

In the algorithm, we first construct a graph G_0 which consists of a single vertex w_0. We then scan the endpoints of the intervals from left to right and construct a digraph $G_i = (V_i, E_i)$ for $i = 1,2,\ldots,2n$, where V_i consists of w_0 and the vertices w_j which correspond to the intervals I_j such that $l_j \le i$, and the edges in E_i are generated in such a way that the directed paths in G_i from vertex w_0 to a vertex w_j correspond to the IIF's in YL_j. During the construction of G_i, we determine a set $PRECEDE_{i+1}$ as $\{w_0\}$ if $PRE_{i+1} = \phi$ and as $\{w_j \mid I_j \in PRE_{i+1}\}$ otherwise.

Algorithm 1.

```
1.  $V_0 \leftarrow \{w_0\}$, $E_0 \leftarrow \phi$, $G_0 \leftarrow (V_0, E_0)$. $PRECEDE_1 \leftarrow \{w_0\}$.
2.  for $i \leftarrow 1$ until $2n$ do
      a) Let $I_j$ be the interval which has point $i$ as one of its endpoints.
      b) if point $i$ is the left endpoint of $I_j$ then
         (1) Create a new vertex $w_j$.
         (2) $V_i \leftarrow V_{i-1} \cup \{w_j\}$, $E_i \leftarrow E_{i-1} \cup \{ (x \rightarrow w_j) \mid x \in PRECEDE_i \}$, $G_i \leftarrow (V_i, E_i)$.
         (3) $PRECEDE_{i+1} \leftarrow PRECEDE_i$.
      c) if point $i$ is the right endpoint of $I_j$ then
         (1) $V_i \leftarrow V_{i-1}$, $E_i \leftarrow E_{i-1}$, $G_i \leftarrow (V_i, E_i)$.
         (2) $PRECEDE_{i+1} \leftarrow (PRECEDE_i - PRECEDE_{l_j}) \cup \{w_j\}$.
3.  $Q \leftarrow \{ \{I_{q_1}, I_{q_2}, \ldots, I_{q_k}\} \mid$ there is a directed path $[w_0, w_{q_1}, w_{q_2}, \ldots, w_{q_k}]$ in $G_{2n}$ such that
        $w_{q_k} \in PRECEDE_{2n+1}\}$.
4.  output $Q$. □
```

Suppose that a point i is the left endpoint of some interval I_j. When we visit such a point during the execution of Step 2, $PRECEDE_i$ has already been determined so as to represent the corresponding intervals in PRE_i, and graph G_{i-1} has been constructed in such a way that the directed paths from w_0 to the vertices in $PRECEDE_i$ correspond to the IIF's in $\bigcup \{YL_k \mid I_k \in PRE_i\}$. Recall that $XL_j = \bigcup \{YL_k \mid I_k \in PRE_i\}$ from Lemma 1 and $YL_j = \{X \cup \{I_j\} \mid X \in XL_j\}$ by definition. Therefore, in the graph G_i which is constructed at Step 2.b) (2), the directed paths from w_0 to w_j correctly represent the IIF's in YL_j.

When we visit the right endpoint i of some interval, we determine $PRECEDE_{i+1}$ based on the aforementioned relationship between PRE_i and PRE_{i+1}. Since $G_i = G_{i-1}$, the graph still maintains the correspondence between the directed paths from w_0 to each vertex $w_k \in PRECEDE_{i+1}$ and the IIF's in YL_k. As noted before, $PRE_{2n+1} = END(F)$. Therefore, from Theorem 1, we will eventually obtain the set of all MLIF's of F by generating all directed paths in G_{2n} from w_0 to the vertices in $PRECEDE_{2n+1}$.

Consider the family of intervals $\{I_1, I_2, ..., I_7\}$ shown in Fig. 1(a). Figs. 3(a), (b), (c), (d), and (e) show graphs G_0, G_2, G_3, G_4, and G_{14}, respectively, which are constructed by Algorithm 1. Then, $Q = \{ \{I_1, I_3, I_6\}, \{I_1, I_3, I_7\}, \{I_1, I_4, I_7\}, \{I_1, I_5\}, \{I_2, I_6\}, \{I_2, I_7\} \}$ is obtained by generating all directed paths in G_{14} from v_0 to the vertices in $PRECEDE_{15} = \{w_5, w_6, w_7\}$.

The main difference between our algorithm and Leung's [16] is in the way to collect the information needed for the generation process. In fact, with a slight modification, his generation procedure could be used in our algorithm in place of Steps 3 and 4. However, since we have already constructed the graph G_{2n} in Step 2, we can easily enumerate all MLIF's of F by finding all the maximal paths in G_{2n} rather than using Leung's list manipulation method [16].

The following procedure searches the graph G_{2n} in a DFS-like manner, but the method of marking vertices is different from that used in the usual depth-first search algorithm (see, *e.g.*, Aho, *et al.* [1]). In this procedure, when we backtrack from vertex x to z, we mark x "old" and each vertex w such that $(x \rightarrow w) \in E_{2n}$ "new". This method enables us to find later, if any, directed paths which contain a path from z to w that does not pass through x.

Procedure 1.

```
1.  Create an empty stack $STACK$. $Q \leftarrow \phi$. Mark all vertices in $V_{2n}$ "new".
2.  Add $w_0$ into $STACK$. $CP \leftarrow \phi$.
3.  while $STACK$ is not empty do
      a) $x \leftarrow$ the top element of $STACK$.
      b) if there is an edge $(x \rightarrow w) \in E_{2n}$ such that $w$ is marked "new"
           then add $w$ into $STACK$. $CP \leftarrow CP \cup \{w\}$.
           else execute the following statements (1) to (3).
                (1) if $x \in PRECEDE_{2n+1}$ then $Q \leftarrow Q \cup \{ \{I_j \mid w_j \in CP\} \}$.
                (2) for each vertex $w$ such that $(x \rightarrow w) \in E_{2n}$ do mark $w$ "new".
                (3) Mark $x$ "old" and delete it from $STACK$. $CP \leftarrow CP - \{x\}$. □
```

We now analyze the time complexity of Algorithm 1. Let $\beta_I(F)$ denote the total sum of the cardinalities of all MLIF's of F, or equivalently, the total sum of the cardinalities of all MLIS's of $G_I(F)$. Furthermore, let γ denote the total sum of the numbers of vertices on the directed paths in $G_{2n} = (V_{2n}, E_{2n})$ from w_0 to the vertices in $PRECEDE_{2n+1}$.

Lemma 2. $|E_{2n}| = O(\gamma)$ and $\gamma = O(\beta_I(F))$.

Proof. It is clear that $\gamma = O(\beta_I(F))$ from the description of Algorithm 1. As can be seen from the construction method of G_{2n}, for any directed edge $(x \rightarrow w) \in E_{2n}$, there is a directed path from w_0 to x if $x \neq w_0$ and a path from w to a vertex in $PRECEDE_{2n+1}$ if $w \notin PRECEDE_{2n+1}$. Therefore, there is a directed path from w_0 to a vertex in $PRECEDE_{2n+1}$ which passes through $(x \rightarrow w)$. This implies that $|E_{2n}| < \gamma$. □

We are now ready to show the following lemma and theorem.

Lemma 3. The time complexity of Algorithm 1 is $O(\beta_I(F))$.

Proof. The time required to execute Step 1 is clearly $O(n)$. The vertices and edges once added to the graphs G_i's are not removed later. Furthermore, the vertices once deleted from the sets $PRECEDE_i$'s will never be added to them again. Since $|PRECEDE_i| = |E_i| - |E_{i-1}|$ for $i = 1,2,...,2n$, $\Sigma_{i=1}^{2n+1} |PRECEDE_i| = |E_{2n}| + |PRECEDE_{2n+1}| \leq n + |E_{2n}|$. Thus, Step 2 can be performed in $O(n + |E_{2n}|)$ time. It is easy to verify that Procedure 1 determines Q in $O(n + |E_{2n}| + \gamma)$ time. Thus, from Lemma 2, Step 3 of Algorithm 1 can be carried out in $O(n + \beta_I(F))$ time. Finally, Step 4 can be executed in $O(\beta_I(F))$ time.

We know that $|E_{2n}| = O(\beta_I(F))$ from Lemma 2. Furthermore, $n \leq \beta_I(F)$ since, for each interval $I_j \in F$, there is at least one MLIF of F which contains I_j. Therefore, the overall time complexity of Algorithm 1 is $O(\beta_I(F))$. □

Theorem 2. For any family of n intervals F, all MLIS's of $G_I(F)$ can be generated in $O(n \cdot \log n + \beta_I(F))$ time. This time complexity is optimal to within a constant factor.

Proof. As claimed in the preceding section, we can construct, in $O(n \cdot \log n)$ time, a canonical family of intervals F' such that $G_I(F) = G_I(F')$. We know from Lemma 3 that the application of Algorithm 1 to the resultant family of intervals requires $O(\beta_I(F))$ time. Therefore, we can generate all MLIS's of $G_I(F)$ in $O(n \cdot \log n + \beta_I(F))$ time.

It is clear that the generation of all MLIS's requires $\Omega(\beta_I(F))$ time. The interval graph $G_I(F)$ has an MLIS whose cardinality is equal to n if and only if any two intervals in F are independent from each other. Since testing whether any two intervals are independent or not requires $\Omega(n \cdot \log n)$ time [20], Algorithm 1 is optimal to within a constant factor. □

4. Enumeration of Maximum Independent Sets of an Interval Graph

Let $F = \{I_1, I_2, ..., I_n\}$ be a canonical family of intervals. We define α_j to be $Max\{\,|X| \mid X \in YL_j\}$ for $j = 1,2,...,n$ and $\alpha(F)$ to be the cardinality of an MMIF of F. Note that $\alpha(F) = Max\{\alpha_j \mid I_j \in F\}$. For $i = 1,2,...,\alpha(F)$, we define S_i as $\{I_j \in F \mid \alpha_j = i\}$. As an example, consider the family of intervals of Fig. 4. In this particular case, $\alpha_1 = \alpha_2 = \alpha_3 = 1$, $\alpha_4 = \alpha_5 = 2$, and $\alpha_6 = \alpha_7 = 3$. Therefore, $S_1 = \{I_1, I_2, I_3\}$, $S_2 = \{I_4, I_5\}$ and $S_3 = \{I_6, I_7\}$.

Suppose that each interval $I_j \in F$ is assigned an integer as its weight. For an IIF, X of F, we denote by $weight(X)$ the sum of the weights of the intervals in X. For this weighted case, an algorithm developed by Asano, Asano, and Imai [3] can be used to find $Max\{weight(X) \mid X$ is an IIF of $L_j\}$ for $j = 1,2,...,n$. Since the weight of every interval is one in our case, their algorithm finds $Max\{\,|X| \mid X \in XL_j\} = \alpha_j - 1$ for $j = 1,2,...,n$. By modifying their algorithm, we can obtain a procedure which partitions F into $S_1, S_2, ..., S_{\alpha(F)}$. In the following procedure, a variable M is used to maintain the value of $Max\{\alpha_k \mid r_k \leq i\}$ for $i = 1,2,...,2n$. It should be noted that the final value of M becomes $\alpha(F)$.

Procedure 2.

```
1.  $M \leftarrow 0$.
2.  for $i \leftarrow 1$ until $2n$ do
        a) Let $I_j$ be the interval which has point $i$ as one of its endpoints.
        b) if point $i$ is the left endpoint of $I_j$ then $\alpha_j \leftarrow M+1$.
        c) if (point $i$ is the right endpoint of $I_j$) and ($M < \alpha_j$) then $M \leftarrow \alpha_j$.
3.  for $i \leftarrow 1$ until $M$ do $S_i \leftarrow \phi$.
4.  for each interval $I_j \in F$ do $S_{\alpha_j} \leftarrow S_{\alpha_j} \cup \{I_j\}$. □
```

The following lemma is obvious from the description of the procedure.

Lemma 4. The time complexity of Procedure 2 is $O(n)$. □

Assume that we have already partitioned F into $S_1, S_2, \ldots, S_{\alpha(F)}$ by applying Procedure 2 to F. For $i = 1,2,\ldots,\alpha(F)$, we define XM_i to be the set of all MMIF's of $\bigcup_{k=i}^{\alpha(F)} S_k$. Clearly, our goal here is to find XM_1. We have the following lemma and its subsequent corollaries.

Lemma 5. For any IIF, X of F, $|X \cap S_i| \leq 1$ for $i = 1,2,\ldots,\alpha(F)$.

Proof. Assume that $|X \cap S_q| \geq 2$ for some integer q such that $1 \leq q \leq \alpha(F)$. Let I_j and I_k with $l_j < l_k$ be two elements of $X \cap S_q$. From the definition of S_q, there exists an IIF, Y of L_j such that $|Y| = q-1$. Since I_j and I_k are independent, we have $l_j < r_j < l_k < r_k$. This implies that $Y \cup \{I_j\}$ is an IIF of L_k with q elements, and hence $\alpha_k \geq q+1$. This contradicts the assumption that $I_k \in S_q$. Therefore, $|X \cap S_i| \leq 1$ for $i = 1,2,\ldots,\alpha(F)$. □

Corollary 1. For any MMIF, X of F, $|X \cap S_i| = 1$ for $i = 1,2,\ldots,\alpha(F)$.

Proof. It is clear from Lemma 5 and the fact that $|X| = \alpha(F)$. □

Corollary 2. Let i be an integer such that $1 \leq i \leq \alpha(F)$. For any IIF, X in XM_i, $|X| = \alpha(F) - i + 1$.

Proof. Let Y be any MMIF of F. Since $Y \cap (\bigcup_{k=i}^{\alpha(F)} S_k)$ is an IIF of $\bigcup_{k=i}^{\alpha(F)} S_k$, we have $|X| \geq |Y \cap (\bigcup_{k=i}^{\alpha(F)} S_k)| = \alpha(F) - i + 1$ from Corollary 1. On the other hand, Lemma 5 implies that $|X| \leq \alpha(F) - i + 1$. Therefore, $|X| = \alpha(F) - i + 1$. □

Corollary 3. Let X be an IIF of $\bigcup_{k=i}^{\alpha(F)} S_k$ for some integer i such that $1 \le i \le \alpha(F)$. If $|X| = \alpha(F) - i + 1$, then $X \in XM_i$.

Proof. This corollary clearly holds from Corollary 2. □

Lemma 5 and the above corollaries lead to the following theorem.

Theorem 3. $XM_{\alpha(F)} = \{ \{I_j\} \mid I_j \in S_{\alpha(F)}\}$. And, for $i = \alpha(F)-1, \alpha(F)-2, \ldots, 1$, $XM_i = \{Y \cup \{I_j\} \mid Y \in XM_{i+1}, I_j \in S_i \text{ and } r_j < Min\,\{l_h \mid I_h \in Y\}\ \}$.

Proof. The first equality of the theorem clearly holds from Corollaries 2 and 3. Let i be an integer such that $1 \le i \le \alpha(F) - 1$ and let $X = \{I_{j_1}, I_{j_2}, \ldots, I_{j_k}\}$ with $l_{j_1} < r_{j_1} < l_{j_2} < r_{j_2} < \cdots < l_{j_k} < r_{j_k}$ be any IIF in XM_i. It is obvious that $r_{j_1} < Min\,\{l_q \mid I_q \in X - \{I_{j_1}\}\ \}$ and that $i \le \alpha_{j_1} < \alpha_{j_2} < \cdots < \alpha_{j_k} \le \alpha(F)$. We know from Corollary 2 that $k = |X| = \alpha(F) - i + 1$ and from Lemma 5 that $|X \cap S_q| \le 1$ for $q = i, i+1, \ldots, \alpha(F)$. Therefore, $\alpha_{j_1} = i$, that is, $I_{j_1} \in S_i$, and $X - \{I_{j_1}\}$ is an IIF of $\bigcup_{q=i+1}^{\alpha(F)} S_q$. Since $|X - \{I_{j_1}\}| = \alpha(F) - i$, we have $X - \{I_{j_1}\} \in XM_{i+1}$ by Corollary 3. Thus, X belongs to the set defined by the right hand side of the second equality of Theorem 3.

On the other hand, let X be any element of XM_{i+1}. If there is an interval $I_j \in S_i$ such that $r_j < Min\,\{ l_q \mid I_q \in X\}$, then $X \cup \{I_j\}$ is an IIF of $\bigcup_{q=i}^{\alpha(F)} S_q$. Since $|X| = \alpha(F) - i$ from Corollary 2, $|X \cup \{I_j\}| = \alpha(F) - i + 1$. Therefore, $X \cup \{I_j\} \in XM_i$ from Corollary 3. □

For $i = \alpha(F), \alpha(F)-1, \ldots, 1$, let S_i' be defined as $\{I_j \in S_i \mid$ there exists an IIF in XM_i which contains $I_j\}$. From Theorem 3, $S'_{\alpha(F)} = S_{\alpha(F)}$ and $S_i' = \{I_j \in S_i \mid$ there is an interval $I_h \in S'_{i+1}$ such that $r_j < l_h\}$ for $i = \alpha(F)-1, \alpha(F)-2, \ldots, 1$.

We now describe an algorithm for generating all MMIF's of F. In the algorithm, a digraph $G_i = (V_i, E_i)$ is constructed for $i = \alpha(F), \alpha(F)-1, \ldots, 1$. It has a unique vertex w_0 and the vertices which correspond to the intervals in $S'_{\alpha(F)} \cup S'_{\alpha(F)-1} \cup \cdots \cup S_i'$, and the edges are determined in such a way that its maximal paths correspond to the IIF's in XM_i.

Algorithm 2.

```
1.  $S'_{\alpha(F)} \leftarrow S_{\alpha(F)}$.  $V'_{\alpha(F)} \leftarrow \{w_j \mid I_j \in S_{\alpha(F)}\}$.
2.  $V_{\alpha(F)} \leftarrow \{w_0\} \cup V'_{\alpha(F)}$,  $E_{\alpha(F)} \leftarrow \{ (w_0 \rightarrow w_j) \mid I_j \in S_{\alpha(F)}\}$,  $G_{\alpha(F)} \leftarrow (V_{\alpha(F)}, E_{\alpha(F)})$.
3.  for $i \leftarrow 1$ until $\alpha(F)$ do
        Sort the intervals in $S_i$ in descending order of the coordinates of their left endpoints.
4.  for $i \leftarrow \alpha(F) - 1$ step $-1$ until 1 do
        a) $I_q \leftarrow$ the interval in $S'_{i+1}$ whose left endpoint has the largest coordinate.
        b) $S'_i \leftarrow \{I_j \in S_i \mid r_j < l_q\}$,  $V'_i \leftarrow \{w_j \mid I_j \in S'_i\}$,  $E'_i \leftarrow \phi$.
        c) for each element $I_j \in S'_i$ do  $E'_i \leftarrow E'_i \cup \{ (w_k \rightarrow w_j) \mid w_k \in S'_{i+1}$ and $r_j < l_k \}$.
        d) $V_i \leftarrow V_{i+1} \cup V'_i$,  $E_i \leftarrow E_{i+1} \cup E'_i$,  $G_i \leftarrow (V_i, E_i)$.
5.  $Q \leftarrow \{ \{I_{q_1}, I_{q_2}, \ldots, I_{q_{\alpha(F)}}\} \mid$ there is a directed path $[w_0, w_{q_1}, w_{q_2}, \ldots, w_{q_{\alpha(F)}}]$ in $G_1$ such that
        $w_{q_{\alpha(F)}} \in V'_1$ \}.
6.  output $Q$. □
```

As noted before, XM_1 is the desired set of all MMIF's of F. Therefore, in order to prove the correctness of the algorithm, it suffices to show that, for $i = \alpha(F), \alpha(F)-1, \ldots, 1$, the directed paths in G_i from w_0 to the vertices in V'_i correspond to the IIF's in XM_i. This can in fact be done by an easy induction proof on the value of i using Theorem 3. Thus, Algorithm 2 generates all MMIF's of F.

We provide here a simple example to show how Algorithm 2 works. Consider the family of intervals $\{I_1, I_2, \ldots, I_7\}$ of Fig. 4 again. Since $\alpha(F) = 3$ and $S_3 = \{I_6, I_7\}$, graph G_3 is constructed as shown in Fig. 5(a) in Step 2. During the first execution of the **for**-loop in Step 4, the value of q is determined as 7 and S'_2 becomes $\{I_4, I_5\}$. After the graph G_2 of Fig. 5(b) is constructed, q changes in value to 5 and $S'_1 = \{I_1, I_2\}$ is found during the second execution of the **for**-loop. Since G_1 is constructed as shown in Fig. 5(c), Q is determined as $\{ \{I_1, I_4, I_6\}, \{I_1, I_4, I_7\}, \{I_1, I_5, I_6\}, \{I_1, I_5, I_7\}, \{I_2, I_5, I_6\}, \{I_2, I_5, I_7\} \}$.

Let $\delta_I(F)$ denote the total sum of the cardinalities of all MMIF's of F, that is, the total sum of the cardinalities of all MMIS's of $G_I(F)$.

Lemma 6. The time complexity of Algorithm 2 is $O(n + \delta_I(F))$.

Proof. It takes $O(\,|S_{\alpha(F)}|\,)$ time to execute Steps 1 and 2. Step 3 can be carried out in

$O(n)$ time by using a bucket sort algorithm [1]. Each execution of Step 4 requires $O(|V_i'|+|E_i'|+|S_i|)$ time for $i=\alpha(F)-1,\alpha(F)-2,\ldots,1$ since the intervals in S_{i+1}' have already been sorted in descending order of the coordinates of their left endpoints. It is easy to see that $\Sigma_{i=1}^{\alpha(F)-1}|V_i'| \le \Sigma_{i=1}^{\alpha(F)-1}|S_i| = n - |S_{\alpha(F)}|$ and that $\Sigma_{i=1}^{\alpha(F)-1}|E_i'| = |E_1| - |E_{\alpha(F)}|$. Thus, the total time required for Step 4 is $O(n+|E_1|)$. Using Procedure 1, we can execute Step 5 in $O(n+|E_1|+\gamma)$ time, where γ denotes the total sum of the numbers of vertices on the directed paths in G_1 from w_0 to the vertices in V_1'. Clearly the final step can be performed in $O(\delta_I(F))$ time. Since it can be shown that $|E_1| = O(\gamma)$ and $\gamma = O(\delta_I(F))$ (see the proof of Lemma 2), the overall time complexity of Algorithm 2 is $O(n+\delta_I(F))$. □

Theorem 4. For any family of n intervals F, one can generate all MMIS's of $G_I(F)$ in $O(n\cdot\log n+\delta_I(F))$ time. This time complexity is optimal to within a constant factor.

Proof. As claimed before, one can construct in $O(n\cdot\log n)$ time a canonical family of intervals F' such that $G_I(F)=G_I(F')$. The applications of Procedure 2 and Algorithm 2 to the resultant family of intervals require $O(n+\delta_I(F))$ time in total due to Lemmas 4 and 6. Therefore, all MMIS's of $G_I(F)$ can be generated in $O(n\cdot\log n+\delta_I(F))$ time. Using the same argument as in the second part of the proof of Theorem 2, we can conclude that this time complexity is optimal to within a constant factor. □

5. Definitions for Circular-Arc Graphs

Let $S=\{A_1, A_2, \ldots, A_n\}$ be a family of arcs on a circle C. Each endpoint of the arcs in S is assigned a real number, called a *coordinate*. The endpoints are located on the circumference of C in ascending order of their coordinates in the clockwise direction. Without loss of generality, we assume that (i) all endpoints of the arcs in S are distinct, and (ii) no single arc in S covers the entire circle C by itself.

For simplicity, we call the point with coordinate i as point i. Suppose that an arc

$A_j \in S$ begins at point i and ends at point k in the clockwise direction. Then, we call points i and k the *head* and *tail*, respectively, of A_j. For $j = 1,2,\ldots,n$, let h_j and t_j denote the coordinates of the head and tail, respectively, of A_j. If $h_j < t_j$, then A_j is called a *forward arc*; otherwise it is called a *backward arc*. For example, the family of arcs shown in Fig. 6(a) contains three backward arcs A_6, A_7 and A_8.

For an arc $A_j \in S$ and an endpoint i of another arc in S, we say that A_j *contains* point i if points h_j, i, and t_j appear on the circumference of C in this order in the clockwise direction. For two distinct arcs A_i and A_j in S, we say that they *intersect* with each other if one of them contains at least one endpoint of the other arc; otherwise A_i and A_j are said to be *independent* from each other. The *circular-arc graph* for S, which we denote by $G_C(S)$, is defined as follows :

$G_C(S) = (V, E)$, where

$V = \{v_1, v_2, \ldots, v_n\}$, and

$E = \{ (v_i, v_j) \mid A_i$ and A_j intersect with each other$\}$.

As an example, we show in Fig. 6(b) the circular-arc graph for the family of arcs of Fig. 6(a). Note that any two backward arcs in S intersect with each other. Thus, the set of vertices corresponding to the backward arcs forms a clique in the graph $G_C(S)$.

S is said to be *canonical* if (i) h_j's and t_j's for $j = 1,2,\ldots,n$ are all distinct integers between 1 and $2n$, and (ii) point 1 is the head of some arc. For instance, the family of arcs shown in Fig. 6(a) is canonical. Similarly to the case of interval graphs, when S is not canonical, using a regular sorting algorithm [1], one can construct in $O(n \cdot \log n)$ time a canonical family of arcs S' such that $G_C(S) = G_C(S')$.

A subfamily X of S is called an *independent arc family* (abbreviated to an *IAF*) if any two arcs in X are independent from each other. An IAF, X of S is called a *maximal independent arc family* (abbreviated to an *MIAF*) if no proper superfamily of X is an IAF of S, and it is called a *maximum independent arc family* (abbreviated to an *MMAF*) if its cardinality is the largest among all IAF's of S. For example, the family of arcs shown in Fig. 6(a) has three MMAF's, $\{A_1, A_3, A_4\}$, $\{A_1, A_3, A_5\}$ and $\{A_3, A_4, A_7\}$. Clearly, the MLAF's (resp., MMAF's) of S and the MLIS's (resp., MMIS's) of $G_C(S)$ are in one-to-one correspondence. In Sections 6 and 7, we will present efficient algorithms for generating all MLAF's and all MMAF's, respectively, of S.

6. Enumeration of Maximal Independent Sets of a Circular-Arc Graph

Let $S = \{A_1, A_2, ..., A_n\}$ be a canonical family of arcs on a circle C. We denote by FA_S and BA_S the families of all forward and backward arcs, respectively, in S. Leung [16] proposed an algorithm for generating all MLAF's of S. And he gave an $O(n^2+\beta_C(S))$ time implementation, where $\beta_C(S)$ denotes the size of an output, that is, the total sum of the cardinalities of the MLAF's of S. In this section, we present a more efficient implementation of his algorithm. In fact, our implementation is optimal to within a constant factor. We first show Leung's algorithm below.

Algorithm 3. [16]

```
1.  $Q \leftarrow \phi$.
2.  Partition $S$ into $FA_S$ and $BA_S$.
3.  Generate all MLAF's of $FA_S$.
4.  for each MLAF, $X$ of $FA_S$ do
        if there exists no arc $A_j \in BA_S$ such that $X \cup \{A_j\}$ is an IAF then  $Q \leftarrow Q \cup \{X\}$.
5.  for each arc $A_j \in BA_S$ do
        $Q \leftarrow Q \cup \{X \mid X$ is an MLAF of $S$ which contains $A_j\}$.
6.  output $Q$. □
```

Let $\beta_C(FA_S)$ denote the total sum of the cardinalities of all MLAF's of FA_S. We start with the following lemma.

Lemma 7. $\beta_C(S) \geq \beta_C(FA_S)$.

Proof. For any MLAF, X of FA_S, there exists at least one MLAF of S which contains X. Let Y be such an MLAF of S. Then, it is clear that X is the unique MLAF of FA_S which is contained by Y. Therefore, $\beta_C(S) \geq \beta_C(FA_S)$. □

Steps 1 and 2 of Algorithm 3 take $O(1)$ and $O(n)$ time, respectively, to execute. It is obvious that the circular-arc graph for FA_S, $G_C(FA_S)$, is an interval graph. In fact, we can construct in $O(n)$ time a canonical family of intervals F^S such that $G_I(F^S) = G_C(FA_S)$. This can be done by mapping the arcs in FA_S onto the real line **R** in such a way that the endpoint with the i-th smallest coordinate is assigned integer i as its coordinate on **R**. Thus, by applying Algorithm 1 to F^S, we can execute Step 3 in $O(n+\beta_C(S))$ time due to Lemmas 3 and 7. Recall that, during the execution of Algorithm 1, we obtain a digraph, which we will refer to as G^S. This graph has vertex w_0 and the vertices which correspond to the arcs in FA_S. It also contains a directed edge $(w_j \rightarrow w_k)$ for $1 \leq j \neq k \leq n$ if and only if $t_j < h_k$ and there is no arc $A_q \in FA_S$ such that $t_j < h_q < t_q < h_k$. This graph will be utilized again in Step 5.

In order to execute Step 4, we first sort the backward arcs in ascending order of the coordinates of their tails. Since S is canonical, this sorting can be done in $O(n)$ time by a bucket sort algorithm [1]. Suppose that $BA_S = \{A_{j_1}, A_{j_2}, ..., A_{j_k}\}$ with $t_{j_1} < t_{j_2} < \cdots < t_{j_k}$. For $i = 1,2,...,2n$, we determine $INDEX(i)$ as follows :

$$INDEX(i) \begin{cases} = 0 & (1 \leq i < t_{j_1}), \\ = Max\ \{h_{j_q} \mid 1 \leq q \leq p\} & (t_{j_p} \leq i < t_{j_{p+1}},\ p = 1,2,...,k-1), \\ = Max\ \{h_{j_q} \mid 1 \leq q \leq k\} & (t_{j_k} \leq i \leq 2n). \end{cases}$$

For example, for the family of arcs of Fig. 6(a), $BA_S = \{A_6, A_7, A_8\}$ and $t_7 = 2 < t_6 = 3 < t_8 = 7$. Since $h_7 = 14$, $h_6 = 12$ and $h_8 = 16$, we have $INDEX(1) = 0$, $INDEX(2) = INDEX(3) = \cdots = INDEX(6) = 14$, and $INDEX(7) = INDEX(8) = \cdots = INDEX(16) = 16$.

For an IAF, $X = \{A_{i_1}, A_{i_2}, ..., A_{i_q}\}$ of FA_S, we call the coordinate of the head (resp., tail) of arc A_{i_1} (resp., A_{i_q}) the *starting* (resp., *ending*) *coordinate* of X and denote it by $sc(X)$ (resp., $ec(X)$). Lemma 8 follows directly from the definitions.

Lemma 8. For an IAF, X of FA_S, there exists an arc $A_j \in BA_S$ such that $X \cup \{A_j\}$ is an IAF if and only if $INDEX(sc(X)) > ec(X)$. □

It takes $O(n)$ time to compute $INDEX(i)$ for every integer i such that $1 \le i \le 2n$. And, for every MLAF, X of FA_S, both $sc(X)$ and $ec(X)$ can be determined in $O(\beta_C(FA_S))$ time, which is bounded by $O(\beta_C(S))$ due to Lemma 7. Therefore, based on Lemma 8, we can execute Step 4 of Algorithm 3 in $O(n+\beta_C(S))$ time. It is clear that Step 6 of Algorithm 3 requires only $O(\beta_C(S))$ time. In the remainder of this section, we will describe an efficient implementation of Step 5.

For each arc $A_j \in S$, $NEXTSET_j$, $NEXTLIST_j$ and $NEXT_j$ are defined as follows. $NEXTSET_j$ is the set of arcs $\{A_k \in FA_S \mid A_k$ is independent from A_j, $t_j < h_k$, and there is no arc $A_q \in FA_S$ such that $t_j < h_q < t_q < h_k\}$. $NEXTLIST_j$ is the list in which the arcs in $NEXTSET_j$ are stored in ascending order of the coordinates of their tails. $NEXT_j$ is the first element of $NEXTLIST_j$ if $NEXTSET_j \neq \phi$; otherwise it is defined as λ. For the family of arcs of Fig. 6(a), for example, $NEXTSET_1 = \{A_2, A_3\}$, $NEXT_1 = A_3$; $NEXTSET_4 = \phi$, $NEXT_4 = \lambda$; and $NEXTSET_8 = \{A_4, A_5\}$, $NEXT_8 = A_4$.

Lemma 9. $\Sigma_{j=1}^{n} \mid NEXTSET_j \mid \le \beta_C(S)$.

Proof. For two arcs A_j and A_k in S, if $A_k \in NEXTSET_j$, then there exists at least one MLAF of S which contains both A_j and A_k. Furthermore, for any other arc A_q in $NEXTSET_j$, there is no MLAF that contains both A_k and A_q since they intersect with each other. These imply that, for each arc $A_j \in S$, there are at least $\mid NEXTSET_j \mid$ MLAF's which contain A_j. This completes the proof. □

If $NEXTSET_j$ is found for every arc $A_j \in S$, then it is easy to determine $NEXTLIST_j$'s and $NEXT_j$'s. Using a bucket sort algorithm [1], this can be done in $O(n+\beta_C(S))$ time due to

Lemma 9. Furthermore, once $NEXT_j$ and $NEXTLIST_j$ are found for every arc A_j, one can execute Steps 5 of Algorithm 3 in $O(n+\beta_C(S))$ time by using a procedure given by Leung [16]. Thus, the only remaining problem is to efficiently find $NEXTSET_j$ for every arc A_j.

For two arcs A_j and A_k in FA_S, if $t_j < h_k$, they are independent from each other. This implies that $A_k \in NEXTSET_j$ if and only if the digraph G^S has an edge $(w_j \rightarrow w_k)$. Therefore, we can determine $NEXTSET_j$ for every forward arc A_j in $O(n+\beta_C(S))$ time.

Suppose that $BA_S = \{A_{j_1}, A_{j_2}, ..., A_{j_k}\}$ with $t_{j_1} < t_{j_2} < \cdots < t_{j_k}$. For $i = 1,2,...,k$, the following procedure first discards all forward arcs A_q such that $h_q < t_{j_i}$ since they intersect with A_{j_i}. Let FA_S' denote the set of all the remaining forward arcs. The procedure then finds an arc A_p whose tail has the smallest coordinate among all arcs in FA_S'. If A_p intersects with A_{j_i}, there is no forward arc which is independent from A_{j_i}, and hence $NEXTSET_{j_i} = \phi$. On the other hand, if A_p does not intersect with A_{j_i}, we know that $A_p \in NEXTSET_{j_i}$. In this case, the procedure finds all arcs in FA_S' that contain the tail of A_p and are independent from A_{j_i}.

Procedure 3.

```
1.  Sort the arcs in $FA_S$ in ascending order of the coordinates of their tails.
2.  $FA_S' \leftarrow FA_S$.
3.  for each arc $A_j \in BA_S$ do $NEXTSET_j \leftarrow \phi$.
4.  for $i \leftarrow 1$ until $k$ do
        a) $FA_S' \leftarrow FA_S' - \{A_q \in FA_S' \mid h_q < t_{j_i}\}$.
        b) if $FA_S' \neq \phi$ then
            (1) Find an arc $A_p$ such that $t_p = Min\{t_q \mid A_q \in FA_S'\}$.
            (2) if $t_p < h_{j_i}$ then
                    $NEXTSET_{j_i} \leftarrow NEXTSET_{j_i} \cup \{A_p\} \cup \{A_q \in FA_S' \mid h_q < t_p < t_q < h_{j_i}\}$. □
```

In order to execute this procedure efficiently, we maintain the arcs in FA_S' by means of two kinds of data structures. The first one is a doubly-linked list $DLIST$ which stores all arcs in FA_S' in ascending order of the coordinates of their tails. The second one is a segment tree T_S whose root corresponds to interval $[1, 2n]$. (We refer the reader to Bentley and Wood [4] for the definition of and basic operations on a segment tree.) Initially, for each arc $A_j \in FA_S$, a segment $I_j = [h_j, t_j]$ is created and inserted into T_S. Each node v of T_S represents an interval I^v and is associated with a set of segments S^v which cover I^v. We maintain S^v in the form of a sorted list in which the segments are stored in ascending order of the coordinates of their right endpoints, that is, the coordinates of the tails of the corresponding arcs. For example, Fig. 7 shows the segment tree for the segments which correspond to the forward arcs in the family of arcs of Fig. 6(a).

Step 1 of Procedure 3 can be carried out in $O(n)$ time by a bucket sort algorithm [1]. In Step 2, we can insert, in $O(n \cdot \log n)$ time, all the segments that correspond to the arcs in FA_S', into the segment tree T_S by the method developed by Bentley and Wood[4]. Thus, Step 2 including the insertions of the elements into *DLIST* requires only $O(n \cdot \log n)$ time. Step 3 takes $O(n)$ time to execute. We show below that Step 4 can be executed in $O(n \cdot \log n + \beta_C(S))$ time.

To execute Step 4.a), we visit the endpoints of the arcs in FA_S' one by one in ascending order of their coordinates. More specifically, in the case of $i=1$, we scan points 1, 2, ..., t_{j_1-1} and remove from FA_S' all arcs whose endpoints are encountered. In the case of $i \geq 2$, we visit points $t_{j_{i-1}+1}, t_{j_{i-1}+2}, \ldots, t_{j_i}-1$ and remove all the unnecessary arcs. Thus, each endpoint is visited at most once. Each time we find an arc A_j which is to be removed from FA_S', we can delete it from *DLIST* in $O(1)$ time and I_j from T_S in $O(\log n)$ time. Therefore, the total time required for Step 4.a) is $O(n \cdot \log n)$.

Each execution of Step 4.b)(1) requires only $O(1)$ time since the desired arc A_p is the first element of *DLIST*. Recall here that, for each node v of T_S, the segments in S^v are previously sorted in ascending order of the coordinates of the tails of the corresponding arcs. Thus, for $i = 1,2,\ldots,k$, we can carry out Step 4.b)(2) in $O(\log n + |NEXTSET_{j_i}|)$ time by searching T_S in the manner described in Bentley and Wood[4]. Therefore, the total time required for Step 4 of Procedure 3 is $O(n \cdot \log n + \Sigma_{i=1}^{k} \mid NEXTSET_{j_i} \mid)$, which is bounded by $O(n \cdot \log n + \beta_C(S))$ due to Lemma 9.

Since the other steps of Procedure 3 require $O(n \cdot \log n)$ or less time, its overall time complexity is $O(n \cdot \log n + \beta_C(S))$. Therefore, for the reasons mentioned before, we can execute Step 5 of Algorithm 3 in $O(n \cdot \log n + \beta_C(S))$ time.

We have already shown that the other part of Algorithm 3 can be executed in $O(n \cdot \log n + \beta_C(S))$ time. Therefore, we have the following lemma and theorem.

Lemma 10. The time complexity of Algorithm 3 is $O(n \cdot \log n + \beta_C(S))$. □

Theorem 5. For any family of n arcs S, one can generate all MLIS's of $G_C(S)$ in $O(n \cdot \log n + \beta_C(S))$ time. This time complexity is optimal to within a constant factor.

Proof. As claimed in the preceding section, one can construct in $O(n \cdot \log n)$ time a canonical family of arcs S' such that $G_C(S) = G_C(S')$. Thus, from Lemma 10, all MLIS's of $G_C(S)$ can be generated in $O(n \cdot \log n + \beta_C(S))$ time. It is obvious that every interval graph is a circular-arc graph. Therefore, for the same reason as mentioned in the second part of the proof of Theorem 2, our algorithm is optimal to within a constant factor. □

7. Enumeration of Maximum Independent Sets of a Circular-Arc Graph

Let $S = \{A_1, A_2, ..., A_n\}$ be a canonical family of arcs on a circle C. For a subset S', we denote by $\alpha(S')$ the cardinality of an MMAF of S' and by $\delta_C(S')$ the total sum of the cardinalities of all MMAF's of S'. Furthermore, for each backward arc A_j, we define α_j^* to be $Max\{\mid X \mid \mid X$ is an IAF of S which contains $A_j\}$. Let AD_j be the set of arcs in S which intersect with A_j. Then, it is easy to see that $\alpha_j^* = \alpha(S - \{A_j\} - AD_j) + 1$. Thus, we have the following algorithm for generating all MMAF's of S.

Algorithm 4.

```
1.  $Q \leftarrow \phi$.
2.  Determine $\alpha(S)$ and $\alpha(FA_S)$.
3.  if $\alpha(S) = \alpha(FA_S)$ then generate all MMAF's of $FA_S$ and add them to $Q$.
4.  for each backward arc $A_j \in BA_S$ do determine $\alpha_j^*$.
5.  for each backward arc $A_j \in BA_S$ such that $\alpha_j^* = \alpha(S)$ do
        a) $Q_j \leftarrow$ the set of all MMAF's of $S - \{A_j\} - AD_j$.
        b) $Q \leftarrow Q \cup \{X \cup \{A_j\} \mid X \in Q_j\}$.
6.  output $Q$. □
```

We now explain how we execute Algorithm 4 in $O(n^2+\delta_C(S))$ time. Since the Masuda-Nakajima algorithm [18] finds an MMAF of S in $O(n)$ time, $\alpha(S)$ can be determined in $O(n)$ time in Step 2. As explained in the preceding section, we can construct in $O(n)$ time a canonical family of intervals F^S such that $G_I(F^S) = G_C(FA_S)$. Therefore, Procedure 2 can be used to determine $\alpha(FA_S)$ in $O(n)$ time due to Lemma 4. If $\alpha(S) = \alpha(FA_S)$, all MMAF's of FA_S are MMAF's of S and they are generated in Step 3. Such generation can be done by applying Algorithm 2 to F^S, which requires $O(\mid FA_S \mid + \delta_C(FA_S))$ time from Lemma 6. For each arc $A_j \in BA_S$, a family of arcs $S - \{A_j\} - AD_j$ does not contain any backward arcs. Thus, as is the case for FA_S, such a family of arcs can be treated as a family of intervals, and hence Procedure 2 can determine α_j^* in $O(n)$ time. Furthermore, if $\alpha_j^* = \alpha(S)$ for a backward arc A_j, we can find Q_j in $O(n + \delta_C(S - \{A_j\} - AD_j))$ time by using Algorithm 2. Therefore, Steps 4 and 5 can be executed in $O(n^2+\delta_C(S))$ time. Consequently, we have the following theorem.

Theorem 6. For any family of n arcs S, one can generate all MMIS's of $G_C(S)$ in $O(n^2+\beta_C(S))$ time. □

8. Conclusion

In this paper, we have presented algorithms for enumerating the maximal and maximum independent sets of an interval graph and a circular-arc graph. The algorithms are optimal to within a constant factor except the one for generating all maximum independent sets of a circular-arc graph. An optimal algorithm for this case remains to be found. It is also interesting to consider the same generation problems for other classes of graphs.

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] A. Apostolico and S. Hambrusch, "Finding Maximum Cliques on Circular-Arc Graphs," to appear in *Information Processing Letters.*

[3] T. Asano, T. Asano and H. Imai, "Partitioning a Polygonal Region into Trapezoids," *J. of the Association for Computing Machinery*, Vol. 33, pp. 290-312, 1986.

[4] J. L. Bentley and D. Wood, "An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles," *IEEE Trans. on Computers*, Vol. C-29, pp. 571-576, July 1980.

[5] K. S. Booth and G. S. Lueker, "Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms," *J. on Computer and System Sciences*, Vol. 13, pp. 335-379, 1976.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, CA, 1979.

[7] M. R. Garey, D. S. Johnson, G. L. Miller and C. H. Papadimitriou, "The Complexity of Coloring Circular Arcs and Chords," *SIAM J. on Algebraic and Discrete Methods*, Vol. 1, pp. 216-227, 1980.

[8] F. Gavril, "Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph," *SIAM J. on Computing*, Vol. 1, pp. 180-187, 1972.

[9] F. Gavril, "Algorithms for a Maximum Clique and a Maximum Independent Set of a Circle Graph," *Networks*, Vol. 3, pp. 261-273, 1973.

[10] F. Gavril, "Algorithms on Circular-Arc Graphs," *Networks*, Vol. 4, pp. 357-369, 1974.

[11] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, NY, 1980.

[12] U. I. Gupta, D. T. Lee and J. Y.-T. Leung, "An Optimal Solution for the Channel-Assignment Problem," *IEEE Trans. on Computers*, Vol. C-28, pp. 807-810, 1979.

[13] U. I. Gupta, D. T. Lee and J. Y.-T. Leung, "Efficient Algorithms for Interval Graphs and Circular-Arc Graphs," *Networks*, Vol. 12, pp. 459-467, 1982.

[14] W.-L. Hsu, "Maximum Weight Clique Algorithms for Circular-Arc Graphs and Circle Graphs," *SIAM J. on Computing*, Vol. 14, pp. 224-231, 1985.

[15] J. M. Keil, "Finding Hamiltonian Circuits in Interval Graphs," *Information Processing Letters*, Vol. 20, pp. 201-206, 1985.

[16] J. Y.-T. Leung, "Fast Algorithms for Generating all Maximal Independent Sets of Interval, Circular-Arc and Chordal Graphs," *J. of Algorithms*, Vol. 5, pp. 22-35, 1984.

[17] G. S. Lueker and K. S. Booth, "A Linear Time Algorithm for Deciding Interval Graph Isomorphism," *J. of the Association for Computing Machinery*, Vol. 26, pp. 183-195, 1979.

[18] S. Masuda and K. Nakajima, "An Optimal Algorithm for Finding a Maximum Independent Set of a Circular-Arc Graph," to appear in *SIAM J. on Computing*, 1987.

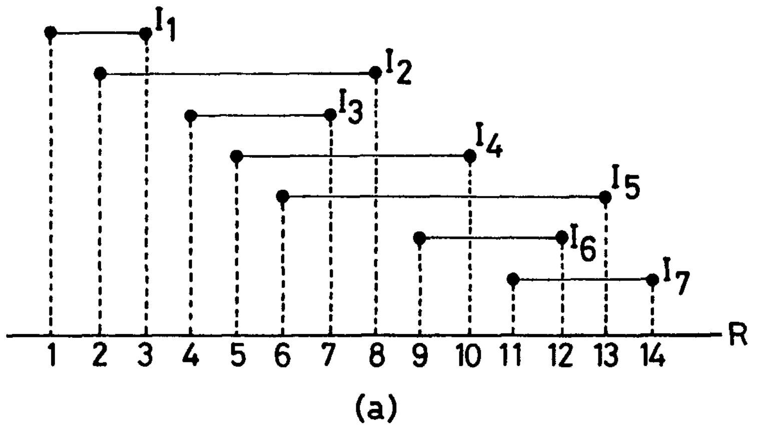
[19] T. Ohtsuki, H. Mori, E. S. Kuh, T. Kashiwabara, and T. Fujisawa, "One-Dimensional Logic Gate Assignment and Interval Graphs," *IEEE Trans. on Circuits and Systems*, Vol. CAS-26, pp. 675-684, 1979.

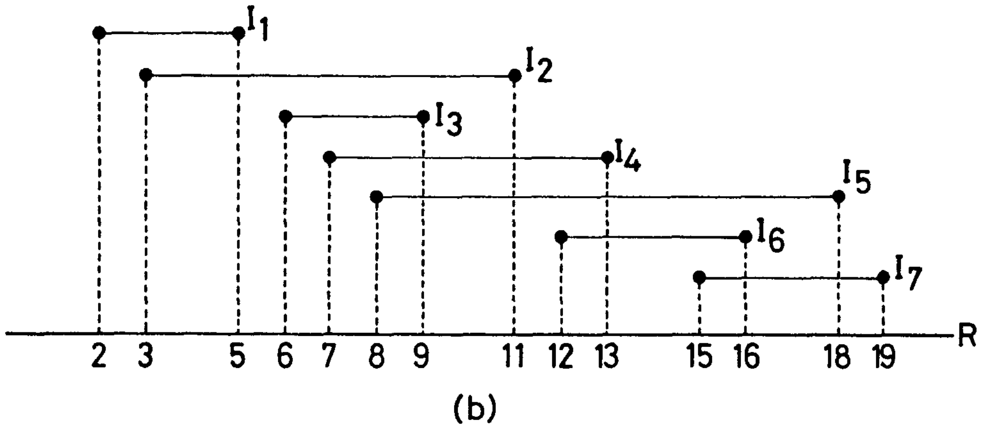
[20] M. I. Shamos and D. Hoey, "Geometric Intersection Problems," *Proc. of the 17th Annual IEEE Symposium on Foundations of Computer Science*, Long Beach, CA, 1976, pp. 208-215.

[21] A. Tucker, "An Efficient Test for Circular-Arc Graphs," *SIAM J. on Computing*, Vol. 9, pp. 1-24, 1980.

[22] O. Wing, S. Huang, and R. Wang, "Gate Matrix Layout," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-4, pp. 220-231, July 1985.

[23] O. J. Yu and O. Wing, "Interval-Graph-Based PLA Folding," *Proc. of the 1985 IEEE International Symposium on Circuits and Systems*, Kyoto, Japan, June 1985, pp. 1463-1466.





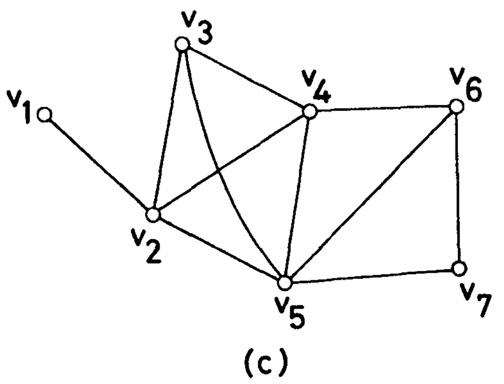
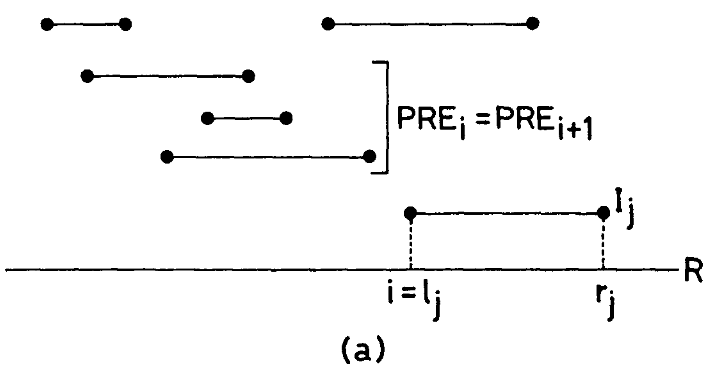


Fig. 1. (a) A canonical family of intervals.
(b) A noncanonical family of intervals.
(c) The corresponding interval graph.



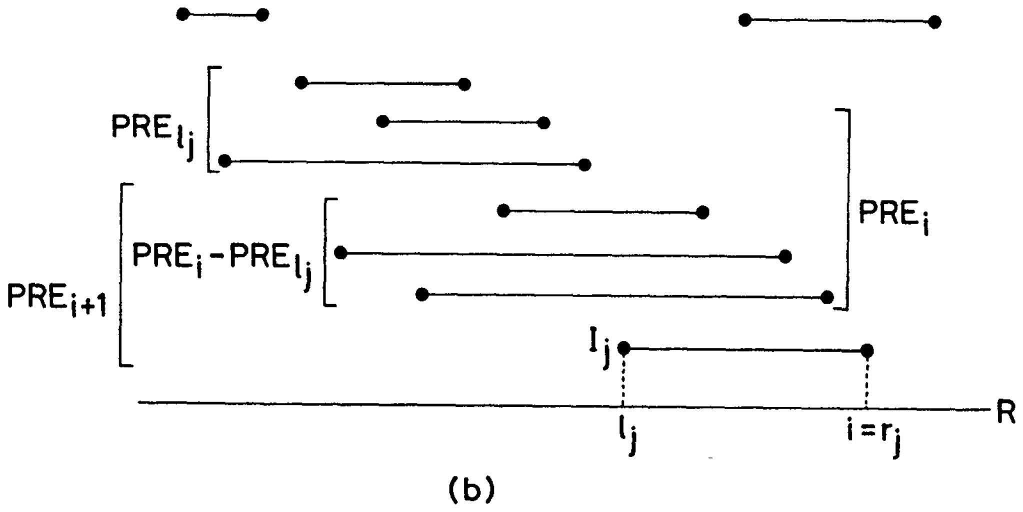


Fig. 2. The relationship between PRE_i and PRE_{i+1}.
(a) The case in which point i is the left endpoint of I_j.
(b) The case in which point i is the right endpoint of I_j.

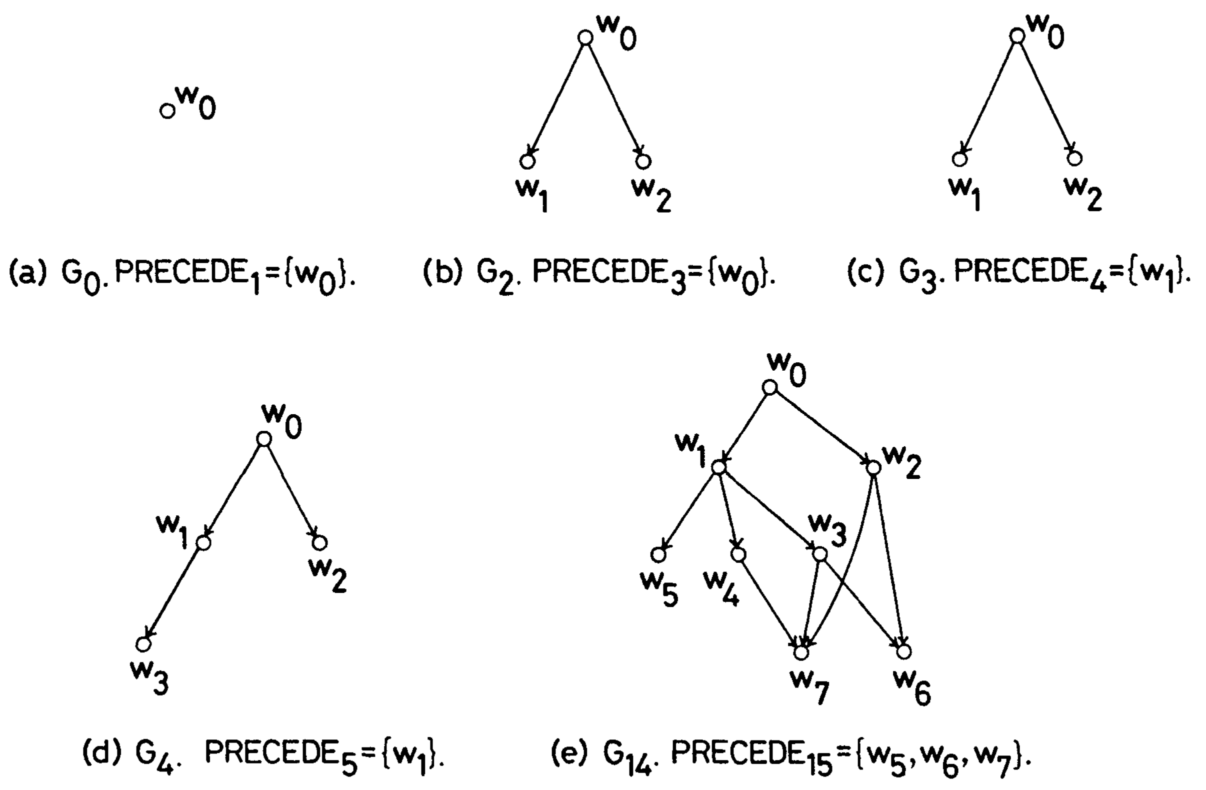
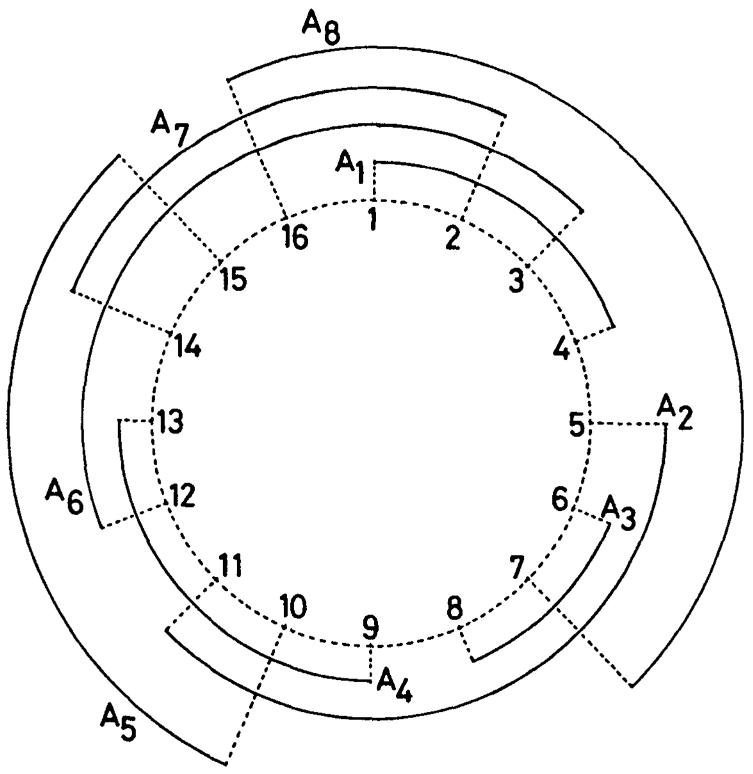
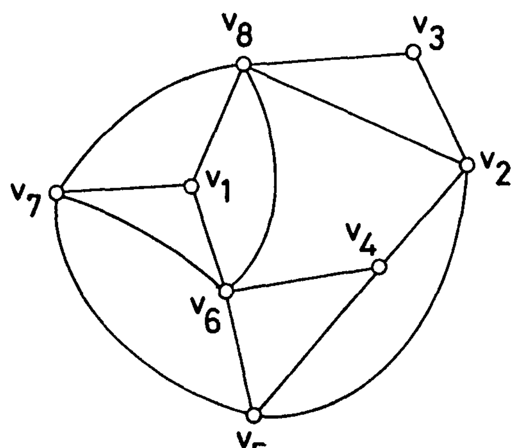

w_0
(a) G_0. PRECEDE$_1$={w_0}.
w_0 w_1 w_2
(b) G_2. PRECEDE$_3$={w_0}.
w_0 w_1 w_2
(c) G_3. PRECEDE$_4$={w_1}.
w_0 w_1 w_2 w_3
(d) G_4. PRECEDE$_5$={w_1}.
w_0 w_1 w_2 w_3 w_4 w_5 w_6 w_7
(e) G_{14}. PRECEDE$_{15}$={w_5,w_6,w_7}.

Fig. 3. Generation of all MLIF's of the family of intervals of Fig. 1(a).



(a)



(b)

Fig. 6. (a) A canonical family of arcs.
(b) The corresponding circular-arc graph.

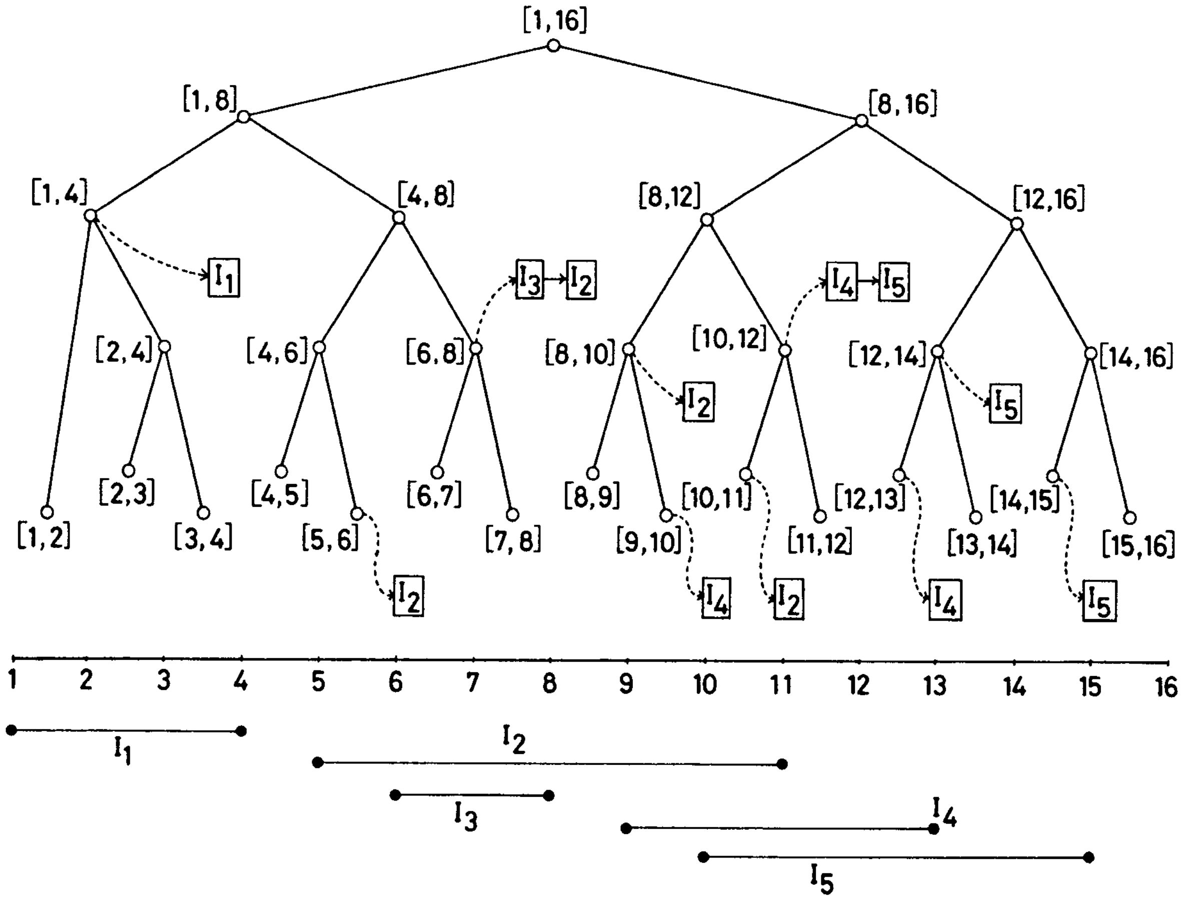


Fig. 7. The segment tree for the forward arcs in the family of arcs of Fig. 6(a).