SRC TR 86-58-r1

Efficient Algorithms for Finding Maximum Cliques of an Overlap Graph

by

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

Efficient Algorithms for Finding Maximum Cliques of an Overlap Graph*

by

Sumio MASUDA†

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.

† This author was on leave with the Electrical Engineering Department and Systems Research Center at the University of Maryland, College Park, Maryland 20742.

Efficient Algorithms for Finding Maximum Cliques of an Overlap Graph

by

Sumio Masuda, Kazuo Nakajima,

Toshinobu Kashiwabara and Toshio Fujisawa

Abstract

Let $F = \{I_1, I_2, ..., I_n\}$ be a finite family of closed intervals on the real line. Two distinct intervals I_j and I_k in F are said to overlap each other if they intersect but neither one of them contains the other. A graph $G = (V, E)$ is called an overlap graph for F 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 intervals in F overlap each other. In this paper, we present two efficient algorithms for finding maximum cliques of an overlap graph when the graph is given in the form of a family of intervals. The first algorithm finds a maximum clique in $O(n \cdot \log n + Min\{m, n \cdot \omega\})$ time, where n and m are the numbers of vertices and edges, respectively, and ω is the size of a maximum clique of the graph. The second algorithm generates all maximum cliques of the graph in $O(n \cdot \log n + m + \gamma)$ time, where γ is the total sum of their sizes.

1. INTRODUCTION

Let $F = \{I_1, I_2, ..., I_n\}$ be a finite family of closed intervals on the real line $\mathbf{R}$. Two distinct intervals I_j and I_k in F are said to *overlap* each other if they intersect each other but neither one of them contains the other. A graph $G = (V, E)$ is called an *overlap graph* for F 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 intervals in F overlap each other. We denote by $G(F) = (V_F, E_F)$ the overlap graph for F. It is known that any overlap graph is a circle graph, which is an intersection graph defined for a family of chords of a circle, and vice versa [7].

Let $G = (V, E)$ be a graph. A subset S of V is called a *clique* of G if any two distinct vertices in S are adjacent to each other. The number of vertices in S is called the *size* of S. A *maximum clique* of G is a clique whose size is the largest among all cliques of G. The problem of finding a maximum clique of G is, in general, NP-hard [1,5]. However, it is solvable in polynomial time when G is restricted to such a graph as a chordal graph [6], comparability graph [8], or circular-arc graph [9]. Furthermore, in the case of an overlap graph, namely, a circle graph, several polynomial time algorithms [3,7,9,12] have been developed for finding a maximum clique of G under the assumption that the graph is given in the form of its corresponding family of intervals F.

The first such algorithm was due to Gavril [7] and requires $O(n^3)$ time, where n is the number of intervals in F, or equivalently that of vertices in its corresponding overlap graph $G(F)$. Later, Buckingham [3] presented an $O(n \cdot \log n + m \cdot \log \omega)$ time algorithm, where m is the number of edges and ω the size of a maximum clique of $G(F)$. In both algorithms, n subproblems, each consisting of finding a maximum clique of a *permutation graph* [7,8], are generated and solved independently. The essential difference between them is that Gavril's algorithm needs $O(n^2)$ time for each subproblem while Buckingham's solves

it more efficiently by applying an algorithm for finding a *longest increasing subsequence* of a sequence [4]. The concept of longest increasing subsequences is also used in the $O(n^2)$ time algorithm developed by Rotem and Urrutia [12]. Their decomposition into subproblems is different from Gavril's or Buckingham's, but the number of the resultant subproblems is still $O(n)$. Since their algorithm maintains a data structure of $O(n)$ space for every subproblem, its overall space complexity is $O(n^2)$.

On the other hand, several authors [7,3,9] considered a more general case in which each interval in F is assigned an integer as its *weight* and developed algorithms for finding a *maximum weight clique* of $G(F)$. The best algorithm is due to Hsu [9] and requires $O(n+m \cdot \log \log n)$ time if the endpoints of the intervals in F are already sorted with respect to their coordinates and the graph $G(F)$ itself as well as F is given. Since $G(F)$ can easily be constructed from F in $O(n \cdot \log n + m)$ time (see, e.g., Sec. 4.2 of Buckingham [3]), Hsu's algorithm can find a maximum weight clique in $O(n \cdot \log n + m \cdot \log \log n)$ time.

The first goal of this paper is to present a new algorithm for finding a maximum clique of an overlap graph for the *unweighted case*. Like the above algorithms [7,3,12,9], we assume that the graph is given in the form of a family of n intervals F. We first create a sequence of subproblems, each consisting of finding a maximum clique of a permutation graph, which are slightly different from those generated by the above algorithms. We then investigate relationships between two consecutive subproblems so that we can solve each subproblem using the results of the preceding one. The time and space complexities of our algorithm are $O(n \cdot \log n + Min\{m, n \cdot \omega\})$ and $O(n)$, respectively.

In the same paper [12] as mentioned above, Rotem and Urrutia extended their algorithm for finding a maximum clique of $G(F)$ to an $O(n^2+\gamma)$ time algorithm for generating all maximum cliques of $G(F)$, where γ is the total sum of the sizes of the cliques. Their algorithm constructs auxiliary acyclic digraphs and, by scanning them with a backtracking

method, generates all the maximum cliques. A similar approach is used in Leung's algorithms [11] for generating all maximal independent sets of an interval graph and a circular-arc graph. The second goal of this paper is to extend our first algorithm to an $O(n \cdot \log n + m + \gamma)$ time algorithm for generating all maximum cliques of $G(F)$. Like Rotem and Urrutia's algorithm [12], our second algorithm constructs and searches auxiliary acyclic digraphs. However, our digraphs differ from theirs because of the different problem decomposition methods and data structures used. Moreover, a different construction method of the digraphs is developed in order to maintain the superiority in the time and space complexities of our first algorithm over theirs.

2. CANONICAL FAMILY OF INTERVALS

Let $F = \{I_1, I_2, ..., I_n\}$ be a finite family of closed intervals on the real line $\mathbf{R}$. For each interval $I_j \in F$, let l_j and r_j denote the coordinates of its left and right endpoints, respectively, that is, $I_j = [l_j, r_j]$. Note that $l_j = r_j$ if and only if the interval I_j is a point. For simplicity, we call the endpoint with coordinate i point i.

Two distinct intervals I_j and I_k in F are said to *intersect* each other if there exists a real number c such that $l_j \leq c \leq r_j$ and $l_k \leq c \leq r_k$, and I_j is said to *contain* I_k if $l_j \leq l_k \leq r_k \leq r_j$. Furthermore, if I_j and I_k intersect but neither one of them contains the other, in other words, if $l_j < l_k \leq r_j < r_k$ or $l_k < l_j \leq r_k < r_j$, then they are said to *overlap* each other. The *overlap graph* for F, denoted by $G(F)$, is defined as follows:

$G(F) = (V_F, E_F)$, where

$V_F = \{v_1, v_2, ..., v_n\}$, and

$E_F = \{(v_j, v_k) \mid I_j$ and I_k overlap each other $\}$.

As an example, a family of intervals and its corresponding overlap graph are shown in Fig. 1.

F is said to be *canonical* if the coordinates of the endpoints of its intervals are all distinct integers between 1 and $2n$. If F is not canonical, we can construct a canonical family of intervals F' such that $G(F') = G(F)$ in the following manner.

Procedure 1.

1. Construct a list of indices $L_l = [i_1, i_2, ..., i_n]$ such that (i) $l_{i_k} \leq l_{i_{k+1}}$ and (ii) if $l_{i_k} = l_{i_{k+1}}$, then $r_{i_k} \geq r_{i_{k+1}}$, for $k = 1,2,...,n-1$.

2. Construct a list of indices $L_r = [j_1, j_2, ..., j_n]$ such that (i) $r_{j_k} \leq r_{j_{k+1}}$ and (ii) if $r_{j_k} = r_{j_{k+1}}$, then j_k appears after j_{k+1} in the list L_l, for $k = 1,2,...,n-1$.

3. Merge L_l and L_r to construct a list of indices $L = [h_1, h_2, ..., h_{2n}]$ such that for $k = 1,2,...,n$, $i_k = h_{k+p_k}$ and $j_k = h_{k+q_k}$, where $p_k = |\{j \mid 1 \leq j \leq n, r_j < l_{i_k}\}|$ and $q_k = |\{i \mid 1 \leq i \leq n, l_i \leq r_{j_k}\}|$.

4. **for** $k \leftarrow 1$ **until** n **do**

   4-1) Let i and j be the integers such that $i < j$ and $h_i = h_j = k$ in L.

   4-2) $l_k' \leftarrow i$, $r_k' \leftarrow j$, $I_k' \leftarrow [l_k', r_k']$.

5. $F' \leftarrow \{I_1', I_2', ..., I_n'\}$. □

For example, if Procedure 1 is applied to the family of intervals shown in Fig. 2(a), we obtain $L_l = [1, 2, 5, 6, 3, 4]$ in Step 1 and $L_r = [6, 3, 1, 5, 4, 2]$ in Step 2. In Step 3, they are merged and $L = [1, 2, 5, 6, 3, 6, 4, 3, 1, 5, 4, 2]$ is created. Therefore, the family of intervals shown in Fig. 2(b) is obtained.

Theorem 1. For any family of intervals F, Procedure 1 correctly constructs a canonical family of intervals F' such that $G(F) = G(F')$.

Proof. Suppose that two intervals I_j and I_k with $l_j \leq l_k$ overlap each other in F. Since $l_j < l_k \leq r_j < r_k$ by definition, j appears before k in both L_l and L_r, which implies

that $l_j' < l_k'$ and $r_j' < r_k'$. Furthermore, it is clear from the descriptions of Steps 3 and 4 that $l_k' < r_j'$. Therefore, $l_j' < l_k' < r_j' < r_k'$, and thus I_j' and I_k' overlap each other in F'.

On the other hand, suppose that I_j and I_k do not overlap in F. Since Procedure 1 always yields $l_j' < r_j'$ and $l_k' < r_k'$, if I_j and I_k do not intersect each other, then $l_j' < r_j' < l_k' < r_k'$ or $l_k' < r_k' < l_j' < r_j'$. It is easy to see that, if I_j (resp., I_k) properly contains I_k (resp., I_j), then $l_j' < l_k' < r_k' < r_j'$ (resp., $l_k' < l_j' < r_j' < r_k'$). Finally, if I_j and I_k are identical, that is, $l_j = l_k \leq r_k = r_j$, then j appears before k in L_l if and only if j appears after k in L_r, which implies that either $l_j' < l_k' < r_k' < r_j'$ or $l_k' < l_j' < r_j' < r_k'$. Thus, I_j' and I_k' do not overlap in F' in any case.

Clearly no two endpoints have the same coordinate in F'. Therefore, we can conclude that Procedure 1 works correctly. □

In order to carry out Steps 1 and 2 of Procedure 1, we use *list merge sort* (see pp. 165-168 in Knuth [10]). This sorting algorithm has time complexity $O(N \cdot \log N)$, where N is the number of items to be sorted, and is *stable* (see also pp. 380-381 in Knuth [10]), that is, preserves the relative order of the items with equal keys.

In Step 1 of Procedure 1, we first sort the indices of the intervals in descending order of the coordinates of their right endpoints. Then, the list merge sorting algorithm is applied with the coordinates of the left endpoints as keys. In this manner, we can obtain $L_l = [i_1, i_2, ..., i_n]$ in $O(n \cdot \log n)$ time. Similarly, we can find L_r in $O(n \cdot \log n)$ time by applying the list merge sorting algorithm to the list $[i_n, i_{n-1}, ..., i_1]$ with the coordinates of the right endpoints as keys. The list L can be constructed in $O(n)$ time in Step 3 by using typical list merging techniques (see, e.g., pp. 159-160 in Knuth [10]). Finally, Steps 4 and 5 can easily be carried out in $O(n)$ time. Thus, we have the following lemma.

Lemma 1. The time and space complexities of Procedure 1 are $O(n \cdot \log n)$ and $O(n)$, respectively. □

3. PROPERTIES OF LONGEST INCREASING SUBSEQUENCES

The concept of longest increasing subsequences [4] of a sequence of numbers plays an important role in the development of our algorithms. This section is devoted to some definitions and lemmas on longest increasing subsequences.

Let $\pi = [\pi(1), \pi(2), ..., \pi(s)]$ be a sequence of distinct numbers. For $i = 1,2,...,s$, we say that $\pi(i)$ is an element of π and denote its position in π by $pos(\pi(i))$, that is, $pos(\pi(i)) = i$. The *length* of π, denoted by $|\pi|$, is the number, s, of elements of π. A sequence $Y = [\pi(i_1), \pi(i_2), ..., \pi(i_t)]$ of elements of π is called a *subsequence* of π if $1 \leq i_1 < i_2 < \cdots < i_t \leq s$. If, in addition, Y satisfies $\pi(i_1) < \pi(i_2) < \cdots < \pi(i_t)$, then it is called an *increasing subsequence* (abbreviated to an IS) of π. A *longest increasing subsequence* (abbreviated to an LIS) of π is an IS of the maximum length among all IS's of π.

For each element x of π, let $lseq(\pi, x)$ be defined to be $Max\{|Y| \mid Y$ is an IS of π which contains x as its first element$\}$. We define the *L-decomposition* of π as an ordered collection of sets $LD(\pi) = <X_1, X_2, ..., X_t>$, where t is the length of an LIS of π and $X_i = \{x \mid x$ is an element of π and $lseq(\pi, x) = i\}$ for $i = 1,2,...,t$. For example, for sequence $\pi = [2, 6, 4, 3, 5, 1]$, $lseq(\pi, 1) = 1$, $lseq(\pi, 2) = 3$, $lseq(\pi, 3) = 2$, $lseq(\pi, 4) = 2$, $lseq(\pi, 5) = 1$, and $lseq(\pi, 6) = 1$. Therefore, $LD(\pi) = <\{1, 5, 6\}, \{3, 4\}, \{2\}>$. For convenience, the number, t, of sets in $LD(\pi)$ is denoted by $|LD(\pi)|$. We show below five lemmas and a corollary on π and $LD(\pi) = <X_1, X_2, ..., X_t>$.

Lemma 2. If $[x, y]$ is an IS of π, then $lseq(\pi, x) > lseq(\pi, y)$.

Proof. It is clear from the definitions. □

Lemma 3. For an integer k such that $1 \leq k \leq t$, let x and y be any elements of X_k. If $x < y$, then $pos(x) > pos(y)$.

Proof. Since $x, y \in X_k$, $[x, y]$ is not an IS of π from Lemma 2. Therefore, if $x < y$, then $pos(x) > pos(y)$. □

Lemma 4. For an integer k such that $1 \leq k \leq t$, let x_1 be any element of X_k and $Y = [x_1, x_2, ..., x_k]$ be an IS of π. Then, $x_i \in X_{k-i+1}$ for $i = 2,3,...,k$.

Proof. Let i be an integer such that $2 \leq i \leq k$. From Lemma 2, $lseq(\pi, x_1) > lseq(\pi, x_2) > \cdots > lseq(\pi, x_i)$, and hence $lseq(\pi, x_i) \leq k-i+1$. On the other hand, since $[x_i, x_{i+1}, ..., x_k]$ is an IS of π, $lseq(\pi, x_i) \geq k-i+1$. Thus, $lseq(\pi, x_i) = k-i+1$, that is, $x_i \in X_{k-i+1}$. □

Corollary 1. For $k = 1,2,...,t$, $X_k \neq \phi$.

Proof. It is clear from Lemma 4. □

Lemma 5. For an integer k such that $2 \leq k \leq t$, let x_1 be an element of X_k. Let $y = Min\{z \mid z \in X_{k-1} \text{ and } z > x_1\}$. Then, $[x_1, y]$ is an IS of π.

Proof. Let $X = [x_1, x_2, ..., x_k]$ be an IS of π. From Lemma 4, $x_2 \in X_{k-1}$, and hence $y \leq x_2$. Therefore, $pos(x_2) \leq pos(y)$ by Lemma 3, and thus $pos(x_1) < pos(y)$. Since $x_1 < y$, $[x_1, y]$ is an IS of π. □

Lemma 6. Let k be an integer such that $1 \leq k \leq t$. For two elements x_1 and y_1 of X_k, let $[x_1, x_2, ..., x_k]$ and $[y_1, y_2, ..., y_k]$ be any IS's of π. Let $z_i = Min\{x_i, y_i\}$ for $i = 1,2,...,k$. Then, $[z_1, z_2, ..., z_k]$ is an IS of π.

Proof. For $i = 1,2,...,k$, $x_i, y_i \in X_{k-i+1}$ from Lemma 4. Thus, we can derive from Lemma 3 that $pos(z_i) = Max\{pos(x_i), pos(y_i)\} < Max\{pos(x_{i+1}), pos(y_{i+1})\} = pos(z_{i+1})$ for $i = 1,2,...,k-1$. Furthermore, since $x_i < x_{i+1}$ and $y_i < y_{i+1}$, $z_i = Min\{x_i, y_i\} < Min\{x_{i+1}, y_{i+1}\} = z_{i+1}$. Therefore, $[z_1, z_2, ..., z_k]$ is an IS of π. □

4. RELATIONSHIPS BETWEEN MAXIMUM CLIQUES AND LONGEST INCREASING SUBSEQUENCES

Let $F = \{I_1, I_2, ..., I_n\}$ be a canonical family of intervals. Without loss of generality, we can assume that $r_1 < r_2 < \cdots < r_n$. (The renumbering of the indices can be

performed in $O(n)$ time by using bucket sort [1], if necessary.) This assumption implies that, if two intervals I_j and I_k with $j<k$ overlap each other, then $l_j<l_k<r_j<r_k$.

For $i=1,2,...,2n$, we define a set of intervals CUT_i to be $\{I_j \in F \mid l_j<i+0.5<r_j\}$. Let τ_i be defined as the sequence of the indices of intervals in CUT_i sorted in ascending order of the coordinates of their left endpoints. For example, for the canonical family of intervals shown in Fig. 3, $CUT_6=\{I_2, I_3, I_4, I_5\}$ and $\tau_6=[2, 4, 5, 3]$. For any subset S of F, we denote by $\omega(S)$ the size of a maximum clique of the corresponding overlap graph $G(S)$.

Theorem 2. Let OPT be an integer such that $\omega(CUT_{OPT})=Max\{\omega(CUT_i) \mid i=1,2,...,2n\}$. A maximum clique of $G(CUT_{OPT})$ is also a maximum clique of $G(F)$.

Proof. Since any clique of $G(CUT_{OPT})$ is a clique of $G(F)$, $\omega(CUT_{OPT})\leq\omega(F)$. Let C be a maximum clique of $G(F)$ and let $j=Min\{k \mid v_k \in C\}$. For any integer $k\neq j$ such that $v_k \in C$, $l_j<l_k<r_j<r_k$ by the assumption on the indices, and hence $I_k \in CUT_{r_j-1}$. Thus, C is a clique of $G(CUT_{r_j-1})$ and $\omega(F)=|C|=\omega(CUT_{r_j-1})\leq\omega(CUT_{OPT})$. Therefore, $\omega(CUT_{OPT})=\omega(F)$, and a maximum clique of $G(CUT_{OPT})$ is also a maximum clique of $G(F)$. □

Based on this theorem, we can find a maximum clique of $G(F)$ by performing the following steps (I) and (II).

(I) Find an integer OPT such that $\omega(CUT_{OPT})=Max\{\omega(CUT_i) \mid i=1,2,...,2n\}$ $(=\omega(F))$.

(II) Find a maximum clique of the graph $G(CUT_{OPT})$.

For $i=1,2,...,2n$, $G(CUT_i)$ is a permutation graph [7,8]. It is known that the problem of finding a maximum clique of a permutation graph can be transformed to that of finding an LIS of a sequence [8]. In fact, we have the following theorem.

Theorem 3. For $i=1,2,...,2n$, there is a one-to-one correspondence between the

maximum cliques of $G(CUT_i)$ and the LIS's of τ_i.

Proof. Let i be an integer such that $1 \leq i \leq 2n$ and $C = \{v_{j_1}, v_{j_2}, ..., v_{j_k}\}$ with $j_1 < j_2 < \cdots < j_k$ be a subset of vertices in $G(CUT_i)$. By assumption, C forms a clique if and only if $l_{j_1} < l_{j_2} < \cdots < l_{j_k} < i+0.5 < r_{j_1} < r_{j_2} < \cdots < r_{j_k}$ (see Fig. 4). Therefore, the set of the cliques of $G(CUT_i)$ and the set of the IS's of τ_i are in one-to-one correspondence. This completes the proof. □

One can find an LIS of a sequence of N numbers in $O(N \cdot \log N)$ time [4]. However, if an LIS of every τ_i is determined independently, as is done for the subproblems in Buckingham's algorithm [3], then a total of $O(n^2 \cdot \log n)$ time would be needed. Since the length of an LIS of τ_i is equal to $|LD(\tau_i)|$, it is sufficient to find $LD(\tau_1)$, $LD(\tau_2)$, ..., $LD(\tau_{2n})$ to carry out Step (I). Let i be an integer such that $1 \leq i \leq 2n-1$. Let $LD(\tau_i) = <X_1, X_2, ..., X_t>$ and $LD(\tau_{i+1}) = <X_1', X_2', ..., X_u'>$. In what follows, we will establish some relationships between $LD(\tau_i)$ and $LD(\tau_{i+1})$.

Remark 1. If point $i+1$ is the left endpoint of some interval I_j, then τ_{i+1} is obtained by appending j at the end of τ_i (see Fig. 5(a)). □

Remark 2. If point $i+1$ is the right endpoint of some interval I_j, then j is the smallest element of τ_i, and τ_{i+1} is obtained by deleting j from τ_i (see Fig. 5(b)). □

Theorem 4. Suppose that point $i+1$ is the right endpoint of some interval I_j. If $lseq(\tau_i, j) = t$ and $|X_t| = 1$, then $u = t-1$ and $X_k' = X_k$ for $k = 1,2,...,t-1$. On the other hand, if $lseq(\tau_i, j) \neq t$ or $|X_t| \neq 1$, then $u = t$, $X'_{lseq(\tau_i, j)} = X_{lseq(\tau_i, j)} - \{j\}$, and $X_k' = X_k$ for $k = 1,2,...,lseq(\tau_i, j) - 1$, $lseq(\tau_i, j)+1$, $lseq(\tau_i, j)+2,...,t$.

Proof. It is clear from Remark 2 that $lseq(\tau_{i+1}, x) = lseq(\tau_i, x)$ for any element x of τ_{i+1}. Therefore, the theorem holds. □

In the remainder of this section, we will consider the case in which point $i+1$ is the

left endpoint of some interval I_j. We will assume that $t = |LD(\tau_i)| \geq 1$ since, if $LD(\tau_i)$ is empty, we can easily determine $LD(\tau_{i+1})$ as $<\{j\}>$.

Lemma 7. For any element x of τ_i, $lseq(\tau_i, x) \leq lseq(\tau_{i+1}, x) \leq lseq(\tau_i, x)+1$.

Proof. Since τ_i is a subsequence of τ_{i+1}, any IS of τ_i is an IS of τ_{i+1}. Thus, $lseq(\tau_i, x) \leq lseq(\tau_{i+1}, x)$. Let Y be any IS of τ_{i+1} which contains x as its first element. If it does not contain j, then Y is an IS of τ_i. If it does, then the sequence obtained by deleting j from Y is an IS of τ_i. Therefore, $lseq(\tau_{i+1}, x) \leq lseq(\tau_i, x)+1$. □

For $k = 0,1,2,...,t$, we define a set $CARRY_k$ as follows:

$CARRY_0 = \{j\}$,

$CARRY_k = \{x \mid lseq(\tau_i, x) = k \text{ and } lseq(\tau_{i+1}, x) = k+1\}$, for $k = 1,2,...,t$.

The following theorem is an immediate consequence of Lemma 7.

Theorem 5. If $CARRY_t = \phi$, then $u = t$, and otherwise, $u = t+1$. For $k = 1,2,...,t$, $X_k' = (X_k \cup CARRY_{k-1}) - CARRY_k = (X_k - CARRY_k) \cup CARRY_{k-1}$. And if $u = t+1$, then $X_{t+1}' = CARRY_t$. □

Theorem 5 implies that, if $CARRY_1$, $CARRY_2$, ..., $CARRY_t$ are obtained, one can determine $LD(\tau_{i+1})$ from $LD(\tau_i)$. The following theorems play an important role in finding $CARRY_k$'s efficiently.

Theorem 6. If $CARRY_k = \phi$ for some integer k such that $1 \leq k \leq t-1$, then $CARRY_{k+1} = CARRY_{k+2} = \cdots = CARRY_t = \phi$.

Proof. Assume that $CARRY_k = \phi$ and $CARRY_{k+1} \neq \phi$. Let x_1 be an element of $CARRY_{k+1}$. Note that $x_1 \in X_{k+1} \cap X_{k+2}'$ by definition. Let $[x_1, x_2, ..., x_{k+2}]$ be an IS of τ_{i+1}. Since $x_1 \in X_{k+2}'$, $x_2 \in X_{k+1}'$ from Lemma 4. Furthermore, since $x_1 \in X_{k+1}$, $x_{k+2} = j$ from Remark 1, and hence $[x_1, x_2, ..., x_{k+1}]$ is an IS of τ_i. This implies by Lemma 4 that $x_2 \in X_k$. Therefore, $x_2 \in X_k \cap X_{k+1}' = CARRY_k$, which contradicts the assumption that $CARRY_k = \phi$.

By repeatedly using this argument, the proof will be completed. □

Theorem 7. For $k = 1,2,...,t$, $CARRY_k = \{x \in X_k \mid x < Max(CARRY_{k-1})\}$ if $CARRY_{k-1} \neq \phi$.

Proof. Let k be an integer such that $1 \leq k \leq t$.

("$\subseteq$"*part*) Let x_1 be an element of $CARRY_k$. By definition, $x_1 \in X_k \cap X'_{k+1}$. Let $[x_1, x_2, ..., x_{k+1}]$ be an IS of τ_{i+1}. From Lemma 4 and Theorem 5, $x_2 \in X'_k = (X_k - CARRY_k) \cup CARRY_{k-1}$. Assume that $x_1 \geq Max(CARRY_{k-1})$. Since $x_1 < x_2$, $x_2 \notin CARRY_{k-1}$, and hence $x_2 \in X_k - CARRY_k \subseteq X_k$. This contradicts the fact that $x_1 \in X_k$ by Lemma 2. Therefore, $x_1 \in \{x \in X_k \mid x < Max(CARRY_{k-1})\}$.

("$\supseteq$"*part*) Let $y_1 = Max(CARRY_{k-1})$ and let x_1 be an element of $\{x \in X_k \mid x < y_1\}$. There are two cases to be considered separately.

Case 1: $k = 1$.

Since $CARRY_0 = \{j\}$, $x_1 < y_1 = j$. Thus, $[x_1, j]$ is an IS of τ_{i+1} from Remark 1, and hence $lseq(\tau_{i+1}, x_1) \geq 2$. On the other hand, since $x_1 \in X_1$, $lseq(\tau_{i+1}, x_1) \leq 2$ by Lemma 7. Therefore, $x_1 \in X'_2$, which implies that $x_1 \in X_1 \cap X'_2 = CARRY_1$.

Case 2: $k \geq 2$.

Since $y_1 \in CARRY_{k-1}$, $y_1 \in X_{k-1} \cap X'_k$. Let $[y_1, y_2, ..., y_k]$ be an IS of τ_{i+1}. Since $y_1 \in X_{k-1}$, we have $y_k = j$ and $[y_1, y_2, ..., y_{k-1}]$ is an IS of τ_i. Furthermore, since $x_1 \in X_k$, there exists an IS, $[x_1, x_2, ..., x_k]$ of τ_i. From Lemma 4, $y_q, x_{q+1} \in X_{k-q}$ for $q = 1,2,...,k-1$. Let $z_q = Min\{x_{q+1}, y_q\}$ for $q = 1,2,...,k-1$. Then, $[z_1, z_2, ..., z_{k-1}]$ is an IS of τ_i due to Lemma 6. Since $y_{k-1} < y_k = j$, $z_{k-1} < j$. Therefore, $[z_1, z_2, ..., z_{k-1}, j]$ is an IS of τ_{i+1}.

If $pos(x_2) \leq pos(y_1)$ in τ_i, then clearly $pos(x_1) < pos(z_1)$. On the other hand, suppose that $pos(y_1) < pos(x_2)$ in τ_i. Since $y_1, x_2 \in X_{k-1}$, $x_2 < y_1$ from Lemma 3, and hence $z_1 = x_2$. Thus, in either case, $pos(x_1) < pos(z_1)$ in τ_{i+1}. Furthermore, $x_1 < z_1$ since $x_1 < y_1$ and $x_1 < x_2$. Therefore, $[x_1, z_1, z_2, ..., z_{k-1}, j]$ is an IS of τ_{i+1}. Since $x_1 \in X_k$, we have $x_1 \in X'_{k+1}$ from Lemma 7, and thus $x_1 \in X_k \cap X'_{k+1} = CARRY_k$. □

5. ALGORITHM FOR FINDING A MAXIMUM CLIQUE

Our algorithm for finding a maximum clique of $G(F)$ is composed of three steps.

Algorithm 1.

I. Find an integer OPT such that $\omega(CUT_{OPT}) = \omega(F)$ by comparing $|LD(\tau_1)|$, $|LD(\tau_2)|$, ..., $|LD(\tau_{2n})|$.

II. Find $LD(\tau_{OPT})$.

III. Find a maximum clique of $G(CUT_{OPT})$ by extracting an LIS of τ_{OPT} from $LD(\tau_{OPT})$. □

In the first step of Algorithm 1, we successively finds $LD(\tau_i)$ for $i = 1,2,...,2n$. However, keeping all such L-decompositions would require a large amount of space. Instead, we maintain, for each $i = 1,2,...,2n$, only the current L-decomposition $LD(\tau_i)$ and two integers $LOCAL\_OPT$ and MAX_SIZE such that $MAX\_SIZE = |LD(\tau_{LOCAL\_OPT})| = Max\{|LD(\tau_k)| \mid k = 1,2,...,i\}$.

In what follows, we describe how the algorithm generates $LD(\tau_i)$'s. 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 $i = 1$ or $LD(\tau_{i-1})$ is empty, then $LD(\tau_i)$ is determined as $<\{j\}>$. In the former case, we also initialize both $LOCAL\_OPT$ and MAX_SIZE to be 1.

Suppose that $i > 1$ and $|LD(\tau_{i-1})| \geq 1$. Also suppose that $LD(\tau_{i-1})$ has been determined as an ordered collection of sets $<X_1, X_2, ..., X_{LAST}>$. If $i = l_j$, then $LD(\tau_i)$ is determined as $<X_1, X_2, ..., X_{LAST}>$ which is obtained by the following procedure.

Procedure 2.

1. $CARRY_0 \leftarrow \{j\}$. $k \leftarrow 1$.
2. **while** $CARRY_{k-1} \neq \phi$ **do**

2-1) **if** $k = LAST+1$ **then** $LAST \leftarrow LAST+1$ and $X_{LAST} \leftarrow \phi$.

2-2) $CARRY_k \leftarrow \{x \in X_k \mid x < Max(CARRY_{k-1})\}$.

2-3) $X_k \leftarrow (X_k - CARRY_k) \cup CARRY_{k-1}$.

2-4) $k \leftarrow k+1$. □

The correctness of Procedure 2 follows from Theorems 5, 6 and 7. If $|LD(\tau_i)|(= LAST) \geq MAX\_SIZE$ after the execution of the procedure, we update $LOCAL\_OPT$ to be i and MAX_SIZE to be $LAST$.

If $i = r_j$, then $LD(\tau_i)$ is obtained by the following procedure. Its correctness is due to Theorem 4. In this case, we do not have to update $LOCAL\_OPT$ and MAX_SIZE.

Procedure 3.

1. **if** $lseq(\tau_{i-1}, j) = LAST$ and $|X_{LAST}| = 1$

   then $LAST \leftarrow LAST-1$

   else $X_{lseq(\tau_{i-1},j)} \leftarrow X_{lseq(\tau_{i-1},j)} - \{j\}$. □

The value of $LOCAL\_OPT$ after having determined $LD(\tau_{2n})$ is the desired integer OPT. In Step II of Algorithm 1, using the above procedures, we repeatedly determine $LD(\tau_i)$'s again until $LD(\tau_{OPT}) = <X_1, X_2, ..., X_{LAST}>$ is found. Then, in Step III, an LIS of τ_{OPT} is obtained by applying the following procedure to $LD(\tau_{OPT})$.

Procedure 4.

1. $x_{LAST} \leftarrow$ any element of X_{LAST}.
2. **for** $k \leftarrow LAST-1$ **until** 1 **step** -1 **do** $x_k \leftarrow Min\{z \in X_k \mid z > x_{k+1}\}$.
3. Output $[x_{LAST}, x_{LAST-1}, ..., x_1]$. □

Lemma 8. Procedure 4 correctly finds an LIS of τ_{OPT} in $O(n)$ time.

Proof. The repeated application of Lemma 5 proves the correctness of the procedure. Since $|X_1|+|X_2|+\cdots+|X_{LAST}|=|CUT_{OPT}|\leq n$, the time complexity is $O(n)$. □

In the remainder of this section, we evaluate the complexities of Step I of Algorithm 1. For $i=1,2,...,2n$, the current L-decomposition $LD(\tau_i)=<X_1, X_2, ..., X_{|LD(\tau_i)|}>$ is represented by the following data structures.

1) $LIST_k$: A list which stores the elements of X_k in ascending order of their values. Each element x in $LIST_k$ has a pointer $NEXT(x)$ to the next element in $LIST_k$.

2) $FIRST_k$: A pointer to the first element in $LIST_k$.

3) $MEMBER$: An array such that $MEMBER(x)=loeq(\tau_i, x)$ for each element x of τ_i.

For example, Fig. 6 illustrates the contents of $LIST_k$'s and $FIRST_k$'s for L-decomposition $<\{3, 5\}, \{4\}, \{1, 2\}>$.

It is obvious that, for any L-decomposition $LD(\tau_i)=<X_1, X_2, ..., X_{|LD(\tau_i)|}>$ which is generated in Step I, $|X_1|+|X_2|+\cdots+|X_{|LD(\tau_i)|}|\leq n$. Therefore, the space complexity of Step I is $O(n)$.

Let i be an integer such that $1\leq i\leq 2n$. If point i is the right endpoint of some interval I_j, then j is the smallest element of $X_{loeq(\tau_{i-1},j)}$ in $LD(\tau_{i-1})$. Thus, using the above data structures and Procedure 3, we can find $LD(\tau_i)$ in $O(1)$ time. It also takes $O(1)$ time to determine $LD(\tau_i)$ if $i=1$ or $LD(\tau_{i-1})$ is empty. Therefore, the total time needed for these cases is $O(n)$.

If point i is the left endpoint of some interval I_j such that $2\leq j\leq n$ and $|LD(\tau_{i-1})|\geq 1$, Procedure 2 is executed to determine $LD(\tau_i)$. Let k be an integer such that $1\leq k\leq |LD(\tau_{i-1})|$ and $CARRY_{k-1}\neq\phi$. Let m_1 and m_2 be the smallest and the largest element of $CARRY_{k-1}$, respectively. (These elements can easily be determined when we find

$CARRY_{k-1}$.) Since the elements of X_k are already sorted, $CARRY_k$ can be determined in $O(|CARRY_k|+1)$ time in Step 2-2). Let m_3 be the largest element of $CARRY_k$. In Step 2-3), we can update $FIRST_k$ and $LIST_k$ by making two assignments, "$FIRST_k \leftarrow m_1$" and "$NEXT(m_2) \leftarrow NEXT(m_3)$" (see Fig. 7). Thus, the execution of Step 2-3) including the update of *MEMBER* can be done in $O(1+|CARRY_{k-1}|)$ time. Therefore, Procedure 2 determines $LD(\tau_i)$ in $O(1+\sum_{1\le q\le |LD(\tau_{i-1})|} |CARRY_q|)$ time.

Each integer j, $2\le j\le n$, is first added to $CARRY_0$ and becomes an element of X_1 when $LD(\tau_{l_j})$ is determined. Thereafter, for $k=1,2,...,lsec(\tau_{r_j-1}, j)-1$, it is added to $CARRY_k$ exactly once and moves from X_k to X_{k+1}. When $LD(\tau_{r_j})$ is determined, j is removed from $X_{lseq(\tau_{r_j-1}, j)}$ and never be added later to any $CARRY_k$. Therefore, from the above argument, the total time required for the execution of Procedure 2 is $O(n+\Sigma_{j=1}^{n} lseq(\tau_{r_j-1}, j))$. As shown in the proof of Theorem 3, graph $G(CUT_{r_j-1})$ has a clique of size $lseq(\tau_{r_j-1}, j)$ which contains v_j. This implies that $lseq(\tau_{r_j-1}, j) \le Min\{1 + (\text{the degree of } v_j \text{ in } G(F)), \omega(F)\}$ for $j=1, 2, ..., n$. Thus, $\Sigma_{j=1}^{n} lseq(\tau_{r_j-1}, j) \le Min\{n + 2\cdot|E_F|, n\cdot\omega(F)\}$. Consequently, we have the following theorem.

Theorem 8. The time and space complexities of Step 1 of Algorithm 1 are $O(n+Min\{|E_F|, n\cdot\omega(F)\})$ and $O(n)$, respectively. □

It is obvious that Step II of Algorithm 1 requires neither more time nor more space than Step I. Therefore, we have the following theorem from Lemma 8 and Theorem 8.

Theorem 9. Given a canonical family of n intervals F, Algorithm 1 finds a maximum clique of $G(F)$ in $O(n+Min\{|E_F|, n\cdot\omega(F)\})$ time and with $O(n)$ space. □

By combining this theorem and Lemma 1, the following theorem is obtained.

Theorem 10. For any family of n intervals F, a maximum clique of $G(F)$ can be

found in $O(n \cdot \log n + Min\{\,|E_F|, n \cdot \omega(F)\})$ time and with $O(n)$ space. □

6. ALGORITHM FOR GENERATING ALL MAXIMUM CLIQUES

Let $F = \{I_1, I_2, ..., I_n\}$ be a canonical family of intervals with $r_1 < r_2 < \cdots < r_n$. For $j = 1,2,...,n$, we define $OVL(I_j)$ as a set of intervals $\{I_k \in F \mid I_k$ and I_j overlap each other and $k > j\}$. Let σ_j be defined as the sequence of the indices of intervals in $OVL(I_j) \cup \{I_j\}$ sorted in ascending order of the coordinates of their left endpoints. For example, for the family of intervals shown in Fig. 3, $OVL(I_1) = \{I_3, I_4, I_5\}$, $\sigma_1 = [1, 4, 5, 3]$ and $OVL(I_3) = \phi$, $\sigma_3 = [3]$. Note that $CUT_{r_1 - 1} = \{I_1, I_2, I_3, I_4, I_5\}$, $\tau_{r_1 - 1} = [2, 1, 4, 5, 3]$ and $CUT_{r_3 - 1} = \{I_3, I_4, I_5\}$, $\tau_{r_3 - 1} = [4, 5, 3]$.

Lemma 9. Let j be an integer such that $1 \le j \le n$. For any element j_1 of σ_j, $lseq(\tau_{r_j - 1}, j_1) = lseq(\sigma_j, j_1)$.

Proof. It is clear that σ_j is a subsequence of $\tau_{r_j - 1}$. Thus, any IS of σ_j is an IS of $\tau_{r_j - 1}$, and hence $lseq(\tau_{r_j - 1}, j_1) \ge lseq(\sigma_j, j_1)$. On the other hand, let $[j_1, j_2, ..., j_k]$ be an IS of $\tau_{r_j - 1}$. It is easy to see that $l_j \le l_{j_1} < l_{j_q} < r_j \le r_{j_1} < r_{j_q}$ for $q = 2,3,...,k$. Therefore, $[j_1, j_2, ..., j_k]$ is an IS of σ_j, and thus $lseq(\tau_{r_j - 1}, j_1) \le lseq(\sigma_j, j_1)$. □

Corollary 2. For $j = 1,2,...,n$, $lseq(\sigma_j, j) = lseq(\tau_{r_j - 1}, j)$. □

Let j be an integer such that $1 \le j \le n$ and $lseq(\sigma_j, j) = \omega(F)$. Let $LD(\sigma_j) = <X_1, X_2, ..., X_{\omega(F)}>$. We introduce an auxiliary acyclic digraph $H(j) = (V(j), E(j))$ as follows:

$V(j) = V_{\omega(F)}(j) \cup V_{\omega(F)-1}(j) \cup \cdots \cup V_1(j)$, where

$V_{\omega(F)}(j) = \{w_j\}$, and

$V_k(j) = \{w_p \mid p \in X_k$ and there exists an integer q such that $w_q \in V_{k+1}(j)$, $q < p$ and $l_q < l_p\}$, for $k = \omega(F) - 1, \omega(F) - 2, ..., 1$,

and $E(j) = E_{\omega(F)-1}(j) \cup E_{\omega(F)-2}(j) \cup \cdots \cup E_1(j)$, where

$$E_k(j) = \{(w_q \rightarrow w_p) \mid w_q \in V_{k+1}(j),\ w_p \in V_k(j),\ q < p \text{ and } l_q < l_p\},$$

$$\text{for } k = \omega(F)-1,\ \omega(F)-2,\ldots,1.$$

For a directed path $P = [w_{j_1}, w_{j_2}, \ldots, w_{j_k}]$ in $H(j)$, we define its *length* to be the number of the edges on P, that is, $k-1$. From the definition of $H(j)$, we know that any longest directed path in $H(j)$ connects w_j and a vertex in $V_1(j)$ and contains $\omega(F)$ vertices. Let $PATH\_SET(j)$ be the set of all such paths in $H(j)$. It is easy to show that there is a one-to-one correspondence between $PATH\_SET(j)$ and the set of LIS's of σ_j.

Let $C = \{v_{j_1}, v_{j_2}, \ldots, v_{j_k}\}$ with $j_1 < j_2 < \cdots < j_k$ be a subset of vertices in $G(F)$. As mentioned earlier, C forms a clique if and only if $l_{j_1} < l_{j_2} < \cdots < l_{j_k} < r_{j_1} < r_{j_2} < \cdots < r_{j_k}$. This implies that C is a maximum clique of $G(F)$ if and only if $lesq(\sigma_{j_1}, j_1) = \omega(F)$ and $[j_1, j_2, \ldots, j_k]$ is an LIS of σ_{j_1}. Therefore, we have the following theorem, which provides the basis of our algorithm for generating all maximum cliques of $G(F)$.

Theorem 11. There is a one-to-one correspondence between the set of maximum cliques of $G(F)$ and $\bigcup_{\substack{1 \le j \le n \\ lseq(\sigma_j, j) = \omega(F)}} PATH\_SET(j)$. □

Consider, for example, the canonical family of intervals shown in Fig. 8(a), where $lseq(\sigma_1, 1) = lseq(\sigma_3, 3) = \omega(F) = 3$. Since $LD(\sigma_1) = <\{5, 6, 10\}, \{2, 9\}, \{1\}>$, $H(1)$ becomes as shown in Fig. 8(b) and has four longest directed paths, $[w_1, w_2, w_5]$, $[w_1, w_2, w_6]$, $[w_1, w_2, w_{10}]$, and $[w_1, w_9, w_{10}]$. Fig. 8(c) depicts the graph $H(3)$, which has five longest directed paths $[w_3, w_5, w_7]$, $[w_3, w_5, w_8]$, $[w_3, w_6, w_7]$, $[w_3, w_6, w_8]$, and $[w_3, w_9, w_{10}]$. As mentioned in Theorem 11, all of these paths correspond to the maximum cliques of $G(F)$.

We are now ready to show the framework of our algorithm for finding all maximum cliques of $G(F)$. Its correctness follows directly from Theorem 11 and Corollary 2.

Algorithm 2.

I. **for** $j \leftarrow 1$ **until** n **do** determine $lseq(\tau_{r_{j-1}}, j)$. Determine $\omega(F)$.

II. **for** each integer j such that $lseq(\tau_{r_{j-1}}, j) = \omega(F)$ **do**

a) Determine $LD(\sigma_j)$.

b) Construct digraph $H(j)$.

c) Find $PATH\_SET(j)$ and generate its corresponding maximum cliques of $G(F)$. □

As mentioned in Introduction, Rotem and Urrutia's algorithm [12] also constructs and searches auxiliary acyclic digraphs. However, their digraphs are slightly different from $H(j)$'s. Furthermore, our graph construction method, which will be shown later, is rather different from theirs. In their algorithm, for each of the $O(n)$ subproblems, a data structure of size $O(n)$ is maintained. And, during the construction of the data structures, every element is assigned pointers which are later used to determine the edges of their digraphs. On the other hand, our algorithm always maintains only one L-decomposition and repeatedly updates it. As in Algorithm 1, it is represented by $LIST_k$'s, $FIRST_k$'s and $MEMBER$ only. It might be possible to use pointers similar to those mentioned above, but if so, we would have to update them dynamically, and hence our algorithm would be unnecessarily complicated. Furthermore, it would be necessary to update such pointers attached to the elements which do not belong to any IS of length $\omega(F)$.

In what follows, we briefly describe an efficient implementation of Algorithm 2. We first determine $OVL(I_j)$ for $j = 1,2,...,n$ as a list in which the intervals are sorted in ascending order of their subscripts. This preprocessing can be performed in $O(n + |E_F|)$ time, since the construction of $G(F) = (V_F, E_F)$ needs only $O(n + |E_F|)$ time (see, e.g., Section 4.2 of Buckingham [3]). We then carry out Step I of Algorithm 2 by executing Step I

of Algorithm 1. For $j = 1,2,...,n$, using the array *MEMBER*, $lseq(\tau_{r_j-1}, j)$ can trivially be obtained just after $LD(\tau_{r_j-1})$ is determined. Furthermore, $\omega(F)$ can easily be found since it is equal to the value of *MAX_SIZE* after the determination of $LD(\tau_{2n})$. Thus, Step I of Algorithm 2 can be completed in $O(n + Min\{|E_F|, n \cdot \omega(F)\})$ time due to Theorem 8.

In order to carry out Step II, we execute Step I of Algorithm 1 again. Each time $LD(\tau_{r_j-1})$ is determined for such an integer j that $lseq(\tau_{r_j-1}, j) = \omega(F)$, we suspend the computation of $LD(\tau_i)$'s and find $PATH\_SET(j)$ by a method described below. Then, the execution of Step I of Algorithm 1 is resumed.

Suppose that $LD(\tau_{r_j-1})$ has been determined for an integer j such that $lseq(\tau_{r_j-1}, j) = \omega(F)$ during the second execution of Step I of Algorithm 1. We know from Lemma 9 that $lseq(\sigma_j, x) = lseq(\tau_{r_j-1}, x)$ for each element x of σ_j. Furthermore, the value of $lseq(\tau_{r_j-1}, x)$ can easily be obtained by using the array *MEMBER*. Since $OVL(I_j)$ has already been determined at the preprocessing stage, we can construct $LD(\sigma_j) = <X_1, X_2, ..., X_{\omega(F)}>$ in $O(|\sigma_j|)$ time in such a way that X_k is represented by a sorted list for $k = 1,2,...,\omega(F)$.

We now explain how digraphs $H(j)$'s are created in Step II b). It is trivial to determine $V_{\omega(F)}(j)$. Suppose that, for an integer k such that $1 \leq k \leq \omega(F) - 1$, we have already obtained $V_{k+1}(j)$ as a sorted list in which the vertices are stored in ascending order of their subscripts. The following procedure constructs $V_k(j)$ from $V_{k+1}(j)$. In the procedure, a list *SMALLER* is maintained and repeatedly updated so that, for each integer $p \in X_k$, it stores such integers q that $q < p$ and $w_q \in V_{k+1}(j)$ in their descending order. By Lemma 3, for any two elements x and x' in *SMALLER*, $x > x'$ if and only if $l_x < l_{x'}$. Thus, using this sorted list, we can test, in $O(1)$ time, whether $w_p \in V_k(j)$ or not. Furthermore, we can create the set of edges $\{(w_q \rightarrow w_p) \in E_k(j)\}$, if any, in time proportional to its cardinality plus one.

Procedure 5.

1. $V_k(j) \leftarrow \phi$, $E_k(j) \leftarrow \phi$.

2. $X'_{k+1} \leftarrow \{q \mid w_q \in V_{k+1}(j)\}$. *SMALLER* $\leftarrow$ an empty list.

3. **while** $X_k \neq \phi$ **do**

   3-1) Let p be the minimum element in $X'_{k+1} \cup X_k$.

   3-2) **if** $p \in X'_{k+1}$

   then insert p at the beginning of *SMALLER* and $X'_{k+1} \leftarrow X'_{k+1} - \{p\}$.

   else execute the following statements a) and b).

   a) $X_k \leftarrow X_k - \{p\}$.

   b) **if** $\{q \in SMALLER \mid l_q < l_p\} \neq \phi$ **then** $V_k(j) \leftarrow V_k(j) \cup \{w_p\}$ and

   $E_k(j) \leftarrow E_k(j) \cup \{(w_q \rightarrow w_p) \mid q \in SMALLER$ and $l_q < l_p\}$. □

This procedure finds the elements of $V_k(j)$ in ascending order of their subscripts. Thus, we can repeatedly apply this procedure and eventually obtain the digraph $H(j) = (V(j), E(j))$. For the reasons mentioned before, Procedure 5 can be performed in $O(|X_k| + |E_k(j)| + |X'_{k+1}|)$ time for $k = \omega(F) - 1, \omega(F) - 2, \ldots, 1$. By summing up these execution times, we can see that $H(j)$ can be constructed in $O(|\sigma_j| + |E(j)|)$ time.

In Step II c) of Algorithm 2, we can find $PATH\_SET(j)$ by searching $H(j)$ starting from w_j with a backtracking method. This requires $O(|V(j)| + |E(j)| + \gamma_j)$ time, where γ_j is defined as the total sum of the lengths of the longest directed paths in $H(j)$. Clearly $|V(j)| \leq |\sigma_j|$. It is easy to see that, for any directed edge $(w_q \rightarrow w_p)$ in $E(j)$, there exists a longest directed path from w_j which passes along this edge. Thus, $|E(j)| \leq \gamma_j$. Therefore, for any integer j such that $1 \leq j \leq n$ and $lseq(\sigma_j, j) = \omega(F)$, the time required by Steps II a), b) and c) is $O(|\sigma_j| + \gamma_j)$. Furthermore, from Theorem 8, we can find all such integers j in $O(n + Min\{|E_F|, n \cdot \omega(F)\})$ time.

Let γ denote the total sum of the sizes of all maximum cliques of $G(F) = (V_F, E_F)$. From Theorem 11, $\sum_{\substack{1 \le j \le n \\ lseq(\sigma_j, j) = \omega(F)}} \gamma_j = O(\gamma)$. Moreover, it is clear by definition that $\Sigma_{j=1}^{n} \mid \sigma_j \mid = n + \mid E_F \mid$. Therefore, the total time needed for Step 2 is $O(n + \mid E_F \mid + \gamma)$.

Theorem 12. Given a canonical family of n intervals F, Algorithm 2 is implementable to run in $O(n + \mid E_F \mid + \gamma)$ time and with $O(n + \mid E_F \mid)$ space.

Proof. As explained above, the preprocessing, Steps I and II can be carried out in $O(n + \mid E_F \mid)$, $O(n + Min\{\mid E_F \mid, n \cdot \omega(F)\})$ and $O(n + \mid E_F \mid + \gamma)$ time, respectively. Therefore, the time complexity of Algorithm 2 is $O(n + \mid E_F \mid + \gamma)$. As for the space complexity of Algorithm 2, we know that the preprocessing uses $O(n + \mid E_F \mid)$ space since $G(F)$ is constructed. Step I needs only $O(n)$ space due to Theorem 8. For any integer j such that $lseq(\sigma_j, j) = \omega(F)$, the space complexity of Step II is proportional to $\mid \sigma_j \mid + \mid E(j) \mid \le n + \mid E_F \mid$ if each longest directed path in $H(j)$ is not maintained but released just after it is found. Therefore, Step II can be carried out with $O(n + \mid E_F \mid)$ space. □

Remark 3. As stated in the above proof, each time a longest directed path in $H(j)$ is found, we generate its corresponding maximum clique of $G(F)$ and discard the path. Otherwise, it would need $O(n + \mid E_F \mid + \gamma)$ space to maintain all the cliques. □

From Theorem 12 and Lemma 1, we have the following theorem.

Theorem 13. For any family of n intervals F, all maximum cliques of $G(F)$ can be generated in $O(n \cdot \log n + \mid E_F \mid + \gamma)$ time and with $O(n + \mid E_F \mid)$ space. □

7. CONCLUSION

In this paper, we first developed an algorithm for finding a maximum clique of an overlap graph $G(F) = (V_F, E_F)$ when the graph is given in the form of its corresponding

family of intervals F. The algorithm runs in $O(n \cdot \log n + Min\{ \mid E_F \mid, n \cdot \omega(F)\})$ time and with $O(n)$ space, where $n = \mid F \mid$ and $\omega(F)$ is the size of a maximum clique of $G(F)$. We then presented an algorithm for generating all maximum cliques of $G(F)$. This algorithm requires $O(n \cdot \log n + \mid E_F \mid + \gamma)$ time and $O(n + \mid E_F \mid)$ space, where γ is the total sum of their sizes. We feel it difficult to develop faster algorithms, say $O(n \cdot \log n)$ and $(n \cdot \log n + \gamma)$ time algorithms, for finding a single and all maximum cliques, respectively, of an overlap graph.

Recently, Apostolico and Hambrusch [2] presented algorithms for finding a maximum clique of an overlap graph for both unweighted and weighted cases. Their algorithm for the unweighted case is similar to our first algorithm. However, their implementation is different from ours and requires $O(n \cdot \omega(F) \cdot \log(2n/\omega(F)))$ time, which is slightly worse than our result. On the other hand, their algorithm for the weighted case can be executed in $O(n^2)$ time. It is comparable to the best known algorithm developed by Hsu [9], especially for the case when the graph $G(F)$ is dense.

Acknowledgement We would like to express our gratitude to the referees for their suggestions which have significantly improved the presentation of this paper.

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, "New Clique and Independent Set Algorithms for Circle Graphs," Technical Report CSD-TR-608, Dept. of Computer Sciences, Purdue University, West Lafayette, IN, 1986.

[3] M. A. Buckingham, "Circle Graphs," Technical Report NSO-21, Courant Institute of Mathematical Sciences, New York University, New York, NY, 1980.

[4] M. L. Fredman, "On Computing the Length of the Longest Increasing Subsequences," *Discrete Mathematics*, Vol. 11, 1975, pp. 29-35.

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

[6] 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, 1972, pp. 180-187.

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

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

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

[10] D. E. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[11] 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.

[12] D. Rotem and J. Urrutia, "Finding Maximum Cliques in Circle Graphs," *Networks*, Vol. 11, 1981, pp. 269-278.

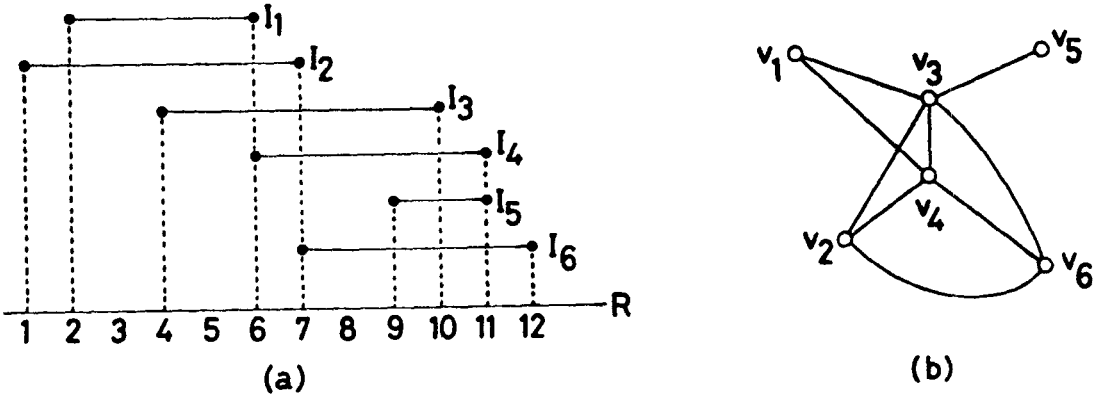

Fig. 1. (a) A family of intervals.
(b) Its corresponding overlap graph.

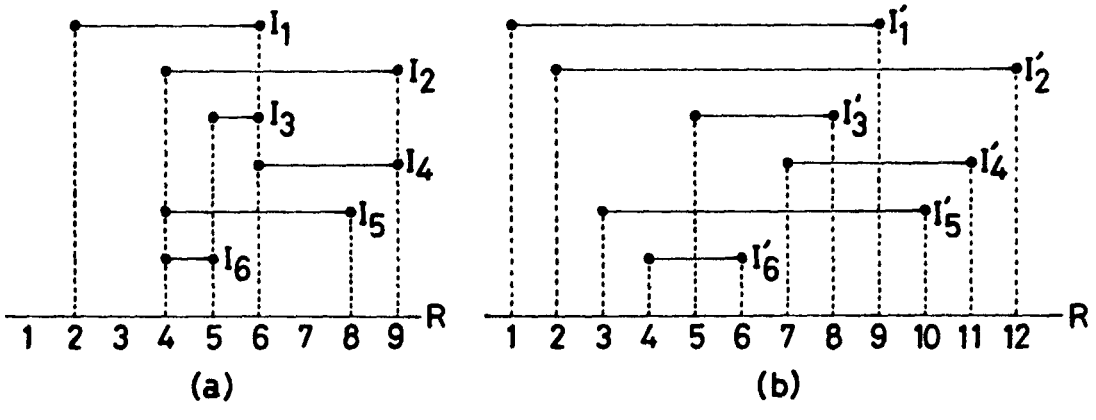

Fig. 2. (a) A non-canonical family of intervals.
(b) A canonical family of intervals.

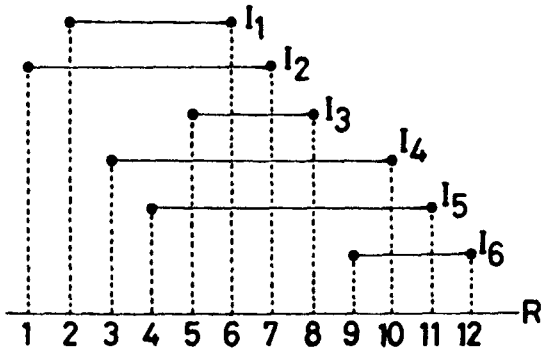

Fig. 3. A canonical family of intervals.

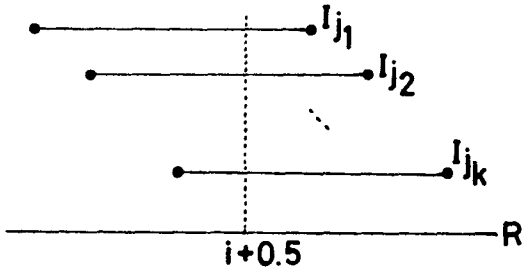

Fig. 4. A family of intervals which corresponds to a clique.

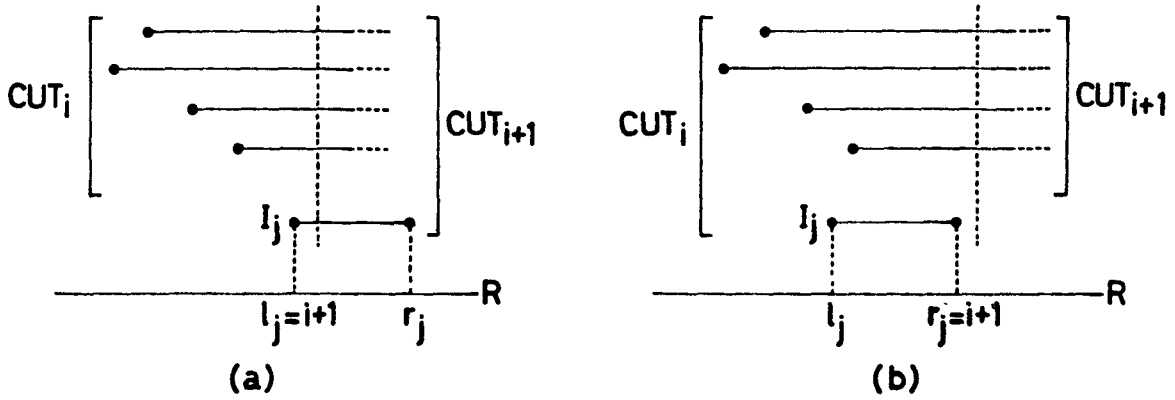

Fig. 5. The relationship between CUT_i and CUT_{i+1}.
(a) Point $i+1$ is the left endpoint of I_j.
(b) Point $i+1$ is the right endpoint of I_j.

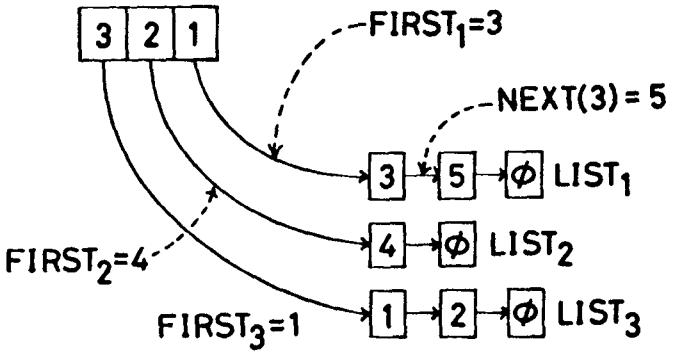

Fig. 6. Data structures for L-decomposition $<\{3, 5\}, \{4\}, \{1, 2\}>$.

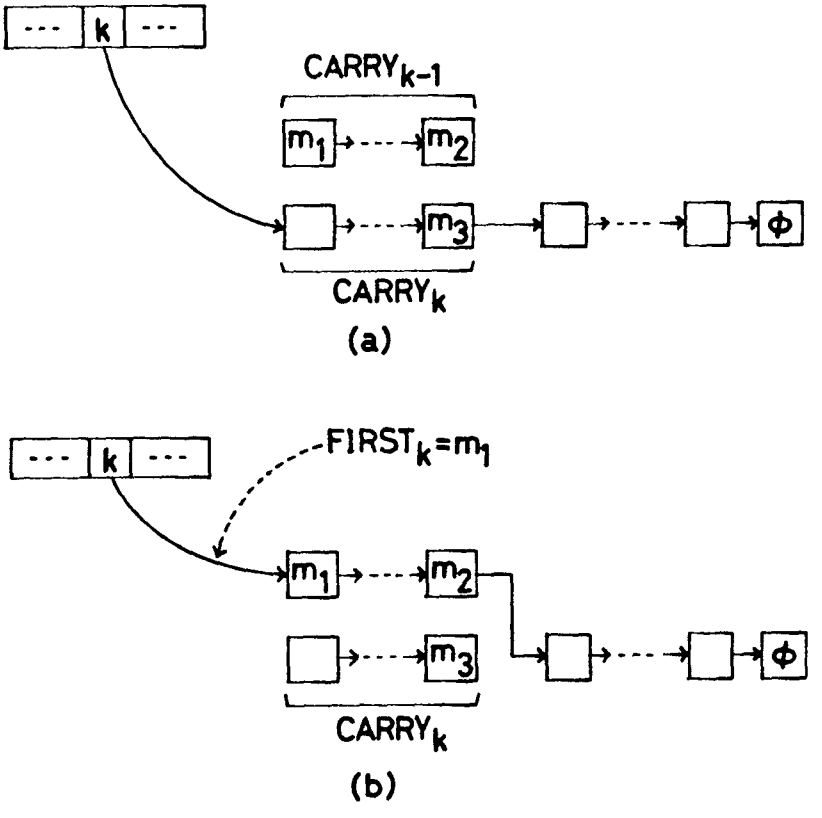

Fig. 7. Pointer manipulations for the execution of Step 2-3) of Procedure 2. (a) Before the execution. (b) After the execution.

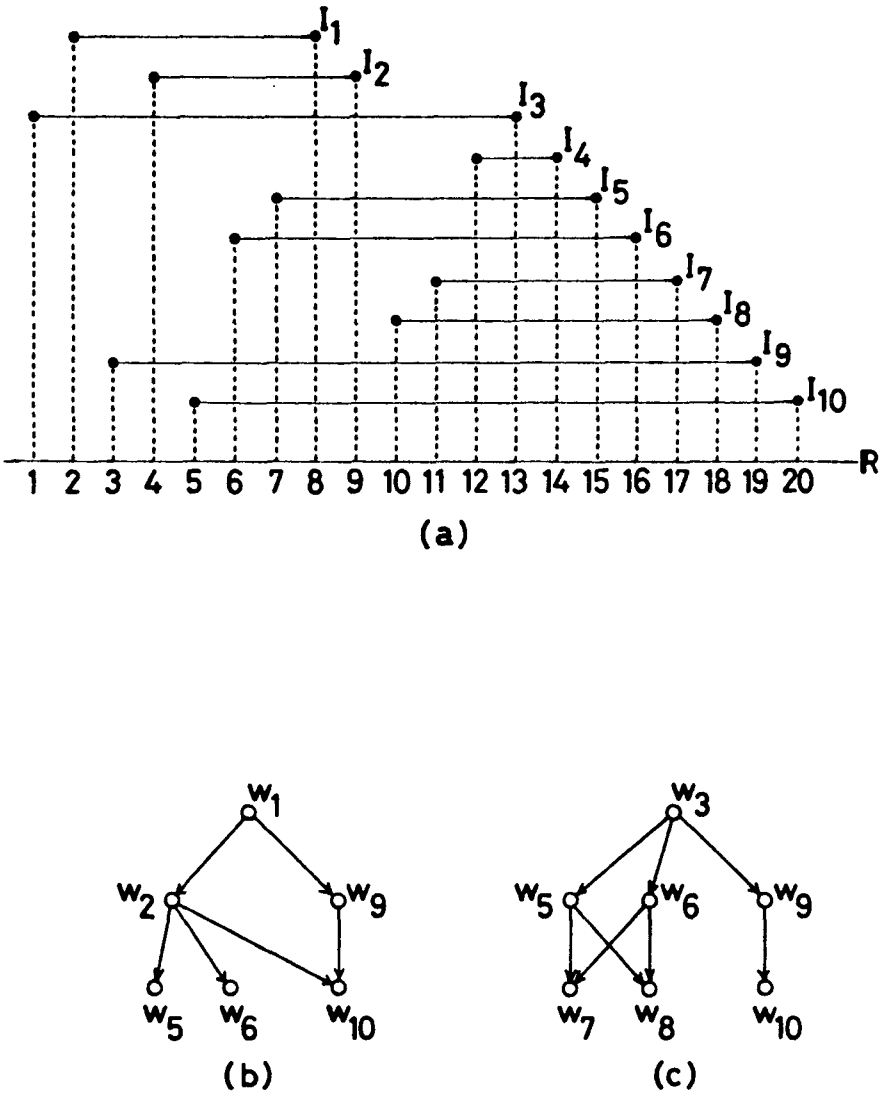

Fig. 8. Generation of all maximum cliques of $G(F)$.
(a) A family of intervals F. (b) Graph $H(1)$. (c) Graph $H(3)$.