FINDING MAXIMUM CLIQUES IN CIRCLE GRAPHS

by

D. Rotem[+] and J. Urrutia[*]

Research Report CS-80-38

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada

[+] Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

[*] Department of Combinatorics and Optimization
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

This research was supported by the National Sciences and Engineering Research Council of Canada under grant numbers A8142 and A3055 and the Consejo Nacional de Ciencia Technologia (CONACT) of Mexico.

To appear in NETWORKS

A B S T R A C T

A circle diagram consists of a circle C and a set of n chords. This diagram defines a graph with n vertices where each vertex corresponds to a chord, and two vertices are adjacent if their corresponding chords intersect in C. A graph G is called a circle graph if it is defined by some circle diagram.

An algorithm which requires $O(n^2)$ steps to generate one maximum clique is presented. The algorithm can also be used to generate all maximum cliques where the number of steps needed to generate each additional maximum clique is linear in its size. This compares favourably with Gavril's algorithm [4] which works in $O(n^3)$ steps.

1. Introduction

A circle diagram $C(\bar{v}_1,\bar{v}_2,\ldots,\bar{v}_n)$ consists of a circle C with a set of chords $\bar{v}_1,\bar{v}_2,\ldots,\bar{v}_n$. This diagram defines a graph G with a vertex set $V(G) = \{v_1,v_2,\ldots,v_n\}$ such that v_i is adjacent to v_j if their corresponding chords $\bar{v}_i$ and $\bar{v}_j$ intersect in $C(\bar{v}_1,\ldots,\bar{v}_n)$. (See Fig. 1).

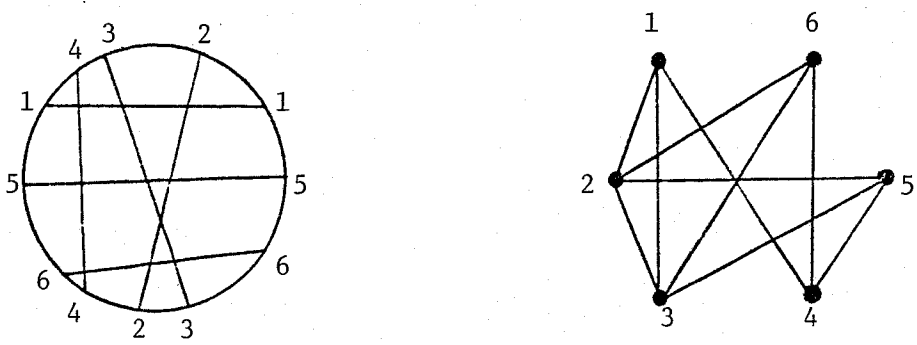


Figure 1 : A Circle diagram and its corresponding graph.

A graph G is called a circle graph if there exists a circle diagram which defines G. These graphs, sometimes under different names, have appeared from time to time in the literature (Read [8], Read and Rosenstiehl [9] and Zelinka [10]). Algorithms for finding a

maximum clique and a maximum independent set of a circle graph are given in Gavril [4]. These algorithms require $O(n^3)$ steps.

In this paper we present an algorithm for finding a maximum clique in $O(n^2)$ steps. This algorithm is based on a representation of circle graphs by Even and Itai [1] and some properties of permutation graphs ([2] [6]) which are a special type of circle graphs.

In Section 3, we present the algorithm of [1] for representing a circle graph by a permutation P in order to prove some properties of P which are required in our algorithm. In Section 4 we generalize this algorithm and show how to generate all (or any number of) maximum cliques of a circle graph. In this algorithm, the cost of transforming from one maximum clique to the next is linear in the size of the maximum clique and no clique is generated more than once.

2. Preliminaries and Definitions

A circle diagram $C(\bar{v}_1,\bar{v}_2,\ldots,\bar{v}_n)$ is called a permutation diagram if it is possible to draw a line L inside the circle C such that L crosses all the chords $\bar{v}_1,\ldots,\bar{v}_n$. (See Fig. 2.) A circle graph G which can be defined by a permutation diagram is called a permutation graph (PG). The class of PG was studied in Pnueli et al [6] and Even et al [2] where it is shown that the vertices of a PG G can be labelled with the set $N = \{1,2,\ldots,n\}$ to obtain a labelled graph G(N) such that there exists a permutation $P = \langle P(1),\ldots,P(n)\rangle$ where vertices i and j are adjacent in G(N) if and only if

$$(i-j)(P^{-1}(i)-P^{-1}(j)) < 0,$$

or in words i and j form an inversion in P.

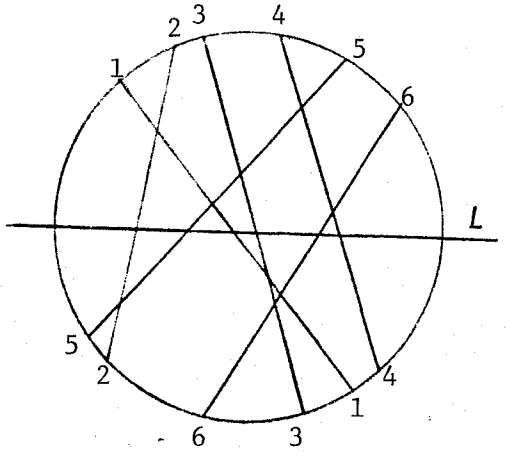


Figure 2. A circle graph which is a permutation graph. The representing permutation is $P = \langle 5,2,6,3,1,4 \rangle$. A maximum clique is $\langle 5,3,1 \rangle$ which is an LDS in P.

Clearly the vertices of a PG can be labelled with any set of numbers $I = \{i_1,\ldots,i_n\}$ with $i_1 < i_2,\ldots, < i_n$ such that there exists a permutation P on I where i_k is adjacent to i_ℓ if and only if i_k and i_ℓ form an inversion in P.

A decreasing subsequence in P is a sequence of elements $i_1 > i_2 > \ldots > i_k$ such that $P^{-1}(i_1) < P^{-1}(i_2) < \ldots < P^{-1}(i_k)$. It follows from these definitions that the vertices of a clique in G(N) correspond to a decreasing subsequence in P and vice versa. A maximum clique corresponds to a longest decreasing subsequence (LDS) in P. (See Fig. 2.) Clearly an LDS becomes an LIS (longest increasing subsequence) if P is read from right to left. An element $x \epsilon P$ is a left to right maximum in P if all elements on its left are smaller than it. The above observations are used in the algorithm of Section 3 where the problem of finding a maximum clique in a circle graph G is reduced to finding maximum cliques in certain permutation subgraphs of G.

3. Representation of Circle Graphs

In [1] Even and Itai present an algorithm which constructs a permutation P from a given circle diagram $C(\bar{v}_1,\bar{v}_2,\ldots,\bar{v}_n)$. They also show that $C(\bar{v}_1,\ldots,\bar{v}_n)$ can be reconstructed from P. By using P, it will be shown that we can find permutation subgraphs $P_1,\ldots,P_k$ whose union is G (the circle graph defined by $C(\bar{v}_1,\ldots,\bar{v}_n)$). We only present here the algorithm for the construction of P, the interested reader is referred to [1] for proof of correctness and other related results.

Given $C(\bar{v}_1,\bar{v}_2,\ldots,\bar{v}_n)$:

Step 1: $i \leftarrow 1$.

Step 2: Mark a point (not an endpoint of a chord) on the circle. This is the current artificial vertex.

Step 3: From the current artificial vertex move clockwise along C. Label each unlabelled endpoint of a chord encountered by i' and its other endpoint by i, incrementing i by 1 after each labelling, until a labelled endpoint j is met.

Step 4: Assign the label i to the current artificial vertex and increment i by 1.

Step 5: If all endpoints are labelled go to Step 6; else continue moving clockwise from j ignoring all labelled endpoints until an unlabelled endpoint p is met. Mark a point between p and the last labelled endpoint found. This point becomes the current artificial vertex. Go to Step 3.

Step 6: The representing permutation P for G is obtained by reading the unprimed labels from C moving clockwise from the first artificial vertex. (See Fig. 3 .) The permutation P will be called a representing permutation of G (clearly there are many such permutations) and its size is $n+k$ where k is the number of artificial vertices introduced. □

Remark 1: Note that all the endpoints labelled in Step 3 get labels which are smaller than the current artificial vertex. Therefore, for every primed label j' there exists at least one artificial vertex with a label bigger than j', which precedes j' on C.

Remark 2: During the labelling process, when an artificial vertex is assigned a label, it is bigger than any of the unprimed labels which precede it on C. Therefore, an artificial vertex has a label which is bigger than all elements on its left in P, and hence the artificial vertices form the sequence of left to right maxima in P.

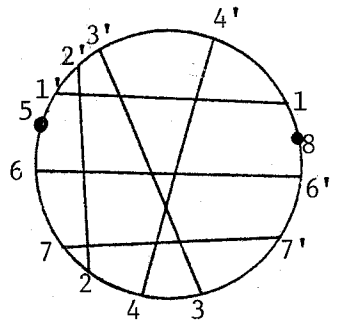


Figure 3.

Labelling of a circle graph. $P = \langle 5,1,8,3,4,2,7,6 \rangle$ with artifical vertices 5 and 8. $RS_P(5) = \langle 1,3,4,2 \rangle$

Remark 3: After the first artificial vertex is created in step 2, a new artificial vertex is introduced in step 5 only when an unlabelled endpoint is met. Therefore $k \leq n$, and $k = n$ only when $C(\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_n)$ has no intersections and it is drawn such that an artificial vertex appears before each primed label.

In what follows, chords or artificial vertices are simply called by the label i assigned to them by the above algorithm. For an element m in P, we denote by $RS_P(m)$ the subsequence of elements in P which appear to the right of m in P and are smaller than m (See Fig. 3.).

In the remaining part of this section we prove some properties of P which allow us to obtain a set of permutation subgraphs of G such that any maximum clique of G is contained in at least one such subgraph.

Lemma 1: For an artificial vertex i, the chords whose labels are elements of $RS_P(i)$ form a permutation graph.

Proof: Assume that during the labelling process the current artificial vertex is labelled i (in Step 4) after encountering a label j (in Step 3). We construct a line L passing through the first artificial vertex $P(1)$ and an interior point z_i in the arc between $(i-1)'$ and j on C (see Fig. 4). We now show that L crosses all chords whose labels are in $RS_P(i)$ which proves the Lemma by the definition of a permutation graph.

An element x is in $RS_P(i)$ if it corresponds to a chord with endpoints x and x' such that x appears after i on C and $x < i$. Since all labels on the arc from i to z_i are primed, x appears on the arc from z_i to $P(1)$. The other endpoint x' cannot appear on the arc from z_i to $P(1)$ since in that case we would have $x > i$. Hence the chord x must cross L.

□

Theorem 1: Let G be a circle graph with a representing permutation P and artificial vertices $a_1, a_2, \ldots, a_k$ then:

(a) An induced subgraph K of G is a complete subgraph if the labels of $V(K)$ form an increasing subsequence in at least one of the subsequences

$$RS_P(a_1), RS_P(a_2), \ldots, RS_P(a_k).$$

(b) For $1 \leq j \leq k$, any increasing subsequence in $RS_P(a_j)$ corresponds to a complete subgraph in G.

Proof

(a) Let the vertices of a complete subgraph K in G be labelled $k_1 < k_2 < \ldots < k_m$. Then $\langle k_1, k_2, \ldots, k_m \rangle$ forms an increasing subsequence in P. Also, the endpoints labelled $k_1, \ldots, k_m$ appear after k_m' on C. By Remark 1, there exists an artificial vertex q, where $q > k_m$ which precedes k_m' on C. Therefore $k_1, \ldots, k_m$ belong to $RS_P(q)$.

(b) This part follows from Lemma 1 and the 1-1 correspondence between complete subgraphs in a permutation graph and increasing subsequences in its representing permutation mentioned in Section 2. Note that $RS_P(a_j)$, read from right to left, is a representing permutation for the chords whose labels belong to this set.

□

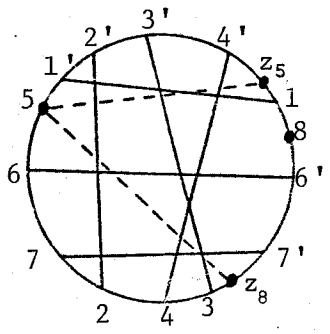


Figure 4
Permutation graphs for $RS_P(5)$ and $RS_P(8)$

We observe that by using Theorem 1, the problem of finding a maximum clique in G is now reduced to that of finding the longest LIS in the subsequences $RS_P(a_1), RS_P(a_2), \ldots, RS_P(a_k)$.

In Algorithm - 2, we present a well known method for finding the length of an LIS of a given sequence. This problem is closely related to the problem of constructing the first row of a Standard Young Tableaux (Knuth [5, Sec. 5.1.4], Fredman [3] and Schensted [7]).

Algorithm - 2

The input sequence $S = \langle s_1, s_2, \ldots, s_n \rangle$ is scanned from left to right and an ordered set of queues $Q_1, Q_2, \ldots, Q_\ell$ is formed from it.

Step 1: s_1 is inserted as the first element of Q_1.

Step 2: Assume $Q_1, Q_2, \ldots, Q_{i-1}$ were formed from $s_1, s_2, \ldots, s_{j-1}$. The element s_j is attached to the first queue which has its last element bigger than s_j. If no such queue exists, s_j is inserted as the first element of a new queue Q_i.

□

The next lemma summarizes the properties of this construction and is proved in [7].

Lemma 2: If ℓ queues are formed by the above construction from S then:

(a) The length of the LIS in S is ℓ.

(b) The elements in Q_i $(1 \leq i \leq \ell)$ form a decreasing subsequence of S.

(c) During the execution of the algorithm, if m non-empty queues were formed, the last element of Q_i is bigger than the last element of Q_{i-1} for $2 \leq i \leq m$. □

Let $Q^j = \langle Q_1^j, Q_2^j, \ldots, Q_m^j \rangle$ be the ordered set of queues generated from an input sequence $RS_P(a_j)$ by Algorithm 2. It follows from Lemma 2 that the size of a maximum clique in G is equal to the cardinality of a largest set among the k sets $Q^1, Q^2 \ldots, Q^k$. Next, we prove some properties of P which allow us to generate all the sets $Q^1, Q^2, \ldots, Q^k$ using one scan of P from left to right.

For an element $p \in P$ which does not correspond to an artificial vertex, let a_r and a_{r+s} be the smallest and biggest artificial vertices such that $p \in RS_P(a_r) \cap RS_P(a_{r+s})$. By Remark 2 it follows that $p \in RS_P(a_j)$ for $r \leq j \leq r+s$. Let us assume that by applying Algorithm-2 to $RS_P(a_j)$ p joins $Q_{m_j}^j$ for $r \leq j \leq r+s$. Then p is called <u>monotonic</u> if $m_r \geq m_{r+1} \geq \ldots \geq m_{r+s}$.

<u>Lemma 3</u>: Every $p \in P$ is monotonic.

Proof: Let $x \in P$ be the leftmost non-monotonic element in P. Then there exist two sequences $RS_P(a_j)$ and $RS_P(a_{j+1})$ such that $x \in RS_P(a_j) \cap RS_P(a_{j+1})$ and x is in Q_m^j and $Q_{m'}^{j+1}$ with $m < m'$

We consider the situation at the time when x is inserted into $Q_{m'}^{j+1}$. Let q be the last element in $Q_{m'-1}^{j+1}$ at this time, then $q < x$ and $q \in RS_P(a_j)$. Also, since q is to the left of x in P, q must be monotonic and therefore $q \in Q_\ell^j$ where $\ell \geq m' - 1$. It follows from Lemma 2(c) that the last elements in $Q_1^j, Q_2^j, \ldots, Q_{m'-1}^j$ are all smaller than x and therefore the queue Q_m^j into which x can be inserted satisfies $m > m' - 1$. This contradicts the assumption that x is non-monotonic.

□

Algorithm - 3

Given $P = \langle p_1,\ldots,p_{n+k}\rangle$, we generate a vector $\langle a_1,a_2,\ldots,a_k\rangle$ of artificial vertices and the sets $Q^1,Q^2,\ldots,Q^k$ by scanning P from left to right. The algorithm inserts each non artificial vertex p_i into every Q^j such that $p_i \in RS_P(a_j)$. The variable 'temp' is used for recording the queue in Q^j into which p_i was inserted. By Lemma 3 and Lemma 2(c), the search in Q^{j-1} can continue from Q^{j-1}_{temp} thus saving all the comparisons with the last elements of Q^{j-1}_m with $m <$ 'temp'.

```
1. a_1 ← p_1; m ← 1; (p_1 is the first artificial vertex)
2. For      2 ≤ i ≤ n+k     do
3. begin
4.    temp ← 1;
5.    if  p_i > a_m  then do (check whether p_i is an artificial vertex)
6.       begin
7.            m ← m+1 ;
8.            a_i ← p_i ;
9.       end
10.   else begin   (insert  p_i  into all Q^j such that  p_i ∈ RS_P (a_j) )
11.         j ← m ;
12.         while  p_i < a_j  and  j > 0  do
13.            begin
14.               insert  p_i  into  Q^j  using Algorithm-2 ;
15.               In Step 2 of Algorithm-2, start the search by comparing
16.               p_i  to the last element of  Q^j_temp  and assume  p_i  joins  Q^j_t;
17.                  temp ← t;
18.                  j ← j-1;
19.            end;
20.      end
21. end
```

An upper bound on the total number of comparisons made by Algorithm 3 can be obtained as follows:

Let G be a circle graph with n vertices, k artificial vertices and a maximum clique of size ℓ.

The insertion of each non artificial vertex into the queues requires at most $\ell + k + 1$ comparisons (lines 12-16) whereas for each artificial vertex we make only 1 comparison (line 5). This gives a total of $n(k+\ell+1) + k = O(n^2)$ comparisons. The number of assignment statements is easily seen to be also $O(n^2)$.

4. Generating Maximum Cliques

In this section we present a method to generate any required number of maximum cliques in a circle graph. It was shown in the previous section that given a sequence S, Algorithm - 2 generates $Q_1,\ldots,Q_\ell$ from it where ℓ is the length of an LIS in S. First, we shall have to modify Algorithm - 2 in order to keep track of elements which belong to an LIS such that subsequently all LIS of S can be generated.

It follows from Lemma 2 that if L is an LIS of S then L has exactly one element in each Q_i. In order to generate an LIS we need a method to select correctly an element from each queue. To this end, we introduce the following extension which follows Step 2 in Algorithm - 2 and call the resulting algorithm Algorithm - 2*.

Step 2b: Attach two pointers to the element s_j when it is inserted into Q_i $(i > 1)$. These pointers called FIRST(s_j) and LAST(s_j) show the positions of the first and last elements in Q_{i-1} which are smaller than s_j. Clearly, LAST(s_j) is equal to the number of elements in Q_{i-1} at the time of insertion.

Lemma 4: Let S be an input sequence to Algorithm - 2* from which ℓ queues are formed. A subsequence $L = \langle s_{i_1}, s_{i_2}, \ldots, s_{i_\ell} \rangle$ of S is an LIS if and only if:

(a) $s_{i_j} \in Q_j$ for $1 \leq j \leq \ell$,

(b) the position m of $s_{i_{j-1}}$ in Q_{j-1} satisfies

$$\text{FIRST}(s_{i_j}) \leq m \leq \text{LAST}(s_{i_j})$$

for $2 \leq j \leq \ell$.

Proof: A subsequence L of length ℓ in S is an LIS if and only if its elements satisfy:

(1) $s_{i_{j-1}} < s_{i_j}$ and (2) $s_{i_{j-1}}$ is to the left of s_{i_j} in S for $2 \leq j \leq \ell$.

Lemma 2 and (2) imply that $s_{i_{j-1}}$ is present in Q_{j-1} when s_{i_j} is inserted, also by (1) the position m of $s_{i_{j-1}}$ in Q_{j-1} must satisfy (b).

Conversely, if (a) and (b) are satisfied, then from (b) $s_{i_{j-1}} < s_{i_j}$ and $s_{i_{j-1}}$ must appear on the left of s_{i_j} in S. Therefore L is an LIS.

□

We now proceed to generate all LIS (or any required number) of S as follows. We pick the first element s_{i_ℓ} in Q_ℓ, and by Lemma 4 all its immediate successors in an LIS must be selected from $Q_{\ell-1}$ from positions ranging from $\text{FIRST}(s_{i_\ell})$ to $\text{LAST}(s_{i_\ell})$. After

selecting an element $s_{i_{\ell-1}}$ from $Q_{\ell-1}$ we can proceed to select elements from queues $Q_{\ell-2}, Q_{\ell-3},\ldots,Q_1$ in the same way, thus obtaining an LIS. The problem of generating all LIS is equivalent to a depth first search of a tree T such that the root of T on level 0 is a dummy element with its sons being all the elements of Q_ℓ. In general, on the i^{th} level of T we have elements of $Q_{\ell-i+1}$ such that the sons of an element in level i are all its immediate successors in an LIS arranged from left to right according to their position in $Q_{\ell-i}$. The leaves of T are elements of Q_1. A new LIS is generated each time that a leaf of T is reached during this traversal.

EXAMPLE 1

Consider an input sequence S = <2,5,1,3,6,4,9,8,7>. The queues formed from S are given below where the underlined numbers are elements of S followed by the two pointers FIRST and LAST.

Q_1 = $\underline{2}$,$\underline{1}$

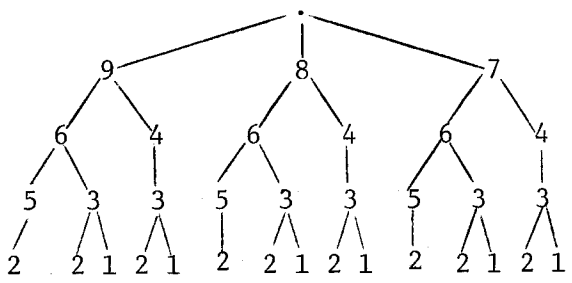
Q_2 = $\underline{5}$: 1 : 1, $\underline{3}$: 1 : 2

Q_3 = $\underline{6}$: 1 : 2, $\underline{4}$: 2 : 2

Q_4 = $\underline{9}$: 1 : 2, $\underline{8}$: 1 : 2, $\underline{7}$: 1 : 2

This system of queues can be shown as the tree T (repetition of subtrees is given only for illustration purposes).

T =



The LIS will be generated in the following order

1) <9,6,5,2> 4) <9,4,3,2> 7) <8,6,3,2> 10) <8,4,3,1> 13) <7,6,3,1>

2) <9,6,3,2> 5) <9,4,3,1> 8) <8,6,3,1> 11) <7,6,5,2> 14) <7,4,3,2>

3) <9,6,3,1> 6) <8,6,5,2> 9) <8,4,3,2> 12) <7,6,3,2> 15) <7,4,3,1>

□

In order to generate all maximum cliques in a circle graph G we use Algorithm 3 which now uses Algorithm - 2* in line 14.

Assume that maximum $\{|Q^i|\} = \ell$ for $1 \le i \le k$. By Lemma 2(a), the size of a maximum clique in G is ℓ. We therefore proceed by taking the sets $Q^{i1}, Q^{i2}, \ldots, Q^{ij}$ with cardinality ℓ, and then generate all the LIS contained in each of them using the depth first search scheme.

Note that while processing Q^{it} we can avoid generating any LIS which appeared previously for $Q^{i1}, Q^{i2}, \ldots, Q^{it-1}$. We observe that if the biggest element in an LIS generated from Q^{it} is smaller than the artificial vertex $a_{i_{t-1}}$ then this LIS is already included in Q^{it-1} and should not be generated again. To implement this, we can check the elements of Q^{it}_{ℓ}, the last queue in Q^{it}. This queue contains the largest elements of every LIS generated from Q^{it}, therefore if any element in Q^{it}_{ℓ} is smaller than $a_{i_{t-1}}$ it can be omitted. If no elements remain in Q^{it}_{ℓ}, then Q^{it} does not contain any new maximum clique.

EXAMPLE 2

Consider a circle graph which is represented by the permutation $P = \langle 9,3,11,2,4,1,6,10,8,7,5\rangle$. The artificial vertices here are $a_1 = 9$ and $a_2 = 11$, (the left to right maxima in P).

$RS_P(9) = \langle 3,2,4,1,6,8,7,5\rangle$ and $RS_P(11) = \langle 2,4,1,6,10,8,7,5\rangle$;

Applying Algorithm-3 we get

$$Q^1 = \begin{array}{l} \underline{3},\ \underline{2},\ \underline{1} \\ \underline{4} : 1 : 2 \\ \underline{6} : 1 : 1\ ,\ \underline{5} : 1 : 1 \\ \underline{8} : 1 : 1\ ,\ \underline{7} : 1 : 1 \end{array}$$

and

$$Q^2 = \begin{array}{l} \underline{2},\ \underline{1} \\ \underline{4} : 1 : 1 \\ \underline{6} : 1 : 1,\ \underline{5} : 1 : 1 \\ \underline{10} : 1 : 1,\ \underline{8} : 1 : 1,\ \underline{7} : 1 : 1 \end{array}$$

The LIS generated from Q^1 are:

$\langle 8,6,4,3\rangle$

$\langle 8,6,4,2\rangle$

$\langle 7,6,4,3\rangle$

$\langle 7,6,4,2\rangle$

We observe that in Q^2_4, 8 and 7 are smaller than 9, the previous artificial vertex, and can be omitted. The only new maximum clique is therefore $\langle 10,6,4,2\rangle$. □

Let us consider now an upper bound on the total number of steps which are needed to generate all maximum cliques. First, we show that step 2(b) which is now added to Algorithm - 2 can be implemented efficiently so that Algorithm 3 still runs in $O(n^2)$ steps. We recall that step 2(b) updates two pointers FIRST and LAST for each element that joins a queue. Assume that the queue Q_i contains the elements $q_1, q_2, \ldots, q_m$. Then by Lemma 2

$$\text{FIRST } (q_1) \leq \text{FIRST } (q_2) \leq \ldots \leq \text{FIRST } (q_m).$$

This can be used to conduct a 'catenated' search in Q_{i-1} so that the search for FIRST (q_2) starts from position FIRST (q_1) etc. In this way, updating all FIRST pointers in Q_i requires $|Q_i| + |Q_{i-1}|$ comparisons.

By the same arguments used in the analysis of Algorithm 3, it can be shown that in total at most $O(k.n)$ comparisons are required by Algorithm 3, to update the pointer FIRST for all elements in $Q^1, Q^2, \ldots, Q^k$. The pointer LAST requires only 1 assignment statement per each insertion. We conclude that Algorithm 3 uses at most $O(n^2)$ steps as claimed.

The number of steps for generating each additional maximum clique is clearly bounded by ℓ which is the height of the 'depth first search' tree whereas the number of such cliques can be exponential [4].

6. Conclusions

A method of finding all maximum cliques of a circle graph was presented. The algorithm works in $O(n^2)$ steps in order to find the size of a maximum clique, and then requires at most $O(\ell)$ steps to generate each additional maximum clique where ℓ is size of a maximum clique in the graph. The efficiency of the method depends also on the number k of artificial vertices where in all non-trivial cases $k<n$. However, for a given circle diagram, we can obtain at most $2n$ different representing permutations depending on the choice of the first artificial vertex on C. Different labellings can result in a different number of artificial vertices and it is still not known which of those labellings will give the best performance.

Acknowledgement

The authors thank the referee for a suggestion which helped to improve the running time from $O(n^2 lg_2 n)$ to $O(n^2)$.

References

[1] Even, S. and A. Itai, "Queues, Stacks and Graphs", Theory of Machines and Computations, Z. Kohavi and A. Paz, ed., Academic Press, New York, 1971, pp.71-86.

[2] Even, S., A. Lempel and A. Pnueli, "Permutation Graphs and Transitive Graphs", J. ACM, 19, 1972, pp. 400-410.

[3] Fredman, M.L., "On Computing the length of longest increasing subsequences. Discrete Math. 11, (1975), 29-35.

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

[5] Knuth, D.E., "The art of Computer Programming, Vol. 3, Addison Wesley 1973, Reading Mass.

[6] Pnueli, A., A. Lempel, and S. Even, "Transitive Orientation of Graphs and Identification of Permutation Graphs", Canadian J. Math. 23 (1971) pp. 160-175.

[7] Schensted, C., "Longest Increasing and Decreasing Subsequences", Canadian J. Math 13 (1961) 179-191.

[8] Read, R.C., "The Chord Intersection Problem". To appear in Anuals N.Y. Academic Sciences.

[9] Read, R.C. and P. Rosenstiehl, "On the Gauss Crossing Problem" Collog. Math. Societatis Janos Bolyai, 18, Combinatorics, Keszthely (Hungary) 1976, pp.843-876.

[10] Zelinka, B., "Graf Systemu Tetiv Dane Kruznice", Matematicko-Fyzikalny Casopis SAV. 15.4, 1965, pp.273-279.