LETTER Special Issue on Discrete Mathematics and Its Applications

Finding a Maximum Weight Independent Set of a Circle Graph

Takao ASANO†, Hiroshi IMAI††, *Members*
and Akira MUKAIYAMA†, *Nonmember*

SUMMARY We present an algorithm for finding a maximum weight independent set of a circle graph. For a circle graph of a set of n chords with N endpoints, the algorithm finds a maximum weight independent set in $O(nN)$ time and $O(n)$ space.

1. Introduction

The problem of decomposing a polygon into simpler components is of fundamental importance in computational geometry and often arises in VLSI layouts. Closely related to this problem is the problem of finding a maximum set of independent chords in a circle (i. e., the problem of finding a maximum independent set of a circle graph). In fact, algorithms for finding a maximum set of independent chords in a circle are used in decomposing a simple polygon into a minimum number of trapezoids[1] and into a minimum number of uniformly monotone polygons[6].

Gavril first obtained an $O(n^3)$ time algorithm for finding a maximum set of independent chords for a set of n chords with N endpoints on a circle. Later, Read et al.[7], Buckingham[2], and Asano et al.[1] proposed $O(n^2)$ time algorithms. Recently, Liu and Ntafos proposed an $O(N^3)$ time and $O(N^2)$ space algorithm[6]. Note that $n=O(N^2)$ (and $n^2=O(N^4)$) in the worst case.

In this paper, we present another efficient algorithm which is a slight modification of the algorithm in Ref. (1). The proposed algorithm runs in $O(nN)$ time and $O(n)$ space. Thus, it may be considered to be an improvement of $O(n^2)$ time algorithms as well as Liu and Ntafos's algorithm.

2. Preliminaries

For a set of chords each of which connects two points on a circle C, a subset of chords is independent if no two of them have a point in common. Here, we consider each chord is a closed set, i. e., both of its endpoints on C belong to it (the argument below can be easily modified in case each chord is an open set). Then the problem we consider is to find a maximum set of independent chords in C. This can be restated as a problem on special graphs known as circle graphs, intersection graphs of chords in circles. That is, in a circle graph, each vertex represents a chord in a circle C and two vertices are connected by an edge iff the corresponding chords intersect (i.e., have a point in common). Thus, finding a maximum set of independent chords is equivalent to finding a maximum independent set of a circle graph (an independent set of a graph is a subset of vertices no two of which are connected by an edge). When each vertex has a weight, the weight of an independent set is defined to be the sum of the weights of vertices in the set.

A circle graph is closely related to an overlap graph defined below. Let S be a set of n intervals on a line with N endpoints. For intervals $I=[x^-, x^+]$ and $J=[y^-, y^+]$, we say I is completely contained in J if $y^- < x^-$ and $x^+ < y^+$. Two intervals are said to overlap if they intersect without one being completely contained in the other. An overlap graph $G(S)$ of S is obtained by identifying each interval with a vertex and connecting two vertices by an edge iff the corresponding intervals overlap. An interval graph, an intersection graph of intervals, is obtained by identifying each interval with a vertex and connecting two vertices by an edge iff the corresponding intervals intersect. There is a slight difference between an overlap graph and an interval graph. Since the class of circle graphs coincides with the class of overlap graphs[4], we will concentrate on finding a maximum weight independent set of an overlap graph.

3. Finding a Maximum Weight Independent Set of an Overlap Graph

Consider a set S of n intervals with N endpoints. Each interval I of S has a nonnegative weight $w(I)$. In this section we will describe an algorithm for finding a maximum weight independent set of the overlap graph $G(S)$ of S. We assume that N endpoints are sorted and labeled from 1 to N. This can be done in $O(N \log N)$ time and $O(N)$ space (if the set of endpoints is considered to be a multiset, this sorting can be done in $O(n \log n)$ time and $O(n)$ space). We ignore this complexity

Manuscript received September 14, 1990.

† The authors are with the Faculty of Science and Technology, Sophia University, Tokyo, 102 Japan.
†† The author is with the Faculty of Engineering, The University of Tokyo, Tokyo, 113 Japan.

because it does not dominate the complexity of our algorithm (note that $N=\Omega(\sqrt{n})$). For convenience sake, we introduce a new interval $I_0=[0, N+1]$ with weight $w(I_0)=0$. All intervals of S are of course completely contained in I_0.

For an interval I of S, let $S(I)$ be the set of intervals of S completely contained in I. Define $W(I)$ to be the weight of a maximum weight independent set of the overlap graph of $S(I)\cup\{I\}$ (that always contains I if $w(I)$ is positive). Thus, it is easily observed that $W(I)$ satisfies the following[1),(4),(5)]:

$W(I)=w(I)+$[the weight of a maximum weight independent set of the interval graph of $S(I)$ where the weight of each interval J of $S(I)$ is $W(J)$].

Thus, $W(I_0)$ becomes the weight of a maximum weight independent set of the overlap graph $G(S)$ of S (and $G(S\cup\{I_0\})$).

Gavril proposed an $O(n^3)$ time algorithm based on the above observation[4]. In fact, he has iteratively solved the problem of finding a maximum weight independent set of an interval graph for a subset of those intervals to compute $W(I)$'s. Note that, to compute $W(I)$, we have only to compute $W(J)$ in advance for each J of S completely contained in I. Thus, there are many possibilities about the order to compute $W(I)$'s. In this paper we adopt a method of computing $W(I)$'s in decreasing order of the left endpoints of the intervals. Note also that, for intervals I's having their left endpoint in common, we can compute $W(I)$'s simultaneously.

Let L_k $(k=0, 1, \cdots, N-1)$ be the set of intervals of $S\cup\{I_0\}$ having k as their left endpoint. Let I_k be the interval in L_k with the largest right endpoint. $S(I_k)$ is the set of intervals of S completely contained in I_k. Let $\{x_1, \cdots, x_{a(k)}\}$ be the set of endpoints of the intervals in $L_k\cup S(I_k)$. We assume $x_1<\cdots<x_{a(k)}$. Then our algorithm for computing $W(I)$'s $(I\in L_k)$ can be described as follows (we assume that, for each J in $S(I_k)$, $W(J)$ is already computed).

```
Procedure COMP_W(L_k, I_k, S(I_k));
begin
 1  sort a(k) numbers x_1, ..., x_a(k) in increasing
      order;
      {comment: we assume x_1<...<x_a(k). x_1 is the
      left-most endpoint of the intervals in L_k.}
 2  U:=0;
 3  for i:=2 to a(k) do
      begin
 4      for each interval J in S(I_k) with x_i as
          its left endpoint do
          {comment: U is the weight of a
          maximum weight independent set of
          the interval graph of intervals in
          S(I_k) whose right endpoints are
          less than x_i}
 5        U(J):=U+W(J);
 6      if x_i is the right endpoint of an
          interval I in L_k then
 7        W(I):=U+w(I);
 8      UMAX:=0;
 9      for each interval J in S(I_k) with x_i as
          its right endpoint do
10        if UMAX<U(J) then
11          UMAX:=U(J);
          {comment: UMAX=max[U(J)]}
12      if UMAX>U then
13        U:=UMAX;
      end;
end;
```

The validity of the algorithm can be easily shown by an argument similar to the one in Ref. (1) (since, at each x_i, U is the weight of a maximum weight independent set of the interval graph of those intervals in $S(I_k)$ whose right endpoints are less than x_i). Thus, the whole algorithm for computing $W(I_0)$, the weight of a maximum independent set of the overlap graph $G(S)$ of S, as well as all other $W(I)$'s, can be written as follows:

```
Procedure COMP_WEIGHT;
begin
  for k:=N-1 downto 0 do
      if k is the left endpoint of an interval of S
      then
          begin
            find L_k, I_k and S(I_k);
            COMP_W(L_k, I_k, S(I_k));
          end;
end;
```

Now we consider the time complexity. Line 1 in COMP_$W(L_k, I_k, S(I_k))$ can be done in $O(N)$ time since the endpoints are labeled from 1 to N. Lines 3–13 can be done in $O(|L_k|+|S(I_k)|)$ time. Thus, COMP_$W(L_k, I_k, S(I_k))$ requires only $O(n)$ time. Since L_k, I_k and $S(I_k)$ can be found in $O(n)$ time, COMP_WEIGHT requires only $O(nN)$ time. Similarly, it can be shown that the required space is $O(n)$. It is also easy to modify the above algorithm in such a way that it actually finds a maximum weight independent set. Thus by summarizing we have:

[Theorem] A maximum weight independent set of an overlap graph of n intervals with N endpoints (or, of a circle graph of n chords with N endpoints on a circle) can be found in $O(nN)$ time and $O(n)$ space.

4. Remarks

In Ref. (5) an $O(n\log n+m_c\,\mathrm{loglog}\,n)$ time algorithm for finding a maximum weight independent set

of an overlap graph of n intervals is described, where m_c is the number of the pairs of intervals one of which is completely contained in the other. Employing the technique described there, we can similarly refine the complexity analysis of our algorithm by using the persistent search tree proposed by Sarnak and Tarjan[8] (the hive graph proposed by Chazelle[3] can also be used). The persistent search tree is one of the most powerful data structures in computational geometry. It can be used to efficiently solve the following (static) interval search problem: Given a set S of n intervals on a line and a query interval, find the set of intervals in S which are completely contained in the query interval. In fact, by using the persistent search tree, the set of intervals in S which are completely contained in the query interval can be enumerated in $O(t+\log n)$ time in such a way that all the endpoints of the enumerated t intervals are sorted in nondecreasing order. We can construct the persistent search tree in $O(n \log n)$ time and $O(n)$ space.

Thus, by choosing I_k as a query interval, we can find $S(I_k)$ in COMP_WEIGHT in $O(|S(I_k)|+\log n)$ time. L_k and I_k can also be obtained in $O(|L_k|)$ time. Thus, the time complexity of COMP_W(L_k, I_k, $S(I_k)$) (i.e., computing $W(I)$'s for all I's in L_k) becomes $O(\log n+|L_k|+|S(I_k)|)$ since sorting in Line 1 can be done by merging the two sorted lists of endpoints of $S(I_k)$ and L_k. From the above discussion, we see that a maximum weight independent set of the overlap graph $G(S)$ can be found in $O\left(n \log n+\sum_{k=0}^{N}(\log n+|L_k|+|S(I_k)|)\right)=O(n \log n+m)$ time and $O(n)$ space, where $m=\sum_{k=0}^{N}|S(I_k)|$. Clearly, $m \leqq m_c$ and $m \leqq nN$.

It should be noted that the algorithm in this paper can be easiy modified even if we consider each interval (chord) is an open set, i. e., it contains none of its endpoints.

References

(1) Asano T., Asano T. and Imai H.: "Partitioning a polygonal region into trapezoids", J. of ACM, 33, pp. 290-312 (1986).

(2) Buckingham M.: "Circle Graphs", Ph. D. Dissertation, Courant Institute, Rept. NSO 21 (Oct. 1980).

(3) Chazelle B.: "Filtering search: A new approach to query-answering", SIAM J. Computing, 15, pp. 703-724 (1986).

(4) Gavril F.: "Algorithms for a maximum clique and a maximum independent set of a circle graph", Networks, 3, pp. 261-273 (1973).

(5) Imai H. and Asano T.: "Applications of the priority search tree to circle graph problems", IECEJ Technical Reports, **CAS86**-112 (1986).

(6) Liu R. and Ntafos S.: "On decomposing polygons into uniformly monotone parts", Inform. Process. Lett., 27, pp. 85-88 (1988).

(7) Read R. C., Rotem D. and Urrutia J.: "Orientations of circle graphs", J. Graph Theory, 6, pp. 325-341 (1982).

(8) Sarnak N. and Tarjan R. E.: "Planar point location using persistent search trees", C. of ACM, 29, pp. 669-679 (1986).