Information Processing Letters 41 (1992) 99–102
North-Holland

14 February 1992

Finding a maximum set of independent chords in a circle *

R.C. Chang

Institute of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan 30050, ROC

H.S. Lee

Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan 30050, ROC

Communicated by K. Ikeda
Received 26 october 1990
Revised 24 September 1991

Abstract

Chang, R.C. and H.S. Lee, Finding a maximum set of independent chords in a circle, Information Processing Letters 41 (1992) 99–102.

In this note we propose an $O(nm)$ algorithm for finding a maximum independent set of m chords which are incident to n vertices on a circle. This result can be applied to improving the time complexity of the algorithm for partitioning simple polygons into a minimum number of uniformly monotone polygons.

Keywords: Circle graph, combinatorial problems, computational geometry, maximum independent set, polygon decomposition

1. Introduction

Let C be a circle which has n vertices and m chords connecting these n vertices. Two chords are said to be independent if they do not intersect. The problem considered in this paper is to find a maximum set of independent chords (MSIC) in C. The problem is equivalent to finding an independent set with maximum cardinality in a circle graph. An independent set of a graph is a subset of its vertices where no two vertices are joined by an edge. A circle graph corresponding to the set of chords in a circle can be constructed as follows. If there are chords sharing the same end-point, we can expand the circle slightly so that no chords are sharing the same end-point. Every vertex in the circle graph corresponds to a chord in the circle and two vertices are connected if their corresponding chords intersect. There are m vertices in the circle graph constructed in this way.

Gavril [3] first proposed an $O(m^3)$ algorithm for finding a maximum independent set of a circle graph with m vertices. Buckingham [2], Read et al. [5] and Supowit [6] independently proposed various $O(m^2)$ algorithms for this problem. Applying these $O(m^2)$ algorithms to the case where chords are allowed to have common end-points would result in a complexity of $O(n^4)$ in the worst case. Liu and Ntafos [4] presented a dynamic programming approach to solve this problem in $O(n^3)$. However, the time complexity

* This research work was partially by the National Science Council of the Republic of China under grant No. NSC80-0408-E009-03.

of Liu and Ntafos' algorithm depends only on the number of vertices in a circle no matter how many chords there are. Later an $O(nm)$ time and $O(n)$ space algorithm was given by Asano et al. [1].

In this paper, we propose another $O(nm)$ algorithm to find an MSIC in a circle with n vertices and m chords. But our algorithm is conceptually simpler than that of Asano et al.

2. The algorithm

Let E denote the set of chords in C, the set of vertices in C be numbered from v_1 to v_n clockwise, and $e(i, j)$ denote the chord that connects v_i and v_j. Also define $V(i, j) = \{v_k \mid i \leqslant k \leqslant j\}$ and $E(i, j) = \{e(l, m) \mid i \leqslant l < m \leqslant j \text{ and } e(l, m) \in E\}$. $V(i, j)$ denotes the set of vertices between v_i and v_j, and $E(i, j)$ denotes the set of chords connecting only vertices in $V(i, j)$. Hence, $E(1, n) = E$. Let $C(i, j)$ denote an MSIC of $E(i, j)$ and let $M(i, j)$ be the cardinality of $C(i, j)$. It follows that $C(1, n)$ is an MSIC of E and $C(i, j) = \emptyset$ when $j \leqslant i$.

Our algorithm is also based on the principle of dynamic programming, which is stated in the following lemma.

Lemma 2.1.

$$C(i, j) = \max_{\substack{i \leqslant k \leqslant j-1,\\ e(k, j) \in E}} (C(i, j-1), C(i, k-1) \cup C(k+1, j-1) \cup \{e(k, j)\}),$$

where max(·) *is a function to take the set with maximum cardinality in the argument list.*

Proof. Since $C(i, j)$ is an MSIC between v_i and v_j, it follows that all chords of $C(i, j)$ do not intersect. $C(i, j)$ must contain either one or none of $e(k, j) \in E$ for $i \leqslant k \leqslant j-1$.

For the first case, assume that $C(i, j)$ contains the chord $e(k, j)$. Because the chords in $C(i, j)$ do not intersect, the chords in $C(i, j)$ except $e(k, j)$ can be classified into two parts: chords of the first part connect only vertices in $V(i, k-1)$ and chords of the second part connect only vertices in $V(k+1, j-1)$. Furthermore, these two parts are MSICs of $C(i, k-1)$ and $C(k+1, j-1)$ respectively. Hence, the MSIC of $V(i, j)$ can be computed as follows:

$$C(i, j) = C(i, k-1) \cup C(k+1, j-1) \cup \{e(k, j)\}.$$

For the latter case in which $C(i, j)$ contains none of $e(k, j) \in E$ for $i \leqslant k \leqslant j-1$, chords in $C(i, j)$ connect only vertices in $V(i, j-1)$. Vertex v_j can be discarded without affecting the MSIC of $V(i, j)$. Therefore,

$$C(i, j) = C(i, j-1).$$

From above, if $C(i, k-1) \cup C(k+1, j-1) \cup \{e(k, j)\}$ is the largest set for all possible k between i and $j-1$, we can conclude that $C(i, j)$ equals $C(i, j-1)$ or $C(i, k-1) \cup C(k+1, j-1) \cup \{e(k, j)\}$ depending on which one is larger. □

$C(1, n)$ an MSIC of E, can be computed according to the recurrence relation in Lemma 2.1 above. Based on Lemma 2.1, Algorithm MSIC works in a bottom-up manner. MSICs are constructed incrementally along the circle. Once MSICs for smaller intervals have been computed, MSICs for larger intervals can be computed according to Lemma 2.1. Intervals to be computed are expanded incrementally until the interval contains all the vertices on C. Algorithm MSIC is stated in the following:

Algorithm MSIC

```
For $i = 1$ to $n$ Do $C(i, i) = \emptyset$;
For $l = 1$ to $n - 1$ Do
    For $k = 1$ to $n - l$ Do
        Begin
            $j = i + l$;
            $C(i, j) = C(i, j - 1)$;
            For all $e(k, j) \in E$ where $i \leqslant k \leqslant j - 1$
                Do
                    if $M(i, k - 1) + M(k + 1, j - 1) + 1 >$
                        $M(i, j)$ then
                        $C(i, j) = C(i, k - 1) \cup C(k + 1, j -$
                        $1) \cup \{e(k, j)\}$
        End
End of MSIC.
```

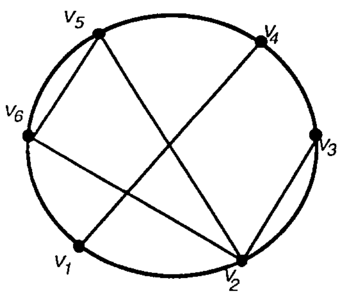
An adjacent matrix A can be used to represent E. Let $a(i, j)$ denote an entry of A; $a(i, j)$ is equal to 1 if $e(i, j) \in E$ and $a(i, j)$ is equal to 0 otherwise. Since only those $e(i, j)$'s that belong to e and satisfy that $i < j$ are considered in Algorithm MSIC, entries above the diagonal of A would be sufficient. That is, we can leave the entries on and below the diagonal of A undefined. In order to implement the inmost loop in the Algorithm MSIC efficiently, a slight modification to A, however, is needed. Associate each entry of A above the diagonal with a pointer. As shown in Fig. 1, entries with 1 in a column are chained together, and entries with 0 in a column point to the first entry with 1 following them in the same column. For each entry in A, if there is no entry with 1 following it in the same column, the pointer of this entry is set to ground, that is, the diagonal entry on this column. For example, both $a(2, 6)$ and $a(3, 6)$ point to $a(5, 6)$ and $a(3, 5)$ points to $a(5, 5)$, the ground. With such modification, the inmost loop can be executed efficiently because only entries with 1 are scanned. Note that A can be constructed in $O(n^2)$ time. The complexity of our algorithm is analyzed in the following theorem.

Theorem 2.2 *Algorithm MSIC runs in* $O(nm)$ *steps, where* $m = |E|$.

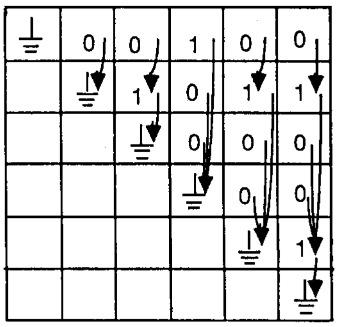
Proof. The complexity of Algorithm MSIC is equal to the number of entries with 1 in A scanned by Algorithm MSIC, and it is given by the following equation.

$$\begin{aligned}
&\sum_{l=1}^{n-1} \sum_{i=1}^{n-l} \sum_{k=i}^{i+l-1} a(k, i+l) \\
&\quad = \sum_{l=1}^{n-1} \sum_{j=1+l}^{n} \sum_{k=j-l}^{j-1} a(k, j) \\
&\quad = \sum_{j=2}^{n} \sum_{l=1}^{j-1} \sum_{k=j-1}^{j-1} a(k, j) \\
&\quad = \sum_{j=2}^{n} \sum_{k=1}^{j-1} k \cdot a(k, j) \\
&\quad \leqslant n \sum_{j=2}^{n} \sum_{k=1}^{j-1} a(k, j) \\
&\quad = nm. \qquad \square
\end{aligned}$$

Consider a restricted case in which $V(1, n)$ can be divided into two parts, $V(1, k)$ and $V(k+1, n)$ such that all chords of E connect one vertex in $V(1, k)$ and another vertex in $V(k+1, n)$. In other words, $E(1, k) = E(k+1, n) = \emptyset$. We call this a bipartite case. For the bipartite case, we have the following corollary.



(a)



(b)

Fig. 1. (a) A circle with six vertices and five chords. (b) The variant adjacent matrix corresponding to the chord set in (a).

Corollary 2.3. *For the bipartite case stated above, Algorithm MSIC can be improved to run in* $O(n^2)$.

Proof. Assume that the set of chords E is in a bipartite case. Consider the inmost loop of Algorithm MSIC. Let $K = \{k \mid e(k, j) \in E \text{ and } i \leqslant k \leqslant j-1\}$. Suppose $k_1, k_2 \in K$ and $k_1 < k_2$. We want to show that

$$E(i, k_1 - 1) = E(i, k_2 - 1) = \emptyset.$$

If $E(i, k_1 - 1) \neq \emptyset$, then there exists a chord $e \in E(i, k_1)$. However, there will be no such k' that all chords in $\{e(k, j) \mid k \in K\} \cup \{e\}$ connect one vertex in $V(1, k')$ and another vertex in $V(k'+1, n)$. In other words, this is not a bipartite case. Hence $E(i, k_1 - 1) = \emptyset$ and $E(i, k_2 - 1) = \emptyset$ similarly. We have $C(i, k_1 - 1) = C(i, k_2 - 1) = \emptyset$. Furthermore,

$$M(k_1 + 1, j - 1) \geqslant M(k_2 + 1, j - 1).$$

It follows that

$$\begin{aligned}
&\max\big(C(i, k_1 - 1) \cup C(k_1 + 1, j - 1) \\
&\qquad \cup \{e(k_1, j)\}, \\
&\quad C(i, k_2 - 1) \cup C(k_2 + 1, j - 1) \\
&\qquad \cup \{e(k_2, j)\}\big) \\
&\ = C(i, k_1 - 1) \cup C(k_1 + 1, j - 1) \\
&\quad \cup \{e(k_1, j)\};
\end{aligned}$$

therefore,

$$\begin{aligned}
C(i, j) = \max\big(&C(i, j-1), C(i, k-1) \\
&\cup C(k+1, j-1) \cup \{e(k, j)\}\big),
\end{aligned}$$

where k is the smallest integer in K. Since the inmost loop requires only one comparison, total time requires $O(n^2)$ steps. □

3. Conclusion

Finding a maximum set of independent chords on a circle is a crucial step in partitioning simple polygons into the minimum number of uniformly monotone polygons in [4]. In this note, we have presented an $O(nm)$ algorithm for this problem, which is faster than $O(n^3)$ when m is less than $O(n^2)$. When applying the MSIC-finding algorithm to partitioning simple polygons into the minimum number of uniformly monotone polygons, we may find that the vertices on the circle are in a bipartite case. Then we can apply Corollary 2.3 to this bipartite case to obtain an $O(n^2)$ algorithm. As for the case where the vertices on the circle are not in a bipartite case, it remains open whether there exists an algorithm which has a time complexity better than $O(nm)$ for the MSIC problem.

References

[1] T. Asano, H. Imai and A. Mukaiyama, A faster algorithm for finding a maximum weight independent set of a circle graph, *Inform. Process. Soc. Japan SIGAL Report AL5-18* (1989) 133–138.

[2] M. Buckingham, Circle graphs, Ph.D. Dissertation, Courant Institute, Rept. NSO #21, October 1980.

[3] F. Gavril, Algorithms for a maximum clique and a maximum independent set of a circle graph, *Networks* **3** (1973) 261–273.

[4] R. Liu and S. Ntafos, On decomposing polygons into uniformly monotone parts, *Inform. Process. Lett.* 27 (2) (1988) 85–89.

[5] R.C. Read. D. Rotem and Urrutia. Orientations of circle graphs, *J. Graph Theory* **6** (1982) 325–341.

[6] K.J. Supowit, Finding a maximum planar subsets of a set of nets in a channel, *IEEE Trans. Computer-Aided Design* **6** (1) (1989) 93–94.