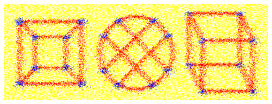
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 21, no. 4, pp. 547–554 (2017)
DOI: 10.7155/jgaa.00428

A Maximum Weight Clique Algorithm For Dense Circle Graphs With Many Shared Endpoints

Max Ward[1] *Andrew Gozzard*[1] *Amitava Datta*[1]

[1]School of Computer Science and Software Engineering
University of Western Australia
Perth, WA 6009, Australia

Abstract

Circle graphs are derived from the intersections of a set of chords. They have applications in VLSI design, bioinformatics, and chemistry. Some intractable problems on general graphs can be solved in polynomial time on circle graphs. As such, the study of circle graph algorithms has received attention. State-of-the-art algorithms for finding the maximum weight clique of a circle graph are very efficient when the graph is sparse. However, these algorithms require $\Theta(n^2)$ time when the graph is dense. We present an algorithm that is a factor of $\sqrt{n}$ faster for dense graphs in which many chord endpoints are shared. We also argue that these assumptions are practical.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
June 2016	October 2016	October 2016	February 2017	February 2017
		Published: April 2017		
	Article type: Concise paper		Communicated by: S. Kobourov	

E-mail addresses: max.ward-graham@research.uwa.edu.au (Max Ward) gozzaa01@student.uwa.edu.au (Andrew Gozzard) amitava.datta@uwa.edu.au (Amitava Datta)

1 Introduction

A circle graph is defined by the intersections of a set of chords on the interior of a circle (depicted in Figure 1). Each chord is represented by a vertex, and the intersections of chords define edges. We will call the number of vertices in a circle graph n. Circle graphs are often used to model problems in VLSI design; such as channel routing and switch-box routing [9]. In addition they have natural analogues in bioinformatics and chemistry. The possible base pairings of RNA nucleotides constitute a circle graph. Finding the maximum independent set of a circle graph is equivalent to finding a valid RNA secondary structure with maximum bonds. This is a classic computational problem in RNA bioinformatics solved by Nussinov *et al.* [7]. Similarly, finding the maximum clique of a circle graph corresponds to finding the most nested pseudoknot for an RNA molecule. Bonsma & Breuer [3] modeled a similar problem in chemistry using circle graphs; the enumeration of benzenoid hydrocarbons and fullerenes. In both of these applications, the endpoints of chords are shared by many other chords. Indeed, Bonsma & Breuer [3] used density and shared endpoints to define an algorithm that was faster than conventional circle graph algorithms for their problem, as did Nussinov *et al.* [7]. Therefore, we propose that considering algorithms for dense circle graphs with many shared endpoints is practical.

Algorithms for finding the maximum weight clique in a circle graph all assume that endpoints are not shared [1, 5, 8]. It is possible to use these algorithms even when endpoints are shared by permuting the endpoints [4]. For dense graphs (containing $\Theta(n^2)$ edges) the best known algorithms for this problem require $\Theta(n^2)$ time [1, 5, 8]. We present an algorithm that runs in $\Theta(k^3)$ time where k is the number of distinct endpoints used by chords. For dense circle graphs with many shared endpoints $n = \Theta(k^2)$, so our algorithm is the fastest known algorithm for finding the maximum weight clique for this circle graph type. Note that this bound is actually smaller than the size of the graph, as a dense circle graph might contain $\Theta(k^4)$ edges. Our findings echo the results of Bonsma & Breuer [3] and Nussinov *et al.* [7], who found fast algorithms for computing the maximum independent set of a dense circle graph with many shared endpoints.

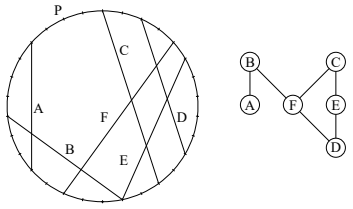

Figure 1: The relationship between a set of chords and a circle graph.

2 An Algorithm For Maximum Weight Clique

2.1 Interval Model & Definitions

To facilitate discussion of our algorithm, we will assume the interval model of a circle graph without loss of generality. This is a convenient representation in which the circle is cut at an arbitrary point. Thus we are left with a set of intervals on a line. An example can be found in Figure 2. An interval is analogous to a vertex, and two vertices share an undirected edge iff their corresponding intervals overlap, but neither is contained by the other. Let us define an interval as a tuple of its left and right endpoints respectively. If we have two intervals (i, j) and (i', j'), then there is an edge between their corresponding vertices iff $i < i' < j < j'$ or $i' < i < j' < j$. A clique is correspondingly a set of mutually overlapping intervals, such that no interval is contained within another. Note that when we say that an interval is left or right of another interval, or leftmost or rightmost in a clique, we do so with the assumption that intervals are ordered by ascending left endpoint, then by ascending right endpoint.

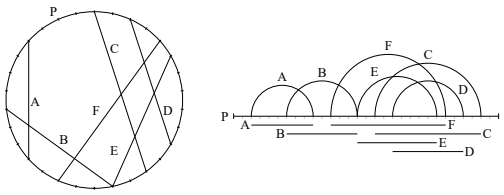

Figure 2: A circle graph and its interval model representation.

An important observation about our representation is that intervals having touching endpoints are defined not to overlap, and thus do not share an edge in their graph representation. In existing maximum clique algorithms on circle graphs, it has been assumed that endpoints are never shared, and so this part of the definition is unspecified. Clearly, it would make just as much sense to define intervals that touch at their endpoints as overlapping. Indeed, this is the definition used by Nussinov *et al.* [7], and Bonsma & Breuer [3]. We deviate from existing definitions on this point, so we shall explain the reason. For the known applications of algorithms that find a maximum (possibly weighted) clique on a circle graph, it is more useful to have each endpoint used by at most one interval in the output. To illustrate, let us consider RNA secondary structure; endpoints represent nucleotides, chords represent potential bonds, and a clique is a pseudoknotted structure. Now, observe that, in a valid RNA secondary structure, a nucleotide can only be involved in a single bond. Thus, any valid pseudoknotted structure must also have each nucleotide involved in at most one bond. Therefore, our representation is applicable for this use, which is for the same domain tackled by Nussinov *et al.* While we cannot see how

a clique algorithm could be applied usefully to the work of Bonsma & Breuer, the application to VLSI is clear. One is often concerned with wire crossings for VLSI, and for these applications it is useful to consider only intervals that strictly overlap [6, 2].

Using our representation, let us define some useful terminology. As we are dealing with the weighted clique problem, every vertex must have a weight. Let us define $W(i,j)$ to be the weight of the interval (i,j). Furthermore, we will define endpoints to be in the range $[0,k)$ for k endpoints. This can be done without loss of generality, even for real numbered endpoints, at the cost of sorting and relabeling. In addition, we assume that there are no duplicate intervals as we can always take the maximum weight duplicated interval. Also, we say that an interval covering no points (i,i) is not a valid interval. In addition, if the interval (i,j) does not exist, then $W(i,j) = -\infty$. Finally, note that the weight of a clique is the sum of the weights of its intervals.

2.2 Maximum Weight Clique Algorithm

We shall now define an algorithm capable of finding the maximum weight clique of a circle graph.

$$F(l,r,r') = \begin{cases} \max \begin{cases} F(l+1,r,r') \\ F(l,r,r'+1) \\ W(l,r') + F(l+1,r,r'+1) \end{cases} & \text{if } l < r \leq r' < k \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The function $F(l,r,r')$ returns the maximum weight of a clique in which left endpoints are in the range $[l,r)$ and right endpoints are in the range $[r',k)$. Note that we define $F(l,r,r') = 0$ when $l < r \leq r' < k$ is not satisfied. Observe that $l \geq r$ implies that the range of possible left endpoints comprises no endpoints. Similarly, if $r' \geq k$, the algorithm would consider having right endpoints that are not valid endpoints. Finally, if $r > r'$ then intervals that end before others start would be permitted.

The algorithm works by considering three cases. A pithy explanation of these cases is that they find the leftmost interval in an optimal clique that is not currently included, add it, and then recursively construct the rest of the optimal clique.

In the first case $F(l+1,r,r')$, we consider that the next best interval might not have its left endpoint at l, but instead in the range $[l+1,r)$. In the second case $F(l,r,r'+1)$, we consider that the next best interval might not have its right endpoint at r'.

In the third and final case $W(l,r') + F(l+1,r,r'+1)$, we consider that the next best interval is (l,r'). If we are taking (l,r') to be in a clique, the left endpoints of any additional intervals must be in the range $[l+1,r)$, and right endpoints in the range $[r'+1,k)$. If they were not, they would not intersect (l,r').

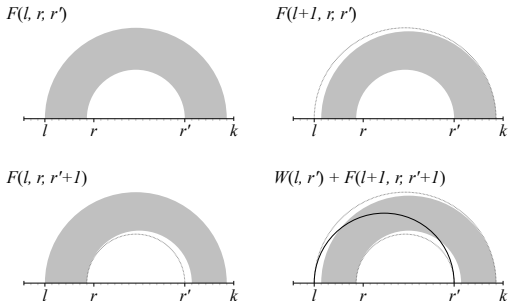

Figure 3: A diagrammatic representation of our algorithm. The diagrams correspond to the terms in the dynamic programming recurrence defined in Equation 1. The grey half-annulus represents an optimal clique such that all left endpoints are in the left range, and all right endpoints are in the right range. The initial extent of the half-annulus is represented by grey arcs; this is to help illustrate a change in half-annulus ranges. Arcs newly added to the clique are denoted by a darker arc.

The only remaining question is how to extract the solution from our dynamic programming recurrence. It is enough to consider every possible leftmost interval in a clique, and call F correspondingly. Therefore the solution is encoded by Equation 2 when S is the set of all intervals in the input circle graph.

$$\max\{F(l+1, r, r+1) + W(l, r) : (l, r) \in S\} \tag{2}$$

2.3 Proof of Correctness

Having defined our algorithm we now provide a formal proof of its correctness. For $l < r \leq r' < k$, call a clique an (l, r, r')-*clique* if its left endpoints are in the range $[l, r)$ and its right endpoints are in the range $[r', k)$. This mirrors the construction for our algorithm (see Equation 1). Now, observe that

$$F(l, r, r') \geq F(l+1, r, r')$$
$$F(l, r, r') \geq F(l, r, r'+1)$$

This is because all $(l{+}1, r, r')$-*cliques* and $(l, r, r'{+}1)$-*cliques* are also (l, r, r')-*cliques*. Next,

$$F(l, r, r') \geq W(l, r') + F(l+1, r, r'+1)$$

This follows since any $(l+1, r, r'+1)$-*clique* is still a clique after adding the interval (l, r').

This proves that $F(l, r, r')$ is at least as large as the cases it considers. To prove that it is also no larger, and therefore equal, consider a maximum weight (l, r, r')-*clique* C. In addition, we consider three cases based on the leftmost interval (i, j) in C which correspond to the cases in the algorithm. If $i = l$ and $j = r'$ then C consists of this interval (l, r') and an $(l + 1, r, r' + 1)$-*clique*, since all other chords intersect (i, j). Thus,

$$F(l, r, r') \leq W(l, r') + F(l + 1, r, r' + 1)$$

Next, if $i > l$, then C is also an $(l + 1, r, r')$-*clique*, so, in this case,

$$F(l, r, r') \leq F(l + 1, r, r')$$

Finally, if $i = l$ and $j > r$' then C is also an $(l, r, r' + 1)$-*clique*, so,

$$F(l, r, r') \leq F(l, r, r' + 1)$$

This covers all cases, so,

$$F(l, r, r') = \max\{F(l + 1, r, r'), F(l, r, r' + 1), W(l, r') + F(l + 1, r, r' + 1)\}$$

The correctness of the algorithm is proved.

2.4 Complexity Analysis

Filling the tables for our dynamic programming recurrence takes $\Theta(k^3)$ time. There is a single table for F with k^3 cells. Each cell requires $O(1)$ time to be computed as per Equation 1. Extracting the solution from the table takes $\Theta(n)$ time as implied by Equation 2. Thus it follows that the requirements for our dynamic programming algorithm are $\Theta(k^3)$ time and $\Theta(k^3)$ space.

2.5 Practical Implementation

The space requirement is unacceptably large so we will provide a way to improve it. Observe that any call to the function F will never refer to more than one value for r. Therefore we can use an $\Theta(k^2)$ sized table, and recompute it for each r. While doing this, we can keep track of the best solution we have seen. This allows us to find the correct maximum. This approach requires only $\Theta(k^2)$ space, but the time requirement remains $\Theta(k^3)$. Note that we also require $\Theta(k^2)$ space to store a table for $W(i, j)$, so this space complexity cannot be improved upon.

3 Final Remarks

We have presented a fast algorithm for dense circle graphs with many shared endpoints. Existing algorithms need to examine all the edges in the graph, and

thus required $\Theta(n^2)$ time for such graphs. Our algorithm is able to exploit this graph type, and scales asymptotically slower than the number of edges encoded in the input. However, previous algorithms are remarkably fast for sparse graphs where $n = O(k)$. Indeed they are much faster than our algorithm. When the graph is sparse, the best lower bound is $\Omega(n \log n)$ [1]. However, a fast algorithm for sparse and dense graphs might be devised by building upon our algorithm, and this is certainly an interesting avenue for future investigations. For the unweighted case, Tiskin recently presented a brilliant algorithm which is able to find the maximum clique in $O(n \log^2 n)$ time [10]. It is possible that a faster algorithm for dense circle graphs with many shared endpoints exists. Indeed, we conjecture that an $O(k^2 \log k)$ algorithm might exist for the unweighted case of the maximum clique problem on circle graphs.

Acknowledgments

The authors wish to thank Eliot Courtney, Zachary Forman, and Connie Pyromallis for their valuable advice and proof reading acumen. Further, we would like the thank an anonymous reviewer who suggested a vastly improved proof of correctness formulation.

References

[1] A. Apostolico, M. J. Atallah, and S. E. Hambrusch. New clique and independent set algorithms for circle graphs. *Discrete Applied Mathematics*, 36(1):1–24, 1992. doi:10.1016/0166-218X(92)90200-T.

[2] S. N. Bhatt and F. T. Leighton. A framework for solving vlsi graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, 1984. doi:10.1016/0022-0000(84)90071-0.

[3] P. Bonsma and F. Breuer. Counting hexagonal patches and independent sets in circle graphs. *Algorithmica*, 63(3):645–671, 2012. doi:10.1007/s00453-011-9561-y.

[4] F. Gavril. Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks*, 3(3):261–273, 1973. doi:10.1002/net.3230030305.

[5] W.-L. Hsu. Maximum weight clique algorithms for circular-arc graphs and circle graphs. *SIAM Journal on Computing*, 14(1):224–231, 1985. doi:10.1137/0214018.

[6] F. T. Leighton. Layouts for the shuffle-exchange graph and lower bound techniques for vlsi. Technical report, DTIC Document, 1982.

[7] R. Nussinov, G. Pieczenik, J. R. Griggs, and D. J. Kleitman. Algorithms for loop matchings. *SIAM Journal on Applied mathematics*, 35(1):68–82, 1978. doi:10.1137/0135006.

[8] D. Rotem and J. Urrutia. Finding maximum cliques in circle graphs. *Networks*, 11(3):269–278, 1981. doi:10.1002/net.3230110305.

[9] N. A. Sherwani. *Algorithms for VLSI physical design automation*. Springer Science & Business Media, 2012.

[10] A. Tiskin. Fast distance multiplication of unit-monge matrices. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1287–1296. Society for Industrial and Applied Mathematics, 2010. doi:10.1007/s00453-013-9830-z.