

A Simpler Linear-Time Recognition of Circular-Arc Graphs

Haim Kaplan and Yahav Nussbaum

School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.
{haimk,nuss}@post.tau.ac.il

Abstract. We give a linear time recognition algorithm for circular-arc graphs. Our algorithm is much simpler than the linear time recognition algorithm of McConnell [10] (which is the only linear time recognition algorithm previously known). Our algorithm is a new and careful implementation of the algorithm of Eschen and Spinrad [4, 5]. We also tighten the analysis of Eschen and Spinrad.

1 Introduction

A *Circular-arc graph* is an intersection graph of arcs on the circle. That is, every vertex is represented by an arc, such that two vertices are adjacent if and only if the corresponding arcs intersect. Many subclasses of circular-arc graphs have also been studied such as *proper circular-arc graphs* [3], and *unit circular-arc graphs* [8]. An extensive overview of circular-arc graphs can be found at the book by Spinrad [13]. Recent applications of circular-arc graphs are in modeling ring networks [15] and item graphs of combinatorial auctions [2].

The first polynomial time algorithm for circular-arc recognition was given by Tucker [16]. This algorithm splits into one of two cases according to whether \bar{G} is bipartite (G is co-bipartite). In case \bar{G} is not bipartite the algorithm finds an odd length induced cycle in \bar{G} , and further splits into one of two subcases according to whether the cycle it found is of length 3 or of length at least 5. Using Tucker's terminology we refer to the first case where G is co-bipartite as Case I. We refer to the subcase where we found in \bar{G} a cycle of length 3, and therefore we found in G an independent set of size 3, as Subcase IIa. We refer to the case where we found in \bar{G} an induced cycle of length at least 5 as Subcase IIb.

Tucker showed how to implement his algorithm in $O(n^3)$ time. One of the bottlenecks in Tucker's implementation is a preprocessing phase where we identify containment relations between the neighborhoods of the vertices. Specifically, for every pair of vertices v and w we determine whether the neighborhood of v is contained in the neighborhood of w or vice versa. Furthermore, Tucker runs his algorithm recursively on particular graphs and this recursive structure also leads to cubic running time.

Spinrad [12] simplified Case I in Tucker's algorithm – the case where G is co-bipartite. Spinrad reduced this case to the problem of recognizing two dimensional posets [14]. We construct the poset using particular relations between the

vertices of G . Two vertices are related in the poset if their corresponding arcs are either disjoint, one is contained in the other, or together they cover the circle. The relations between the arcs are determined from the relations between the neighborhoods of the vertices. In case G is a circular-arc graph then from any two total orders that represent the poset we can construct a representation for G . Spinrad showed that this algorithm runs in $O(n^3)$ time.

Eschen and Spinrad [4, 5] gave an $O(n^2)$ algorithm for recognizing circular-arc graphs by addressing the two bottlenecks in Tucker's implementation. Eschen and Spinrad show how to compute neighborhood containment relations in $O(n^2)$ time. Specifically, they construct four graphs such that if G is indeed circular-arc graph then each of the four graphs is either an interval graph or a chordal bipartite graph. These graphs are constructed such that the neighborhood of v contains the neighborhood of w in G , if and only if the neighborhood of v contains the neighborhood of w in each of these graphs. The quadratic time bound follows since one can compute neighborhood containment relations in interval graphs and chordal bipartite graphs in quadratic time [9, 4, 5].

Eschen and Spinrad also showed that in Case I of the algorithm, when G is co-bipartite, we can use the same reduction to determine all pair of arcs that can cover circle in a model of G in $O(n^2)$ time. Since this was the only bottleneck in Spinrad's algorithm for this case, we obtain an $O(n^2)$ implementation of Case I. To implement Subcases IIa and IIb in $O(n^2)$ time, they changed the recursive structure of Tucker's algorithm. They show how to implement the algorithm such that each recursive call is on a co-bipartite graph (Case I) and therefore does not trigger further recursion. Since the sum of the sizes of the graphs in all recursive calls is proportional to the size of G , the quadratic bound follows.

Recently, McConnell [10] presented the first recognition algorithm for circular-arc graphs that runs in linear time. The algorithm reduced the problem to an interval graph recognition problem where specific intersection types between the intervals are specified. McConnell's algorithm uses the same preprocessing stage of Eschen and Spinrad where it computes neighborhood containment relations. To establish the linear time bound, McConnell tightens the analysis of Eschen and Spinrad's preprocessing stage. He shows that this preprocessing stage can be implemented in linear time since we are interested only in neighborhood containment relations between adjacent vertices, and the associated chordal bipartite graphs cannot be too large.

McConnell's algorithm is quite involved. Its most complicated computation is to find a partition of a graph into a particular kind of modules called Δ modules. Those Δ modules are used to turn the input circular-arc graph into an interval graph with specific types of intersections between the intervals, and to find a representation for this interval graph. McConnell first presents an implementation that runs in $O(m + n \log n)$ time. To get the linear time bound a more complicated partitioning procedure has to be adapted from the linear time transitive orientation algorithm [11] which is by itself quite involved. This algorithm also uses probe interval graphs to find pairs of arcs that can cover the circle in linear time.

Hsu [6] presented a different recognition algorithm for circular-arc graphs that runs in $O(mn)$ time and reduces the problem to recognition of circle graphs.

1.1 Our contribution

We give a careful implementation of the recognition algorithm of Eschen and Spinrad that in fact runs in linear time. Our implementation first either finds an independent set of size 3, and then we can apply Subcase IIa of the algorithm, or it concludes that the graph has $\Theta(n^2)$ edges. In the latter case the implementation of Eschen and Spinrad in fact runs in linear time.

Eschen and Spinrad find in Subcase IIa a particular maximal independent set and place the corresponding arcs on the circle. We show how to find the independent set and place its arcs on the circle in linear time. Our implementation then continues as the implementation of Eschen and Spinrad, but we tighten their analysis to show that the running time is linear. Our main new insight is that each subgraph considered by the algorithm while placing and ordering the arcs on the circle is dense. That is, the number of edges that each subgraph contains is quadratic in the size of its vertex set. Furthermore the total size of these subgraphs is linear in the size of the input graph.

Our algorithm also performs a preprocessing phase where neighborhood containment relations are computed. As proved by McConnell [10] this can be done in linear time. As all previous algorithms, we also require a postprocessing verification step where we check that the representation we obtain is indeed a representation of G . McConnell [10] gave a straightforward linear time implementation of this postprocessing step, which traverse the circular-arc model and extract all the pairs of intersecting arcs from it. The model corresponds to the graph G , only if the intersections of arcs fit the adjacencies of vertices.

We describe a linear time implementation of Subcase IIa. Subcase IIb can also be implemented in linear time in a similar way. We do not describe it here since we apply Subcase IIb only when we are sure that G has $\Theta(n^2)$ edges. Our implementation is much simpler than McConnell's algorithm.

2 Preliminaries

We consider a finite simple graph $G = (V, E)$, Where $|V| = n$ and $|E| = m$. We represent graphs using adjacency-lists. For a vertex v in a graph, the (*closed*) *neighborhood* of v , denoted by $N[v]$ is the set of all vertices adjacent to v together with v itself. For a set of vertices U we define $N_U[v]$ to be $N[v] \cap U$.

A *circular-arc model* of a graph G is a set of arcs on the circle. Each vertex has an arc associated with it, such that two vertices are adjacent if and only if the corresponding arcs intersect. A graph G is a *circular-arc graph* if it has a circular-arc model. Note that a circular-arc graph may have more than one model.

We represent a single arc in a circular-arc model by its clockwise and counter-clockwise endpoints. We assume that no arc covers the entire circle. We represent

a circular-arc model by an ordered cyclic list of the endpoints of its arcs. To simplify we shall refer to the clockwise direction as *right* and to the counterclockwise direction as *left*, as we view them if we stand at the center of the circle.

There are four possible types of intersections between two arcs x and y [16, 6]. Arcs x and y *cross* if each contains a single endpoint of the other. Arcs x and y *cover the circle* if each contains both endpoints of the other. Arc x may be *contained* in arc y . And arc x may *contain* arc y . If x and y either cross or cover the circle, we say that x and y *overlap*.

For convenience, we refer to the vertices of G as arcs even before we decide if G is a circular-arc graph and find a model for it. We would say that two adjacent vertices intersect even before we have a model, because the arcs of adjacent vertices must intersect in every model. Hsu [6] showed that if G is a circular-arc graph then it has a model M such that for every pair of vertices v and u , the arc representing v in M contains the arc representing u in M if and only if $N[u] \subseteq N[v]$. So we would say that v contains u when $N[u] \subseteq N[v]$, even before we have found a model. This relation between u and v is the *neighborhood containment relation*. Additionally we would say that two vertices overlap when they intersect but do not contain each other.

A graph that can be partitioned into two independent sets is called *bipartite*. If G is not bipartite then it must have an odd-length induced cycle. If \bar{G} , the complement of G , is bipartite then G is *co-bipartite*, and is covered by two cliques.

A $(0,1)$ -matrix is said to have the *circular-ones* property if its columns can be ordered such that the 1's in each row are circularly consecutive. Circular-ones arrangement can be found in $O(m+n+r)$ time [1, 7] where m is the number of columns, n is the number of rows, and r is the number of 1's.

In the rest of the paper we show a linear time implementation of the algorithm. Out of the three cases, we present only Subcase IIa, since the other cases are applied only when $m = \Theta(n^2)$.

3 Preprocessing

An arc which represent a universal vertex can be placed on the circle in $O(1)$ time by placing its right endpoint anywhere on the circle and its left endpoint immediately to the right side of it. It is easy to find all universal arcs of G in linear time. Thus, we may assume that G does not have any universal vertices.

Let x be an arc that have the same neighborhood as another arc y that was already placed on the circle. The arc x can be placed on the circle by placing its endpoints next to the endpoints of y , in $O(1)$ time. McConnell [10] showed how to find vertices with the same neighborhood in linear time using a simple process called *radix partitioning*, which is similar to radix sort. Thus, we may assume that there are no two vertices in G that have the same neighborhood.

Before running our algorithm we preprocess the graph and for every pair of adjacent vertices v and u we check whether v contains u or u contains v , that is whether $N[v] \subseteq N[u]$ or $N[u] \subseteq N[v]$. Recall that Eschen and Spinrad [4, 5] showed how to compute neighborhood containment relations in $O(n^2)$ time, and

McConnell [10] tighten the analysis to show that this can be done in linear time. For more details of this part, which are not complicated, see [10].

4 Splitting into Cases

Recall that the algorithms of Tucker [16] and Eschen and Spinrad [4, 5] split into one of three cases. *Case I*, where \bar{G} is bipartite, that is G is co-bipartite. *Case II*, where \bar{G} has an odd-length induced cycle. Case II splits further into two non-exclusive subcases. *Subcase IIa* where \bar{G} has a triangle, that is G has three independent vertices. And *Subcase IIb* where \bar{G} has an induced of odd length 5 or more. Our algorithm, in fact, splits into one of these three cases as well. But we decide on the case to apply more carefully.

Let a_1 be a vertex of minimum degree in G . If $|N[a_1]| > \frac{n}{2}$ then every vertex of G has an edge to at least $\frac{n}{2}$ other vertices, so $m = \Theta(n^2)$. Otherwise, let Y be the set of arcs nonadjacent to a_1 . We look for a pair of nonadjacent vertices in Y . For every vertex $y \in Y$ we traverse its adjacency list, and construct its restriction to Y . The time to traverse all the adjacency lists is $O(n + m)$. If for every $y \in Y$ we found that $N_Y[y] = Y$ then $m = \Theta(n^2)$ since we know that $|Y| \geq \frac{n}{2}$. Otherwise, we find nonadjacent pair of arcs $a_2, a_3 \in Y$.

So either we concluded that $m = \Theta(n^2)$, and thus the $O(n^2)$ time bound of Eschen and Spinrad [5] is linear. Or, we found three independent vertices a_1, a_2, a_3 , and we can apply Subcase IIa. In the rest of the paper we describe a linear time implementation of Subcase IIa.

The algorithm for Subcase IIa consists of the three stages of Tucker's algorithm. In Stage 1, we find a set of arcs that can be ordered easily around the circle and divide it into sections, such that no arc has its both endpoints in the same section. In Stage 2, we place every endpoint of every other arc in its section. And in Stage 3, we order the endpoints within each section. We describe each of these stages in the following three sections.

5 Stage 1: Dividing the circle into sections

The algorithm begins by finding an independent set of arcs, I , that can be easily embedded around the circle, in an order consistent with some model of G . This set of arcs divides the circle into sections, such that no arc has its two endpoints in the same section.

5.1 Finding a maximal independent set

The algorithm of Tucker uses maximal independent set of arcs I of size at least 3 that obeys two requirements. First, no arc of I contains any other arc of G . Second, there is no arc $x \in I$ that has two nonadjacent arcs $y, z \notin I$ such that y and z overlap x and do not overlap any other arc in I . We begin by constructing a maximal independent set I' greedily, which satisfies the first requirement, and then change it to an independent set I that satisfies the second requirement as well.

Before constructing I' , we eliminate any arc that contains another arc from G , since those arcs cannot be in I . Let G' be the subgraph of G without these arcs. Every intersecting pair of arcs in G' overlaps, since no arc of G' contains another. In order to construct I' we maintain a set J consisting of every arc in G' that is nonadjacent to any arc already in I' . For every arc in G' we maintain a counter of the number of arcs in I' that intersect it.

Let $\{a_1, a_2, a_3\}$ be the independent set that we found in Sect. 4. We initialize I' to consist of arcs $\{a'_1, a'_2, a'_3\}$ where a'_i is an arc from G' and may be either a_i or a minimal arc contained in a_i . The set I' is an independent set in G' , since $\{a_1, a_2, a_3\}$ is an independent set in G . For every $a'_i \in I'$, we remove $N[a'_i]$ from J and increase the counters of the members of $N[a'_i]$.

As long as J is not empty, we pick an arbitrary arc $x \in J$ and add x to I' . We increase the counter of every arc y that overlaps x , and set $J = J \setminus \{x\}$. When J is empty, I' is a maximal independent set.

Next we construct I from I' . For every arc $x \in I'$ such that there are two nonadjacent arcs y_1 and y_2 in G' which overlaps only x in I' , we add y_1 and y_2 to I . If such y_1 and y_2 do not exist, we add x to I . To do so in linear time, we find all arcs in G' that overlaps only x by scanning $N[x]$, and identifying all neighbors of x whose counter equals to one. Let $Y \subset N[x]$ consist of these neighbors. For every $y \in Y$ we scan $N[y]$ and construct $N_Y[y]$, if $N_Y[y] \neq Y$ then we find $y' \in Y \setminus N_Y[y]$ which is nonadjacent to y .

The following lemma proves that I satisfies the requirements stated above.

Lemma 1. *If G is a circular-arc graph then I is a maximal independent set in G and we cannot get a larger independent set by replacing an arc $y_1 \in I$ with two nonadjacent arcs $z_1, z_2 \notin I$ that intersect y_1 .*

Proof. First note that I' is a maximal independent set in G , since it is a maximal independent set in G' , and every arc in G which is not in G' contains an arc in G' .

Assume that I is not a maximal independent set in G , then there is an arc $z \notin I$ which is nonadjacent to every arc of I . We may assume that z is in G' . The arc z cannot be in I' because otherwise z or an adjacent arc would be inserted to I . Then, since I' is maximal independent set, z must overlap some $x \in I'$, such that $x \notin I$ and x was replaced by y_1, y_2 in I . It follows that $\{y_1, y_2, z\}$ is an independent set of three arcs that overlap x , but this is impossible since each of them should cover an endpoint of x , and x has only two endpoints.

Assume that $y_1 \in I$ can be replaced by two nonadjacent arcs $z_1, z_2 \notin I$ that overlap only y_1 in I . The arc y_1 cannot be a member of I' , since otherwise we would have added z_1, z_2 to I instead of y_1 . Therefore there are $x \in I'$ and $y_2 \in I$ such that y_1 and y_2 are nonadjacent and overlap x . Arcs z_1 and z_2 do not overlap any $x' \neq x$, $x' \in I'$, because if they do, they must overlap some arc different from y_1 in I . Since I' is maximal, the two arcs z_1 and z_2 must be adjacent to x . Again, we got independent set of three arcs $\{y_2, z_1, z_2\}$, that should overlap x , but x has only two endpoints. \square

Note that if G is not a circular-arc graph, I might not satisfy the requirements stated in Lemma 1. In this case our algorithm continues and will detect that G is not a circular-arc graph later on.

In section 5.2 we show how to place the arcs of I on the circle. We label the arcs of I by $a_1, \dots, a_{|I|}$ according to their cyclic order around the circle, where a_1 is some arbitrary arc in I . The endpoints of the arcs split the circle into sections. Each section is either an arc of I or a gap between two consecutive arcs of I . Let S be a section. The two endpoints of the arcs of I that define S are called *the endpoints of S* . We assume that a section contains its left endpoint, but does not contain its right endpoint. We denote by S_{2i} the section of arc a_i . We denote by S_{2i+1} the section which is the gap between S_{2i} and $S_{2(i+1)}$. Subscripts of arcs are modulo $|I|$ and subscripts of sections are modulo $2|I|$.

Let I^c be the set of arcs of G not in I . For every $x \in I^c$, the arc x cannot be contained in an arc of I and is not universal. And since I is a maximal independent set, the following lemma holds.

Lemma 2. [16] *Let $x \in I^c$. In a model of G consistent with the placement of I , the endpoints of x are in different sections.*

5.2 Placing the independent set around the circle

We now place the arcs of I around the circle. Tucker showed how to order the arcs in I around the circle, using the adjacencies between arcs in I and arcs in I^c , such that there exist a circular-arc model of G consistent with this order.

Lemma 3. [16] *If G is a circular-arc graph then there exists a model of G consistent with every cyclic order of I that satisfies the following two requirements: (1) For each arc $x \in I^c$, the neighborhood of x in I , $N_I[x]$, is consecutive around the circle, with the arcs that are contained in x in the middle and the arcs that overlap x in the ends. (2) For each pair of adjacent arcs $x, y \in I^c$, the union of their neighborhoods in I , $N_I[x] \cup N_I[y]$ is consecutive around the circle.*

Let $x \in I^c$. We define $D(x)$ to be the set consisting of every arc $y \in N_{I^c}[x]$ such that $N_I[x] \cap N_I[y] = \emptyset$. Let $D^m(x)$ be the subset of $D(x)$ consisting of every arc $y \in D(x)$ such that there is no $y' \in D(x)$ for which $N_I[y'] \subset N_I[y]$ (see Fig. 1). Eschen and Spinrad [5] proved the following.

Lemma 4. [5] *Assume that G is a circular-arc graph. Let P be an order of I that satisfies the second requirement of Lemma 3 with respect to every pair of arcs $x, y \in I^c$ such that $y \in D^m(x)$ and $x \in D^m(y)$. Then, P satisfies the second requirement of Lemma 3 with respect to every pair of adjacent arcs in I^c .*

We construct a matrix M such that from a circular-ones arrangement of M we can define the order of I . Every arc of I corresponds to a column of M and every requirement of Lemma 3 has a row. We arrange the matrix such that the ones in every row are cyclically consecutive. The order of the columns will give us an order of I that is consistent with the requirements of Lemma 3. Any arc $x \in I^c$ with $N_I[x] = I$ cannot affect the order of I according to the requirements of Lemma 3, so we ignore those arcs.

For each arc $x \in I^c$ we create a row that have 1's in the columns of the arcs in $N_I[x]$. This row forces the consecutiveness of $N_I[x]$. If G is a circular-arc graph

then there are at most two arcs in I that x overlaps. For each such arc, z , we create a row that have 1's only in the columns of $N_I[x] \setminus \{z\}$. These rows will force $N_I[x]$ to be ordered so that the arcs that x contains are in the middle and the arcs that x overlaps are in the ends. If for some x there are more than two arcs in I that it overlaps then we halt since G is not a circular-arc graph. We created at most three rows for each arc, and a total of at most $3n$ rows with $3m$ ones.

In order to find $D(x)$ and neighborhood containment relation with respect to I , we decide for each pair of arcs $x, y \in I^c$ whether $N_I[x] \cap N_I[y] = \emptyset$ or $N_I[x] \subseteq N_I[y]$. To do so, we find a circular-ones arrangement [1, 7] of M . This arrangement gives us a preliminary cyclic order of the arcs of I . If such an arrangement does not exist then G is not a circular-arc graph, and we halt. For each pair of adjacent arcs in I^c we can detect if their neighborhoods in I do not intersect or one contains the other by looking at the first and last neighbors of both arcs in the cyclic order of I . We find the last neighbor of all arcs of I^c in the cyclic order by scanning the arcs of I starting from an arbitrary arc in the cyclic order. An arc $z \in I$ is the last neighbor of $x \in I^c$ if it is a neighbor of x , but the arc z' following z in the cyclic order is nonadjacent to x . We find the first neighbor in I of each $x \in I^c$ symmetrically.

Let $x \in I^c$ and consider the neighborhood containment relation restricted to I of the arcs in $D(x)$. In any circular-arc model of G , every $y \in D(x)$ covers one endpoint of x and stretches away from x . So $N_I[y]$ consists of a member of I next to x in the model, followed by zero or more members of I consecutive to it, in the direction which y stretches. Therefore, the arcs of $D(x)$ form at most two chains with respect to the neighborhood containment relation restricted to I , each consisting of arcs that cover the same endpoint of x . So, there are at most two distinct neighborhoods in I for arcs in $D^m(x)$. For example, in the illustration of Fig. 1, $N_I[b]$ and $N_I[c]$ are the two distinct neighborhoods in I for arcs in $D^m(x)$.

For each arc of x , we go through $D(x)$ to find $D^m(x)$. We find $D^m(x)$ partitioned into two sets, each consisting of arcs with the same neighborhood in I . We consider the elements in $D(x)$ one by one, in an arbitrary order. While scanning $D(x)$ we maintain at most two sets of minimal elements with respect to the neighborhood containment relation restricted to I . We denote these sets by $M_1(x)$ and $M_2(x)$. If for the next arc $y \in D(x)$, we have that $N_I[y] = N_I[m_i]$ for $m_i \in M_i(x)$ we add y to $M_i(x)$. If $N_I[y] \subset N_I[m_i]$ for $m_i \in M_i(x)$, we replace $M_i(x)$ by $\{y\}$. If $N_I[m_i] \subset N_I[y]$ for $m_i \in M_i(x)$, we skip y . Otherwise, the relation does not form two chains and thus G is not a circular-arc graph and we halt. When we finish scanning $D(x)$, we have identified $D^m(x)$ partitioned into two sets $M_1(x)$ and $M_2(x)$, each set consist of all elements with the same neighborhood in I .

According to Lemma 4, for every pair of arcs $x, y \in I^c$ such that $x \in D^m(y)$ and $y \in D^m(x)$, we should add a row to M with 1's in the columns of $N_I[x] \cup N_I[y]$. Although there could be $\Omega(n^2)$ pairs $x, y \in I^c$ such that $x \in D^m(y)$ and $y \in D^m(x)$, the number of distinct sets $N_I[x] \cup N_I[y]$ is at most n . This is because for every arc $x \in I^c$, the members of $D^m(x)$ have at most two distinct neighborhoods in I . We identify these distinct rows to add to M as follows.

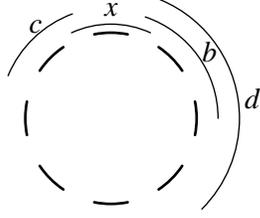


Fig. 1. $D(x)$ and $D^m(x)$. Arcs of I are drawn in boldface. $b, c, d \in D(x)$. Also, $b, c \in D^m(x)$ but $d \notin D^m(x)$, since $N_I[b] \subset N_I[d]$.

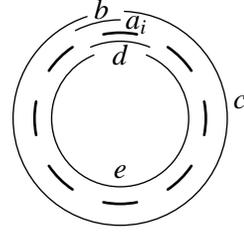


Fig. 2. U_i, W_i, A_i^e and A_i^c . Arcs of I are drawn in boldface. $b \in U_i, c \in W_i, d \in A_i^e$ and $e \in A_i^c$

For every $x \in I^c$, we traverse every set $M_i(x)$ which is not empty. For each $y \in M_i(x)$ we check if $x \in D^m(y)$. If indeed $x \in D^m(y)$, we add a row to M with 1's in the columns of $N_I[x] \cup N_I[y]$. In this case we also set $M_i(x)$ to be empty and stop the traversal, since all other arcs in $M_i(x)$ have the same neighborhood in I as y . To check if $x \in D^m(y)$ in constant time, we pick an arbitrary arc z_i from each $M_i(y)$ that is not empty, and check if $N_I[x] = N_I[z_i]$.

Since we use the neighborhood of each arc to define at most two rows, we add to M at most n rows containing at most $2m$ ones. We can find circular-ones arrangement for M in $O(n + m)$ time. If such an order does not exist then G is not a circular-arc graph. Otherwise, we place the arcs of I in this order clockwise on the circle. We keep the section $S_1, \dots, S_{2|I|}$ that are formed by the endpoint of I in an ordered cyclic list.

6 Stage 2: Placing the endpoints of the arcs in the sections

Consider the order of I found in Sect. 5.2. For every arc $x \in I^c$, the members of $N_I[x]$ are consecutive on the circle. Since there are no universal arcs in G , and I is a maximal independent set, x cannot contain all arcs of I and $N_I[x] \neq \emptyset$. Also, x overlap at most two arcs of I , since otherwise G is not a circular-arc graph and we should have detected it in Stage 1.

Let $x \in I^c$, the way we place the endpoint of x into their sections depends on the relation between x and the arcs of I . In most cases these relations suffice to determine the sections, and in the other cases we apply the algorithm recursively on an appropriate graph. The arc x satisfies one of the following cases (see Fig. 2).

- Arc x contains arc $a_i \in I$ and does not intersect any other arc in I . In this case the left endpoint of x is placed in S_{2i-1} and the right endpoint is placed in S_{2i+1} . For every $a_i \in I$ we accumulate all arcs that contain it and does not intersect any other arc of I in a set which we call A_i^e .

- Arc x overlaps $a_i \in I$ and does not intersect any other arc in I . For every $a_i \in I$ we accumulate these arcs in a set which we call U_i .
- Arc x intersects at least two arcs of I and does not intersect at least one. For all these arcs we identify in $N_I[x]$ the leftmost arc a_i , and the rightmost arcs a_j . We do that as we identified the first and last neighbor of every arc in the preliminary order of I in section 5.2. If x contains a_i then the left endpoint of x is in S_{2i-1} , if x overlaps a_i then this endpoint is in S_{2i} . Similarly, if x contains a_j then the right endpoint of x is in S_{2j+1} , if x overlaps a_j then this endpoint is in S_{2j} . For every arc $a_i \in I$ we accumulate every arc that contains all arcs in I except a_i in a set which we call A_i^c .
- Arc x overlaps two consecutive arcs $a_i, a_{i+1} \in I$ and contains all other arcs of I . In this case, we place the left endpoint of x in $S_{2(i+1)}$ and the right endpoint of x in S_{2i} .
- Arc x overlaps one arc $a_i \in I$ and contains all other arcs of I . For each $a_i \in I$ we accumulate these arcs in a set W_i .

At this point we placed the endpoints of all arcs in I^c into their sections except arcs in U_i and W_i for $i = 1, \dots, |I|$. Consider any arc $a_i \in I$ and the associated sets U_i and W_i . Each arc in $U_i \cup W_i$ has one endpoint in S_{2i} and the other in S_{2i-1} or in S_{2i+1} . Furthermore, all arcs of U_i must form a clique, as otherwise we can get from I a larger independent set by replacing a_i by two nonadjacent arcs in U_i , contradicting Lemma 1.

We place the endpoints of the arcs of $U_i \cup W_i$ in the sections S_{2i-1}, S_{2i} and S_{2i+1} for each $a_i \in I$ separately, by solving a new problem recursively on a graph G_i . The graph G_i which we construct is identical to the graph that Eschen [4] constructs¹. This graph is co-bipartite and therefore when we apply the algorithm to G_i , Case I applies and there would not be further recursion. We contribute the following observations. If G is a circular-arc graph then the recursive application of the algorithm on G_i takes time linear in the size of G_i . Furthermore, the sum of the sizes of all G_i 's is proportional to the size of G .

Let C_i^a be the set of arcs $\{a_i\} \cup A_i^e \cup U_i$. The set C_i^a forms a clique in G , since U_i forms a clique and all A_i^e arcs intersect every arc that a_i intersects. The clique C_i^a consists of all arcs contained in the union of the sections S_{2i-1}, S_{2i} and S_{2i+1} . Let Q_i be the set of arcs adjacent to some but not all arcs in C_i^a .

To define G_i we first define a subgraph of G which we denote by G_i' . The graph G_i' will also be a subgraph of G_i . The graph G_i' is the subgraph induced by $C_i^a \cup Q_i \cup A_i^c \cup W_i$. Note that Q_i is not necessarily disjoint of W_i and A_i^c .

We find Q_i by scanning the adjacency list of each $x \in C_i^a$. We maintain the set Y of arcs encountered during the scan. For each such arc $y \in Y$, we also keep a counter that counts the number of neighbors of y in C_i^a . When we finish scanning the adjacency list of every $x \in C_i^a$, the arcs of Q_i are exactly those arcs $y \in Y$ whose counters are smaller than $|C_i^a|$. We construct G_i' by scanning the adjacency list of each arc in it and restricting the list to contain only arcs inside G_i' .

The following lemma proves that all G_i' 's are constructed in linear time.

¹ Note that this graph is different to the one from [5] which seems to have an error.

Lemma 5. [4] *Every arc x participates in a constant number of graphs G'_i .*

Proof. From the definition of the sets U_i, W_i, A_i^e, A_i^c , it follows that an arc x can belong to at most one such set. If $x \in Q_i$ for some i , then one of the arcs a_{i-1}, a_i, a_{i+1} must be the leftmost or the rightmost arc of $N_I[x]$ in the cyclic order of I . So, x can belong to Q_i only if one of a_{i-1}, a_i or a_{i+1} is the rightmost or leftmost neighbor of it in the cyclic order of I . \square

Let n'_i be the number of vertices in G'_i , and let m'_i be the number of edges in G'_i . Every arc in G'_i covers at least one of the four endpoints of the three consecutive sections $S_{2i-1}, S_{2i}, S_{2i+1}$. Therefore, the arcs of G'_i are covered by four cliques, one for each endpoint. One of these cliques should have at least $\frac{n'_i}{4}$ vertices and therefore has at least $\frac{n'_i}{4}(\frac{n'_i}{4}-1)$ edges. So we check if $m'_i \geq \frac{n'_i}{4}(\frac{n'_i}{4}-1)$. If this inequality does not hold then G is not a circular-arc graph and we halt. Otherwise, we know that $m'_i = \Theta(n_i'^2)$.

We construct G_i from G'_i by adding a constant number of vertices and $O(n_i'^2) = O(m_i')$ edges. The vertices guarantee that any model of G_i can be embedded into a model of G , and the edges make all the vertices which are not in C_i^a into a second clique. So if n_i and m_i denote the number of vertices and edges in G_i respectively, then we also have that $m_i = \Theta(m_i') = \Theta(n_i'^2) = \Theta(n_i^2)$. The details of the construction of G_i are as in Eschen [4].

Since $m_i = \Theta(n_i^2)$, the recursive application of our algorithm to G_i takes $O(m_i)$ time. Since each arc of G belong to at most a constant number of graphs G_i , then each edge of G must belong to at most constant number of graphs G_i . And therefore, $\sum m_i = O(m)$ and the linear time bound for Stage 2 follows.

7 Stage 3: Arranging the endpoints in each section

We now know which sections contain the endpoints of every arc. Next we would arrange the endpoints inside each section. We follow Eschen and Spinrad's algorithm [4, 5], but provide a tighter analysis of it.

Our algorithm goes through the sections and tries to split each section S into ordered list of subsections. If S is split to subsections, those subsections replace S in the cyclic order of sections. When we cannot split sections anymore then each section S has a corresponding section S' such that all arcs that have one endpoint in S have their other endpoint in S' and vice versa. We then use recursion to order the endpoints inside sections containing more than one endpoint.

Our initial list of sections, $S_1, \dots, S_{2|I|}$, are the sections of Stage 2. Let n_i be the number of arcs that have an endpoint in S_i , and let m_i be the number of edges in G between these arcs. If G is a circular-arc graph then the arcs that have their right endpoint in S_i should be a clique in G , since they all cover the left endpoint of S_i . Similarly, the arcs that have their left endpoint in S_i also form a clique in G . So for each of the initial sections m_i should be at least $\frac{n_i}{2}(\frac{n_i}{2}-1)$. We check for all $i = 1, \dots, 2|I|$ that indeed $m_i \geq \frac{n_i}{2}(\frac{n_i}{2}-1)$, and if it does not hold for some i , then G is not a circular-arc graph. Note that since each arc has endpoints in two sections, $\sum m_i = O(m)$.

We split sections in the same way as Eschen and Spinrad [4, 5]. Intuitively, since the order of the endpoints inside a particular section S is not affected by any arc that does not have an endpoint in S , it suffices to determine the order between pairs of endpoints in the same section. Therefore the time it takes to split the sections is $O(\sum n_i^2)$. Since $O(\sum n_i^2) = O(\sum m_i) = O(m)$ this time is linear in the size of G . Details of this stage can be found at [4, 16].

References

1. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.
2. V. Conitzer, J. Derryberry, and T. Sandholm. Combinatorial auctions with structured item graphs. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 212–218, 2004.
3. X. Deng, P. Hell, and J. Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM J. Comput.*, 25(2):390–403, 1996.
4. E. M. Eschen. *Circular-arc graph recognition and related problems*. PhD thesis, Department of Computer Science, Vanderbilt University, 1997.
5. E. M. Eschen and J. P. Spinrad. An $O(n^2)$ algorithm for circular-arc graph recognition. In *SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 128–137, 1993.
6. W.-L. Hsu. $O(mn)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM J. Comput.*, 24(3):411–439, 1995.
7. W.-L. Hsu and R. M. McConnell. PC-trees and circular-ones arrangements. *Theor. Comput. Sci.*, 296(1):99–116, 2003.
8. M. C. Lin and J. L. Szwarcfiter. Efficient construction of unit circular-arc models. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 309–315, 2006.
9. T.-H. Ma and J. P. Spinrad. Avoiding matrix multiplication. In *Graph-Theoretic Concepts in Computer Science: 16th International Workshop WG '90*, Lecture Notes in Computer Science 484, pages 61–71, 1991.
10. R. M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003.
11. R. M. McConnell and J. P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.
12. J. P. Spinrad. Circular-arc graphs with clique cover number two. *Journal of Combinatorial Theory Series B*, 44(3):300–306, 1988.
13. J. P. Spinrad. *Efficient Graph Representations*. Fields Institute Monographs 19. American Mathematical Society, 2003.
14. J. P. Spinrad and J. Valdes. Recognition and isomorphism of two dimensional partial orders. In *Automata, Languages and Programming, 10th Colloquium*, Lecture Notes in Computer Science 154, pages 676–686, 1983.
15. S. Stefanakos and T. Erlebach. Routing in all-optical ring networks revisited. In *Proceedings of the 9th IEEE Symposium on Computers and Communication*, pages 288–293, 2004.
16. A. C. Tucker. An efficient test for circular-arc graphs. *SIAM J. Comput.*, 9(1):1–24, 1980.