Polynomial time recognition of unit circular-arc graphs

Guillermo Durán [1]

Dep. de Ingeniería Industrial, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile, Santiago, Chile. Fax number: (56) (2) 689-7895. gduran@dii.uchile.cl

Agustín Gravano [2]

Dep. de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina. agravano@dc.uba.ar

Ross M. McConnell

Computer Science Department, Colorado State University, Fort Collins, CO 80528, USA. rmm@cs.colostate.edu

Jeremy Spinrad

Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235, USA. spin@vuse.vanderbilt.edu

Alan Tucker

Department of Applied Mathematics, State University of New York at Stony Brook, Stony Brook, NY 11794-3600, USA. atucker@notes.cc.sunysb.edu

Abstract

We present an efficient algorithm for recognizing unit circular-arc (UCA) graphs, based on a characterization theorem for UCA graphs proved by Tucker in the seventies. Given a proper circular-arc (PCA) graph G, the algorithm starts from a PCA model for G, removes all its circle-covering pairs of arcs and determines whether G is a UCA graph. We also give an $O(N)$ time bound for Tucker's 3/2-approximation algorithm for coloring circular-arc graphs with N vertices, when a circular-arc model is given.

Key words: circular-arc graphs, graph algorithms, polynomial recognition, proper circular-arc graphs, unit circular-arc graphs.

1 Introduction

Let G be a finite undirected graph, and let $V(G)$ and $E(G)$ the vertex and edge sets of G, respectively. Denote $|V(G)| = N$ and $|E(G)| = M$.

A graph G is a *circular-arc graph* if there exists a family ϱ of arcs around a circle and a one-to-one correspondence between vertices of G and arcs in ϱ, such that two distinct vertices are adjacent in G if and only if the corresponding arcs intersect in ϱ. Such a family of arcs is called an *arc model* for G.

Circular-arc graphs have a variety of applications in such fields as genetic research [12], compiler design [18] and statistics [8]. The first characterization of circular-arc graphs is due to Tucker [15], who also gave an $O(N^3)$ algorithm for their recognition [19]. Recently, McConnell [10] improved this to $O(N+M)$.

Circular-arc graphs admit some interesting subclasses, such as Helly circular-arc graphs, proper circular-arc graphs and unit circular-arc graphs.

A family S of subsets satisfies the Helly property when every subfamily of S consisting of pairwise intersecting subsets has a common element. A graph G is a *Helly circular-arc* (HCA) graph if there exists an arc model for G such that the arcs satisfy the Helly property. Gavril [4] gave a characterization of these graphs using the clique matrix of a graph. This characterization leads to an $O(N^3)$ algorithm for recognizing HCA graphs.

A graph G is a *proper circular-arc* (PCA) graph if there exists an arc model for G such that no arc is included in another. Tucker [16] gave a characterization and an efficient recognition algorithm, using matrix characterizations, for recognizing PCA graphs. Deng et al. [2] gave a recognition algorithm that runs in linear time, and also produces a PCA model within this time bound when the graph is a PCA graph.

A graph G is a *unit circular-arc* (UCA) graph if there exists an arc model for G such that all the arcs are of the same length. Tucker [17] gave a characterization by forbidden subgraphs for this class of graphs. This characterization shows that UCA graphs are a proper subclass of PCA graphs. They can be useful in traffic control, when it is necessary that the green traffic lights for each lane at a street intersection are on for the same amount of time [6,13]. A polynomial-time algorithm for recognizing UCA graphs has not been given previously in the literature.

Let $G = (V(G), E(G))$ be a PCA graph, and $\varrho = \{A_1, ..., A_{|V(G)|}\}$ a proper circular-arc model for G.

[1] Partially supported by FONDECyT Grant 1030498 and Millennium Science Nucleus "Complex Engineering Systems", Chile and "International Scientific Cooperation Program CONICyT/SETCIP", Chile-Argentina.
[2] Partially supported by UBACyT Grant X127, Argentina.

An *(n, k)-circuit* of G with respect to ϱ is a set $\{x_1, ..., x_n\}$ of vertices $(n \geq 1)$, such that x_i and x_{i+1} are adjacent $(1 \leq i \leq n-1)$, x_n is adjacent to x_1, arc A_{i+1} starts clockwise from the counterclockwise endpoint of arc A_i, and the set of arcs wraps k times around the circle. To count the number of turns around the circle, we walk along the circumference starting at the counterclockwise endpoint of A_1, jumping from A_1 to A_2, from A_2 to A_3, and so on, and each time we pass by the starting point, we count a new turn.

An *(m, l)-independent set* is a set $\{x_1, ..., x_m\}$ of vertices $(m \geq 1)$, such that x_i and x_{i+1} are nonadjacent $(1 \leq i \leq m-1)$, x_m is nonadjacent to x_1, arc A_{i+1} is an arc that starts clockwise from the clockwise endpoint of arc A_i, and the set of arcs wraps l times around the circle. We count the number of turns around the circle in the same manner as before, but now we consider the last turn as complete (i.e., we add 1 to the total number of turns).

We call an (n, k)-circuit C *minimal* if no (n', k')-circuit with $n'/k' < n/k$ is formed by vertices in C (perhaps in another order). If $k = 0$ we assume that the quotient is equal to a big M. A *maximal* (m, l)-independent set is defined analogously.

The graph $CI(n, k)$, with $n > k$, is the circular-arc graph corresponding to the circular-arc model built in the following way: let ε be a positive real number, and r=1 be the radius of the circle. Draw n arcs $A_0, A_1, ..., A_{n-1}$ of length $l_1 = 2\pi k/n + \varepsilon$ such that each A_i starts at $2\pi i/n$ and ends at $2\pi(i+k)/n + \varepsilon$ (i.e., $A_i = (2\pi i/n, 2\pi(i+k)/n + \varepsilon)$). Afterward, draw n arcs $B_0, B_1, ..., B_{n-1}$ of length $l_2 = 2\pi k/n - \varepsilon$, such that each B_i starts at $(2\pi i + \pi)/n$ and ends at $(2\pi(i+k) + \pi)/n - \varepsilon$ (i.e., $B_i = ((2\pi i + \pi)/n, (2\pi(i+k) + \pi)/n - \varepsilon)$). For example, the model of Figure 1 generates the graph $CI(4, 1)$.

Tucker [17] introduced the definition of this family of graphs, and Durán and Lin [3] formalized this construction.

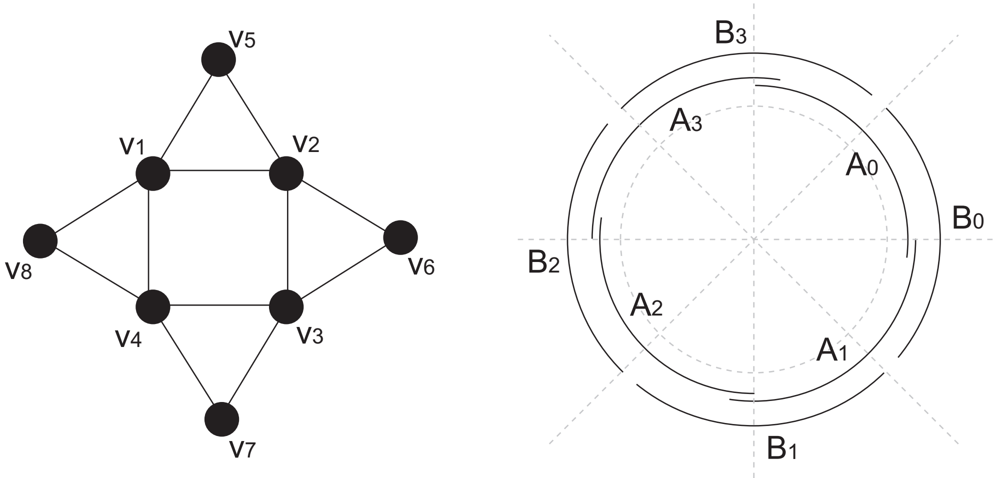

Fig. 1. $CI(4,1)$ and its circular-arc model.

Theorem 1.1 ([17]) *Let G be a proper circular-arc graph. Then G is a unit* *circular-arc graph if and only if G contains no $CI(n,k)$ subgraphs, with n,k relatively prime and $n > 2k$.*

Theorem 1.1 does not trivially lead to a polynomial-time algorithm for the recognition of this class, because we should show that a given PCA graph does not contain any $CI(n,k)$ subgraph, with n,k relatively prime and $n > 2k$. However, the proof of the theorem implicitly suggests an algorithm for recognizing UCA graphs. In this paper, we show that such an algorithm can be implemented to run in polynomial time, and prove its correctness.

In the proof of Theorem 1.1, Tucker uses two lemmas that are also useful in our study:

Lemma 1.1 ([17]) *Let G be a proper circular-arc graph and let ϱ be a proper circular-arc model for G. Then for any (n, k)-circuit C and (m, l)-independent set I (both with respect to ϱ), we have $m/l \leq n/k$.*

Lemma 1.2 ([17]) *Let G be a proper circular-arc graph. For any relatively prime n,k with $n > 2k$, G contains a minimal (n, k)-circuit C and a maximal (n, k)-independent set I with respect to any proper circular-arc model, if and only if G contains the subgraph $CI(n,k)$. For $n < 2k$, or n,k not relatively prime, no such maximal I exists. Any (2,1)-circuit can be eliminated by altering the PCA model.*

Corollary 1.1 *Let G be a proper circular-arc graph. Then the following statements are equivalent:*

(1) G is not a unit circular-arc graph.
(2) G contains a $CI(n,k)$ subgraph, with n,k relatively prime and $n > 2k$.
(3) G has a minimal (n, k)-circuit and a maximal (n, k)-independent set with respect to any PCA model, with n,k relatively prime and $n > 2k$.

Proof: It follows directly from Theorem 1.1 and Lemma 1.2. □

The following lemmas are true for any model ϱ of a proper circular-arc graph G. They have easy proofs that can be deduced from the proof in [17] of Lemma 1.2.

Lemma 1.3 *Let $C = x_1, ..., x_n$ be a minimal (n, k)-circuit, with the first vertex fixed. Then, we can order the other vertices of C such that for each i, A_{i+1} is the farthest arc adjacent to A_i, in the clockwise direction (i.e., among all arcs adjacent to A_i, A_{i+1} is the one which extends farthest from A_i, in the clockwise direction).*

Lemma 1.4 *Let $I = x_1, ..., x_m$ be a minimal (m, l)-independent set, with the first vertex fixed. Then, we can order the other vertices of I such that for each i, A_{i+1} is the first arc nonadjacent to A_i, in the clockwise direction (i.e., among all arcs nonadjacent to A_i, A_{i+1} is the one which starts nearest to A_i, in the clockwise direction).*

The following theorem guarantees that every PCA graph can be represented with no circle-covering pairs of arcs.

Theorem 1.2 ([6],[17]) *If G is a proper circular-arc graph, then G has a proper circular-arc model in which no pair of arcs covers all the circle (i.e., they do not intersect at both ends).*

In section 2 we present a quadratic algorithm for recognizing UCA graphs, using the results reviewed in this section and some ideas briefly introduced in [11]. This algorithm receives as input a PCA model with no circle-covering pairs of arcs. In section 3 we discuss the complexity of constructing a UCA model.

2 Recognition of UCA graphs

In this section, we describe the algorithm for recognizing UCA graphs and analyze its time complexity and correctness. The first step of the algorithm is to use the algorithm of [2] to determine whether the input graph is a proper circular arc graph, rejecting it if it is not, and obtaining a proper circular-arc model if it is.

If the graph passes this test, we then modify the proper circular arc model to obtain a new proper circular arc model that has no pair of arcs that covers the circle. The details of this step are given in Section 2.3.

It remains to search for upper and lower bounds for the length of the circumference for any unit circular-arc model for G with arcs of length 1. The upper bound is the smallest n/k among all minimal (n, k)-circuits, and the lower bound is the largest m/l among all maximal (m, l)-independent sets. These bounds will determine if such a unit circular-arc model can exist or not.

Let us put it in other words. Let Λ be the length of the circumference for a unit circular-arc model for G with arcs of length 1. Then Λ has to be smaller than a value imposed by all minimal (n, k)-circuits, and Λ has to be greater than a value imposed by all maximal (m, l)-independent sets. If that range is empty then a UCA model cannot exist, and vice versa.

(1) SEARCH FOR THE LARGEST LOWER BOUND
   For each arc A_i in ϱ:
   - (a) $c \leftarrow A_i$
   - (b) Mark c.
   - (c) While there are unmarked arcs nonadjacent to c:
     - (i) Find the first unmarked arc nonadjacent to c, in the clockwise direction. Mark it and store it in c.
     - (ii) Calculate m/l, where m is the number of marked arcs, and l is the number of times we have traversed the circle (calculated as

we said above). If this value is the greatest lower bound found so far, save it.

(2) SEARCH FOR THE SMALLEST UPPER BOUND

For each arc A_i in ϱ:

(a) $c \leftarrow A_i$

(b) Mark c.

(c) While there are unmarked arcs adjacent to c:

(i) Find the unmarked arc adjacent to c which extends farthest, in the clockwise direction. Mark it and store it in c.

(ii) If we have completed a new turn, calculate n/k, where n is the number of marked arcs, and k is the number of times we have traversed the circle (calculated as we said above). If this value is the smallest upper bound found so far, save it.

(3) The upper and lower bounds are equal if and only if G is not a UCA graph.

2.1 Correctness

First of all, the algorithm builds an $(m,\, l)$-independent set, starting at every arc and choosing at each step the first nonadjacent arc (in the clockwise direction). So, by Lemma 1.4, it visits every maximal $(m,\, l)$-independent set, and thus also the maximum one. This yields a lower bound m/l for the length of the circumference of any UCA model for G, with arcs of length 1.

Next, the algorithm builds an $(n,\, k)$-circuit, starting at every arc and choosing at each step the neighbor arc which extends farthest (in the clockwise direction). So, by Lemma 1.3, it visits every minimal $(n,\, k)$-circuit, and thus also the minimum one. This yields an upper bound n/k for the length of the circumference of any UCA model for G, with arcs of length 1.

By Lemma 1.1, $m/l \leq n/k$. There are two possible cases to consider:

- $m/l = n/k$. Then, this model contains a minimal $(n,\, k)$-circuit and a maximal $(n,\, k)$-independent set. Also, $n,\, k$ are relatively prime and $n \geq 2k$ because otherwise no maximal $(n,\, k)$-independent set would exist (Lemma 1.2). Lastly, $(n, k) \neq (2, 1)$ because the model has no circle-covering pairs. Therefore, by Corollary 1.1, G is not a unit circular-arc graph.
- $m/l < n/k$. Since the $(m,\, l)$-independent set found is maximum and the $(n,\, k)$-circuit is minimum, there cannot exist a maximal $(m',\, l')$-independent set and a minimal $(n',\, k')$-circuit, such that $m'/l' = n'/k'$. Therefore, there cannot exist a minimal $(n'',\, k'')$-circuit and a maximal (n'',k'')-independent set. By Corollary 1.1, this means that G is a unit circular-arc graph.

As we have seen, $m/l = n/k$ if and only if G is not a UCA graph. The algorithm outputs the truth value of the former equality, thus answering whether G is a UCA graph or not.

2.2 An $O(N^2)$ implementation

The algorithm generates and checks $2N$ sequences of arcs, each of length N. To obtain an $O(N^2)$ bound, when an arc is marked, we must be able to find the next unmarked arc of the sequence in $O(1)$ amortized time. For loop (1), this is the nearest unmarked arc beginning clockwise from the clockwise endpoint of the last marked arc, and for loop (2), this is the unmarked arc that intersects the marked arc and extends farthest in the clockwise direction.

Without loss of generality, we may assume that the endpoints of arcs are all distinct, since they can be perturbed slightly to make this true. Let us think of each counterclockwise endpoint of an arc as a *beginning point*, and associate the letter B with it, and think of each clockwise endpoint as an *ending point*, and associate the letter E with it. This yields a circular list of E's and B's. We label each B that begins an arc with a pointer to the E that ends that arc (refer to Figure 2). This gives a representation of the circular-arc model consisting of a circular list of B's and E's, with one pointer on each B.

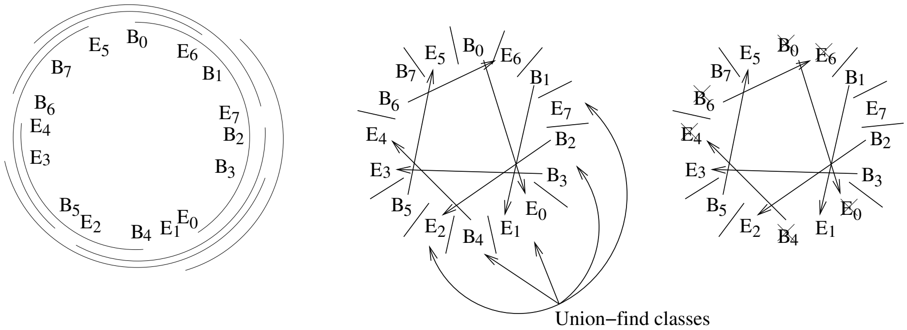

Fig. 2. The beginning point of each arc is represented by a B and the ending point by an E. Each B has a pointer to the corresponding E. Initially, there is one union-find class for each consecutive block of B's or E's (middle illustration). When all elements in a union-find class are marked, the class is merged with its two neighboring classes. The righthand illustration shows the union-find classes after $\{B_0, E_0, B_4, E_4, B_6, E_6\}$ have been marked.

When an arc is marked, let us mark the B and E corresponding to the arc in this representation. To solve the problem of finding the next arc to mark, we use a *union-find data type*. This data type can be used to maintain a partition of a set, which in our application is the set of marked and unmarked letters about the circle. In addition, it supports the following operations: a *union* operation, which, given two partition classes, merges them into a single class, and a *find* operation, which, given a letter, finds the partition class that contains the letter.

We maintain the invariant that each union-find partition class consists of a consecutive block of marked and unmarked letters about the circle. We also augment each class by labeling it with a pointer to a doubly-linked list of the unmarked letters in the block, ordered according to their clockwise order on the circle. Because the list is doubly-linked, a letter can be spliced out of its list in $O(1)$ time when it is marked. In addition, each class is labeled with a *next pointer* to union-find class that lies immediately clockwise from it on the circle, and a *previous pointer* to the union-find class that lies immediately counterclockwise from it. If C is a class, let $next(C)$ and $previous(C)$ denote the classes pointed to by its next and previous pointers.

For example, in the righthand illustration of Figure 2, the class $\{E_0, E_1, B_4, E_2\}$ at the bottom of the figure would have the list (E_1, E_2) as its ordered list of unmarked elements, a *next* pointer to $\{B_5\}$, and a *previous* pointer to $\{B_2, B_3\}$.

Initially, before any letters are marked, the union-find classes are the consecutive blocks of E's and consecutive blocks of B's about the circle. An example is given in the middle illustration of Figure 2. Therefore, when C is such a class, $next(C)$ and $previous(C)$ contain B's if C contains E's and they contain E's if C contains B's. The classes of E's alternate with the classes of B's around the circle.

We wish to maintain the invariant that every union-find class contains at least one unmarked letter. Therefore, when the last unmarked letter in a class C is marked, we merge $previous(C)$, C, and $next(C)$ and into a single union-find class using two union operations. The new class inherits its *previous* pointer from $previous(C)$ and its *next* pointer from $next(C)$. Assigning these pointers takes $O(1)$ time. The list of unmarked elements of the next class is obtained in $O(1)$ time by appending the list of unmarked elements of $next(C)$ to the end of the list of unmarked elements of $previous(C)$. This concatenation also takes $O(1)$ time.

For example, in middle illustration of Figure 2, marking $\{B_0, E_0\}$ causes $\{E_5\}, \{B_0\}$, and $\{E_6\}$ to merge. Marking $\{B_4, E_4\}$ causes $\{E_0, E_1\}$, $\{B_4\}$, and $\{E_2\}$ to merge. Marking $\{B_6, E_6\}$ causes no classes to merge. The situation at this point is depicted in the righthand illustration of Figure 2. At that point, marking $\{B_1, E_1\}$ causes $\{E_5, B_0, E_6\}$, $\{B_1\}$, and $\{E_7\}$ to merge. Marking $\{B_5, E_5\}$ causes $\{E_0, E_1, B_4, E_2\}$, $\{B_5\}$, and $\{E_3, E_4\}$ to merge. Marking $\{B_2, E_2\}$ causes no classes to merge. Marking B_7 causes $\{E_0, E_1, B_4, E_2, B_5, E_3, E_4\}$, $\{B_7, E_7\}$, and $\{E_5, B_0, E_6, B_1, E_7\}$ to merge. At this point, there is only one unmarked element left to mark.

By induction on the number of marked letters, this maintains the following invariants for as long as there is at least one unmarked arc left:

(1) Every class contains at least one unmarked letter.
(2) A class can contain a mixture of B's and E's, but the unmarked elements of a class are either all E's or all B's.
(3) Every union-find class is a consecutive block of (marked and unmarked) letters around the circle.
(4) Classes containing undeleted B's alternate around the circle with classes containing undeleted E's. That is, if C is a class, then $next(C)$ and

$previous(C)$ contain unmarked B's if and only if C contains unmarked E's.

(5) The list of unmarked letters in each class is ordered in clockwise order, starting from the beginning of the block of letters occupied by the class.

For loop (1) above, when the beginning letter B of an arc to be marked is found, we retrieve the ending letter E of the arc, and mark them, updating the union-find structure as described above. To find the beginning letter of the next arc to mark, we may perform a *find* operation on the E to obtain the class C that contains it, get $next(C)$ from C's pointer, and then get the first unmarked letter in $next(C)$ from the beginning of its list of its unmarked elements. For loop (2), the algorithm is the same, except that we get the next B from the end of the list of unmarked members of $previous(C)$; this is the last B that occurs inside the marked arc, hence the beginning of the arc that intersects the marked arc and extends farthest in the clockwise direction. The correctness is immediate from the Invariants 1-5.

The union-find data type can be implemented so that $O(N)$ *union* and $find$ operations on a ground set of size $O(N)$ takes $O(N\alpha(N))$ time [1], where $\alpha()$ is an extremely slow-growing, but unbounded, inverse of Ackermann's function. Together with the $O(1)$-time operations to mark letters, update the $previous$ and $next$ pointers and the lists of unmarked elements after each *union* or $find$ operations, the total time for executing loop (c) of loop (1) or loop (2) is $O(N\alpha(N))$. Since loop (c) is executed N times, this gives an $O(N^2\alpha(N))$ time bound for an implementation of the algorithm.

$O(N^2)$ is the best bound one can hope for for an implementation of the algorithm, since it generates and checks $\Omega(N)$ of lists of $\Omega(N)$ elements each. The $O(N^2\alpha(N))$ bound leaves open the theoretical question of whether the best bound achievable is $O(N^2\alpha(N))$ or $O(N^2)$. We now show that $O(N^2)$ is possible.

When certain constraints can be placed in advance on the allowed *union* operations, the union-find data structure given by Gabow and Tarjan [5] allows $O(N)$ *union* and N *find* operations on a set of N elements to be carried out in $O(N)$ time. The constraints that must be satisfied are that the data structure is initialized in advance with an unrooted tree whose vertices are the N elements of the ground set, such that the subsequent union operations will only produce union-find classes that induce connected subgraphs of this tree.

Let G be the graph obtained by letting each letter around the circle be a vertex, and letting letters be adjacent if they are neighbors in the cyclic order. Because of Invariant 3, we would like to be able to initialize Gabow-Tarjan with G. However, since G is a cycle, it has one too many edges to be used to initialize Gabow-Tarjan, which requires a tree.

Nevertheless, is not hard to solve this problem with Gabow-Tarjan. The trick is to remove one of the edges of the cycle so that the remainder of the cycle is a path, hence a tree. Initializing the Gabow-Tarjan structure with this path permits all of the *union* operations we need, except those that involve union-find classes on opposite sides of the deleted edge. When such a union is called for, we refrain from calling on the Gabow-Tarjan structure to perform it, and instead, simulate the union by letting the two Gabow-Tarjan classes point to each other. A subsequent *find* operation can then be simulated by a Gabow-Tarjan *find*, followed by a pointer traversal. This adds $O(1)$ to the cost of a *find*, so the additional cost can be ignored in the asymptotic analysis.

(Alternatively, since a cycle is a planar graph, we can also use a generalization of Gabow-Tarjan, due to Gustedt, that can be initialized with an arbitrary planar graph, rather with just a tree [7].)

This gives the cost of the $O(N)$ *union* and $O(N)$ $find$ operations to $O(N)$. Multiplying this by the number of iterations of the outer loop yields a total running time of $O(N^2)$.

2.3 An algorithm for eliminating (2,1)-circuits

The algorithm for recognizing UCA graphs uses the algorithm of [2] to produce a PCA model. It remains to describe how to modify this model to eliminate any circle-covering pairs of arcs, in order to satisfy the preconditions of loops (1) and (2) of Section 2.

Here we present an algorithm for eliminating all pairs of arcs that cover all the circle of a PCA model. This algorithm is based on the proof of Theorem 1.2 [6].

(1) From the PCA model, generate a sequence σ of letters, where each letter represents an endpoint of a circular arc. So, every arc A_x has two related letters: x and $\hat{x}$, representing its counterclockwise and clockwise endpoints, respectively.
(2) Circularly find a subsequence in σ:
$$\underbrace{a \dots \hat{b}}_{\tau}\underbrace{\dots b \dots \hat{a} \dots}_{\rho}$$
(3) Replace τ with $\tau_1\tau_2$, where τ_1 are the letters in τ with $^\wedge$, and τ_2 are the letters in τ without $^\wedge$, always preserving their relative order.
(4) Jump to step 2, until there are no more such subsequences in σ.

Let us analyze the idea of the algorithm. At step 2, it looks for a pair of arcs that covers all the circle, and eliminates it with the reorderings of step 3. These reorderings do not generate new circle-covering pairs of arcs, and does not break the PCA model. Therefore, repetition of this process will leave the PCA model free of such pairs.

2.3.1 Time complexity

Let N be the number of arcs of the input PCA model, and let M be the number of edges of the corresponding graph. This algorithm can be implemented using an array L of length $2N$ to represent the sequence σ of letters. Each element of L is formed by two values: one letter (with or without ^) and the index in L of its opposite letter. The algorithm can be written in the following way:

```
1. For i = 0 to 2N-1
2.     If L[i] has no ^ then
3.       If i < L[i].opposite
           Find the largest j such that
               i < j < L[j].opposite < L[i].opposite
         Else
           Find the largest j such that
               L[i].opposite < i <  j < L[j].opposite
4.      If such a j is found then
5.             Rearrange L[i..j]: put the letters with ^
               before the ones without ^, preserving their
               relative order and updating the indices
               to the opposite letters.
6.             Increase i so that it points again to the
               same element in L.
```

The operation of step 5 is performed in a circular fashion around the array. In step 3, if such a j is found, then $L[j]$ must have ^, because the algorithm works on a PCA model.

Suppose that arc A_a has k double overlapping arcs $A_{b_1}, ..., A_{b_k}$. In other words, there is a sequence of letters $a ... \hat{b}_1 ... \hat{b}_2 ... \hat{b}_k ... b_1 ... b_2 ... b_k ... \hat{a} ...$. In step 3, the algorithm searches for A_a's last double overlapping arc (the one corresponding to b_k, or $L[j]$), if there is any. It is easy to see that by breaking this circle-covering pair, every other circle-covering pair containing A_a is also broken.

In order to do so, step 5 rearranges the subsequence $a ... \hat{b}_k$, maintaining the internal indices of L and the relative order of the letters. This can be done in linear time, by adding an auxiliary field in each element of L. The search of step 3 can also be accomplished in linear time. Therefore, the algorithm can be run in $O(N^2)$.

The main cycle stops only at letters without ^, skipping the other ones. Step 6 is necessary because the letter in $L[i]$ is moved to the right in step 5. However, this will cause no trouble because no letters without ^ are skipped, since the rearrangement of step 5 preserves their relative order.

It remains to see whether the beginning letter of every arc is processed or not. Is it possible that a beginning letter c (not processed yet) advances from one of the last positions of L to one of the first positions when an arc at an earlier position is handled? If this case were possible, the outer loop would quit before reaching c. But can it really happen?

This situation would be possible if at some iteration of the algorithm, τ could begin at the end of L and finish at the beginning of the array. But this case is not possible. Suppose the following situation, and that letter a is going to be processed:

$$\underbrace{\hat{b}}_{\tau}\underbrace{\ldots b \ldots \hat{a} \ldots}_{\rho}\underbrace{a\ c}_{\tau}$$

Arcs A_a and A_b cover all the circle. Thus, letter c would pass from position $2N-1$ to position 0 in L. But these two arcs, which cover all the circle, would have been handled when the outer loop reached letter b.

3 Construction of a UCA model

In this paper, we presented a polynomial time algorithm for recognizing UCA graphs. The time complexity of the whole algorithm is $O(N^2)$, starting from a PCA model, and this model can be constructed in $O(M+N)$ time [2]. Unfortunately, this algorithm does not lead directly to an algorithm for constructing a UCA model. However, Tucker's proof for Theorem 1.1 actually constructs a unit circular-arc model for a graph G, and thus gives us a recognition algorithm, which we outline here very briefly.

The circumference of the circle is set to a length between the lower and upper bounds dictated by the recognition algorithm. The algorithm works by successively fixing one more arc A to be unit length, while maintaining a proper circular-arc model. The process involves shrinking or lengthening A; to maintain the proper model, other arcs are lengthened or shortened together with A. However, arcs which have already been set to be unit length must maintain unit length; if such an arc must be modified to maintain the proper circular-arc model, it is rotated rather than shrunk or expanded. The rotation of a unit arc may cause other unit arcs to be rotated; fundamentally, we end up rotating (i, j)-circuits and/or (g, h)-independent sets, and the algorithm works because the circumference can handle the bounds forced by any such circuit or independent set.

Analyzing the time complexity of this algorithm is tricky, because it involves manipulation of large integers, which cannot be assumed to take constant time. Therefore, this leaves two open problems. Is it always possible to construct a unit circular-model with endpoints corresponding to integers of polynomial size? If so, how efficiently can such a model be constructed?

4 Conclusions and Open Problems

No better time bound is possible for an implementation of the algorithm we give here, since it generates and tests N sequences of length N. This does not preclude the possibility that some other algorithm could yield a faster time bound for recognizing unit circular-arc graphs.

As observed in Section 3, a polynomial bound for producing a unit circular-arc model when an input graph is a unit circular arc graph remains an open problem.

Tucker [18] has given an approximation algorithm for finding a proper coloring of an arbitrary circular-arc graph that uses at most $\lfloor 3/2 \rfloor$ times the minimum possible number of colors, providing that no three arcs cover the entire circle. New approximation bounds for the algorithm have recently been shown by Valencia-Pabon [20]. In particular, if c is the minimum number of arcs in the model that cover the circle, the algorithm produces a coloring that uses at most $\lceil (c-1)/(c-2) \rceil$ times the optimum number of colors. The algorithm is a slight variant of the loop (1.c) of the algorithm of Section 2, and the technique we develop in Section 2.2 for implementing this loop efficiently also gives an $O(N)$ bound for the coloring algorithm, provided that the circular-arc model gives the endpoints in sorted cyclic order.

Acknowledgments: We thank Flavia Bonomo for her comments and suggestions, which improved this work.

References

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, "Introduction to Algorithms", McGraw Hill, Boston, 2001.

[2] X. Deng, P. Hell and J. Huang, "Linear time representation algorithms for proper circular-arc graphs and proper interval graphs", SIAM J. Comput. 25 (1996), 390-403.

[3] G. Durán and M. Lin, "On some subclasses of circular-arc graphs", Congressus Numerantium 146 (2000), 201-212.

[4] F. Gavril, "Algorithms on circular-arc graphs", Networks 4 (1974), 357-369.

[5] H. N. Gabow and R. E. Tarjan, "A linear time algorithm for a special case of disjoint set union", Journal of Computer and System Sciences 30 (1985), 209-221.

[6] M. Golumbic, "Algorithm Graph Theory and Perfect Graphs", Academic Press, New York, 1980.

[7] J. Gustedt, "Efficient Union-Find for Planar Graphs and other Sparse Graph Classes", Theoret. Comput. Sci., 203 (1998), 123-141.

[8] L. Hubert, "Some applications of graph theory and related non-metric techniques to problems of approximate seriation: the case of symmetry proximity measures", British J. Math. Statist Pscychology, 27 (1974), 133-153.

[9] H. Kaplan, N. Shafir, and R. E. Tarjan, "Union-find with deletions", Journal of the ACM 34 (1987), 596-615.

[10] R. M. McConnell, "Linear-time recognition of circular-arc graphs", Algorithmica 37(2) (2003), 93-147.

[11] J. Spinrad, "Representations of Graphs", book manuscript (1997).

[12] F. Stahl, "Circular genetic maps", J. Cell Physiol. 70 (Suppl. 1) (1967), 1-12.

[13] K. Stouffers, "Scheduling of traffic lights - A new approach", Transportation Res. 2 (1968), 199-234.

[14] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm", Journal of the ACM 22 (1975), 215-225.

[15] A. Tucker, "Characterizing circular-arc graphs", Bull. Amer. Math. Soc. 76 (1970), 1257-1260.

[16] A. Tucker, "Matrix characterizations of circular-arc graphs", Pacific J. Math. 38 (1971), 535-545.

[17] A. Tucker, "Structure theorems for some circular-arc graphs", Discrete Mathematics 7 (1974), 167-195.

[18] A. Tucker, "Coloring a family of circular-arc graphs", SIAM J. Appl. Math. 29 (1975), 493-502.

[19] A. Tucker, "An efficient test for circular-arc graphs", SIAM J. Comput. 9 (1980), 1-24.

[20] M. Valencia-Pabon, "Revisiting Tucker's algorithm to color circular arc graphs", SIAM J. Comput. 32 (2003), 1067-1072.