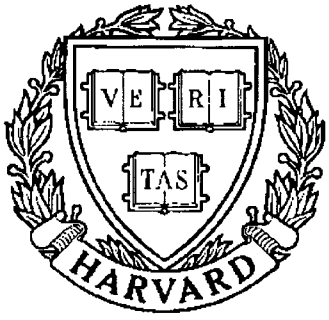
THESIS REPORT
Master's Degree

S Y S T E M S
R E S E A R C H
C E N T E R



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), the University of Maryland, Harvard University, and Industry

Efficient Algorithms for Circular-Arc Containment Graphs

by: M.V. Nirkhe
Advisor: K. Nakajima

M.S. 87-11
Formerly TR 87-211

Efficient Algorithms for Circular-Arc Containment Graphs

by

Madhura Vivek Nirkhe

Thesis submitted to the faculty of the Graduate School
of the University of Maryland in partial fullfillment
of the requirements for the degree of
Master of Science.
1987.

Advisory Committee:

Associate Professor	Dr. Kazuo Nakajima
Professor	Dr. Joseph JáJá
Assistant Professor	Dr. Nariman Farvardin

Abstract

Title of Thesis :

Efficient Algorithms for Circular-Arc Containment Graphs

Madhura Vivek Nirkhe

Master of Science, 1987

Thesis directed by: Dr. Kazuo Nakajima.

In the recent past, a wide variety of algorithms have been developed for a class of intersection graphs, called interval graphs. As a generalization of interval graphs, circular-arc graphs have also been studied extensively. Another category of graphs, namely containment graphs, has recently received some attention. In particular, interval containment graphs have been studied recently and several optimal algorithms have been developed for this class of graphs.

In this thesis we introduce a new class of containment graphs called circular-arc containment graphs. A circular-arc containment graph is a generalization of an interval containment graph and is defined as follows : A graph $G_A = (V_A, E_A)$ is called a circular-arc containment graph for a family $A = \{A_1, \ldots, A_n\}$ of arcs on a circle, if for each $v_i \in V_A$, there is an arc $A_i \in A$, and $(v_i, v_j) \in E_A$ if and only if one of A_i and A_j contains the other.

We characterize this class of graphs by establishing its equivalence to another relatively new class of intersection graphs, called circular permutation graphs. Given a circular-arc containment graph in the form of a family of arcs on a circle, we develop efficient algorithms for finding a maximum clique, a maximum independent set, and a minimum coloring of the graph.

Acknowledgements

I take this opportunity to express my deep gratitude to my advisor, Dr. Kazuo Nakajima for his valuable guidance and encouragement during the course of this thesis.

I would like to extend my thanks to Dr. Sumio Masuda, who, during his stay here as visiting faculty from Osaka University, Japan, gave me help and advice that have been very precious to me in my work.

I must thank two organizations which provided me with financial support; National Science Foundation through grants MIP-84-51510 and CDR-85-00108 (Systems Research Center), and AT&T.

Lastly, I also wish to thank my husband Vivek, without whose support I could have achieved very little.

Contents

List of Figures

Chapter 1

Introduction

1.1. Motivation

In the recent past, a wide variety of algorithms have been developed for a class of intersection graphs, called interval graphs [8,10]. As a generalization of interval graphs, circular-arc graphs have also been studied extensively [10,13]. Another category of graphs, namely containment graphs, has recently received some attention [9]. In particular, interval containment graphs were studied by Masuda and Nakajima [12]. Several optimal algorithms for this class of graphs are provided in [12].

In this thesis, we introduce a new class of containment graphs called circular-arc containment graphs. They are a generalization of interval containment graphs. We define this class of graphs and show its equivalence to another relatively new class of intersection graphs, called circular permutation graphs, which was introduced by Rotem and Urrutia [15].

The problems of finding a maximum clique, a maximum independent set, and a minimum coloring are NP-hard for an arbitrary graph [5]. However, these problems are solvable in polynomial time for various special classes of graphs [3,5,6,7,8,10,16]. In particular, Masuda and Nakajima [12] presented optimal $O(nlogn)$ time algorithms for all the above problems for an interval containment graph. Their algorithms are based on Fredman's algorithm [4] for finding longest increasing subsequences of a sequence of distinct integers. In this thesis, we extend Masuda and Nakajima's techniques to find a solution to the each of the above problems for a circular-arc containment graph.

Besides being of theoretical interest, circular-arc containment graphs can

be foreseen to have some practical applications as well. Interval graphs have found many practical applications [14,17,18]. As the next step to intersection of segments on the real line, the problem of reporting intersections of rectangles has captured the attention of many researchers, among them being Bentley and Wood [2], and is seen to have bearing on many aspects of VLSI design. Algorithms for another category of graphs concerning containment of rectangular regions on a two-dimensional plane, to be called rectangle containment graphs, have the potential to provide valuable insight into approaches to VLSI design and can be envisaged to be of importance. The the problem of rectangle containment subsumes the problem of interval containment. Circular-arc containment graphs as a generalization of interval containment graphs may provide some insight into this problem in the future.

1.2. Outline

In the remainder of this chapter we define the graph terminology needed for dealing with a circular-arc containment graph, and introduce the notation we use in the particular context of the circular-arc containment graph.

Chapter 2 gives the definition of a canonical family of arcs and outlines a procedure to convert any arbitrary family into canonical form. It also describes an algorithm to construct a circular-arc containment graph from a given family of arcs in canonical form. Finally we describe an algorithm due to Fredman [4] for finding longest increasing subsequences of a sequence of distinct integers. The last two algorithms are used in later chapters.

In Chapter 3 we establish the equivalence of a circular-arc containment graph and a circular permutation graph by means of a constructive proof. This implies that a circular-arc containment graph can be recognized in polynomial time since a polynomial time recognition algorithm for a circular permutation

graph has been developed by Rotem and Urrutia [15].

Chapter 4 describes an optimal algorithm for finding a maximum clique of a circular-arc containment graph. In Chapter 5 we present an efficient algorithm for finding a maximum independent set while Chapter 6 is devoted to the development of a fast algorithm to get a minimum coloring. These algorithms are in turn applicable to a circular permutation graph.

Finally Chapter 7 summarizes the contributions of this thesis and future directions for research.

1.3. Graph Terminology and Definitions

A *graph* is a pair $G = (V, E)$, where V is a finite set of $n =\mid V \mid$ elements called *vertices* and $E \subseteq \{(x, y) \mid x, y \in V, x \neq y\}$ is a set of $e =\mid E \mid$ unordered pairs of vertices called *edges*. For an edge (x, y), x and y are said its to the *endpoints*. For distinct vertices x and y we say that x is *adjacent* to y and vice versa, if $(x, y) \in E$; otherwise they are said to be *independent*. An *adjacency set* of a vertex v in $G = (V, E)$, denoted by $Adj_G(v)$, is the set $\{y \mid (v, y) \in E\}$. The *degree* of v, denoted by $d_G(v)$, is the cardinality of its adjacency set. An *adjacency list* of a vertex v is a list of all the edges which have v as one endpoint.

A set $V' \subseteq V$ of vertices is called an *independent set* if the vertices in V' are pairwise independent. A *maximum independent set* (abbreviated as *MIS*) of G is a set with the maximum cardinality among all independent sets of G. A family $Q = \{Q_i\}_{i=1}^{k}$ of sets of vertices is called a *coloring* if $Q_i \cap Q_j = \phi$ for $i \neq j$, $\bigcup_{i=1}^{k} Q_i = V$, and Q_i is an independent set for each $i = 1, 2, \ldots, k$. A *miminum coloring* of G is one with the minimum cardinality among all colorings of G.

A set $V' \subseteq V$ of vertices is called a *clique* if the vertices in V' are pairwise

adjacent. A *maximum clique* of G is one with the maximum cardinality among all cliques of G. A *covering by cliques* of G is a family $R = \{R_i\}_{i=1}^{k}$ of sets of vertices such that $R_i \cap R_j = \phi$ for all $i \neq j$, $\bigcup_{i=1}^{k} R_i = V$, and R_i is a clique for each $i = 1, 2, \ldots, k$. A *minimum covering by cliques* of G is one with the minimum cardinality among all coverings by cliques of G.

A sequence of edges $< (v_{i_1}, v_{i_2}), (v_{i_2}, v_{i_3}), \ldots, (v_{i_{k-1}}, v_{i_k}) >$ is defined to be a *path* from v_{i_1} to v_{i_k}. A set of edges $M \subseteq E$, of a graph $G = (V, E)$, is called a *matching* if every vertex in V is the endpoint of at most one edge in M. A path in G that has alternate edges in M and not in M is called an *alternating path.*

A graph $G = (V, E)$ is called an *intersection graph* for a family of sets $F = \{F_i\}_{i=1}^{n}$ if there is a one-to-one correspondence between V and F such that two vertices are adjacent in G if and only if the corresponding sets have a nonempty intersection. If F is a family of intervals on the real line, G is called an *interval graph.* If F is a family of arcs on a circle, G is called a *circular-arc graph.* The class of interval graphs is properly contained in the class of circular-arc graphs.

A graph $G = (V, E)$ is called a *containment graph* for a family of sets $F = \{F_i\}_{i=1}^{n}$ if there is a one to one correspondence between V and F such that two vertices are adjacent in G if and only if one of the corresponding sets in F is a subset of the other. In this thesis the containment graph for a particular family F is denoted by $G_F = (V_F, E_F)$. If F is a set of intervals on the real line, G_F is called an *interval containment graph* (abbreviated as *ICG*). If F is a family of arcs on a circle, the graph G_F is called a *circular-arc containment graph* (abbreviated as *CACG*). We elaborate on an ICG and a CACG in the next section.

Let π be a permutation of numbers $1, 2, \ldots, n$ and let L_1 and L_2 be two parallel lines. Let n points be labeled on L_1 by $1', 2', \ldots, n'$ and on L_2 by $\pi(1), \pi(2), \ldots \pi(n)$, respectively, from left to right. A *permutation diagram* consists of the labeled lines L_1 and L_2 and a set of segments of lines $\{l_1, l_2, \ldots l_n\}$, such that l_i joins point i' on L_1 to point i on L_2, $1 \leq i \leq n$. A graph is called a *permutation graph* if it represents at least one permutation diagram.

Let π be a permutation of numbers $1, 2, \ldots, n$ and let C_p and C_q be two concentric circles, C_p contained in C_q. Let n points be labeled on the inner circle C_p as $1', 2', \ldots, n'$ and let $\pi(1), \pi(2), \ldots, \pi(n)$ be labeled on the outer circle C_q. Let n chords $W_1, W_2, \ldots, W_n$ be drawn in the annular region between C_p and C_q such that chord W_i joins i' on C_p to i on C_q. Two chords may intersect, but no more than once. The two concentric circles with n points on each, along with the n chords together constitute a *circular permutation diagram* (abbreviated as *CPD*). Note that i' and i can be joined in many ways. A CPD contains only one such connection for each pair (i, i'). For the same permutation π with different choices of connections, different CPD's can be obtained. Also permutations which are cyclic shifts of one another can yield the same CPD. A graph $G_D = (V_D, E_D)$ represents a CPD D if the vertices of G_D can be labeled $v_1, v_2, \ldots, v_n$ such that $(v_i, v_j) \in E_D$ if and only if chords W_i and W_j in D intersect each other. A graph G is called a *circular permutation graph* (abbreviated as *CPG*) if it represents at least one CPD.

1.4. Notation For ICGs and CACGs

Let $I = \{I_1, I_2, \ldots, I_n\}$ be a family of n intervals on the real line. Each interval in I has two endpoints, and each endpoint is assigned a positive integer which corresponds to its co-ordinate on the real line. Among the two

endpoints, the one which is encountered earlier while traversing the interval from left to right is called the *head* and the other is called the *tail.* Each interval is thus uniquely represented by the ordered pair $[h, t]$, where h and t are the co-ordinates of the head and tail, respectively.

For every two intervals I_j and I_k, we say that I_j *contains* I_k, denoted by $I_j \supset I_k$, or alternately $I_k \subset I_j$, if the set of points on I_k is a proper subset of the set of points on I_j. Otherwise, I_j and I_k are said to be *independent* of each other. Let $I' = \{I_{j_1}, \ldots, I_{j_k}\}$ be a subfamily of I. I' is called an *interval containment family* if for every pair of intervals I_{j_i}, I_{j_l}, $1 \leq i, l \leq k$, either $I_{j_i} \subset I_{j_l}$ or $I_{j_l} \subset I_{j_i}$. A *maximum interval containment family* is one with the maximum cardinality among all interval containment families of I. I' is called an *independent interval family* if the intervals of I' are pairwise independent. A *maximum independent interval family* is one with the maximum cardinality among all independent interval families of I. An *interval coloring* is an assignment of colors to the intervals of I such that every two intervals, one of which contains the other are assigned distinct colors. A *minimum interval coloring* of I is a coloring that uses the fewest number of colors. As defined earlier, a graph is called an interval containment graph for I, which we denote by $G_I = (V_I, E_I)$, if for each $v_j \in V_I$, there is an arc I_j in I, and $(v_j, v_k) \in E_I$ if and only if one of I_j and I_k contains the other. The vertices corresponding to an interval containment family (resp., independent interval family) of I form a clique (resp., independent set) of G_I.

A family of n intervals is said to be in *canonical* form if all endpoints of the intervals in the family are assigned distinct integer co-ordinates between 1 and $2n$.

Let C be a circle. Let $A = \{A_1, A_2, \ldots, A_n\}$ be a family of n arcs

on this circle. For every two arcs A_i and A_j, we say that A_i *contains* A_j, denoted by $A_i \supset A_j$, or alternately $A_j \subset A_i$, if the set of points on A_j is a proper subset of the set of points on A_i. Otherwise, A_i and A_j are said to be *independent* of each other. Let $A' = \{A_{i_1}, \ldots, A_{i_k}\}$ be a subfamily of A. A' is called an *arc containment family* if for every pair of arcs A_{i_j}, A_{i_l}, $1 \leq j, l \leq k$, either $A_{i_j} \subset A_{i_l}$ or $A_{i_l} \subset A_{i_j}$. A *maximum arc containment family* is one with the maximum cardinality among all arc containment families of A. A' is called an *independent arc family* if the arcs of A' are pairwise independent. A *maximum independent arc family* is one with the maximum cardinality among all independent arc families of A. An *arc coloring* is an assignment of colors to the arcs of A such that every two arcs, one of which contains the other, are assigned distinct colors. A *minimum arc coloring* of A is a coloring that uses the fewest number of colors. As defined earlier, a graph is called a circular-arc containment graph for A, which we denote by $G_A = (V_A, E_A)$ for a particular family A, if for each $v_i \in V_A$, there is an arc A_i in A, and $(v_i, v_j) \in E_A$ if and only if one of A_i and A_j contains the other. The vertices corresponding to an arc containment family (resp., independent arc family) of A form a clique (resp., independent set) of G_A.

Each arc in A has two endpoints, and each endpoint is assigned a positive integer which corresponds to its co-ordinate on the circumference of the circle. Among the two endpoints, the one which is encountered earlier while traversing the arc in the clockwise direction is called the *head* and the other is called the *tail.* Each arc is thus uniquely represented by the ordered pair $[h, t]$, where h and t are the co-ordinates of the head and tail, respectively.

A family of n arcs is said to be in *canonical* form if :

1. All endpoints of the arcs in the family are assigned distinct integer co-

ordinates between 1 and $2n$.

2. No single arc covers the circle by itself.

3. The co-ordinate assigned to the head of arc A_1 in the family is the positive integer 1 and all other endpoints are assigned co-ordinates in ascending order in the clockwise direction.

In a canonical family of arcs, an arc is said to be a *forward arc* if $h < t$, otherwise it is said to be a *backward arc.*

For an arc $A_i \in A$ and for an endpoint j of another arc in a canonical family A, we say that A_i *contains* the point j if one of the following three conditions holds :

1. $1 \leq h_i < j < t_i \leq 2n.$

2. $1 \leq t_i < h_i < j \leq 2n.$

3. $1 \leq j < t_i < h_i \leq 2n.$

Later in Chapter 2 we show how to convert any family of arcs into canonical form.

1.5. Summary

In this chapter we have established the basic framework needed to deal with the new class of graphs that we have introduced as circular-arc containment graphs. We have described in brief our motivation for undertaking the study of these graphs and defined the terminology and notation to be used in the course of the following chapters.

Chapter 2

Preliminaries

2.1. Overview

It is convenient to define a standard form for a family A of arcs on a circle that we use to represent the circular-arc containment graph G_A. This form was defined as the *canonicalform* in Chapter 1. In this chapter we describe three algorithms that are useful in the later chapters. The first is an $O(nlogn)$ algorithm to convert any family of n arcs into canonical form. The second obtains a CACG G_A corresponding to a family A of arcs in canonical form. The third finds a longest increasing subsequence of a finite sequence of distinct integers and is due to Fredman [4].

2.2. Canonical Family of Arcs

A given family A of n arcs can always be converted into a canonical family A^c in $O(n \log n)$ time in such a fashion that the two CACG's $G_A = (V_A, E_A)$ and $G_{A^c} = (V_{A^c}, E_{A^c})$ are exactly identical. The $O(n \log n)$ bound conforms to the bound for sorting the endpoints of arcs which is the operation involved in the conversion. Since the time complexity of all our algorithms will be $O(n \log n)$ or higher, without loss of generality we can subsequently assume that the family is already available in its canonical form.

Theorem 2.1. *For any family A of n acs on a circle, one can construct, in $O(nlogn)$ time, a canonical family A^c such that the circular-arc containment graphs G_A and G_{A^c} are identical.*

The theorem can be proven by showing that for a given family A the following procedure correctly constructs a new family A^c, which is canonical.

Procedure 2.1.

1. $n \leftarrow | A |$.

2. Construct a list of indices $L_h = (i_1, \ldots, i_n)$ such that one of the following is true for every pair of indices i_k and i_{k+1} :

   (a) $h_{i_k} < h_{i_{k+1}}$.

   (b) $h_{i_k} = h_{i_{k+1}}$ and A_{i_k} is a backward arc and $A_{i_{k+1}}$ is a forward arc.

   (c) $h_{i_k} = h_{i_{k+1}}$ and A_{i_k} and $A_{i_{k+1}}$ are forward arcs and $t_{i_k} > t_{i_{k+1}}$.

   (d) $h_{i_k} = h_{i_{k+1}}$ and A_{i_k} and $A_{i_{k+1}}$ are backward arcs and $t_{i_k} > t_{i_{k+1}}$.

3. Construct a list of indices $L_t = (i_1, \ldots, i_n)$ such that one of the following is true for every pair of indices i_k and i_{k+1} :

   (a) $t_{i_k} < t_{i_{k+1}}$.

   (b) $t_{i_k} = t_{i_{k+1}}$ and A_{i_k} is a forward arc and $A_{i_{k+1}}$ is a backward arc.

   (c) $t_{i_k} = t_{i_{k+1}}$ and A_{i_k} and $A_{i_{k+1}}$ are forward arcs and $h_{i_k} > h_{i_{k+1}}$.

   (d) $t_{i_k} = t_{i_{k+1}}$ and A_{i_k} and $A_{i_{k+1}}$ are backward arcs and $h_{i_k} > h_{i_{k+1}}$.

4. **for** $k \leftarrow 1$ **to** $2n$

   $p \leftarrow$ the first element of L_t

   if $L_h = \phi$

   then

   remove p from L_t

   $t_p' \leftarrow k$

   else

   $q \leftarrow$ the first element of L_h

   if $h_q \leq t_p$

then

remove q from L_h

$h_q' \leftarrow k$

else

remove p from L_t

$t_p' \leftarrow k$.

5. **for** $k \leftarrow 1$ **to** n

   $A_k' \leftarrow [h_k', t_k']$.

6. $A^c \leftarrow \{A_1', A_2', \ldots A_n'\}$. □

Proof of Theorem 2.1 : Consider two arcs A_j and A_k in A. There are two possible cases. Either one of them contains the other or they are independent of each other.

Suppose one contains the other. Without loss of generality we can assume that $A_j \subset A_k$. We will show that in all the following exhaustive subcases, $A_j' \subset A_k'$.

1. Both A_j and A_k are forward arcs or both A_j and A_k are backward arcs. In this subcase, we have three possibilities :

   (a) $h_j = h_k$ and $t_j < t_k$. The procedure numbers h_k' before h_j' and t_j' before t_k' in this case. Thus $h_k' < h_j'$ and $t_j' < t_k'$, and hence $A_j' \subset A_k'$.

   (b) $h_k < h_j$ and $t_j = t_k$. The procedure numbers t_j' before t_k' and h_k' before h_j' in this case. Thus $h_k' < h_j'$ and $t_j' < t_k'$, and hence $A_j' \subset A_k'$.

(c) $h_k < h_j$ and $t_j < t_k$. In this case, $h_k' < h_j'$ and $t_j' < t_k'$ in the new numbering, and hence $A_j' \subset A_k'$.

2. A_j is a forward arc and A_k is a backward arc. There are four possibilities :

(a) $0 < t_k < h_k = h_j < t_j \leq 2n$. Procedure 2.1 numbers h_k' before h_j' in Case 2(b). Hence $0 < t_k' < h_k' < h_j' < t_j' \leq 2n$ and $A_j' \subset A_k'$.

(b) $0 < h_j < t_j = t_k < h_k \leq 2n$. Procedure 2.1 numbers t_j' before t_k' in Case 3(b). Thus $0 < h_j' < t_j' < t_k' < h_k' \leq 2n$ and $A_j' \subset A_k'$.

(c) $0 < t_k < h_k < h_j < t_j \leq 2n$. In this case in the new numbering, $0 < t_k' < h_k' < h_j' < t_j' \leq 2n$, and hence $A_j' \subset A_k'$.

(d) $0 < h_j < t_j < t_k < h_k \leq 2n$. In this case in the new numbering, $0 < h_j' < t_j' < t_k' < h_k' \leq 2n$, and hence $A_j' \subset A_k'$. □

Fig. 2.1 illustrates how the new numbering preserves containment.

Suppose A_j and A_k are independent of each other. Then $h_j \neq h_k$ and $t_j \neq t_k$. The procedure simply preserves the relative order of the co-ordinates of A_j and A_k and hence A_j' and A_k' are also independent of each other.

We have shown that for any two arcs A_i and A_j in A, the relation between them is the same as the relation between A_i' and A_j' in A^c. That is, for the corresponding vertices, $(v_i, v_j) \in E_A$ if and only if $(v_i', v_j') \in E_{A^c}$. Thus G_A is identical to G_{A^c}. □

Remark 2.1. Steps 2 and 3 of Procedure 2.1 involve sorting that takes $O(nlogn)$ time. The procedure further scans the two lists L_h and L_t and renumbers the co-ordinates in Step 4 that takes $O(n)$ time. Hence the total time required for Procedure 2.1 is $O(nlogn)$. □

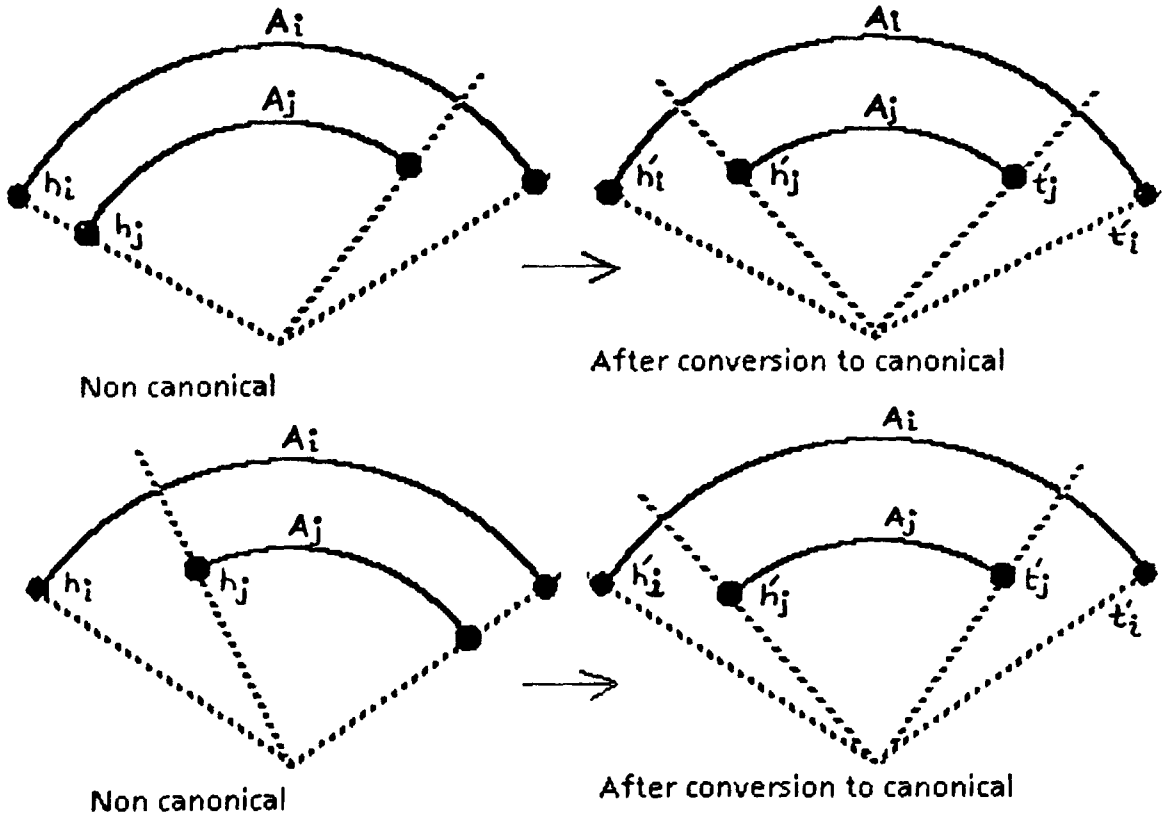


Fig. 2.1. Effect of Procedure 2.1. on a Pair of Arcs

Fig. 2.2 shows a non-canonical family A of arcs. Fig. 2.3 shows the conversion of family A to a canonical family A^c using the above procedure.

2.3. Construction of a CACG

In this section we outline an $O(nlogn + e)$ algorithm to construct a CACG $G_A = (V_A, E_A)$ from a given family A of n arcs, where $e = \mid E_A \mid$. This algorithm can be used later in order to identify a minimum degree vertex of the CACG.

Procedure 2.2. Construction of a CACG

Input : A family $A = \{A_1, \ldots A_n\}$ of arcs, in canonical form, with the arcs sorted in ascending order of their head co-ordinates.

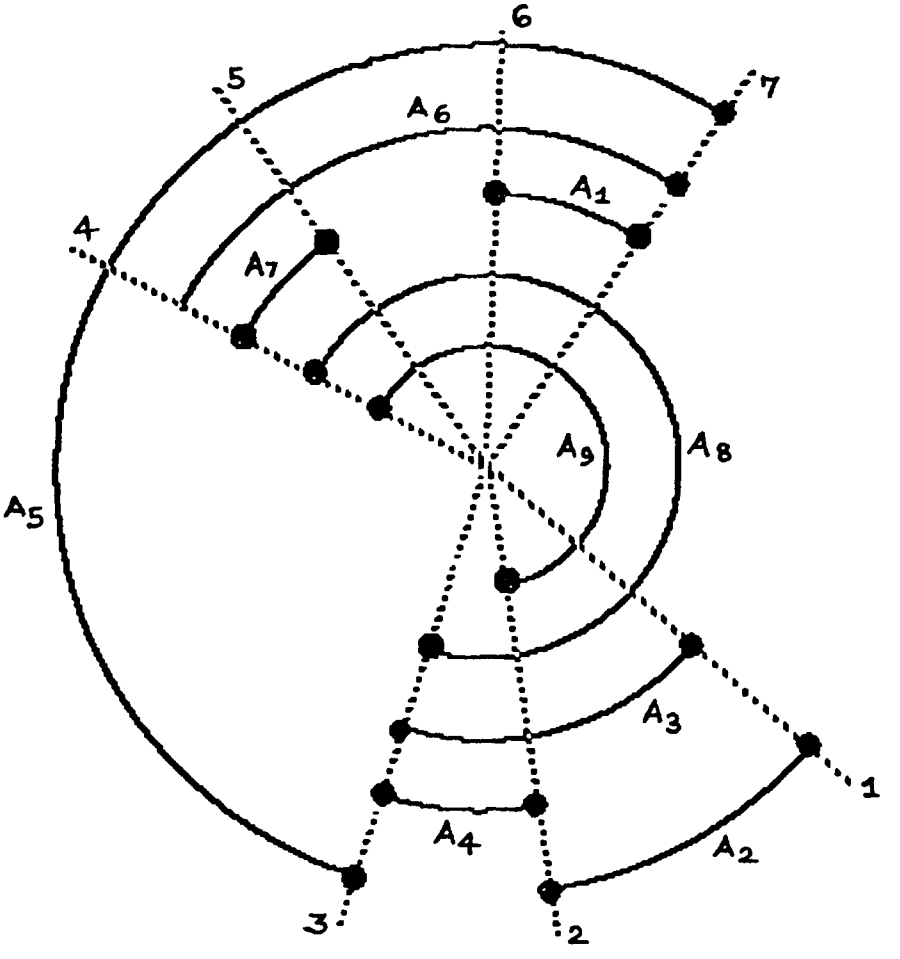


Fig. 2.2. A Non-Canonical Family A of Arcs on a Circle

Output : The CACG $G_A = (V_A, E_A)$.

Method :

1. Push the indices of all the backward arcs into a stack ST in increasing order of their head co-ordinates.

2. Starting with the point marked 1, traverse the circle in the clockwise direction. As soon as a head co-ordinate h_j of an arc A_j, is encountered, push the index j into the stack ST. When a tail point t_j of an arc A_j is

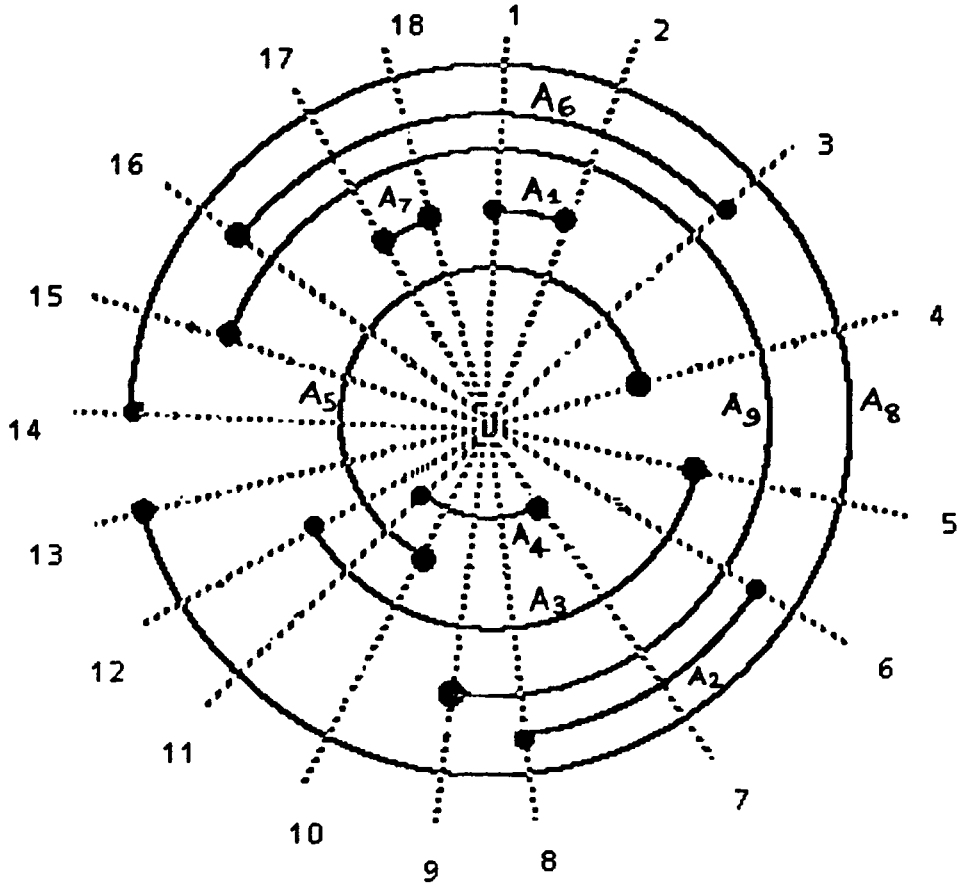


Fig. 2.3. *A* **Converted into a Canonical Family** A^c

encountered,

(a) Delete the index j from the stack.

(b) For each index k below the index j in the stack, add (v_k, v_j) to the adjacency list of v_k, and also add (v_j, v_k) to the adjacency list of v_j.

3. Continue in this manner till the entire circle is traversed and the point 1 is encountered again. At this point, ignore the contents of the stack and output the CACG in the form of the adjacency lists of the vertices. □

Remark 2.2. The procedure takes a total of $O(n + e)$ time, if the arcs are

already sorted, otherwise it takes $O(nlogn+e)$ time, as each of the n arcs makes an entry in the stack at most twice, and the adjacency lists are updated $O(e)$ times. Since initially all the backward arc indices are loaded in the stack, the edges between the vertices corresponding to the backward arcs will be added to the adjacency lists when the backward arc tail co-ordinates are encountered. Subsequently the forward arcs will make an entry into the stack when their head co-ordinates are encountered and the adjacency lists involving them are updated on encountering their tail co-ordinates. The backward arcs in effect enter the stack twice, once at the start of the algorithm and again when their head co-ordinates are actually encountered during the traversal of the circle. They remain in the stack until the last step of the algorithm. □

2.4. Longest Increasing Subsequences

Let $S = < x_1, x_2, \ldots, x_k >$ be a sequence of k distinct numbers. An *increasing subsequence* of S is defined to be a sequence $S' = < x_{i_1}, \ldots, x_{i_h} >$ such that $x_{i_1} < x_{i_2} < \ldots < x_{i_h}$ and $1 \leq i_1 < \ldots < i_h \leq k$. A *longest increasing subsequence* is one with the maximum number of elements among all increasing subsequences of S. A method for computing the first row of the Young tableau was devised by Knuth [11]. By modifying Knuth's method Fredman [4] suggested a solution to the longest increasing subsequence problem that takes $O(klogk)$ time. He also proved that the algorithm is optimal. We describe Fredman's algorithm. We subsequently use it to compute a maximum clique, a maximum independent set, and a minimum coloring of an ICG.

Algorithm 1 (Fredman [4]).

Input : A sequence of distinct integers $S = < x_1, x_2 \ldots, x_k >$.

Output : A longest increasing subsequence S' of S.

Method :

1. Create an empty stack ST_1 and push x_1 into it. $h \leftarrow 1$.

2. **for** $j \leftarrow 2$ **to** k

   if the top element of stack ST_h is larger than x_j

   then

   Find the first stack ST_p whose top element
   is larger than x_j.
   Push x_j into stack ST_p.
   if $p \neq 1$ **then**
   $PRE(x_j) \leftarrow$ the top
   element of stack ST_{p-1}.

   else

   $h \leftarrow h + 1$
   Create a new empty stack ST_h
   and push x_j into it.
   $PRE(x_j) \leftarrow$ the top
   element of stack ST_{h-1}.

3. $x_{i_h} \leftarrow$ any element in ST_h.

4. Output a sequence $S' = < x_{i_1}, x_{i_2}, \ldots, x_{i_h} >$ such that $x_{i_t} \leftarrow PRE(x_{i_{t+1}})$ for $t = 1, 2, \ldots, h-1$. □

Remark 2.3. By maintaining the top elements of the stacks at any time using appropriate data structures, we can find stack ST_p of Step 2, if one exists, in $O(klogk)$ time by a binary search for each $j = 2, 3, \ldots, k$. Therefore, Algorithm 1 runs in $O(klogk)$ time and is optimal to within a constant factor. □

2.5. Summary

In this chapter we have outlined three basic algorithms that will be used in some of the later chapters. The first obtains a canonical form of any given family of arcs. The second is a procedure to get the CACG G_A corresponding to a family A of arcs in canonical form. The third finds a longest increasing subsequence of a sequence of distinct integers.

Chapter 3

Equivalence of a CACG and a CPG

3.1. Overview

It is known that every ICG is a permutation graph and vice versa [9]. In this chapter we establish the same relationship between a CACG and a CPG. We present a constructive proof for this equivalence. Since Rotem and Urrutia [15] developed a polynomial time algorithm for testing whether a graph is a CPG or not, a CACG can be recognized in polynomial time. Furthermore, all the algorithms for a CACG which we present in this thesis are immediately applicable to a CPG.

Recall that a graph G is a CPG if there exists at least one CPD D such that G is the intersection graph for D. Similarly a graph G is a CACG if there exists at least one family A of arcs such that G is the containment graph for A. In order to establish the equivalence of a CACG and a CPG, it suffices to show that for every canonical family A of arcs on a circle there exists a CPD D and vice versa, satisfying the following properties :

1. For every arc A_i in A there is a chord W_i in D and vice versa.

2. For every pair of arcs A_i and A_j, one of which contains the other, there is a pair of chords W_i and W_j which intersect each other, and vice versa.

We say that such A and D are *functionally equivalent.*

In Section 3.2, we describe how to construct such a CPD D from a given canonical family A. In Section 3.3, we show how to convert a CPD D into a canonical family A of arcs. The results of the two sections establish the equivalence of a CACG and a CPG.

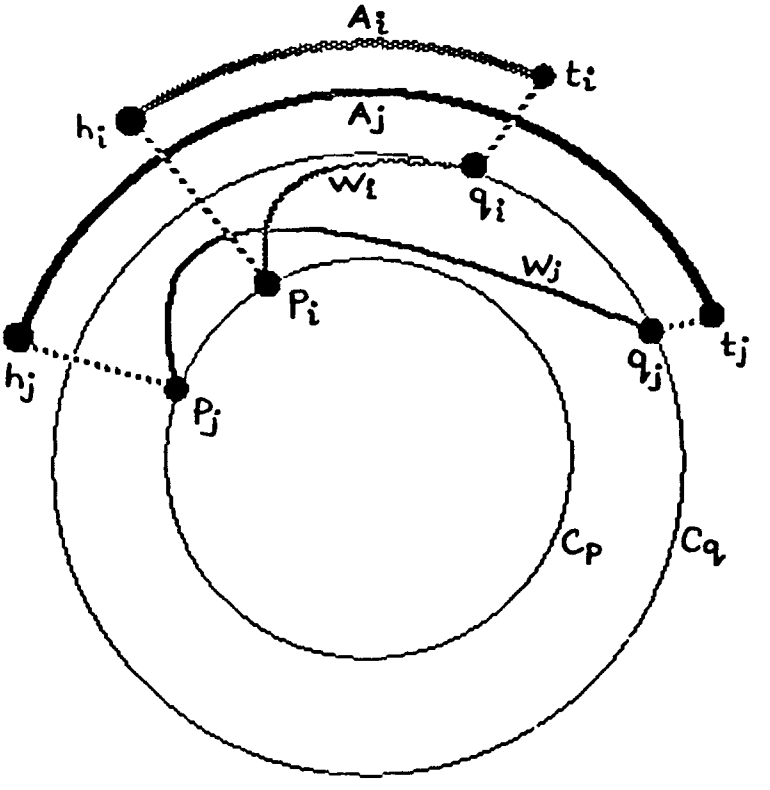


Fig. 3.1. A CPD from a Family of Arcs

3.2. Conversion from a Family of Arcs to a CPD

For every canonical family A of arcs on a circle C, represented by a CACG G_A, the following procedure constructs a CPD D :

Procedure 3.1.

1. Draw two circles C_p and C_q concentric with the circle C. Both C_p and C_q are smaller in diameter than C, C_p having a smaller diameter than C_q.

2. For every arc $A_k = [h_k, t_k]$ project h_k radially onto the inner circle C_p and call the point on it p_k. Project t_k radially onto the outer circle C_q

and call the point on it q_k.

3. In the annular region between C_p and C_q draw a chord W_k in the clockwise sense, joining p_k with q_k. □

Fig. 3.1 illustrates the procedure.

We observe that in the circular permutation diagram thus constructed, W_i and W_j intersect each other if and only if one of A_i and A_j contains the other. □

Therefore, we have shown that for every family A of arcs on a circle there is a corresponding CPD D such that the CACG G_A is exactly identical to the CPG G_D. Thus we have the following theorem :

Theorem 3.1. *Every CACG is a CPG.* □

Remark 3.1. The above conversion apparently takes $O(n)$ time since two endpoints h_k and t_k of an arc A_k are radially projected to points p_k and q_k, respectively, on two concentric circles C_p and C_q. A chord is drawn in the clockwise sense joining p_k to q_k. Thus each arc is converted into a chord in the CPD in constant time, and for n arcs the procedure takes $O(n)$ time. □

3.3. Conversion from a CPD to a Family of Arcs on a Circle

The CPD obtained from the family of arcs by Procedure 3.1 consists of only clockwise sensed chords in the annular region. We call a CPD with all the chords oriented in the same sense (either clockwise or counterclockwise), a *monosense* CPD. A chord in a CPD that spans more than 2π radians and hence appears to overlap itself is called an *XL-chord.* Fig. 3.2 shows an XL-chord. A CPD without XL-chords is called a *proper* CPD. The application of

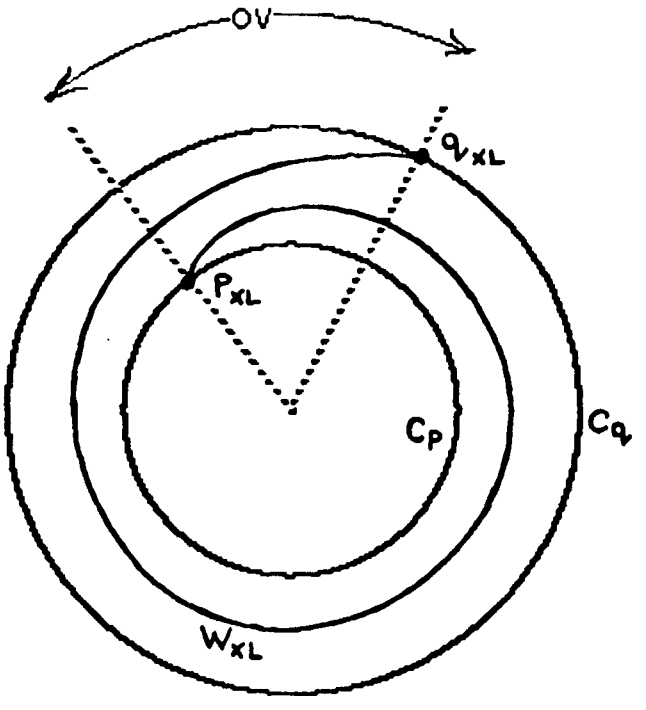


Fig. 3.2. An XL-Chord

Procedure 3.1 in reverse order to a proper monosense CPD yields a family A of arcs such that D and A are functionally equivalent.

Let D be a CPD with chords arbitrarily oriented in both senses. If D can be transformed into a proper monosense CPD, then one can prove that every CPG is a CACG. We show that by the execution of the following two steps D can be converted into a proper monosense CPD. The first step converts D to a monosense CPD. In the second step we convert the monosense CPD to a proper monosense CPD.

Step 1 : Conversion to a monosense CPD

Let W_k be the largest counterclockwise sensed chord in D. Keeping the endpoints of all the chords on the inner circle C_p in their positions, shift all the other endpoints of the chords on the outer circle C_q in the clockwise

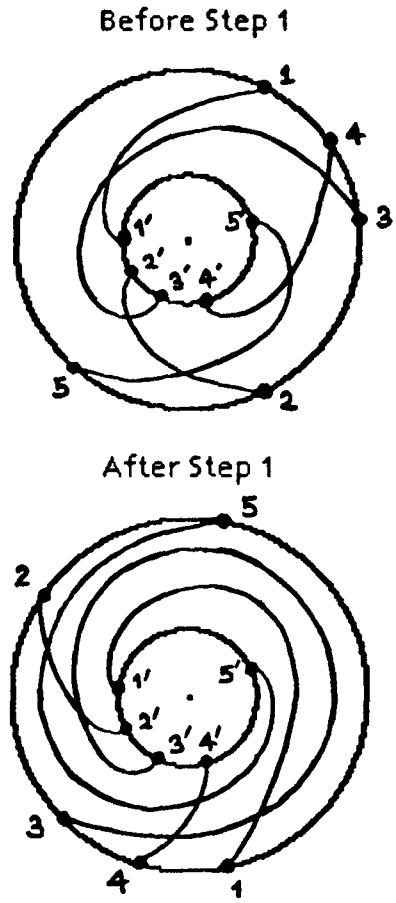


Fig. 3.3. Execution of Step 1 in the Proof of Theorem 3.2.

direction until q_k passes p_k. We call this intermediate diagram D_i. With q_k in a new position on C_q, W_k is now a clockwise sensed chord. By virtue of this operation, all counterclockwise chords become clockwise, since W_k was the largest counterclockwise chord. Thus D_i is monosense. Note that all clockwise chords become even more clockwise, and some may even turn into XL-chords. Note however, that none of the intersections were affected, and that no two chords intersect at more than one point. Fig. 3.3 illustrates an example of the execution of Step 1 on a CPD.

Step 2 : Conversion to a proper monosense CPD

Now that all chords are clockwise, the problem is to fix the chords in D_i

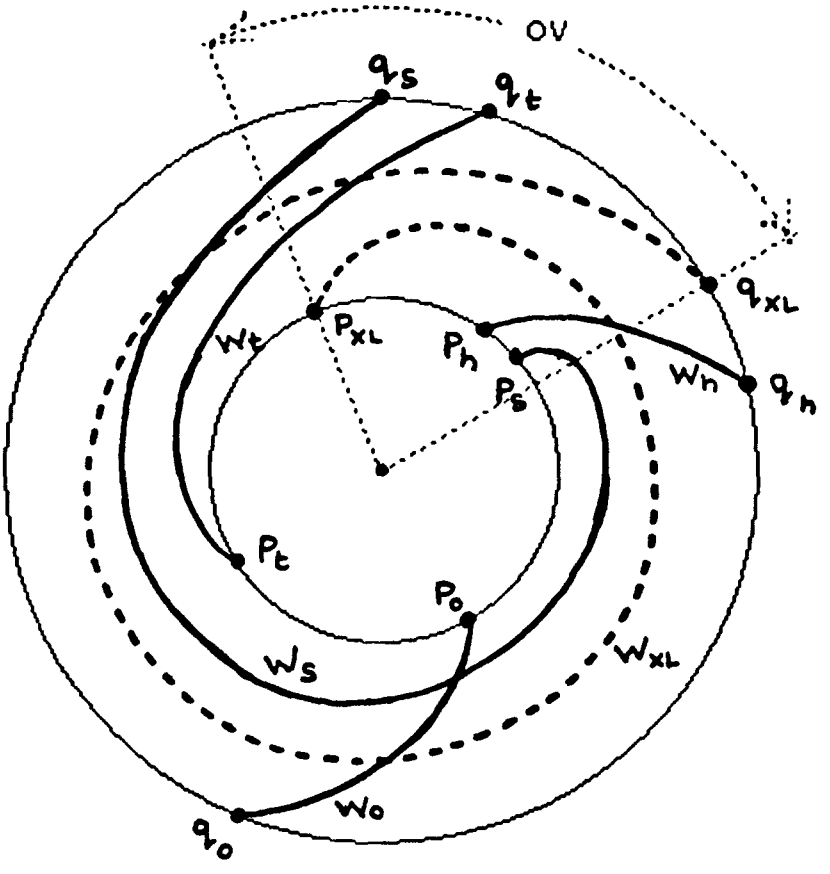


Fig. 3.4. An XL-Chord and Other Chords Relative to It

that have got longer than necessary as they were stretched in the clockwise direction. Consider one such XL-chord W_{XL} shown in Fig. 3.4.

As shown in the figure this chord creates a region between the two points p_{XL} and q_{XL} where the chord essentially appears to overlap itself. We denote this overalp region by OV. A chord which is not an XL-chord can not be completely contained in OV, since if it were, it would intersect W_{XL} twice, which is not legitimate. Thus there are four kinds of chords possible relative to W_{XL} as shown in Fig. 3.4. Chords such as W_t have their tail points in the region OV but the head points outside it. Chords such as W_h have their head points in region OV, but the tail points outside it. The third kind of chords

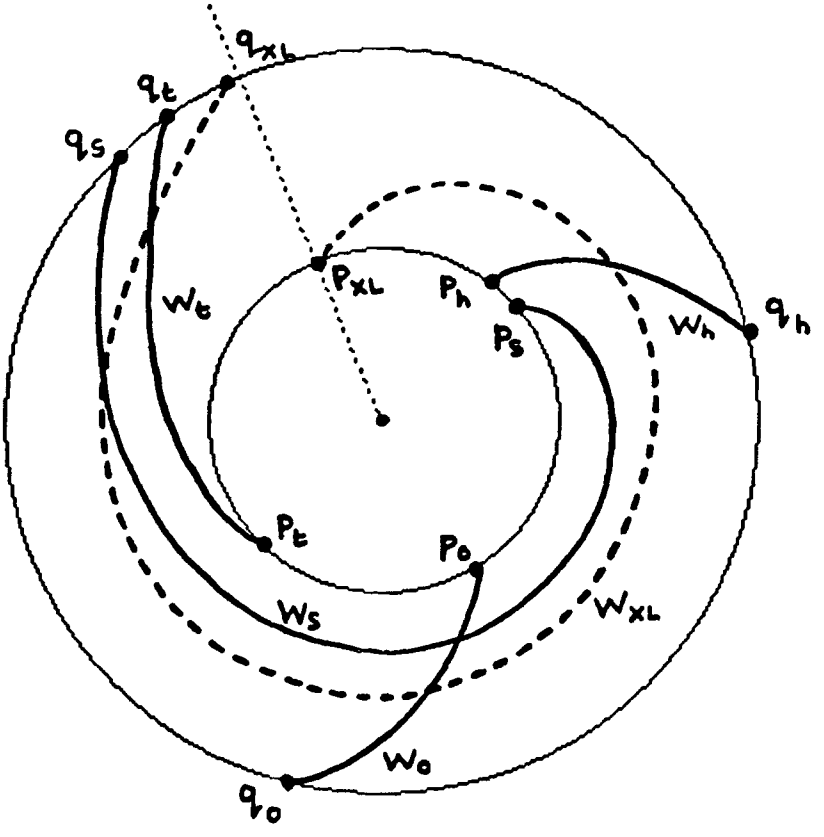


Fig. 3.5. 'Shrinking' of the XL-Chord of Fig. 3.4.

such as W_o lie entirely outside region OV. The fourth kind of chords such as W_s spiral in and out of region OV but are not completely contained in the region OV.

If we were to slide the point q_{XL} in the counterclockwise direction, along with all other points on the circumference of the outer circle in the region OV so that q_{XL} would just get past p_{XL}, we would in effect be only shrinking some chords. In no event will this operation create a counterclockwise chord. Note that none of the other chords are adversely affected. Chords such as W_t and W_s only shrink a little, still remaining clockwise and chords such as W_h and W_o are unaffected. Fig. 3.5 shows the *shrinking* of the XL-chord of Fig. 3.4.

As for the other XL-chords, we shrink them too one by one. We show

that given two XL-chords which are to be shrunk, the order in which the shrinking is done is entirely inconsequential. Let B_1 and B_2 be two arbitrary XL-chords with respective overlap regions OV_1 and OV_2. The following three possibilities arise :

1. OV_1 is contained in OV_2.

2. OV_1 and OV_2 are disjoint.

3. OV_1 and OV_2 have a non-empty intersection region.

As is evident from Figs. 3.6, 3.7, and 3.8, the order in which two XL-chords are shrunk is not consequential in all three cases. Therefore, by shrinking all the XL-chords, one by one, we eventually obtain a proper monosense CPD.

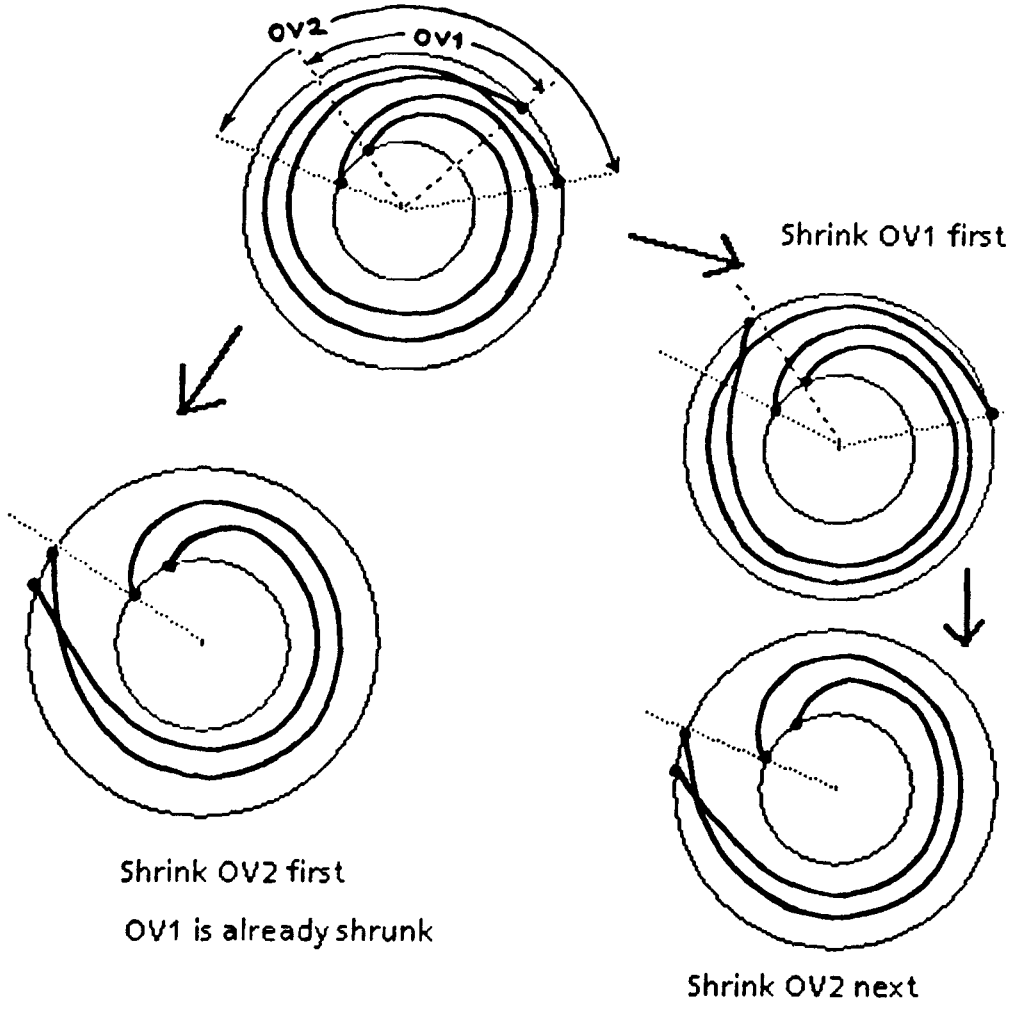


Fig. 3.6. OV_1 is Contained in OV_2.

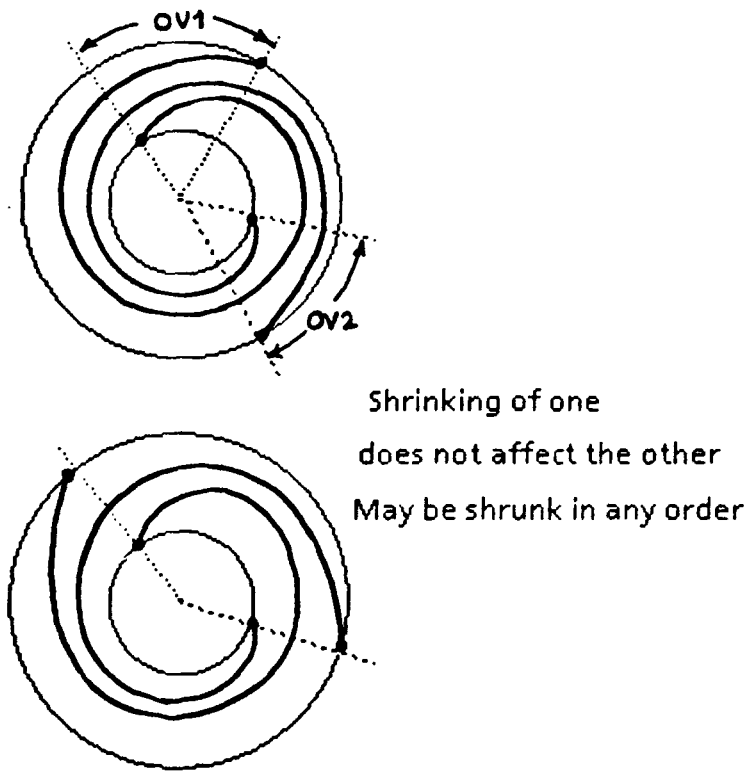


Fig. 3.7. OV_1 **and** OV_2 **are Disjoint.**

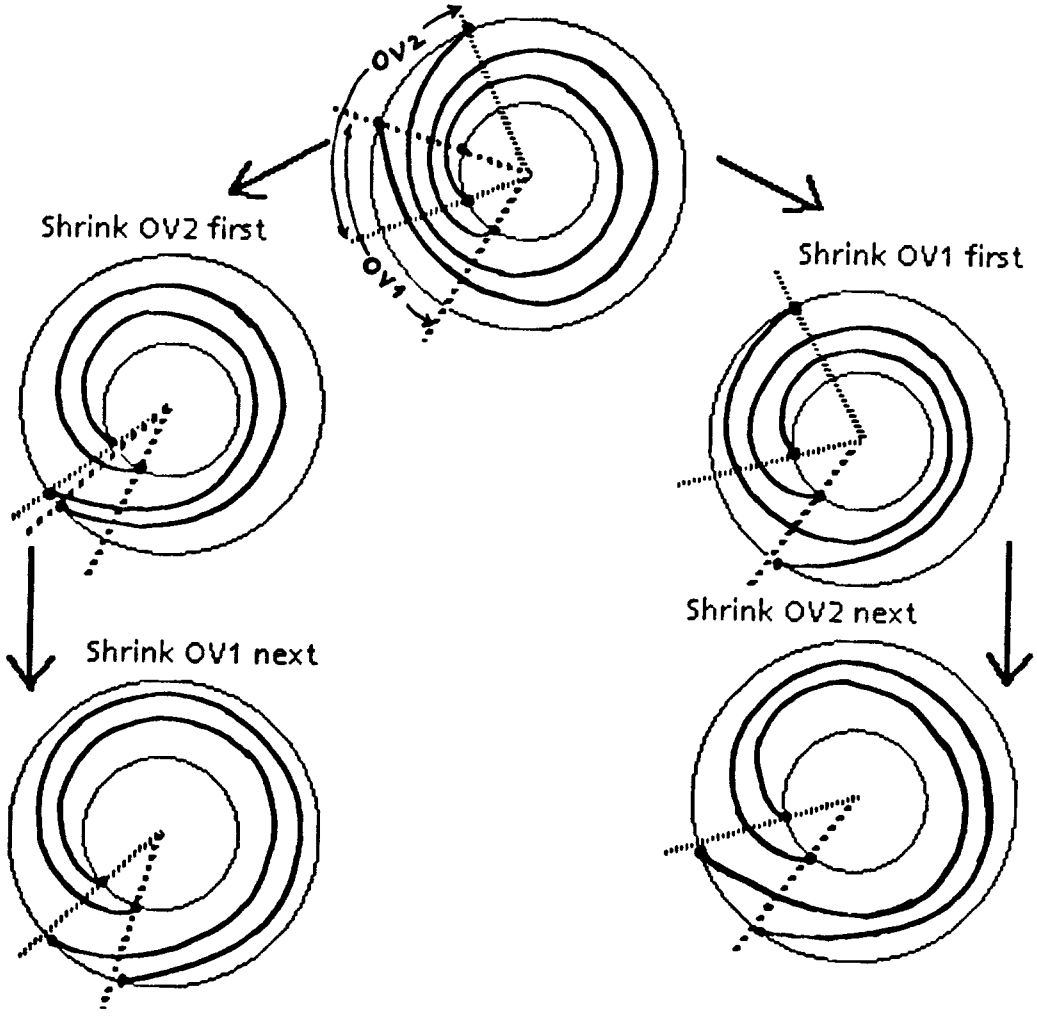


Fig. 3.8. OV_1 **and** OV_2 **have a Non-empty Intersection.**

By applying Procedure 3.1 in reverse order, we can convert this CPD into a family of arcs on a circle. This family of arcs may be further converted into a canonical family as shown in Section 2.2. Thus, we have established the following :

Theorem 3.2. *Every CPG is a CACG.* □

Remark 3.2. In the above conversion, Step 1 takes $O(n)$ time, since it involves shifting all the endpoints of chords on C_q. Step 2 needs $O(n)$ time to shrink each XL-chord, and there are at most n XL-chords. Thus it requires a total of $O(n^2)$ time. Finally, Procedure 3.1 in reverse order takes $O(n)$ time to execute. Conversion of the family of arcs into a canonical family takes $O(nlogn)$ time as shown in Section 2.2. Therefore, the conversion of a CPD to a canonical family of arcs on a circle can be done in $O(n^2)$ time. □

3.4. Summary

We have shown the equivalence of a CACG and a CPG by a constructive proof. Since Rotem and Urrutia [15] have given an $O(n^2)$ recognition algorithm for a CPG, a CACG can be recognized in $O(n^2)$ time. The algorithms that we present later in this thesis are all applicable to a CPG.

Chapter 4

Maximum Clique of a CACG

4.1. Overview

In this chapter, we show that a maximum arc containment family of a family A of arcs on a circle, or equivalently a maximum clique of the CACG G_A, can be found in polynomial time. Our approach is to open out the arc family A into a family I of intervals on the real line in a special way. We show that a maximum clique of the CACG G_A can be obtained by finding a maximum clique of the ICG G_I.

4.2. Conversion of a Family of Arcs into a Family of Intervals

Let $A = \{A_1, A_2, \ldots, A_n\}$ be a family of n arcs on a circle. We assume that A is given in canonical form, for if not, we can always convert it into its canonical form by Procedure 2.1 in $O(nlogn)$ time. We transform the family A of arcs into a family I of intervals on the real line in the following manner :

1. For each backward arc $A_j = [h_j, t_j]$, where $h_j > t_j$, create an interval of type *back*, denoted by $I_j^b = [h_j, t_j + 2n]$.

2. For each forward arc $A_k = [h_k, t_k]$, where $h_k < t_k$, create two intervals of types fwd and $fwd - dup$, denoted by
   $I_k^f = [h_k, t_k]$ and $I_k^{fd} = [h_k + 2n, t_k + 2n]$, respectively.

Thus, if there are b backward arcs and f forward arcs in A ($n = b + f$), we have a total of $m = 2f + b$ intervals in I. $2f$ of these intervals correspond to the f forward arcs and b of the intervals correspond to the b backward arcs.

4.3. Maximum Interval Containment Family of a Family of Intervals

Given a family I of intervals on the real line, Masuda and Nakajima [12] presented a method for finding a maximum interval containment family of I, or equivalently a maximum clique of the ICG G_I. It utilizes the Fredman algorithm for finding a longest increasing subsequence of a sequence of integers [4]. We briefly outline their method here.

Algorithm 2 [12].

Input : A family $I = \{I_1, \ldots, I_n\}$ of n closed intervals, in canonical form, with each interval represented by an ordered pair $I_k = [h_k, t_k]$.

Output : A maximum interval containment family of I.

Method :

1. Renumber the intervals to get a family $I' = \{I'_1, I'_2, \ldots, I'_n\}$ such that $h'_1 < h'_2 < \ldots < h'_n$.

2. Construct a sequence of indices of the intervals $S = [i_1, i_2, \ldots, i_n]$ such that $t'_{i_1} > t'_{i_2} > \ldots > t'_{i_n}$.

3. Find a longest increasing subsequence X of S by using Algorithm 1 of Section 2.4.

4. Output the set of intervals I^X which corresponds to X. □

Masuda and Nakajima [12] showed that the above algorithm runs in $O(nlogn)$ time. The sorting at Step 2 can be done in $O(n)$ time by bucket sort [1] because all co-ordinates are integers 1 through $2n$. In Step 3, Algorithm 1 of Section 2.4 can find I^X in $O(nlogn)$ time. They also gave a proof that I^X is indeed a maximum interval containment family of I.

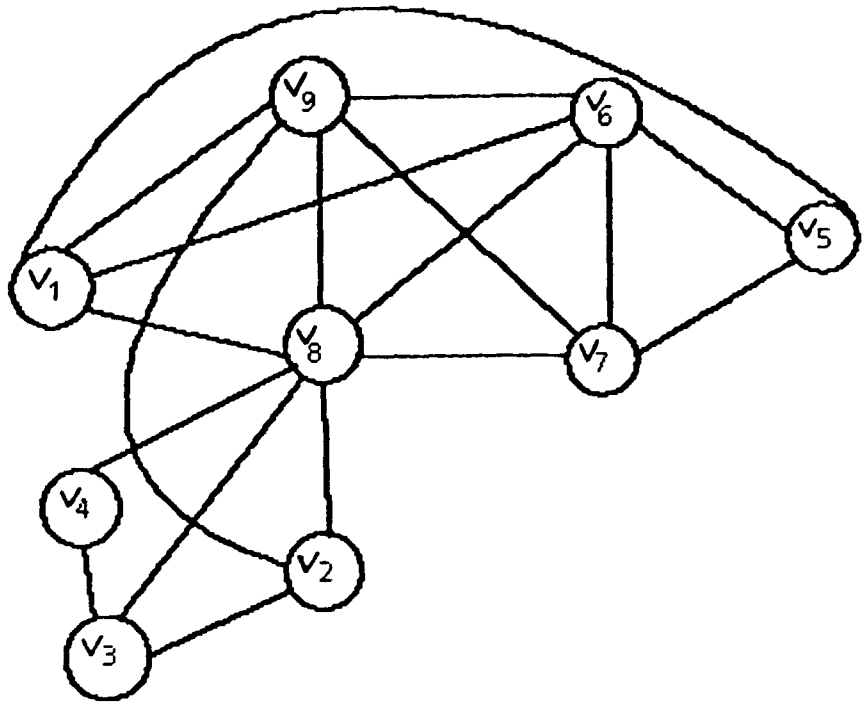


Fig. 4.1. CACG for the family A^c of Fig. 2.3.

4.4. Maximum Arc Containment Family of a Family of Arcs on a Circle

We now describe an algorithm for finding a maximum arc containment family of a given family of arcs on a circle.

Algorithm 3.

Input : A family $A = \{A_1, \ldots, A_n\}$ of n arcs on a circle, in canonical form, with each arc represented by an ordered pair $A_k = [h_k, t_k]$.

Output : A maximum arc containment family of A.

Method :

1. Convert the $n = b + f$ arcs in A into $m = b + 2f$ intervals in I on the real line as described in Section 4.2. Renumber the m intervals $I_1, \ldots, I_m$

in ascending order of their head co-ordinates. For every backward arc A_j, for which the interval I_j^b is assigned the index l, create an ordered pair (j, l). For every forward arc A_j, for which the two intervals I_j^f and I_j^{fd} are assigned indices k^f and k^{fd} respectively, create two ordered pairs (j, k^f) and (j, k^{fd}). Let T be the set of ordered pairs thus created.

2. Proceed to Step 3 of Algorithm 2 and find a set X of indices of the intervals which corresponds to a maximum interval containment family of I.

3. Form the set Z of arcs A_z such that

$$Z = \{A_z \mid (z, x) \in T \wedge x \in X\}.$$ □

We prove that the set Z is a maximum arc containment family of A. Fig. 4.1 shows the CACG G_{A^c} for the canonical family A^c shown in Fig. 2.3. Fig. 4.2 shows the conversion of the family A^c to a family I of intervals. The sequence S of Algorithm 2 and the maximum arc containment family obtained by Algorithm 3 are depicted in Fig. 4.3.

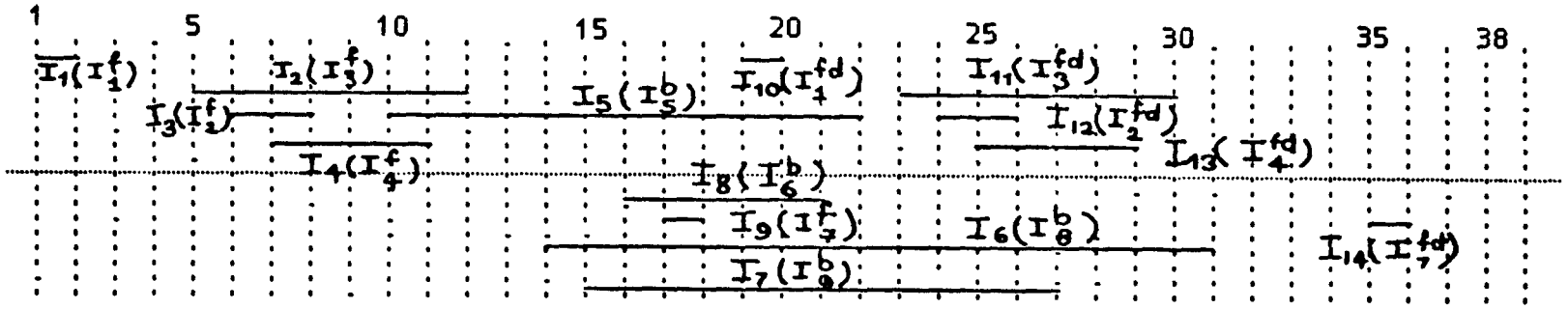
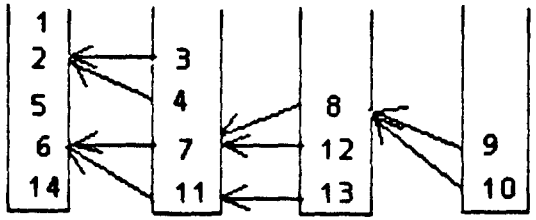


Fig. 4.2. Conversion of A^c into a Family of Intervals

$S = <\ 14, 6, 11, 13, 7, 12, 5, 8, 10, 9, 2, 4, 3, 1\ >$



Maximum Interval Containment Family =

{I6, I7, I8, I9}

Maximum Arc Containment Family =

{A8, A9, A6, A7}

Fig. 4.3. Illustration of the Execution of Algorithm 3

Lemma 4.1. *Every interval containment family of I corresponds to an arc containment family of A.*

Proof: Every interval in I is derived from a unique arc in the family A of arcs. Let I_p and I_q be two intervals in I. I_p and I_q are derived from arcs $A_i = [h_i, t_i]$ and $A_j = [h_j, t_j]$, respectively, for some $i, j \in \{1, 2, \ldots, n\}$. Suppose that one of the two intervals I_p and I_q contains the other, say $I_p \supset I_q$. Note that since I_p contains I_q, it is not possible that A_i is a forward arc and A_j is a backward arc. There are three cases to consider.

1. Both I_p and I_q are derived from backward arcs. In this case, $I_p = [h_i, 2n + t_i]$ and $I_q = [h_j, 2n + t_j]$. If $I_p \supset I_q$, then $h_i < h_j$ and $t_i > t_j$, and hence $A_i \supset A_j$.

2. Both I_p and I_q are derived from forward arcs. In this case, since $I_p \supset I_q$, either $I_p = [h_i, t_i]$ and $I_q = [h_j, t_j]$ or $I_p = [h_i + 2n, t_i + 2n]$ and $I_q = [h_j + 2n, t_j + 2n]$. In either case, $h_i < h_j$ and $t_i > t_j$, and hence $A_i \supset A_j$.

3. I_p is derived from a backward arc and I_q from a forward arc. In this case, $I_p = [h_i, 2n + t_i]$ and either $I_q = [h_j, t_j]$ or $I_q = [h_j + 2n, t_j + 2n]$. In either event, since I_p contains I_q, $A_i \supset A_j$.

In all three cases, if $I_p \supset I_q$, then $A_i \supset A_j$. Since this is true for every pair of intervals in I, an interval containment family of I corresponds to an arc containment family of A. □

Lemma 4.2. *Every arc containment family of A corresponds to one or two interval containment families of I.*

Proof: Let $A^Q = \{A_{j_1}, \ldots, A_{j_l}\}$ be an arc containment family of A. Without loss of generality, we can assume that $A_{j_1} \supset A_{j_2} \supset \ldots \supset A_{j_l}$.

There are three cases to consider :

1. A^Q consists of backward arcs only.

2. A^Q consists of forward arcs only. In this case there are two interval containment families of I which correspond to A^Q.

3. A^Q consists of both forward and backward arcs.

In Case 1, since $h_{j_i} < h_{j_k}$ and $t_{j_i} > t_{j_k}$, for $1 \leq i < k \leq l$, $h_{j_i} < h_{j_k}$ and $t_{j_i} + 2n > t_{j_k} + 2n$ for $1 \leq i < k \leq l$, $I^b_{j_1} \supset I^b_{j_2} \supset \ldots \supset I^b_{j_l}$. Therefore, an interval containment family of the same cardinality l exists in I.

In Case 2, since $h_{j_i} < h_{j_k}$ and $t_{j_i} > t_{j_k}$, and $h_{j_i} + 2n < h_{j_k} + 2n$ and $t_{j_i} + 2n > t_{j_k} + 2n$ for $1 \leq i < k \leq l$, $I^f_{j_1} \supset I^f_{j_2} \supset \ldots \supset I^f_{j_l}$ and also, $I^{fd}_{j_1} \supset I^{fd}_{j_2} \supset \ldots \supset I^{fd}_{j_l}$, respectively. Thus two interval containment families of cardinality l exist in I.

As for Case 3, let A^Q consist of s backward arcs and r forward arcs. Since no forward arc contains a backward arc, it is clear that $A_{j_1}, \ldots, A_{j_s}$ are backward arcs and $A_{j_{s+1}}, \ldots, A_{j_{s+r}}$, where $s + r = l$, are forward arcs. Then we have either

(a) $t_{j_s} < \ldots < t_{j_1} < h_{j_1} < \ldots < h_{j_s} < h_{j_{s+1}} < \ldots < h_{j_{s+r}} < t_{j_{s+r}} < \ldots < t_{j_{s+1}}$

or

(b) $h_{j_{s+1}} < \ldots < h_{j_{s+r}} < t_{j_{s+r}} < \ldots < t_{j_{s+1}} < t_{j_s} < \ldots < t_{j_1} < h_{j_1} < \ldots < h_{j_s}$

If (a) is true, then $I^b_{j_1} \supset I^b_{j_2} \supset \ldots \supset I^b_{j_s} \supset I^f_{j_{s+1}} \supset \ldots I^f_{j_{s+r}}$.

If (b) is true, then $I^b_{j_1} \supset I^b_{j_2} \supset \ldots \supset I^b_{j_s} \supset I^{fd}_{j_{s+1}} \supset \ldots I^{fd}_{j_{s+r}}$.

In either case, l intervals also form an interval containment family of I.

Thus every arc containment family of A corresponds to one or two interval containment families of I. □

Theorem 4.1. *Let Z be an output of Algorithm 3. Z is a maximum arc containment family of A.*

Proof : Let $I^Q{}_m$ be a maximum interval containment family of I obtained by Algorithm 2. After the execution of Algorithm 3 we have

$Z = A^Q{}_m = \{A_p \mid (p,c) \in T \wedge I_c \in I^Q{}_m\}$.

By Lemma 4.1 $A^Q{}_m$ is an arc containment family of A and $\mid I^Q{}_m \mid = \mid A^Q{}_m \mid$. We now prove that $A^Q{}_m$ is a *maximum* arc containment family of A.

Suppose that it is not a maximum. Then there is a proper superset $A^Q{}_{m'}$ of $A^Q{}_m$ that is also an arc containment family of A. By Lemma 4.2 $A^Q{}_{m'}$ corresponds to at least one interval containment family $I^Q{}_{m'}$ of I such that $\mid A^Q{}_{m'} \mid = \mid I^Q{}_{m'} \mid$. As a consequence, $\mid I^Q{}_{m'} \mid > \mid I^Q{}_m \mid$. Hence $I^Q{}_m$ can not be a maximum interval containment family of I, a contradiction. Therefore, the set $Z = A^Q{}_m$ obtained by Algorithm 3 is indeed a maximum arc containment family of A. □

The conversion of the family A of arcs on a circle to a family I of intervals on the real line in Step 1 of Algorithm 3 takes $O(m)$ time. Since $m \leq 2n$, it is $O(n)$. In Step 2 we essentially execute Algorithm 2 and it takes $O(mlogm) = O(nlogn)$ time. Step 3 is a lookup from the set T and requires $O(n)$ time. Thus, Algorithm 3 takes $O(nlogn)$ time to compute a maximum arc containment family of A. Therefore, we have the following theorem :

Theorem 4.2. Given a family A of n arcs on a circle, Algorithm 3 finds, in $O(nlogn)$ time, a maximum arc containment family of A, or equivalently a maximum clique of the CACG G_A. □

Remark 4.1. It was shown by Masuda and Nakajima [12] that $O(nlogn)$ is the lower time bound for finding a maximum clique of an ICG. Since every ICG is a CACG, Algorithm 3 is optimal to within a constant factor. □

4.5. Summary

In this chapter we have developed an optimal $O(nlogn)$ algorithm to find a maximum arc containment family of a family A of arcs on a circle. The algorithm opens out the family A of arcs into a family I of intervals on the real line and in turn finds a maximum interval containment family of I. In Theorem 4.1 we proved that it suffices to find a maximum interval containment family of I in order to find a maximum arc containment family of A.

Chapter 5

Maximum Independent Set of a CACG

5.1. Overview

In this chapter we present a polynomial time algorithm for finding a maximum independent set of a CACG. Given a family A of n arcs, our algorithm finds a maximum independent arc family of A in $O(\delta n \log n + e)$ time, where δ is the minimum vertex degree for the corresponding CACG $G_A = (V_A, E_A)$ and $e = \mid E_A \mid$. We first describe how to find in $O(nlogn)$ time a maximum independent arc family containing a given arc. We then use the result to obtain a maximum independent arc family of A.

5.2. Maximum Independent Arc Family Containing a Given Arc

Let A_k be an arc in a given family A of arcs. We denote by Y_{A_k}, a maximum independent arc family of A which contains A_k. Consider an arc A_k and other arcs relative to it as shown in Fig. 5.1. Those arcs that are either contained in A_k or that contain A_k are not independent of A_k and can clearly be disregarded in the computation of Y_{A_k}.

All the other arcs on the circle can be classified into the following categories :

1. Arcs such as A_h where only the head h_h is contained in A_k.
2. Arcs such as A_t where only the tail t_t is contained in A_k.
3. Arcs such as A_d which are entirely disjoint from A_k.
4. Arcs such as A_s where the head h_s and the tail t_s are both contained in A_k but the arc A_s itself is not contained in A_k.

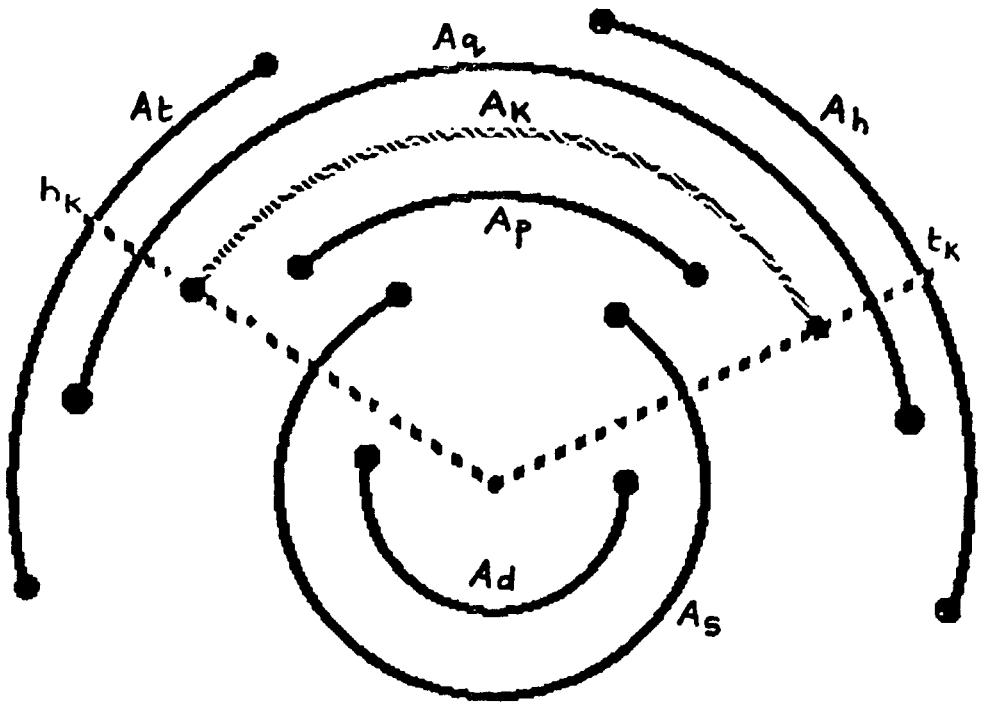


Fig. 5.1. An Arc A_k and Other Arcs with Respect to It

Notice that after those arcs that are not independent of A_k are removed, the circle can be *split open*, in $O(n)$ time, by the steps outlined below, transforming each of the remaining arcs into an interval so that :

1. The tail of arc A_k becomes the starting reference, and the head of A_k becomes the ending reference.

2. All arcs such as A_h are opened with their heads to the left of the starting reference and all arcs such as A_t are opened with their tails to the right of the ending reference. Arcs such as A_d appear between the reference marks while arcs such as A_s are stretched out with their heads to the left of the starting reference and their tails to the right of the ending

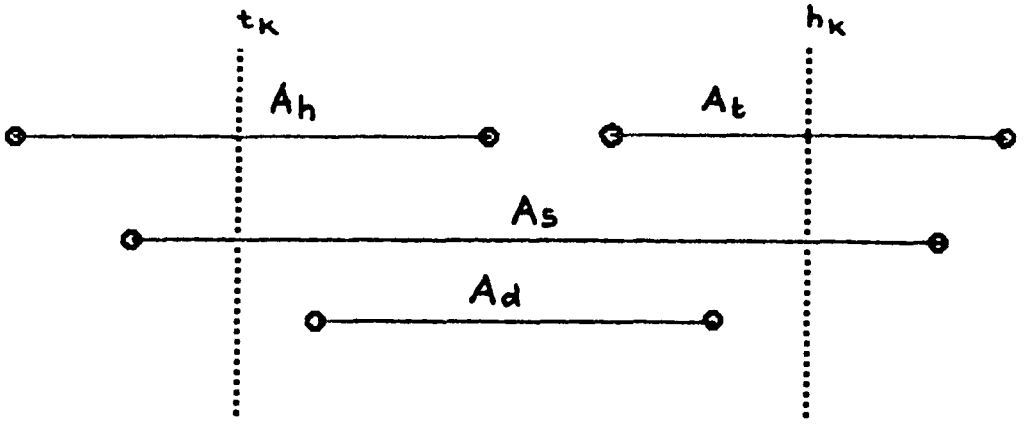


Fig. 5.2. Splitting the Circle Open

reference.

The above operations result in a family of closed intervals along the real line, derived from those arcs that are independent of A_k. The circle split open is shown in Fig. 5.2. We denote this family of intervals by I^k.

Masuda and Nakajima [12] have developed an algorithm for finding a maximum independent interval family in $O(nlogn)$ time, given a family of n intervals on the real line. Their algorithm is based on Algorithm 1 for obtaining a longest increasing subsequence of a sequence of integers. We can obtain a maximum independent interval family of I^k in $O(nlogn)$ time using Masuda and Nakajima's algorithm. Let it be denoted by Y_{I^k}. Let Y^k be the family of arcs in A that correspond to the family Y_{I^k}. Since every arc which corresponds to an interval in I^k is independent of A_k by our construction, a maximum independent arc family of A containing the arc A_k is given by

$$Y_{A_k} = Y^k \cup \{A_k\}.$$

5.3. Maximum Independent Arc Family of A

We can obtain $G_A = (V_A, E_A)$ from a family of n arcs A on a circle in $O(n+e)$ time, by using Procedure 2.2, where $e = \mid E_A \mid$. Let v_α be a vertex of G_A with the minimum degree δ. Let A_α be the arc corresponding to v_α. δ is thus the cardinality of $Adj_{G_A}(v_\alpha)$, or equivalently the number of those arcs that are not independent of A_α.

Let Y be a maximum independent arc family of A. There are two possibilities :

1. Y contains the arc A_α in it, that is, $Y = Y_{A_\alpha}$.

2. Y does not contain the arc A_α.

As we have shown in the preceding section, we can find Y_{A_α} in $O(nlogn)$ time. Therefore, in Case (1) a maximum independent arc family Y of A can be computed in $O(nlogn)$ time.

Now consider the second case. Y does not contain A_α. Therefore, it must contain at least one of the arcs that are not independent of A_α. If this were not the case, all arcs in Y would be independent of A_α and we could easily add the arc A_α itself to the family Y to obtain a larger family, thus contradicting the maximality of Y. Thus, in the second case, Y must contain at least one of the δ arcs that are not independent of A_α. For each of these arcs $A_{i_1}, A_{i_2}, \ldots, A_{i_k}, \ldots, A_{i_\delta}$, we can find a maximum independent arc family $Y_{A_{i_k}}$ in $O(nlogn)$ time by splitting open the circle as shown earlier.

A maximum independent arc family Y is then given by Y_{A_γ} where $\gamma \in \{\alpha, i_1, \ldots, i_\delta\}$ and

$\mid Y_{A_\gamma} \mid = \max \{ \mid Y_{A_\alpha} \mid, \mid Y_{A_{i_1}} \mid, \ldots, \mid Y_{A_{i_\delta}} \mid \}$.

The above approach thus takes a total of $O(\delta nlogn)$ time. Given a family

A of arcs on a circle, it takes $O(nlogn + e)$ time to obtain G_A and determine a minimum degree vertex v_α. Thus the entire maximum independent arc family algorithm runs in $O(\delta nlogn + e)$ time. We have the following theorem :

Theorem 5.1. *Given a family A of n arcs on a circle, one can find, in $O(\delta nlogn + e)$ time, a maximum independent arc family of A, or equivalently, a maximum independent set of the corresponding CACG $G_A = (V_A, E_A)$ where $e = \mid E_A \mid$ and δ is the minimum vertex degree of G_A.* □

5.4. Summary

In this chapter we have presented an algorithm to find a maximum independent set of a CACG. When the graph is given in the form of its corresponding family A of arcs, the algorithm runs in $(\delta nlogn + e)$ time where δ is the minimum vertex degree of G_A, and e is the number of edges in G_A.

Chapter 6

Minimum Coloring of a CACG

6.1. Overview

A graph $G = (V, E)$ is called a *comparability graph* if its edges can be oriented in such a way that for $x, y, z \in V$, $< x, y > \in E$ and $< y, z > \in E$, imply $< x, z > \in E$, where $< x, y >$ denotes an ordered pair of vertices x and y. It is easy to show that a CACG is a comparability graph. It is known that the cardinality of a maximum clique of a comparability graph is the same as the minimum number of colors required to color the graph [8]. Therefore, the problem of finding a minimum coloring of a CACG is closely related to the problem of finding a maximum clique of the graph.

We have already described an optimal $O(nlogn)$ algorithm to find a maximum clique of a CACG in Chapter 4. In this chapter we present an efficient algorithm to find a minimum coloring of a CACG, or more precisely, a minimum coloring of its corresponding family of arcs. The technique we employ here is similar to that used in the algorithm to find a maximum arc containment family. We open out a family A of arcs on a circle into a family I of intervals along the real line in exactly the same manner as described in Section 4.2. We first find a minimum coloring of I. We then derive a coloring of A from that of I by way of a special auxiliary graph.

6.2. Minimum Coloring of I and A

Masuda and Nakajima [12] showed that a minimum coloring for a family I of intervals on the real line is obtained in the following manner. Applying Steps 1 and 2 of Algorithm 2 to I, we generate a set of ω stacks,

$\{ST_1, \ldots, ST_\omega\}$. We then assign color c_i to the intervals whose indices lie in stack ST_i, for $i = 1, \ldots, n$. We have already seen in Chapter 4 how a maximum interval containment family of I stretches horizontally across these ω stacks. Since all the intervals whose indices lie in the same stack are independent of each other, the above procedure produces a minimum coloring of I. Let $color(I_j)$ denote the color c_i assigned in this way, to interval I_j.

Let $\omega(G)$ (resp., $\chi(G)$) denote the cardinality of a maximum clique (resp., minimum coloring) of a graph G. Since G_I and G_A are both comparability graphs, $\omega(G_I) = \chi(G_I)$ and $\omega(G_A) = \chi(G_A)$. In obtaining I from the family A of arcs we duplicate the forward arcs. I has $m = b + 2f$ intervals, while A has $n = b + f$ arcs. We have already shown in Chapter 4 that $\omega(G_I) = \omega(G_A)$. Therefore, the size of a minimum coloring of I is the same as that of a minimum coloring of A.

In the remainder of this chapter we show that we can find a minimum coloring of A from that of I. The basic idea is as follows. A backward arc A_p is assigned a unique color from its corresponding interval I_p. However, for a forward arc A_q, there may be a choice between $color(I_q^f)$ and $color(I_q^{fd})$. We claim that exactly one of the colors can be selected to color every forward arc A_q without creating a conflict. To make such a selection we use an auxiliary graph.

6.3. Construction of an Auxiliary Graph $G^\star$

From the family I of intervals and its coloring, we construct an auxiliary graph $G^\star = (V^\star, E^\star)$ in the manner described below. With each vertex $v \in V^\star$ we associate a property called $type(v)$ derived from the type of the interval which it represents.

1. $V^\star$ consists of $b + 2f$ vertices.

We have a vertex N_k^b for each interval I_k^b, with $type(N_k^b) = back$. We have vertices N_k^f and N_k^{fd}, respectively, for each pair of intervals I_k^f and I_k^{fd}, with $type(I_k^f) = fwd$ and $type(I_k^{fd}) = fwd - dup$. There are b vertices of type $back$ and f vertices each of type fwd and $fwd - dup$ in $G^\star$.

2. $E^\star$ consists of two kinds of edges.

   (a) For every pair of vertices N_k^f and N_k^{fd}, we create an edge(N_k^f, N_k^{fd}) in $E^\star$.

   (b) The remaining edges are created as follows: :

      i. $(N_j^b, N_k^f) \in E^\star$ if $color(I_j^b) = color(I_k^f)$ and $I_j^b \supset I_k^{fd}$.

      ii. $(N_j^b, N_k^{fd}) \in E^\star$ if $color(I_j^b) = color(I_k^{fd})$ and $I_j^b \supset I_k^f$.

      iii. $(N_j^f, N_k^{fd}) \in E^\star$ if $color(I_j^f) = color(I_k^{fd})$ and $either I_j^f \supset I_k^f$ or $I_j^f \subset I_k^f$. □

Fig. 6.1 shows an example of an auxiliary graph $G^\star$ obtained from the family of intervals I shown in Fig. 4.2. Fig. 4.3 showed the stacks which correspond to a coloring of I.

Note that $G^\star$ does not contain any edge whose endpoints are of the same type. Namely, $(N_j^b, N_k^b) \notin E^\star$, $(N_j^f, N_k^f) \notin E^\star$, and $(N_j^{fd}, N_k^{fd}) \notin E^\star$ for any $j, k \in \{1, \ldots, n\}$.

6.4. Properties of $G^\star$

In this section we study some properties of the auxiliary graph $G^\star = (V^\star, E^\star)$ constructed as above. We show that $G^\star$ does not contain any cycle. Using this property, we can partition the set $V^\star$ of vertices into two mutually disjoint independent sets $Y^\star$ and $Z^\star$ such that $Y^\star$ contains all the b vertices

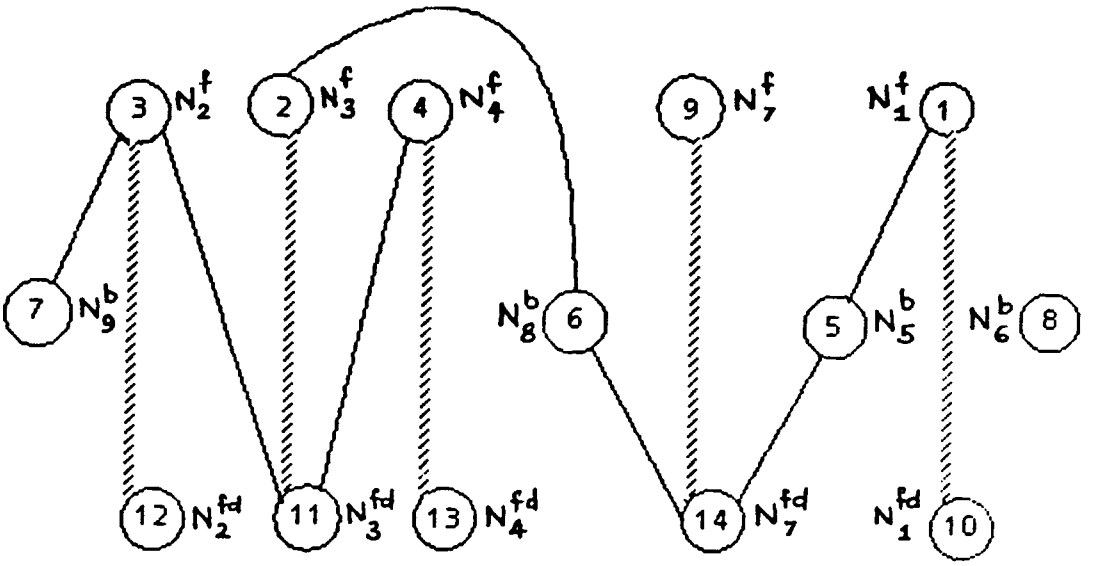


Fig. 6.1. An Example of an Auxiliary Graph G^*.

of type *back* and f vertices of type fwd and/or $fwd - dup$ while Z^* contains f vertices of type fwd and/or $fwd - dup$ only. We begin with the following lemma :

Lemma 6.1. *Let a, b, c and d be four numbers which appear in a sequence S in this order, such that $a < b$ and $c < d$. When Algorithm 1 is applied to S, it is never possible that a and d occur in the same stack and simultaneously b and c appear in the same stack.*

Proof : Suppose that a appears in stack ST_i. Since $b > a$, b appears in stack ST_j, where $j > i$. c appears in stack ST_j if and only if the top element of each stack ST_k, for $k = 1, 2, \ldots, j-1$, is less than c. Since $c < d$, d can never appear in stack ST_k with $k < j$. Thus d can never appear in ST_i. □

We now proceed to describe and justify some properties of $G^* = (V^*, E^*)$.

Without loss of generality, we can assume that the intervals in I are already sorted in the ascending order of their head co-ordinates.

Property 6.1. If $(N_k^f, N_j^b) \in E^\star$, then $(N_k^{fd}, N_l^b) \notin E^\star$, and if $(N_k^{fd}, N_j^b) \in E^\star$, then $(N_k^f, N_l^b) \notin E^\star$, for any $j, k, l \in \{1, \ldots, n\}$.

Proof : Suppose that $(N_k^f, N_j^b) \in E^\star$ and $(N_k^{fd}, N_l^b) \in E^\star$ for some $j, k, l \in \{1, \ldots, n\}$, as shown in Fig. 6.2(a). By construction of $G^\star$, $I_j^b \supset I_k^{fd}$, $I_l^b \supset I_k^{fd}$, $color(I_j^b) = color(I_k^f)$, and $color(I_l^b) = color(I_k^{fd})$. Consider the sequence S obtained during the process of obtaining a minimum coloring of I. The four indices j, k^{fd}, l and k^f must appear in this order in S. The hypothetical situation shown in Fig. 6.2(a) holds only if k^f and j appear in the same stack and k^{fd} and l appear in the same stack. Clearly this is impossible due to Lemma 6.1. □

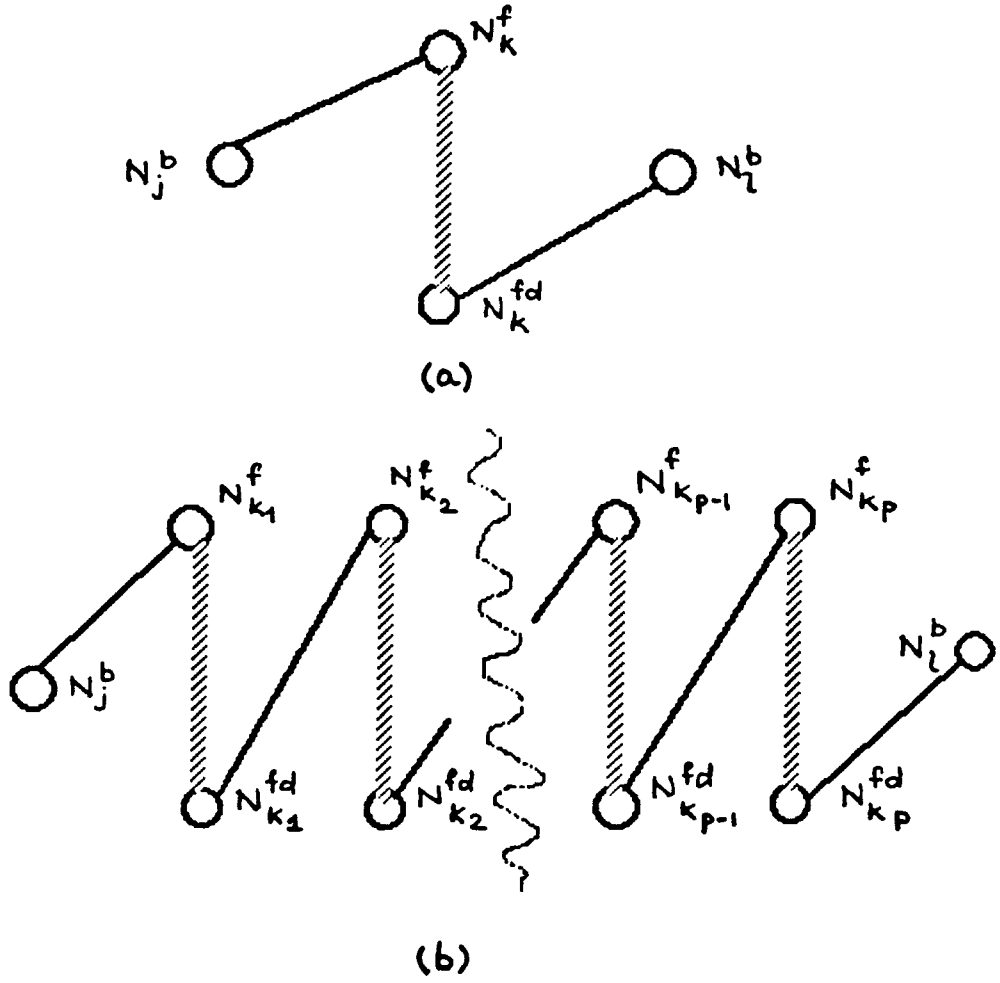


Fig. 6.2. Property 6.1.

A generalized version of Property 6.1 is :

Property 6.1.G. If $(N_j^b, N_{k_1}^f) \in E^\star$, and there exists a path $< (N_{k_1}^f, N_{k_1}^{fd}), (N_{k_1}^{fd}, N_{k_2}^f), \ldots, (N_{k_{p-1}}^{fd}, N_{k_p}^f), (N_{k_p}^f, N_{k_p}^{fd}) >$ in $G^\star$, then $(N_{k_p}^{fd}, N_l^b) \notin E^\star$ for any $j, k_1, \ldots, k_p, l \in \{1, \ldots, n\}$.

Proof : Suppose that $(N_j^b, N_{k_1}^f) \in E^\star$, $< (N_{k_1}^f, N_{k_1}^{fd}), (N_{k_1}^{fd}, N_{k_2}^f), \ldots, (N_{k_{p-1}}^{fd}, N_{k_p}^f), (N_{k_p}^f, N_{k_p}^{fd}) >$ is a path in $G^\star$, and $(N_{k_p}^{fd}, N_l^b) \in E^\star$ for some $j, k_1, \ldots, k_p, l \in \{1, \ldots, n\}$ as shown in Fig. 6.2(b). From the construction of $G^\star$, $color(I_{k_q}^{fd}) = color(I_{k_{q+1}}^f)$ for $q = 1, \ldots, p-1$. Furthermore, it can be easily shown that $I_{k_1}^f \subset I_{k_2}^f \subset \ldots I_{k_{p-1}}^f \subset I_{k_p}^f$. Hence ${k_1}^f > {k_2}^f > \ldots > {k_p}^f$ and ${k_1}^{fd} > {k_2}^{fd} > \ldots > {k_p}^{fd}$. Also, $j < {k_1}^{fd}$. The four numbers $j, {k_1}^{fd}, {k_2}^f$ and ${k_1}^f$ occur in this order in S. By Lemma 6.1 it is not possible that j and ${k_1}^f$ appear in the same stack as well as ${k_2}^f$ and ${k_1}^{fd}$ appear in the same stack. Hence, the hypothetical situation shown in Fig. 6.2(b) does not hold. □

Property 6.2. If $(N_k^b, N_j^{fd}) \in E^\star$ and $(N_k^b, N_l^f) \in E^\star$, then $(N_j^f, N_l^{fd}) \notin E^\star$, for any $j, k, l \in \{1, \ldots, n\}$.

Proof : Suppose that $(N_k^b, N_j^{fd}) \in E^\star$, and $(N_k^b, N_l^f) \in E^\star$, for some $j, k, l \in \{1, \ldots, n\}$ as shown in Fig. 6.3(a). Without loss of generality we can assume that $j < k < l$. Evidently, $0 < h_k^b < h_j^f < t_j^f < 2n < h_l^{fd} < t_l^{fd} < t_k^b$. Hence, I_j^f and I_l^{fd} have no region in common, and by construction of $G^\star$, $(N_j^f, N_l^{fd}) \notin E^\star$. □

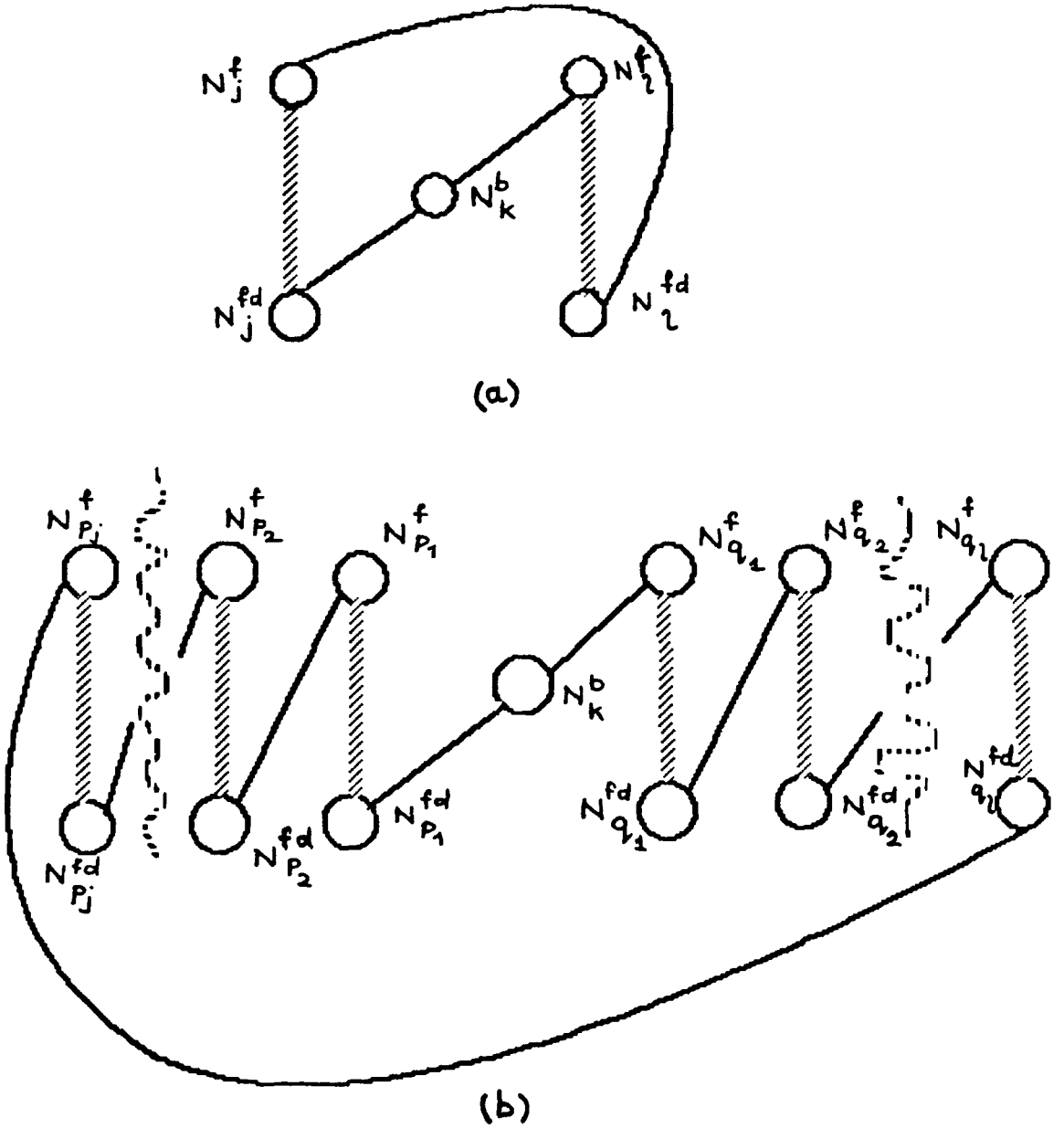


Fig. 6.3. Property 6.2.

A generalized version of Property 6.2 is :

Property 6.2.G If $(N_k^b, N_{p_1}^{fd}) \in E^\star$, $(N_k^b, N_{q_1}^f) \in E^\star$, $< (N_{p_1}^{fd}, N_{p_1}^f)$, $(N_{p_1}^f, N_{p_2}^{fd})$, ..., $(N_{p_{j-1}}^f, N_{p_j}^{fd})$, $(N_{p_j}^{fd}, N_{p_j}^f) >$ and $< (N_{q_1}^f, N_{q_1}^{fd})$, $(N_{q_1}^{fd}, N_{q_2}^f)$, ..., $(N_{q_{l-1}}^{fd}, N_{q_l}^f)$, $(N_{q_l}^f, N_{q_l}^{fd})$, $>$ are paths in $G^\star$, for any $k, p_1, \ldots, p_j, q_1, \ldots, q_l \in \{1, \ldots, n\}$ then $(N_{p_j}^f, N_{q_l}^{fd}) \notin E^\star$.

Proof : Suppose that $(N_k^b, N_{p_1}^{fd}) \in E^\star$, $(N_k^b, N_{q_1}^f) \in E^\star$, and $< (N_{p_1}^{fd}, N_{p_1}^f)$, $(N_{p_1}^f, N_{p_2}^{fd})$, ..., $(N_{p_{j-1}}^f, N_{p_j}^{fd})$, $(N_{p_j}^{fd}, N_{p_j}^f) >$ and $< (N_{q_1}^f, N_{q_1}^{fd})$, $(N_{q_1}^{fd}, N_{q_2}^f)$, ..., $(N_{q_{l-1}}^{fd}, N_{q_l}^f)$, $(N_{q_l}^f, N_{q_l}^{fd})$, $>$ are paths in $G^\star$, as shown in Fig. 6.3(b). $\{I_{p_1}^f, \ldots I_{p_j}^f\}$ is an interval containment family and so do $\{I_{q_1}^f, \ldots I_{q_l}^f\}$. By an argument similar to that used in the proof of Property 6.2, we can conclude that $(N_{p_j}^f, N_{q_l}^{fd}) \notin E^\star$. □

Property 6.3. If $(N_j^f, N_k^{fd}) \in E^\star$, then $(N_j^{fd}, N_k^f) \notin E^\star$, for any $j, k \in \{1, \ldots, n\}$.

Proof : Suppose that $(N_j^f, N_k^{fd}) \in E^\star$ and $(N_j^{fd}, N_k^f) \in E^\star$, for some $j, k \in \{1, \ldots, n\}$ as shown in Fig. 6.4(a). $(I_j^f, I_k^f) \in E_I$. Without loss of generality we can assume that $I_j^f \subset I_k^f$. The indices k^{fd}, j^{fd}, k^f, j^f must occur in this order in the sequence S obtained during the coloring step of I. Since $k^{fd} < j^{fd}$, and $k^f < j^f$, by Lemma 6.1, we can conclude that the hypothetical situation shown in Fig. 6.4(a),wherein k^{fd} and j^f appear in the same stack and j^{fd} and k^f appear in the same stack is not possible. □

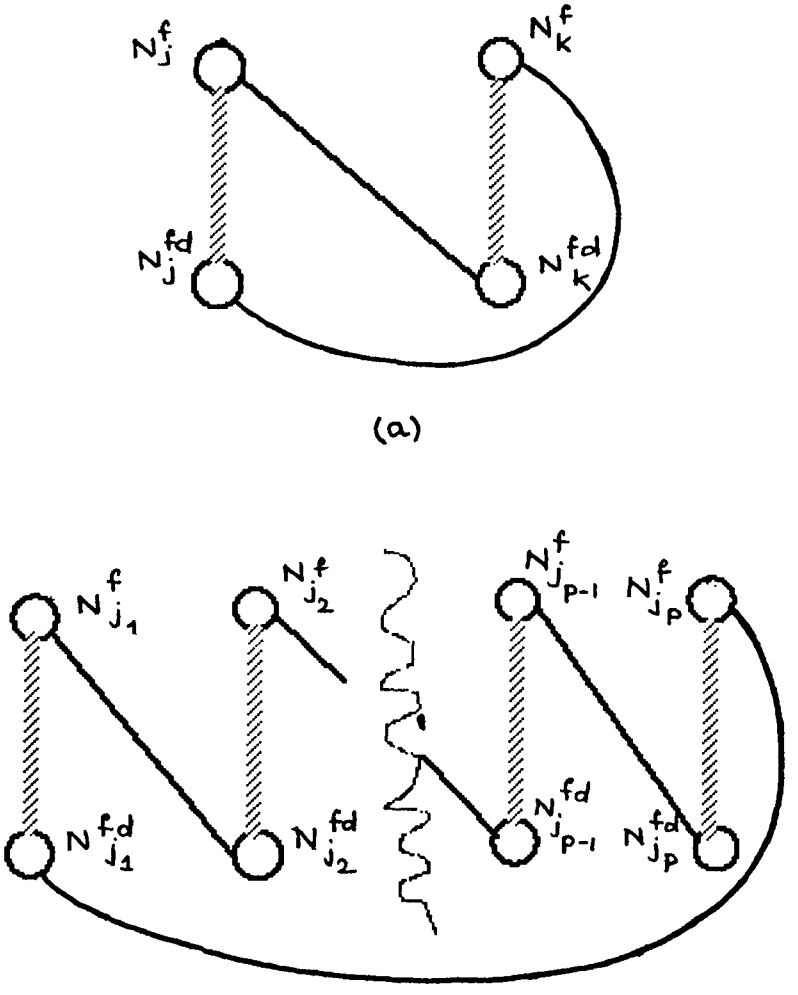


Fig. 6.4. Property 6.3.

A generalized version of Property 6.3 is :

Property 6.3.G If $< (N^f_{j_1}, N^{fd}_{j_2}), (N^{fd}_{j_2}, N^f_{j_2}), \ldots, (N^f_{j_{p-1}}, N^{fd}_{j_p}), (N^{fd}_{j_p}, N^f_{j_p}) >$ is a path in $G^\star$, then $(N^{fd}_{j_1}, N^f_{j_p}) \notin E^\star$, for any $j_1, \ldots, j_p \in \{1, \ldots, n\}$.

Proof : Suppose that $< (N^f_{j_1}, N^{fd}_{j_2}), (N^{fd}_{j_2}, N^f_{j_2}), \ldots, (N^f_{j_{p-1}}, N^{fd}_{j_p}), (N^{fd}_{j_p}, N^f_{j_p}) >$ is a path in $G^\star$ as shown in Fig. 6.4(b) for some $j_1, \ldots, j_p \in \{1, \ldots, n\}$. The arcs $I^f_{j_1}, \ldots, I^f_{j_p}$ form a containment family. By an argument similar to that in the proof of Property 6.3 one can conclude that the hypothetical situation shown in Fig. 6.4(b) is not possible. □

Using the above properties we establish the following theorem :

Theorem 6.1. *$G^\star$ does not have any cycle.*

Proof : Suppose that $G^\star$ has a cycle B. There are two cases to consider.

1. B consists of vertices of type fwd and $fwd - dup$ only.

   Since no edge connecting vertices of the same type exists in $G^\star$, B must contain an edge (N^f_j, N^{fd}_k) for some $j, k \in \{1, \ldots, n\}$. B must consist of a path between N^f_j and N^{fd}_k passing vertices of type fwd and $fwd - dup$ alternately. By Property 6.3.G this is impossible.

2. B contains some vertices of type *back*.

   (a) B contains at least one vertex N^b_j such that $(N^b_j, N^f_k) \in E^\star$ and $(N^b_j, N^{fd}_l) \in E^\star$, for some $j, k, l \in \{1, \ldots, n\}$. The existence of B contradicts Property 6.2.G.

   (b) B does not contain any vertex N^b_j such that $(N^b_j, N^f_k) \in E^\star$ and $(N^b_j, N^{fd}_l) \in E^\star$ for some $j, k, l \in \{1, \ldots, n\}$. By Property 6.1.G, all vertices of type *back* in B must be connected exclusively either to vertices of type fwd or $fwd - dup$, but not both. Since no two

vertices of the same type have an edge between them in $G^\star$, B can not exist.

This completes the proof of Theorem 6.1. □

By construction, all vertices of type *back* are independent of each other in $G^\star$. Starting with these vertices, we perform a breadth-first-search [1] on $G^\star$. Since $G^\star$ has no cycle by Theorem 6.1, the breadth-first-search results in a partition of $V^\star$ into two mutually disjoint independent sets $Y^\star$ and $Z^\star$ such that $Y^\star$ contains all the vertices of type *back* and f vertices of type fwd and/or $fwd - dup$, and $Z^\star$ contains vertices of type fwd and/or $fwd - dup$ only. The breadth-first-search can be performed in $O(m+ \mid E^\star \mid)$ time.

6.5. Minimum Coloring Algorithm for a Family of Arcs

Using the auxiliary graph $G^\star$ of Section 6.3, the following algorithm finds a minimum coloring of A.

Algorithm 4.

Input: A family A of arcs on a circle representing a CACG G_A.

Output: A minimum coloring of A.

Method:

1. Convert the family A of $n = b + f$ arcs into a family I of $m = b + 2f$ intervals on the real line as described in Section 4.2.

2. Find a minimum coloring of I using Masuda and Nakajima's algorithm [12].

3. Construct an auxiliary graph $G^\star$ from I and its coloring by the method described in Section 6.3.

4. Find an independent set $Y^\star$ of $G^\star$ of size $b + f$ as described in Section 6.4.

5. (a) Assign $color(I_k^b)$ to every backward arc A_k.

   (b) As for each forward arc A_j, if $N_j^f \in Y^\star$, then assign $color(I_j^f)$ to A_j, else if $N_j^{fd} \in Y^\star$ then, assign $color(I_j^{fd})$ to A_j.

Step 1 of Algorithm 4 takes $O(n)$ time to obtain a family I of intervals on the real line from a given family A of arcs. Step 2 takes $O(nlogn)$ time to obtain a minimum coloring of I. Step 3 constructs an auxiliary graph $G^\star = (V^\star, E^\star)$ in $O(n+e)$ time, where $e =\mid E_A \mid$ for the CACG $G_A = (V_A, E_A)$. $Y^\star$ can be found in $O(m+ \mid E^\star \mid)$ time, where $\mid E^\star \mid \leq f+ \mid E_A \mid$. Step 5 is a simple lookup and takes $O(n)$ time. Thus the overall complexity of Algorithm 4 is $O(nlogn + e)$. □

Lemma 6.2. *The independent set $Y^\star$ of $G^\star$ induces a coloring of A.*

Proof : We prove that the assignment of colors made with the help of $Y^\star$ produces a coloring of A. Let A_j and A_k be two arcs in A such that $A_j \supset A_k$.

There are three cases to consider separately.

1. Both A_j and A_k are backward arcs. Since no backward arcs were duplicated in I, and $I_j^b \supset I_k^b$, $color(I_j^b) \neq color(I_k^b)$. Thus, A_j and A_k are assigned different colors.

2. Both A_j and A_k are forward arcs. Suppose that $color(I_j^f) = color(I_k^{fd})$. Since $I_j^f \supset I_k^f$, $(N_j^f, N_k^{fd}) \in E^\star$, and so both N_j^f and N_k^{fd} do not appear together in $Y^\star$. Thus, A_j and A_k are never assigned the same color as that assigned to I_j^f and I_k^{fd}.

3. A_j is a backward arc and A_k is a forward arc. There are two subcases to consider.

   (a) $color(I_j^b) = color(I_k^f)$ and $I_j^b \supset I_k^{fd}$. Since $(N_j^b, N_k^{fd}) \in E^\star$, both N_j^b and N_k^{fd} do not appear together in $Y^\star$. As a result, A_j and A_k are never assigned the same color as that assigned to I_j^b and I_k^f.

   (b) $color(I_j^b) = color(I_k^{fd})$ and $I_j^b \supset I_k^f$. Since $(N_j^b, N_k^{fd}) \in E^\star$, both N_j^b and N_k^{fd} do not appear together in $Y^\star$. As a result, A_j and A_k are never assigned the same color as that assigned to I_j^b and I_k^{fd}.

In all the cases, A_j and A_k are assigned distinct colors. Since the b vertices of type *back* and the f vertices of type fwd and/or $fwd - dup$ in $Y^\star$ correspond to all the b backward arcs and f forward arcs, respectively, in A, all the arcs in A get colors. Furthermore, we have shown above that if an arc contains another, they are assigned distinct colors. This completes the proof. □

Lemma 6.2 and the above discussion lead us to the final theorem :

Theorem 6.2. *Given a family A of n arcs on a circle, Algorithm 4 finds correctly, in $O(nlogn + e)$ time, a minimum coloring of the CACG $G_A = (V_A, E_A)$, where $e =\mid E_A \mid$.* □

6.6. Summary

In this chapter we have presented an $O(nlogn + e)$ time algorithm to obtain a minimum coloring of a CACG. The method first opens out a given family A of arcs into a family I of intervals on the real line. It then finds a minimum coloring of I and constructs a special auxiliary graph $G^\star$. The algorithm uses $G^\star$ to find a coloring scheme for A from the minimum coloring of I.

Chapter 7

Conclusion

In this thesis we have introduced a new class of containment graphs, called circular-arc containment graphs(CACGs). A CACG is a containment graph for a family of arcs on a circle. A CACG is a generalization of an interval containment graph(ICG) which is a containment graph for a family of intervals on the real line.

The problems of finding a maximum clique, a maximum independent set, a minimum coloring and a minimum covering by cliques, are NP-hard for a general graph. However, they are solvable in polynomial time for certain special classes of graphs. In particular, Masuda and Nakajima [12] have presented optimal algorithms for all of the above four problems for an interval containment graph. Using their results, we have derived efficient algorithms for the solution of the first three problems for our new graph. The problem of finding a minimum covering by cliques is solvable in polynomial time for a comparability graph [8]. Since a CACG is a comparability graph, the same result also applies to it. The task of finding a more efficient algorithm for finding a minimum covering by cliques for a CACG remains open.

Recently, Rotem and Urrutia [15] introduced and characterized circular permutation graphs (CPGs) as a generalization of permutation graphs. They also gave a polynomial time recognition algorithm for a CPG. However, algorithms for the above four problems have not been developed to date for a CPG. In this thesis we have established the equivalence of a CACG and a CPG by means of a constructive proof. Therefore, all the algorithms described here are applicable to a CPG as well.

A graph representing the containment relation between rectangular regions on a two-dimensional plane can be expected to have some interesting applications in the realm of VLSI design. Some insight into this class of graphs may be obtained from our current knowledge of interval containment graphs and circular-arc containment graphs. A future direction of research might involve the study of rectangle containment graphs.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

[2] J. Bentley and D. Wood, "An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles," *IEEE Transactions on Computers*, Vol. C-29, No. 7, pp. 571-676, 1980.

[3] S. Even, A. Pnueli, and A. Lempel, "Permutation Graphs and Transitive Graphs," *Journal of the ACM*, Vol. 19, pp. 400-410, 1972.

[4] M. L. Fredman, " On Computing the Length of Longest Increasing Subsequences," *Discrete Mathematics*, Vol. 11, pp. 29-35, 1975.

[5] M. R. Garay and D. S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, CA, 1979.

[6] F. Gavril, "Algorithms for a Maximum Clique and a Maximum Independent Set of a Circle Graph," *Networks,* Vol. 3, pp. 261-273, 1972.

[7] F. Gavril, "Algorithms on Circular-Arc Graphs," *Networks,* Vol. 4, pp. 357-369, 1974.

[8] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, New York, 1980.

[9] M. C. Golumbic, "Containment Graphs and Intersection Graphs," IBM Israel Scientific Center, Technical Report 135, 1984.

[10] U. I. Gupta, D. T. Lee, and J. Y-T. Leung, "Efficient Algorithms for Interval Graphs and Circular-Arc Graphs," *Networks*, Vol.12, pp. 459-467, 1982.

[11] D. E. Knuth, *The Art of Programming*,Vol. 3, Addison-Wesley, Reading, MA, 1973.

[12] S. Masuda and K. Nakajima , " Optimal Algorithms for Interval Containment Graphs," *Proc. 29th Midwest Symposium on Circuits and Systems,* Lincoln, NE, August 1986, pp. 431-434.

[13] S. Masuda and K. Nakajima, " An Optimal Algorithm for Finding a Maximum Independent Set of a Circular-Arc Graph," *SIAM Journal on Computing*, Vol. 17, 1988, to appear.

[14] T. Ohtsuki, H. Mori, E. S. Kuh, T. Kashiwabara, and T. Fujisawa, "One Dimensional Logic Gate Assignment and Interval Graphs," *IEEE Transactions on Circuits and Systems*, Vol. CAS-26, pp. 675-684, 1979.

[15] D. Rotem and J. Urrutia, "Circular Permutation Graphs," *Networks*, Vol. 12, pp. 429-437, 1982.

[16] D. Rotem and J. Urrutia, " Finding Maximum Cliques in Circle Graphs,," *Networks*, Vol. 11, pp. 269-278, 1981.

[17] O. Wing, S. Huang, and R. Wang, "Gate Matrix Layout," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-4, pp. 220-231, 1985.

[18] O. J. Yu and O. Wing, "Interval-Graph-Based PLA Folding," *Proc. 1985 International Symposium on Circuits and Systems*, Kyoto, Japan, 1985, pp. 1463-1466.

CURRICULUM VITAE

Name: Madhura Vivek Nirkhe

Permanent Address: 3426 Tulane Drive Apt 3 ,
Hyattsville, MD 20783.

Degree and date to be conferred: M.S, 1987.

Date of birth: October 10, 1962.

Place of birth: Bombay, India.

Secondary education: St. Columba High School, Bombay, India, 1977.

Collegiate institutions attended	Dates	Degree	Year
University of Maryland, College Park, MD 20742	1986–1987	M.S.(EE)	1987
Indian Institute of Technology, Bombay, India.	1979–1984	B.S.(CS)	1984

Major: Computer Engineering.

Professional positions held:

Research Assistant, 1986-1987,

Dept of Physics and Astronomy, Univ. of Maryland, College Park, MD.

Research Fellow, June 1987-present,

Systems Research Center, Univ. of Maryland, College Park, MD.