National Library of Canada

Acquisitions and Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada

Direction des acquisitions et des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

Intersection Graphs, Fraternally Orientable Graphs and Hamiltonian Cycles

by

Claudia C. Iturriaga-Velazquez

A thesis
presented to the University of Ottawa
in fulfilment of the
thesis requirement for the degree of
Master
of
Computer Science

Department of Computer Science,
University of Ottawa,
Ottawa, Ontario, Canada, 1994

The Master of Computer Science program is a joint program with Carleton University, administrated by the Ottawa-Carleton Institute for Computer Science.

ISBN 0-612-11563-1

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Abstract

Consider a graph $G(V,E)$, where V and E denote the vertex and edge sets of $G(V,E)$, respectively. An orientation $\vec{G}$ of $G(V,E)$ is the result of giving an orientation to the edges of G. A directed graph is fraternally oriented if for every three vertices u,v,w, the existence of the edges $u \rightarrow w$ and $v \rightarrow w$ implies that $u \rightarrow v$ or $v \rightarrow u$. A graph G is fraternally orientable if there exists an orientation $\vec{G}$ that is fraternally oriented. In this thesis we study some properties of fraternally orientable graphs, and we describe an algorithm to find a hamiltonian cycle in strongly connected fraternally oriented graphs $\vec{G}$.

Acknowledgements

I would like to thank my boyfriend, Alejandro López-Ortiz, for his continuous encouragement, for cheering me up and above all, for his love.

I would like to thank my supervisor, Jorge Urrutia, for his patience and confidence and for many hours of work. His guidance and counseling have been of great importance in this stage of my academic career.

Special thanks to Margaret Urrutia for reading this thesis.

Also I would like to thank Jorge and Margaret Urrutia for their friendship.

I wish to thank my friends, María Eugenia Perez, Johanne Morin and Rym Mili, for cheering me up during hard times.

I thank my mother, my father and my little sister Juanita for their love.

Lastly, financial support from the Institute of Mathematics at the National Autonomous University of México is gratefully acknowledged.

Contents

List of Figures

Chapter 1

Introduction

A **Hamiltonian Cycle** in a graph G is a cycle that contains every vertex of G. The Hamiltonian Cycle Problem has been studied for several classes of graphs. It consists of testing whether a graph G contains a hamiltonian cycle or not. For general graphs the Hamiltonian Cycle Problem is known to be NP-complete. In addition, it remains NP-complete for the following classes of graphs:

1. planar 3-connected graphs [GJT76],
2. bipartite graphs [Kri76],
3. split graphs [Gol80],
4. edge graphs [Ber81],
5. planar bipartite graphs [IPS82],
6. grid graphs [IPS82],

7. undirected path graphs, thus chordal graphs [Ber86],
8. double interval graphs [IPS82],
9. circle graphs [Dam89].

Polynomial time algorithms for the Hamiltonian Cycle Problem have been developed for:

1. 4-connected planar graphs [Gou82],
2. interval graphs [Kei85],
3. circular-arc graphs [Cha93].

A graph is called Hamiltonian if it has a Hamiltonian cycle. We are interested in studying the Hamiltonian Cycle Problem in a special class of graphs called *fraternally oriented*. In this thesis we show that if a strongly connected digraph $\vec{G}$ is fraternally oriented, then $\vec{G}$ is hamiltonian and one such cycle can be found in $O(|E|)$ time.

Here is an outline of concepts and results introduced in the following chapters.

- In chapter 2 we give some basic definitions and notation that will be used in this thesis.
- In chapter 3 we study comparability graphs. Here we develop useful concepts that familiarize the reader with concepts to be presented in chapter 5. These concepts will be also used in chapter 4.

- We define intersection graphs in chapter 4. We do a brief study of the following classes of intersection graphs: interval graphs, subtree graphs and circular-arc graphs.

- In chapter 5 we introduce our main class of graphs: *fraternally orientable graphs*. Our main result uses this kind of graphs. We also prove here that the graphs presented in the previous chapter — interval graphs, subtree graphs, chordal graphs and circular-arc graphs — are all fraternally orientable.

- The main result is presented in chapter 6. We show that every strongly connected fraternally oriented graph is Hamiltonian and we propose an $O(|E|)$ time algorithm to find a Hamiltonian Cycle in any such graph.

Chapter 2

Basic Definitions and Notation

In this thesis we consider finite connected graphs $G = (V, E)$, where V is the set of vertices of G and E is the set of edges. We denote also the set of vertices of a graph G as $V(G)$ and the set of edges as $E(G)$. In general, we use standard graph theory terminology, as in the book *Graph Theory with Applications* by Bondy and Murty [BMu76]. We give some of these basic definitions here.

We say that vertex u is **adjacent** to vertex v if the edge (u, v) is in the graph. We also denote (u, v) as $u - v$. We do not consider graphs with parallel edges, that is for any pair of vertices u and v of G, there is at most one edge that joins them. We also do not consider self-loops. These are edges from a vertex to itself.

The set of vertices adjacent to a given vertex $v \in V(G)$ is called the **neighborhood** of v. We denote it by $N(v)$.

A walk W in G is a finite non-null sequence $W = v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_k$, whose terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of edge e_i are v_{i-1} and v_i. We say that W is a walk from v_0 to v_k. The vertex v_0 is called the

origin of W and v_k the terminus of W. The vertices $v_1, v_2, \ldots, v_{k-1}$ are called the internal vertices of W. The length of the walk W is k. A walk is determined by the sequence of its vertices $v_0, v_1, \ldots, v_k$; therefore we can denote $W = v_0, v_1, \ldots, v_k$. We say that W is a **closed walk** if $v_0 = v_k$.

A **triangular chord** in a closed walk W is an edge of form $v_i - v_{i+2}$ (in $i+2$ adition is mod k, with $|W| = k$).

A path $P = v_0, v_1, \ldots, v_n$ is a walk where the vertices are distinct. We say that P is a path from vertex v_0 to vertex v_n. The length of P is the number of edges in P, in this case equal to n.

A cycle $C_n = v_0, \ldots, v_n$ is defined as path with $v_0 = v_n$. We also denote $C_n = v_1, \ldots v_n$, where $v_i \neq v_j$, for $j, i \in \{1, \ldots, n\}$, $i \neq j$. A cycle is odd or even as n is odd or even.

A graph H is a **subgraph** of a graph G if $V(H) \subset V(C)$ and $E(H) \subset E(G)$.

We say that a graph H is an **induced subgraph** of a graph G if H is a subgraph of G and for $u, v \in V(H)$, $u - v \in E(H)$ if and only if $u - v \in E(G)$.

A graph G is **connected** if for every pair of vertices $u, v \in G$ there exists a path P with origin u and terminus v. A connected graph with no cycles is called **tree**.

If a graph G is not connected the **components** of G are the maximal connected subgraphs of G.

The **complete graph** G is a graph such that for every pair of vertices $u, v \in V(G)$, $u - v \in E(G)$.

A complete subgraph H of a graph G is a **clique** of G if there is no complete subgraph of G which properly contains H.

The **complement graph** G^c of a graph G is a graph with vertex set $V(G)$ and $u - v \in E(G^c)$ if and only if $u - v \notin E(G)$.

A representation of a graph G in the Euclidian plane for which edges of G intersect only in the points that represent vertices of G is called a **planar representation** of G.

A graph G is a **planar graph** if there exists a planar representation of G.

Given a graph G, we define an **orientation** $\vec{G}$ of G as an assignment of a direction to each edge of G. Each edge $u - v$ in the graph G can be oriented $u \to v$ or $v \to u$; however only one of these orientations is chosen for an orientation $\vec{G}$ of G. We say that a **digraph** is an oriented graph.

We can see that for different choices of orientations of the edges of G we obtain different orientations of G.

A **directed cycle** in an oriented graph $\vec{G}$ is a sequence of vertices and directed edges $v_1, v_2, \ldots, v_n$, such that $v_1 \to v_2, \ldots, v_i \to v_{i+1}, \ldots, v_n \to v_1$.

We say that the orientation $\vec{G}$ is **acyclic** if the oriented graph $\vec{G}$ does not contain any directed cycle.

A **directed path** in a digraph $\vec{G}$ is a path $P = v_0, v_1, \ldots, v_n$ such that $v_0 \to v_1, v_1 \to v_2, \ldots, v_{n-1} \to v_n$. If there is a directed path P in $\vec{G}$ from u to v, we say that v is **reachable** from u.

A digraph $\vec{G}$ is **strongly connected** if every two vertices are reachable from each other.

The set of all vertices u such that $v \to u$ in a digraph $\vec{G}$ will be called the **exterior neighborhood** of v in $\vec{G}$. We denote it by $N^+(v)$.

Similarly, we define the **interior neighborhood** of a vertex v in $\bar{G}$ by

$$N^-(v) = \{u | u \to v\}.$$

As in unoriented graphs the **neighborhood** of a vertex v is

$$N(v) = N^+(v) \cup N^-(v).$$

Chapter 3

Comparability Graphs

Definition 1 *An oriented graph $\vec{G}$ is* **transitively oriented** *if for every three distinct vertices $u, v, w \in V(\vec{G})$ if $u \rightarrow v$ and $v \rightarrow w$ in $\vec{G}$ then $u \rightarrow w$ is also an edge of $\vec{G}$.*

Definition 2 *A graph G is a* **comparability graph** *if it admits a transitive orientation.*

Figure 3.1 a) shows a comparability graph. Figure 3.1 b) is not a comparability graph.

The following characterization of comparability graphs was given in [GHo64]:

Theorem 1 [GHo64] *A graph G is a comparability graph if and only if each odd closed walk has a triangular chord in G.*

Several definitions and results will be needed to prove theorem 1.

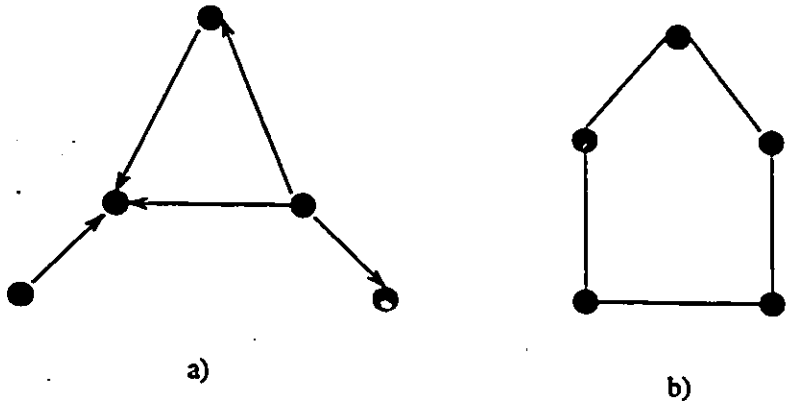

Figure 3.1: a) A comparability graph. b) A non-comparability graph.

The graph in figure 3.2 is not a comparability graph, since $W = d, a, b, e, b, c, f, c, a, d$ is a closed walk with length nine and no triangular chords. Recall that a walk is not necessarily a path (Chapter 2).

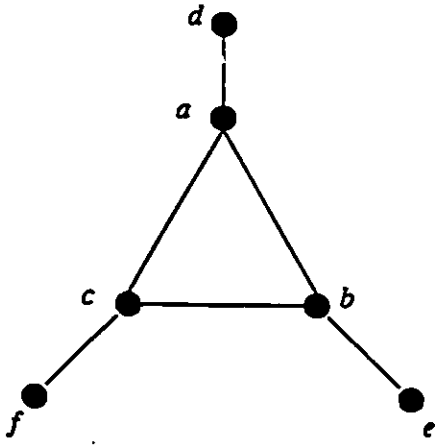

Figure 3.2: A non-comparability graph.

We can notice that in order to orient transitively the graph in figure 3.3, if we orient $v_1 \rightarrow v_2$ then this determines the orientation $v_3 \rightarrow v_2$. Similarly $v_3 \rightarrow v_2$ determines $v_3 \rightarrow v_4$.

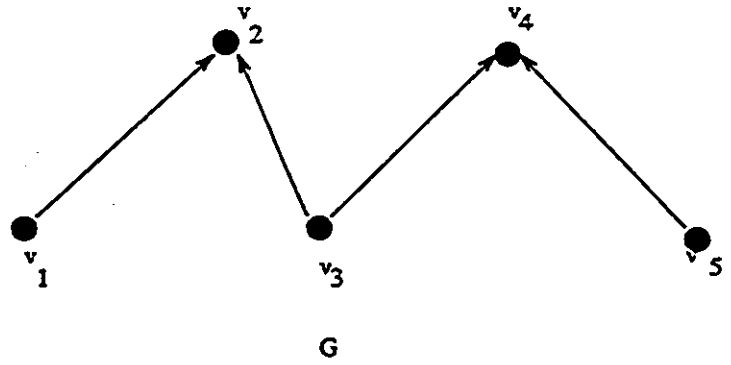

Figure 3.3: A transitive orientation.

To give a transitive orientation on a graph G, we define the following relation.

Definition 3 *In a graph G, we say that an edge $u-v$* **forces** *an edge $w-z$ and denoted it by $u-v\ F\ w-z$, if there exists a walk $W = v_0, v_1, \ldots, v_{n-1}, v_n$ with no triangular chords and with $v_0 = u$, $v_1 = v$ or $v_0 = v$, $v_1 = u$ and $v_{n-1} = w$, $v_n = z$ or $v_{n-1} = z$, $v_n = w$.*

Lemma 1 *The force relation is an equivalence relation on a graph G.*

PROOF. In order to show that this is an equivalence relation we must show that is reflexive, symmetric and transitive.

1. It is reflexive. Let $u-v \in E(G)$. Then $u-v\ F\ u-v$ since $W = u, v, u$ is a walk with no triangular chords.

2. It is symmetric. Let $u-v,\ w-z \in E(G)$. If $u-v\ F\ w-z$ then there is a walk $W = v_0, v_1, \ldots, v_{n-1}, v_n$ with no triangular chords and with $v_0 = u$, $v_1 = v$ or $v_0 = v$, $v_1 = u$ and $v_{n-1} = w$, $v_n = z$ or $v_{n-1} = z$, $v_n = w$. Taking $W' = v_n, v_{n-1}, \ldots, v_1, v_0$, it follows that $w-z\ F\ u-v$.

3. It is transitive. Let $u-v$, $w-z$ and $x-y \in E(G)$ such that $u-v\ F\ w-z$ and $w-z\ F\ x-y$. Since $u-v\ F\ w-z$ there is a walk $W_1 = v_0,\ldots,v_n$ with no triangular chords and without loss of generality we may assume that $v_0 = u$, $v_1 = v$, $v_{n-1} = w$ and $v_n = z$. Similarly since $w-z\ F\ x-y$ there is a walk $W_2 = v'_0,\ldots,v'_m$ with no triangular chords starting with the edge $w-z$ and ending with the edge $x-y$. Two cases arise:

   (a) $v'_0 = w$, $v'_1 = z$,

   (b) $v'_0 = z$, $v'_1 = w$,

   Case (a).

   We have $v'_0 = w$ and $v'_1 = z$. We take a walk $W' = v_0, v_1, \ldots, v_{n-1}, v_n, v'_2, v'_3, \ldots, v'_m$. There are no triangular chords in W' since W_1 and W_2 do not have triangular chords and $v_{n-1} - v'_2 \notin E(G)$, since $v_{n-1} = v'_0$ and $v'_0 - v'_2 \notin E(G)$.

   Case (b).

   We have that $v'_1 = w$ and $v'_0 = z$. We take a walk $W' = v_0, v_1, \ldots, v_{n-1}, v_n, v'_1, v'_2, \ldots, v'_m$. The edge $v_{n-1} - v'_1 \notin E(G)$ since $v_{n-1} = v'_1$. The edge $v_n - v'_2 \notin E(G)$ since $v_n = v'_0$ and $v'_0 - v'_2 \notin E(G)$. Therefore there are no triangular chords in W'.

□

Definition 4 *We call* **forcing classes** *the equivalence classes generated by the equivalence relation "force".*

Definition 5 *In a graph G if $u-v$, $v-x \in E(G)$ and $u-x \notin E(G)$ then we say that*

1. $u \rightarrow v$ **det** $x \rightarrow v$ *and* $v \rightarrow u$ **det** $v \rightarrow x$,

2. $u \rightarrow v$ **det** $x \rightarrow y$ *and* $x \rightarrow y$ **det** $w \rightarrow z$ *then* $u \rightarrow v$ **det** $w \rightarrow z$.

In the case $u \rightarrow v$ det $x \rightarrow v$ this means that if a transitive orientation exists in G and $u \rightarrow v$, the edge $x - v$ must be oriented $x \rightarrow v$, i.e. the orientation in $u - v$ determines the orientation in $x - v$. It is also easy to see that if $u - v \; F \; x - y$ then $u \rightarrow v$ **det** $x \rightarrow y$ or $u \rightarrow v$ **det** $y \rightarrow x$.

Lemma 2 *If we have a walk W with no triangular chords then orienting any edge in W determines an orientation on all the edges in W.*

PROOF. Let $W_n = v_0, v_1, \ldots, v_n$ be a walk with no triangular chords in G. Our proof proceeds by induction on the length of W_n. Letting $n = 2$, if we orient $v_0 \rightarrow v_1$, since $v_0 - v_2 \notin E(G)$, then $v_0 \rightarrow v_1$ **det** $v_2 \rightarrow v_1$. If we orient $v_1 \rightarrow v_0$ then $v_0 \rightarrow v_1$ **det** $v_1 \rightarrow v_2$. Similarly if we give an orientation to the edge $v_2 - v_1$ then this determines an orientation in edge $v_0 - v_1$.

Assume that it is true for $n \leq k - 1$. We proceed to prove that our result is true for $n = k$. Let $W_k = v_0, v_1, \ldots, v_k$ be a walk with no triangular chords of length k. Taking any edge $v_i - v_{i+1}$ in W_k, we build walks W_1 and W_2 from W_k as follows:

1. $W_1 = v_0, v_1, \ldots, v_i$ and
2. $W_2 = v_{i+1}, \ldots, v_k$.

Without loss of generality we orient $v_i \rightarrow v_{i+1}$. Since $v_{i-1} - v_{i+1} \notin E(G)$ then we have $v_i \rightarrow v_{i+1}$ **det** $v_i \rightarrow v_{i-1}$. Now since W_1 has a length less than or equal to $k - 1$, by the induction hypothesis we know that if we orient any edge in W_1 this determines an orientation on all the edges in W_1. Therefore since $v_i \rightarrow v_{i+1}$

det $v_i \rightarrow v_{i-1}$ then $v_i \rightarrow v_{i+1}$ determines an orientation on all the edges in W_1. Similarly in the walk W_2, $v_i \rightarrow v_{i+1}$ det $v_{i+2} \rightarrow v_{i+1}$ since $v_i - v_{i+2} \notin E(G)$. Then $v_i \rightarrow v_{i+1}$ determines an orientation on all the edges in W_2. Therefore giving an orientation to any edge in W_k determines the orientation of all the edges in W_k. □

Definition 6 *We say that a forcing class A of a graph G is* **consistent** *if there are no edges $u - v$, $x - y \in A$ such that $u \rightarrow v$* det *$x \rightarrow y$ and $u \rightarrow v$* det *$y \rightarrow x$.*

The next lemma follows easily:

Lemma 3 *If G is a comparability graph then all its forcing classes are consistent.*

PROOF. Suppose that not all the forcing classes of G are consistent. Then there exists a forcing class A with an edge $w - z \in E(G)$ such that an edge $w \rightarrow z$ det $u \rightarrow v$ and $w \rightarrow z$ det $v \rightarrow u$. Therefore we cannot find a transitive orientation of G. □

Notice that a forcing class may consist of a single edge.

A recognition algorithm of a comparability graph finding a transitive orientation was given by Pnueli, et.al. in [PLE64]. This algorithm takes $O(\Delta|E|)$ time, where Δ is the maximum degree of a vertex. We will prove the correctness of Pnueli et.al.'s algorithm and obtain the result of Theorem 1 as a consequence of this algorithm. The algorithm proceeds as follows:

Algorithm Recognition of Comparability Graphs

Rec-Comp(G)

1. Initialize $G' \leftarrow G$.

2. Chose an edge $u - v \in G'$ and orient $u \rightarrow v$.

3. **Build-Class**($u \rightarrow v$,A).

4. **if** A is not a consistent forcing class **then**

   G is not a comparability graph; STOP.

   else

   if all edges are oriented **then**

   G is a comparability graph; STOP.

   else

   Update $G' \leftarrow G' - A$ and go to step 2.

The subroutine **Build-Class**($u \rightarrow v$,A), whose parameters are a directed edge $u \rightarrow v$ and a forcing class A obtained from $u \rightarrow v$:

Build-Class($u \rightarrow v$,A)

1. Initialize $A \leftarrow \emptyset$.

2. Let $x = u$ and $y = v$.

3. **for** all edges $x - w$, $w \neq v$, with $w - y \notin E(G)$ **do**

   if $x - w$ is unoriented **then**

   orient $x \rightarrow w$.

   if $x - w$ is oriented and $w \rightarrow x$ **then**

   A is not a consistent forcing class; STOP.

4. for all edges $y - w$ with $w - x \notin E(G)$ **do**

   if $y - w$ is unoriented **then**

   orient $w \rightarrow y$.

   if $y - w$ is oriented and $y \rightarrow w$ **then**

   A is not a consistent forcing class; STOP.

5. Update $A \leftarrow A \cup \{x \rightarrow y\}$

6. **if** all edges in $E(G) - A$ are unoriented **then**

   A is a consistent forcing class; STOP.

   else

   Take any directed edge $x' \rightarrow y' \in A$, let $x = x'$, $y = y'$ and go to step 3.

For every edge $u - v$ of G the algorithm takes each edge adjacent to $u - v$. Therefore the algorithm takes $O(\Delta|E|)$ time. We prove the correctness of this algorithm with the following results.

Lemma 4 [GHo64] *If A is a forcing class of a graph G and there exists an edge $x - y$ in $E(G)$ with $x \in V(A)$ and $y \notin V(A)$ then y is adjacent to all the vertices in A.*

PROOF. Since A is a forcing class and $x \in V(A)$ there exists a walk $W = v_0, v_1, \ldots, v_n$ with no triangular chords and $v_0 = x$ that passes through all the vertices in A. Let v_i be the lowest index vertex in A such that $v_i - y \notin E(G)$. Then $W' = y, v_{i-1}, v_i$ is a walk with no triangular chords and thus $y - v_{i-1} \in A$. But this contradicts the fact that $y \notin V(A)$. □

Lemma 5 [GHo64] *If A is a forcing class of a graph G with $|V(G)| > 3$ such that $V(A) = V(G)$ then G does not contain cycles with only one edge in A.*

PROOF. Assume that such cycles exist. Consider the smallest cycle $C = c_1, \ldots, c_n$ among them. Assume that $c_1 - c_2$ is the only edge of C in A. Then the edge $c_1 - c_3 \in E(G)$ since otherwise $c_1 \rightarrow c_2$ determines $c_3 \rightarrow c_2$ and therefore $c_2 - c_3 \in A$ which is a contradiction. The edge $c_1 - c_3$ may or may not be in A. If $c_1 - c_3 \in A$ we can build a cycle $C' = c_1, c_3, \ldots, c_n$ with one edge in A and smaller than C, contradicting the fact that C is the smallest cycle. Therefore $c_1 - c_3 \notin A$ and we have a triangle c_1, c_2, c_3 with only one edge in A. Since C is the smallest cycle then C must be a triangle.

If C is a triangle then we choose the triangle $C = c_1, c_2, c_3$ with $c_1 - c_2 \in A$, having the shortest walk W with no triangular chords from c_1 to c_3 passing through c_2. This walk exists since $V(A) = V(G)$. Let $W = v_1, v_2, \ldots, v_k$, with $v_1 = c_1, v_2 = c_2$ and $v_k = c_3$. Since W does not have triangular chords $v_1 - v_3 \notin E(G)$ and therefore $v_2 - v_3 \in A$. The edge $v_3 - v_k$ is in $E(G)$ since otherwise $v_2 - v_3$ forces $v_2 - v_k$ and then $v_2 - v_k \in A$ and this contradicts the fact that $C = c_1, c_2, c_3$ is a triangle with only one edge in A.

We have obtained that $v_3 - v_k \in E(G)$. Now if the edge $v_3 - v_k \in A$ since $v_1 - v_3 \notin E(G)$, then $v_1 - v_k \in A$ and again the triangle C will have two edges in A instead of one. Thus $v_3 - v_k \notin A$. Then we can get a triangle $C' = v_2, v_3, v_k$, with only one edge in A with a walk $W' = v_3, \ldots, v_k$, smaller than W and then we have a contradiction from the way that we chose the triangle C. □

Theorem 2 [GHo64] *If G is a comparability graph and A is a forcing class then there exists a transitive orientation $\vec{G}'$ such that for a vertex $x \in N(A)$ all edges of*

the form $x - v$ with $v \in V(A)$ are oriented $x \to v$ or all are oriented $v \to x$.

PROOF. Let $\vec{G}$ be a transitive orientation of G such that there exists a vertex $x \in N(A)$ such that $x \to u$ and $v \to x$ with $u, v \in V(A)$. Let X be the set of such vertices x in $\vec{G}$.

Take a vertex $x_0 \in X$ with $v \to x_0$ and $x_0 \to u$, $u, v \in V(A)$. Then the following are true:

1. If we have a vertex w adjacent to x_0 with $w \notin V(A)$ then w must be adjacent to all the vertices in A. To prove this we suppose without loss of generality that we have an orientation $x_0 \to w$ in $\vec{G}$. Since $\vec{G}$ is a transitive orientation and we have $v \to x_0$ and $x_0 \to w$ in $\vec{G}$ then $v \to w \in \vec{G}$. By lemma 4, w is adjacent to all the vertices in A.

2. If there exists a vertex $w \in N(A) - X$ and we have $w \to z$ with $z \in V(A)$ then $w \to x_0$. To prove this, we notice that for the previous result w is adjacent to all the vertices in A. In particular $w \to v$. Since $\vec{G}$ is a transitive orientation the edge $w \to x_0$ is in $\vec{G}$.

Now let $\vec{G}'$ be the orientation of G obtained from $\vec{G}$ by reversing the orientation of all edges $v \to x$ such that $v \in V(A)$ and $x \in X$.

Now we proceed to prove that the orientation $\vec{G}'$ is a transitive orientation. Suppose that there exist edges $a \to b$, $b \to c \in \vec{G}'$ but $a \to c \notin \vec{G}'$. Let Z be the set of edges in $\vec{G}'$ with one end in X and the other in A. All edges are oriented from X to A in $\vec{G}'$. One of the edges $a \to b$ or $b \to c$ must be in Z, otherwise $\vec{G}$ would not be a transitive orientation. Without loss of generality assume that $a \to b \in Z$. Then two cases arise:

1. $c \in A$ or

2. $c \in N(A) - X$.

Case 1. If $c \in A$ since $a \rightarrow b \in Z$ then $a \in N(A)$ and $b \in A$. By lemma 4, $a - c \in G$. If $c \rightarrow a$ in $\vec{G}'$, then this contradicts the way that we chose this orientation.

Case 2. If $c \in N(A) - X$ then $a \in X$ and $b \in A$ we have seen before that $a \rightarrow c$ must be in $\vec{G}$ and therefore must be in $\vec{G}'$. □

Theorem 3 [GHo64] *If G is a comparability graph with forcing class A of G then the graph $G' = G - A$ is a comparability graph.*

PROOF. By theorem 2 we can find a transitive orientation $\vec{G}$ of G such that for a vertex $y \notin V(A)$ all the edges $x - y$ with $x \in A$ are oriented $x \rightarrow y$ or all are oriented $y \rightarrow x$. Let $\vec{G}' = \vec{G} - \vec{A}$. Suppose that $\vec{G}'$ is not a transitive orientation of $G - A$. Then we have $u \rightarrow v, v \rightarrow w \in \vec{G}'$ and $u \rightarrow w \notin \vec{G}'$. Since $\vec{G}$ is a transitive orientation of G then $u \rightarrow w \in \vec{G}$, therefore $u \rightarrow w \in \vec{A}$. It now follows that $v \in V(A)$ otherwise not all edges joining v to A are oriented from v to A or from A to v. This contradicts our selection of $\vec{G}$.

If $u \rightarrow w \in \vec{A}$ and $v \in V(A)$. If $|V(A)| = 3$ then there is no walk without triangular chords that contains u, v and w. This contradicts that A is a forcing class. Therefore $|V(A)| > 3$. Now we take the subgraph H induced by $V(A)$. We have that $V(A) = V(H)$ and we have a triangle u, v, w with only one edge in A. Because of lemma 5 this can not occur. Therefore $u \rightarrow w \in \vec{G}'$. □

The next result follows from the definition of forcing classes:

Lemma 6 *If G is a comparability graph and A is a forcing class of G then the subgraph H of G with edge set A is a comparability graph.*

Theorem 4 *If A is a consistent forcing class of a graph G and $G - A$ is a comparability graph then any transitive orientation of $G \bar{-} A$ joined with a transitive orientation of A is a transitive orientation of G.*

PROOF. Since A is a consistent forcing class we can find a transitive orientation of its edges. Let $\vec{G}$ be an orientation obtained from joining a transitive orientation of $G - A$ with a transitive orientation of A. Suppose that $\vec{G}$ is not a transitive orientation of G. Then there exist edges $u \rightarrow v$ and $v \rightarrow w$ with $u - w \notin E(G)$. Since A and $G - A$ are transitive orientations therefore one of these edges belongs to A and one belongs to $G - A$. Without loss of generality suppose that $u - v \in A$ with orientation $u \rightarrow v$ and $v - w \in E(G - A)$ with orientation $v \rightarrow w$. Since $u - w \notin E(G)$ then $u - v$ and $v - w$ must be in the same forcing class. This contradicts that A is a forcing class. □

Theorem 5 *The algorithm* **Rec-Comp***(G) works correctly.*

PROOF. We have to prove the following:

1. If for a graph G the algorithm Rec-Comp(G) determines that G is a comparability graph, then G is in fact a comparability graph.
2. If G is a comparability graph then Rec-Comp(G) gives us a transitive orientation for it, and tells us that G is a comparability graph.

Case 1.

We prove our result by induction on the number n of calls to the subroutine **Build-Class.**

For $n = 1$ G calls only one time to **Build-Class** so it is obviously a comparability graph. Suppose that the theorem is true for all the graphs for which we are calling $n-1$ times to **Build-Class.** Assume a graph G has n calls to **Build-Class.** Let A be the first consistent forcing class that was found by **Build-Class.** Then $G - A$ is a graph that calls $n - 1$ times to **Build-Class** and by induction $G - A$ is a comparability graph. Similarly the subgraph H of G whose edge set is A is also a comparability graph. Now we can give a transitive orientation for H and another for $G-A$, so by theorem 4 an orientation obtained in G by combining the transitive orientations of H and $G-A$ is a transitive orientation. Hence G is a comparability graph.

Case 2.

Since G is a comparability graph then by lemmas 3 and 6 each application of the subroutine **Build-Class** obtains a consistent forcing class, and by theorem 3 $G - A$ is a comparability graph. Therefore the algorithm determines that G is a comparability graph. It can easily be checked that the orientation induced in G by the algorithm is a transitive orientation of G. □

From theorems 5 and 3 we can obtain the following result:

Theorem 6 *Let A be a consistent forcing class of a graph G. Then G is a comparability graph if and only if $G - A$ is a comparability graph.*

Finally we are able to prove theorem 1.

PROOF. Assume that there exists an odd closed walk with no triangular chords.

Then we can see that the forcing class on this closed walk is not consistent. By lemma 3, G is not a comparability graph.

Conversely if we do not have an odd closed walk with no triangular chords then all the forcing classes A_i of G are consistent. Now we can get this proof from case 1 of the proof for theorem 5.

Chapter 4

Intersection Graphs

The intersection graph of a finite family of sets is defined as follows:

Definition 7 *A graph G is an* **intersection graph** *if there exists a one-to-one correspondence between its vertices and the elements of a family $\mathcal{F}$ of sets such that two vertices are adjacent in the graph if and only if their two corresponding sets intersect.*

Intersection graphs have been studied and classified with respect to the families of sets from which they can be obtained. The following question arose:

"Is every graph an intersection graph?" Marczewski answered this question in 1945 with the following theorem:

Theorem 7 [Mar45] *Every graph G is an intersection graph.*

PROOF. Let G be any graph. For each vertex $v \in V(G)$ we build a set $S(v) = \{v-w | v-w \in E(G)\}$. Now we take as our family of sets $\mathcal{F} = \{S(v) | v \in V(G)\}$; we

have that $S(v) \cap S(w) \neq \emptyset$ if and only if $v - w \in E(G)$. Thus G is an intersection graph of the set family $\mathcal{F}$. □

For example in the graph given in figure 4.1 our sets are the following:

$S(v_1) = \{v_1 - v_2, v_1 - v_3, v_1 - v_4\}$,

$S(v_2) = \{v_1 - v_2, v_2 - v_3\}$,

$S(v_3) = \{v_2 - v_3, v_1 - v_3, v_3 - v_4\}$,

$S(v_4) = \{v_1 - v_4, v_3 - v_4\}$.

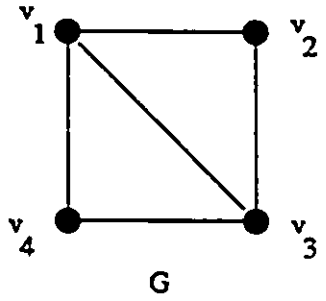

Figure 4.1: Intersection graph of a family of $S(v_i)$ sets.

It is easy to see that if $S(v_i) \cap S(v_j) \neq \emptyset$ then $v_i - v_j \in E(G)$.

In the above theorem there is no restriction on the family of sets $\mathcal{F}$. If we impose restrictions on the elements of $\mathcal{F}$, the problem becomes more interesting; e.g., if we restrict the elements of $\mathcal{F}$ to be curves on the plane we get:

Theorem 8 [EET76] *Not all graphs are intersection graphs of sets of curves in the plane.*

PROOF. Take any non-planar graph G, for example K_5. We build a graph G' by placing a vertex $c_{i,j}$ in the middle of each edge of $v_i - v_j$ of G (figure 4.2).

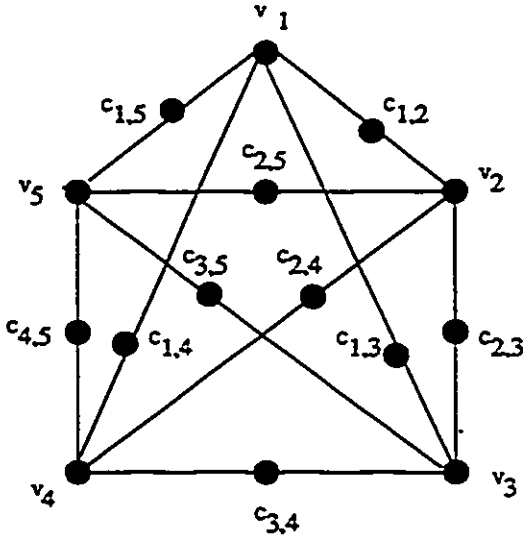

Figure 4.2: The graph G'.

Suppose that the graph G' is an intersection graph of a family $\mathcal{F}$ of curves in the plane.

We can shrink the curves corresponding to the vertices v_i of G' in $\mathcal{F}$ to points x_i without creating new intersections in $\mathcal{F}$ (figures 4.3 and 4.4).

Now the curves corresponding to the vertices c_{ij} intersect at the points x_i into which the curve representing v_j was shrunk. In this way we get a planar representation of G in which the vertices of G correspond to the points x_i and the edges $v_i - v_j$ of G to the curves joining x_i to x_j (the curves $c_{i,j}$, see figure 4.5). □

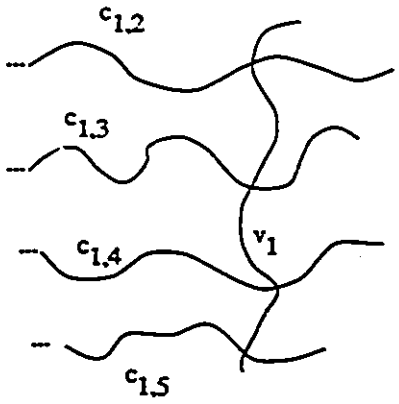

Figure 4.3:

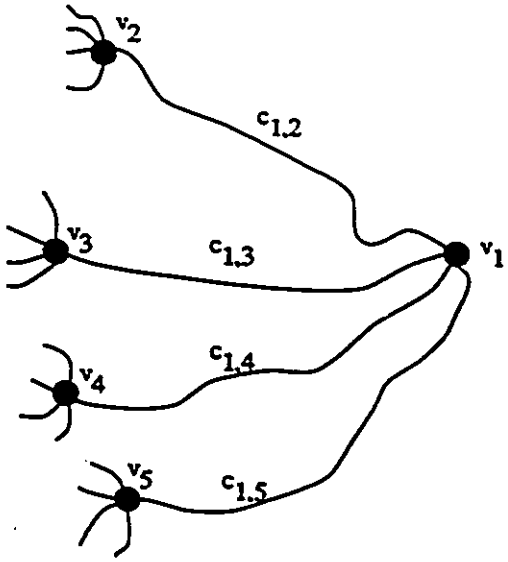

Figure 4.4:

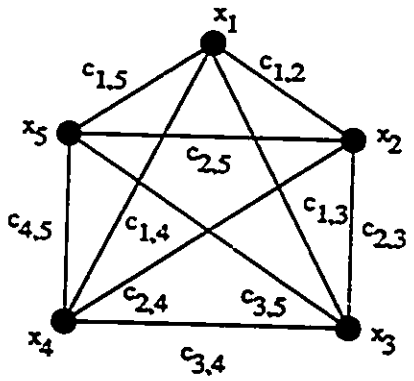

Figure 4.5: The graph obtained when we shrink our family of curves.

Not much is known about the general problem of characterizing intersection graphs of sets of curves on the plane. In this thesis we will survey some restricted classes of intersection graphs. Intersection graphs of special families $\mathcal{F}$ of sets have been studied, some of which are:

1. interval graphs,
2. circular-arc graphs,
3. circle graphs,
4. subtree graphs,
5. permutation graphs.

4.1 Interval Graphs

The problem of characterizing the intersection graph of a family of sets having a defined pattern has been of great interest. In this section we define the following intersection graphs.

Definition 8 *A graph G is called an* **interval graph** *if it is the intersection graph of a family $\mathcal{F}$ of intervals on the real line, that is, if it is possible to set up a one-to-one correspondence between the vertices of G and the intervals in $\mathcal{F}$ such that two vertices are adjacent if and only if the corresponding intervals intersect.*

In figure 4.6 (a) we have an interval family $\mathcal{F} = \{I_1, I_2, I_3, I_4\}$ and in figure 4.6 (b) we have the intersection graph of $\mathcal{F}$ where an interval I_i corresponds to a vertex c_i.

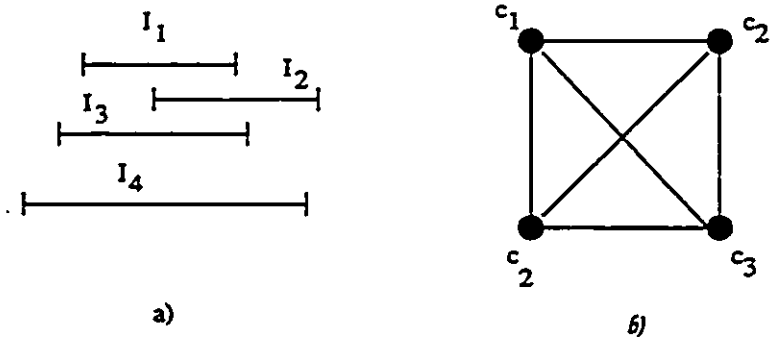

Figure 4.6: a) Interval family $\mathcal{F}$; b) Intersection graph of $\mathcal{F}$.

The problem of characterizing interval graphs was proposed by Hajös in 1957 [Haj57]:

Given a finite number of intervals on a straight line, a graph associated with this set of intervals can be constructed in the following manner: each interval corresponds to a vertex of the graph, and two vertices are connected by an edge if and only if the corresponding intervals overlap at least partially. The question is whether a given graph is isomorphic to one of the graphs just characterized (translated by M.C.G. in [Gol80]).

The biologist Seymour Benzer was studying a structure of the gene and asked the following:

From the classical researches of Morgan and his school, the chromosome is known as a linear arrangement of hereditary elements, the genes. *These elements must have an internal structure of their own. At this finer level, within the* gene *the question arises : ... Are* they [*the subelements within the gene*] *linked together in a linear order analogous to the higher level of integration of the genes in the chromosome?*

A crucial examination of the question should be made from the point of view of topology, *since it is a matter of how parts of the structure are connected to each other, rather than of the distances between them. Experiments to explore the topology should ask* qualitative *questions (e.g., do two parts of the structure touch each other or not?) rather than* quantitative *ones (how far apart are they?)* [Ben59].

Interval graphs have applications in genetics [Ben59], psychophysics [Rob69], archaeology [Ken69] and ecology [Cohen].

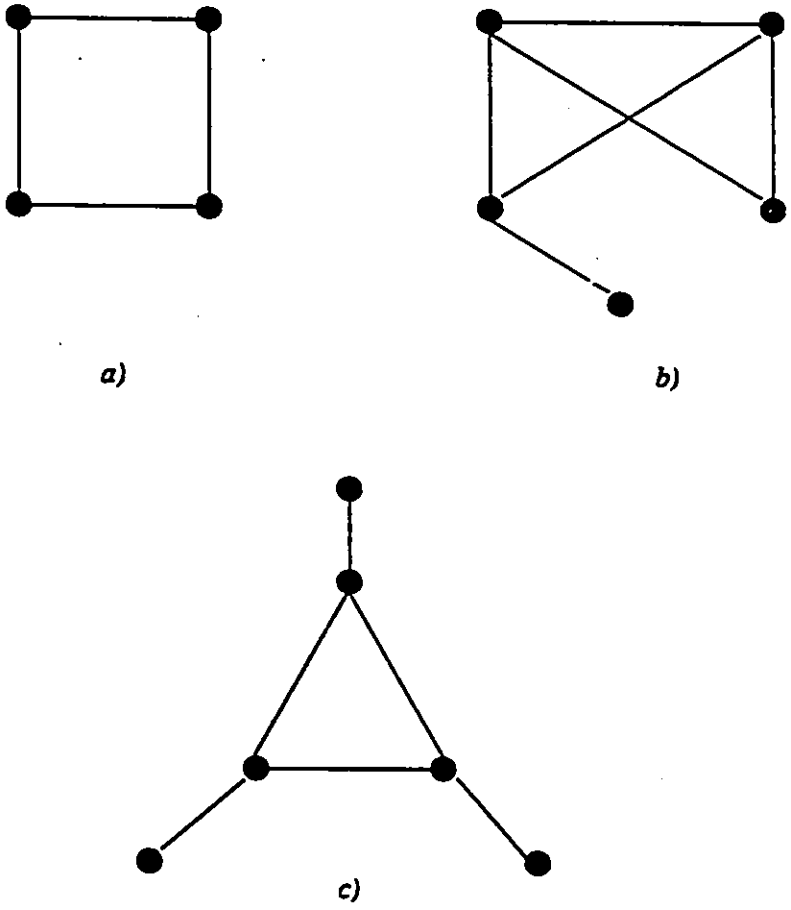

Figure 4.7: a) and c) are not interval graphs; b) is an interval graph.

In figure 4.7 we can see examples of interval graphs.

Several characterizations are known for interval graphs. Some of these were given by :

1. Lekkerkerker and Boland in 1962 [Lek62],
2. Gilmore and Hoffman in 1964 [GHo64],
3. Fulkerson and Gross in 1965 [Ful65].

Now we proceed to introduce a class of graphs called chordal graphs.

Definition 9 *A graph G is* **chordal** *if every cycle of length greater than or equal to four has a chord.*

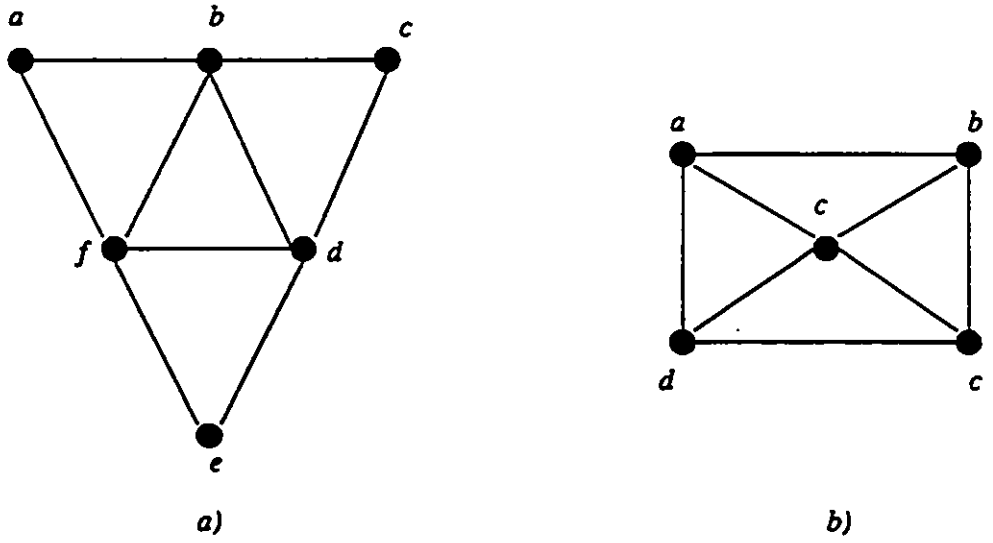

Figure 4.8: a) A chordal graph; b) A non-chordal graph.

We can see an example of a chordal graph in figure 4.8 (a). The graph in figure 4.8 (b) is not a chordal graph since the cycle a, b, c, d does not contain a chord.

We present here the characterization given by Gilmore and Hoffman.

Theorem 9 [GHo64] *A graph G is an interval graph if and only if G is chordal and its complement G^c is a comparability graph.*

PROOF. Let G be an interval graph corresponding to a family $\mathcal{F} = \{I_1, I_2, \ldots, I_n\}$ with intervals $I_i = [l_i, r_i]$. Suppose that $C = c_1, c_2 \ldots, c_s$ $(r \geq 4)$ is a cycle with no triangular chords in G and let I_i be the interval corresponding to the vertex $c_i, i = 1, \ldots, s$. Since $c_1 - c_2 \in E(G)$, the corresponding intervals $I_1 \cap I_2 \neq \emptyset$. Without loss of generality we take $l_1 < l_2$ and since $I_1 \cap I_2 \neq \emptyset$ then $l_2 < r_1$. We now show that $r_1 < r_2$, otherwise $I_2 \subset I_1$ and since $c_2 - c_3 \in E(G)$ then $I_2 \cap I_3 \neq \emptyset$; therefore $I_1 \cap I_3 \neq \emptyset$, but $c_1 - c_3 \notin E(G)$. We have obtained that $r_1 < r_2$. Since $c_1 - c_3 \notin E(G)$ we can set I_3 such that $r_1 < l_3 < r_2 < r_3$ and we maintain the intersection relation among the intervals I_1, I_2, I_3. In general, since $c_i - c_{i+1} \in E(G)$, we can prove $l_{i+1} < r_i$ and since $c_i - c_{i+2} \notin E(G)$ then $r_i < r_{i+1}$; otherwise $I_{i+1} \subset I_i$ and since $c_{i+1} - c_{i+2} \in E(G)$ then $I_{i+1} \cap I_{i+2} \neq \emptyset$ and therefore $I_i \cap I_{i+2} \neq \emptyset$ but $c_i - c_{i+2} \notin E(G)$.

Since $r_i < r_{i+1}$ in particular $r_1 < r_n$. Now in order to set the interval I_{i+2}, since $c_i - c_{i+2} \notin E(G)$ we can set I_{i+2} such that $r_i < l_{i+2} < r_{i+1}$. Now we take the interval I_s. We have $l_s < r_{s-1}$ and since $c_{s-2} - c_s \notin E(G)$ we can set I_s such that $r_{s-2} < l_s < r_{s-1}$ but $r_1 < r_s$ therefore $I_1 \cap I_s = \emptyset$ and this contradicts that G is the interval graph of $\mathcal{F}$ since $c_1 - c_s \in E(G)$.

Now we prove that G^c is a comparability graph. If we have intervals I_i, I_k that do not intersect then one interval is to the left of the other. We give an orientation $i \rightarrow j$ if the interval I_i is to the left of the interval I_j. If we have $i \rightarrow j$ and $j \rightarrow k$, since I_i is to the left of I_j and I_j is to the left of I_k then I_i is to the left of I_k, therefore the edge $i \rightarrow k$ must be in G^c.

Conversely let G be a chordal graph and G^c a comparability graph. Consider a transitive orientation $\vec{G}^c$ of G^c. Two cases arise:

1. G is a clique or
2. G has two or more cliques.

Case 1. Since G is a clique then G^c is only the vertex set of G, therefore G^c is a comparability graph. We take the vertices of G as intervals such that all the intervals intersect each other. Therefore G is an interval graph.

Case 2. G has two or more cliques. Let Q_1 and Q_2 be two maximal cliques in G. There exists at least one edge between Q_1 and Q_2 in G^c. Otherwise $Q_1 \cup Q_2$ is a clique and this contradicts our choice of Q_1 and Q_2.

We now show that in the case that there are two or more edges between Q_1 and Q_2 in G^c then they must have the same orientation in $\vec{G}^c$ between the cliques Q_1 and Q_2. Two cases arise: the edges share a vertex or they do not.

1. The edges in $\vec{G}^c$ between Q_1 and Q_2 have a vertex in common. Let $u - w$, $v - w \in G^c$ with $u, v \in Q_1$ and $w \in Q_2$. Suppose that an orientation on $\vec{G}^c$ of the edges is different between the clique Q_1 and Q_2 in G^c. Without loss of generality, assume that we have $u \rightarrow w$ and $w \rightarrow v$ in $\vec{G}^c$. Since $\vec{G}^c$ is a transitive orientation then $u \rightarrow v \in \vec{G}^c$. This contradicts that Q_1 is a clique since $u, v \in Q_1$ and $u - v \notin Q_1$.
2. Let $u - w, v - z$ be edges in G^c with $u, v \in Q_1$ and $w, z \in Q_2$. Since G is a chordal graph, at least one of the edges $u-z$ or $v-w$ belongs to G^c. Otherwise we can find a cycle u, z, v, w of length four with no triangular chords in G, which contradicts that G is a chordal graph (see figure 4.9).

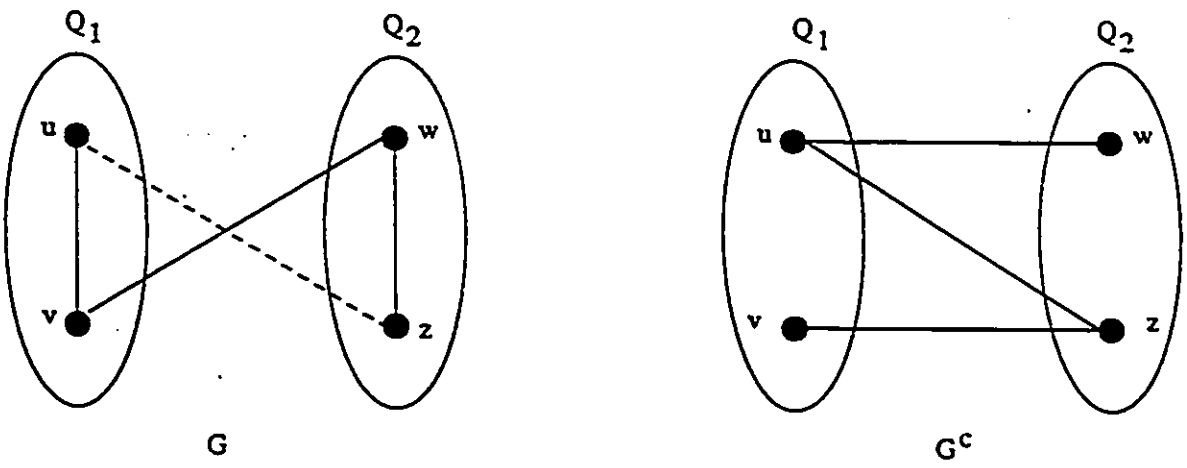

Figure 4.9:

Without loss of generality we suppose that the edge $u - z \in G^c$ and $u \to w$, $v \leftarrow z$ in $\vec{G}^c$. If $u \to z$ by the argument of transitivity $u \to v \in \vec{G}^c$ which contradicts that Q_1 is a clique. Similarly if we have $z \to u$ in $\vec{G}^c$ then $z \to w \in \vec{G}^c$ and this contradicts that Q_2 is a clique (see figure 4.10).

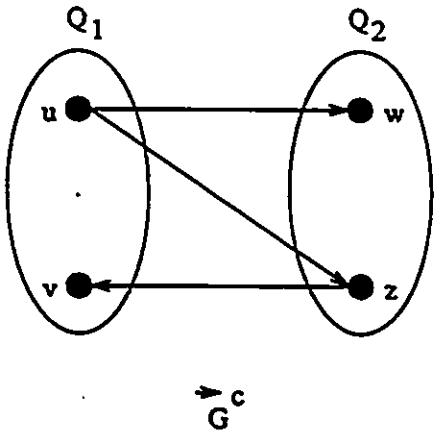

Figure 4.10:

Since for any two cliques Q_i, Q_j of G, any two edges in $\vec{G}^c$ joining elements in Q_i to elements in Q_j have the same orientation, we can define an order relation among the cliques of G by saying that $Q_i < Q_j$ if there are $u \in Q_i$ and $v \in Q_j$ and $u \rightarrow v$ in $\vec{G}^c$. Let $Q_1 < Q_2 < \ldots < Q_p$ be this order. We now proceed to prove that if $u \in G$ such that $u \in Q_i$, $u \in Q_k$ with $Q_i < Q_j < Q_k$ then $u \in Q_j$.

Suppose $u \notin Q_j$. Since Q_i and Q_j are cliques then there exists a vertex $v \in Q_j$ such that $u - v \in G^c$. Since $Q_i < Q_j$ then $u \rightarrow v$. We have also that $Q_j < Q_k$. Then $v \rightarrow u$ but we already have that $u \rightarrow v$.

In order to build an interval representation of G we proceed as follows: for each $v \in V(G)$ let i and k be the smallest and largest indexes such that $v \in Q_i$ and $v \in Q_k$. Then we associate to v the interval $I_v = [i, k]$. We now show that $I_u \cap I_v \neq \emptyset$ if and only if u and v are adjacent in G.

Assume that $I_u \cap I_v \neq \emptyset$. Let $I_u = [i_1, k_1]$ and $I_v = [i_2, k_2]$. If $j \in I_u \cap I_v$ then $i_1 \leq j \leq k_1$ and $i_2 \leq j \leq k_2$. According to the order relation $Q_1, Q_2, \ldots, Q_p$, we have proved that if $u \in Q_{i_1}$ and $u \in Q_{k_1}$ with $Q_{i_1} < Q_j < Q_{k_1}$ then $u \in Q_j$. Similarly we obtain that $v \in Q_j$. Therefore u and v are adjacent in G.

Conversely assume that $I_u \cap I_v = \emptyset$. Let $I_u = [i_1, k_1]$ and $I_v = [i_2, k_2]$. Without loss of generality assume that $k_1 < i_2$. Therefore there is no index j such that $u, v \in Q_j$. Hence u and v are not adjacent in G. □

In [BLe76] Booth and Leuker gave an algorithm to recognize interval graphs in linear time.

4.2 Subtree Graphs

In this section we consider intersection graphs obtained from a family of subtrees of a tree.

Definition 10 *A connected portion of an undirected tree T is called a* **subtree.**

In this definition of subtree of a tree we can obtain subtrees whose end-points are not necessarily vertices of T. These end-points could be in the middle of the edges of T. We can see in figure 4.11 that the end-points of the subtree T_3 of T are in the vertices of T and the end-points of the subtrees T_1 and T_2 of T are not contained in the vertices of T.

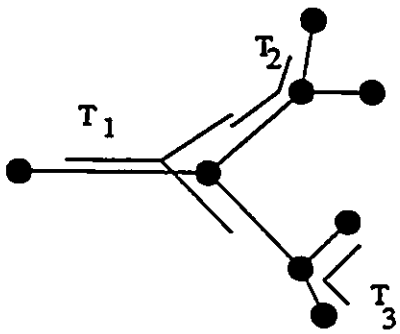

Figure 4.11: Intersection graph of a family of subtrees of a tree.

The intersection graphs that we study here are defined as follows:

Definition 11 *The intersection graph G of a family of subtrees of an undirected tree T is called a* **subtree graph.**

Gavril proved the following result:

Theorem 10 [Gav74] *A graph G is a subtree graph if and only if it is a chordal graph.*

A short proof of this theorem is given in [Wal78].

We denote the set of cliques of a graph G as $\mu(G)$.

Definition 12 *The* **clique graph** $g(G)$ *of a graph G is the intersection graph of the family of cliques of G.*

Definition 13 *A* **clique tree** T *of a graph G is a tree with vertex set $\mu(G)$ and a family $\mathcal{T}$ of subtrees such that each element in $\mathcal{T}$ is a subtree induced by maximal cliques containing a particular vertex of G.*

Gavril gave in [Gav74] an algorithm to build a clique tree for a given chordal graph. This algorithm takes $O(n^4)$. Here we present an algorithm due to Shibata [Shi88], which is shorter than Gavril's algorithm. For this we introduce the following concepts.

Definition 14 *The* **k-overlap clique graph** $g_k(G)$ *of a graph G is a graph with vertex set $\mu(G)$ and two vertices adjacent if and only if the corresponding cliques have an intersection with at least k elements.*

Notice that the 1-overlap clique graph $g_1(G)$ is the clique graph $g(G)$. The graph $g_{k+i}(G)$ is a spanning subgraph of $g_k(G)$.

Now we are ready to present Shibata's algorithm. We call this algorithm **Tree-Clique(G)**, where G is a chordal graph.

Tree-Clique(G)

1. Find the graphs $g_i(G)$ with $1 \leq i \leq k$ such that k is the smallest integer satisfying that $g_k(G)$ is a graph with no edges.
2. Let a tree $T_k = g_k(G)$.
3. Set $i = k$.
4. if $i = 1$ then $T = T_1$ and STOP.

   else

   while $i \geq 2$ do

   i) Find a maximal forest T_{i-1} of $g_{i-1}(G)$ including T_i as subgraph.

   Set $i = i - 1$
5. Set $T = T_1$.

Notice that if G is connected then $g_1(G)$ is connected and the tree T obtained is a spanning tree of $g_1(G)$.

Theorem 11 [Shi88] *If G is a connected chordal graph then a spanning tree T of $g_1(G)$ is a clique tree of G if and only if it can be constructed by the algorithm* **Tree-Clique.**

PROOF. Let T be a clique tree of G. Suppose that T can not be built by the algorithm Tree-Clique. To each edge $Q_i - Q_j$ of T we give a label $|Q_i \cap Q_j|$. Choose the largest k such that the subforest T_k of T is not a spanning forest of $g_k(G)$. Notice that the tree T_k is built with the edges of T with label greater than

or equal to k. Since $T_k \subset g_k(G)$ and T_k is not a spanning forest of $g_k(G)$ then there exist two vertices Q and Q' that are adjacent in $g_k(G)$ but lie in different components of T_k. Therefore some edge $Q_i - Q_j$ in the unique path $P = Q, \ldots, Q'$ on T has label smaller than k. Since $|Q \cap Q'| \geq k$ then $Q \cap Q' \not\subset Q_i \cap Q_j$ and this contradicts that T is a clique tree of G.

Conversely suppose that the algorithm builds a tree T which is not a clique tree. Therefore there exist two cliques Q and Q' such that $Q \cap Q'$ is not contained in some vertex of the unique path between Q and Q' on T. We choose Q and Q' such that $|Q \cap Q'|$ is maximized and the distance between Q and Q' is minimized on T. Let $P = Q_0, Q_1, \ldots, Q_p$ be the path with $Q_0 = Q$ and $Q_p = Q'$; therefore $p \geq 2$. If $Q_0 \cap Q_p \subset Q_i$ for some $i = 1, \ldots, p-1$ then it must be for all $i = 1, \ldots p-1$, otherwise this contradicts that Q_0 and Q_p are the closest points such that $Q_i, i \neq 0, p$ does not contain $Q \cap Q'$. Therefore $(Q_0 \cap Q_p) \cap Q_i \neq \emptyset$ with $i = 1, \ldots, p-1$. Let $|Q_0 \cap Q_p| = k$; then according to the algorithm $|Q_i \cap Q_{i+1}| \geq k$ with $i = 0, \ldots, p-1$. Then we have that $(Q_i \cap Q_{i+1}) - (Q_0 \cap Q_p) \neq \emptyset$ with $i = 0, \ldots, p-1$. Let $P = v_1, v_2, \ldots, v_q$ be the shortest path in the subgraph induced by $\cup_{i=0}^{p-1}((Q_i \cap Q_{i+1}) - (Q_0 \cap Q_p))$, which begins in $(Q_0 \cap Q_1) - (Q_o \cap Q_p)$ and ends in $(Q_p \cap Q_{p-1}) - (Q_0 \cap Q_p)$. Now we take $v \in (Q_0 \cap Q_p)$ and we take the cycle $v, v_1, \ldots, v_q$. Since P is the shortest path and since G is a chordal graph then v is adjacent to v_i for $i = 1, \ldots, q$. Let R_j be a clique containing $\{v_j, v_{j+1}\} \cup (Q_0 \cap Q_p)$ with $i = 1, \ldots, q$ and $R_0 = Q_0$, $R_q = Q_p$. Then $|R_j \cap R_{j+1}| > k$. Since $Q_0 \cap Q_p \subset R_j$ for each j then we have that $Q_0 \cap Q_p$ is contained in each vertex on the path $R_0, \ldots, R_q$; i.e. between $Q_0, \ldots Q_p$. But we chose the points Q_0 and Q_p in such a way that the path $Q_0, \ldots, Q_p$ should have a vertex that does not contain $Q_0 \cap Q_p$. □

In figure 4.12 we can see a graph whose set of cliques is:

$Q_1 = \{1, 4, 6, 7\}$,

$Q_2 = \{2, 4, 5, 6, 7\}$,

$Q_3 = \{3.5, 6, 7\}$ and

$Q_4 = \{4, 5, 6, 7\}$.

The first graph obtained with no edges is $g_5(G)$ and when we build in $g_3(G)$ the forest T_3 we already have the spanning tree T (figure 4.13).

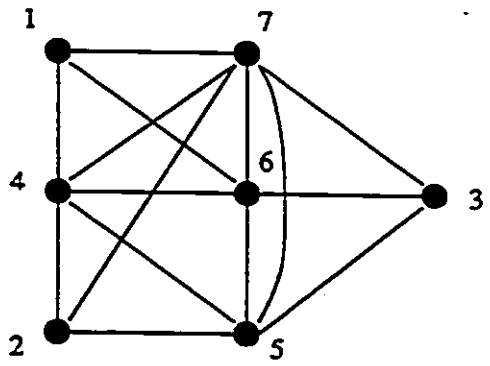

Figure 4.12: A graph G with cliques Q_1,Q_2,Q_3 and Q_4.

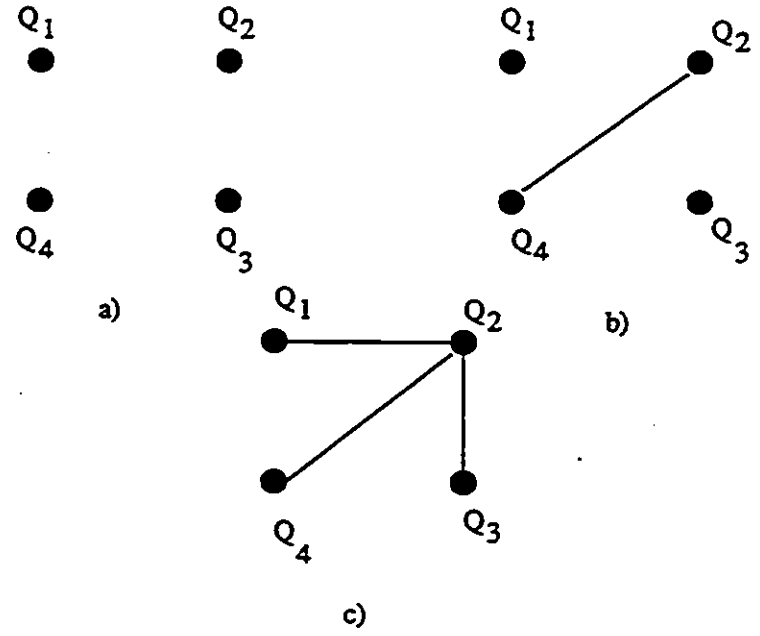

Figure 4.13: a) $g_5(G)$, b) $g_4(G)$ and c) $g_3(G)$.

4.3 Circular-Arc Graphs

Definition 15 *The intersection graph of a family of arcs on a circular ordered set is called a* **circular-arc graph.**

The problem of characterizing these graphs appears in [Had64] and [Kle69]. A characterization of these graphs was given by Tucker in [Tuc71] by a property of the adjacency matrix. Tucker also gave in [Tuc80] a recognition algorithm for circular-arc graphs that works in $O(n^3)$ time.

Definition 16 *A matrix whose entries are* 0 *or* 1 *is called a* **0-1 Matrix.**

Definition 17 *A* **circular 1's matrix** *is a matrix whose columns can be permuted such that the 1's in every row of the matrix appear in consecutive cycle order.*

Theorem 12 *A graph G is a circular-arc graph if and only if its corresponding adjacent matrix is a circular 1's matrix.*

Chapter 5

Fraternally Orientable Graphs (FOG)

We have already mentioned the importance that an orientation of a graph G has in Graph Theory. Among the different types of orientations that we can obtain from a graph G, we are interested in a special type of orientation called *fraternal*. In this chapter we proceed to study properties of graphs that have a fraternal orientation. We prove that the intersection graphs presented in the previous chapter — interval graphs, subtree graphs and circular-arc graphs — are all fraternally orientable.

Now we proceed to define this special orientation as follows:

Definition 18 *An orientation $\vec{G}$ of a graph G is called* **fraternal** *if for every three vertices u, v, w the existence of the edges $u \rightarrow w$ and $v \rightarrow w$ in $\vec{G}$ implies that u and v are adjacent in G.*

Definition 19 *An undirected graph G is* **fraternally orientable** *if there exists*

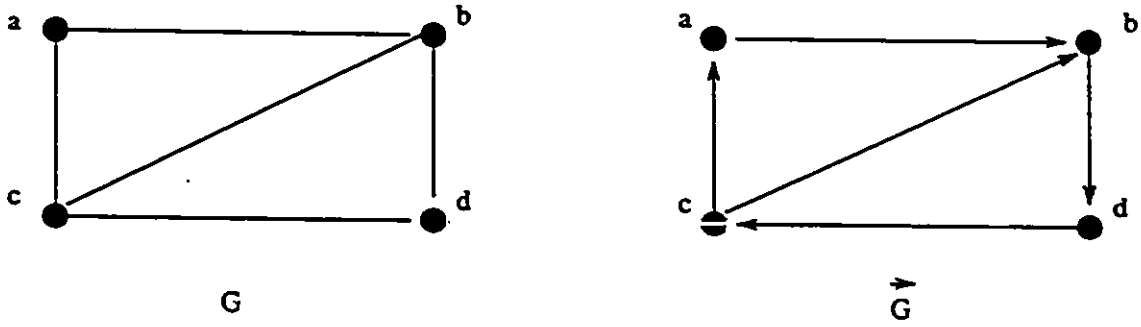

Figure 5.1: A fraternally oriented graph.

an orientation of its edges in such a way that the directed graph $\vec{G}$ obtained is fraternally oriented.

These graphs have also been studied under the name of $B_1 - graphs$ in [GaH92, GaH02, Skr82]. In figure 5.1 we can notice that a fraternally oriented graph can have directed cycles.

A characterization between fraternally orientable graphs and chordal graphs was given in [Ros70] by the following theorem:

Theorem 13 [Ros70] *A graph G is chordal if and only if there exists an acyclic fraternal orientation $\vec{G}$ of G.*

For example the graph G in figure 5.1 is a chordal graph. An acyclic orientation $\vec{G'}$ is given in figure 5.2.

A polynomial time algorithm for finding a fraternal orientation of an undirected graph G, if one exists, is given in [UrG92] and takes $O(|V||E|)$ time.

The intersection graphs that we have presented; interval graphs, subtree graphs and circular-arc graphs, are all fraternally orientable.

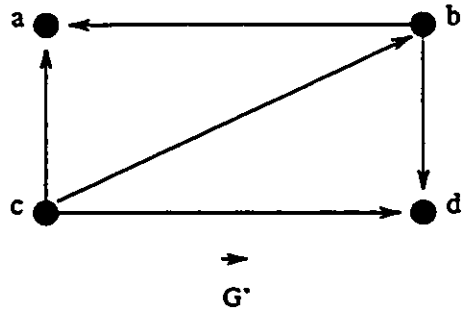

Figure 5.2: An acyclic fraternally oriented graph.

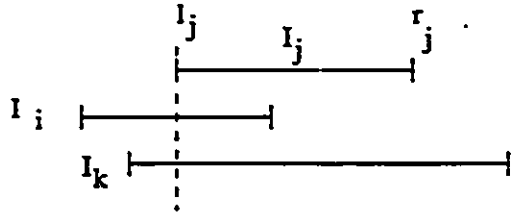

Figure 5.3: Intervals $I_j = [l_j, r_j]$, $l_j \in I_i, I_k$.

Theorem 14 *If G is an interval graph then G is fraternally orientable.*

PROOF. Let $\mathcal{F} = \{I_1, I_2, \ldots, I_n\}$ be the family of intervals that correspond to the interval graph G with $I_i = [l_i, r_i]$. If $I_i \cap I_j \neq \emptyset$ then $l_j \in I_i$ or $l_i \in I_j$. If $l_j \in I_i$ and $l_i \notin I_j$ then we orient the edge $v_i - v_j \in E(G)$ with $v_i \rightarrow v_j$. In the case that $l_i \in I_j$ and $l_j \in I_i$ orient $v_i - v_j$ either way. Suppose that this orientation is not a fraternal orientation. Then we have $v_i \rightarrow v_j$ and $v_k \rightarrow v_j$ such that $v_i - v_k \notin E(G)$. Since $v_i \rightarrow v_j$ then $l_j \in I_i$ and since $v_k \rightarrow v_j$ then $l_j \in I_k$, therefore $l_j \in I_i \cap I_k$ (figure 5.3); this contradicts the fact that G is an interval graph. □

In order to proceed with the proof that subtree graphs are fraternally orientable graphs we will need the following notation. Given a tree T we mark a point $v \in T$ as the root of T. We denote v by $r(T)$. We call $T(v)$ a *rooted tree* with root $r(T) = v$.

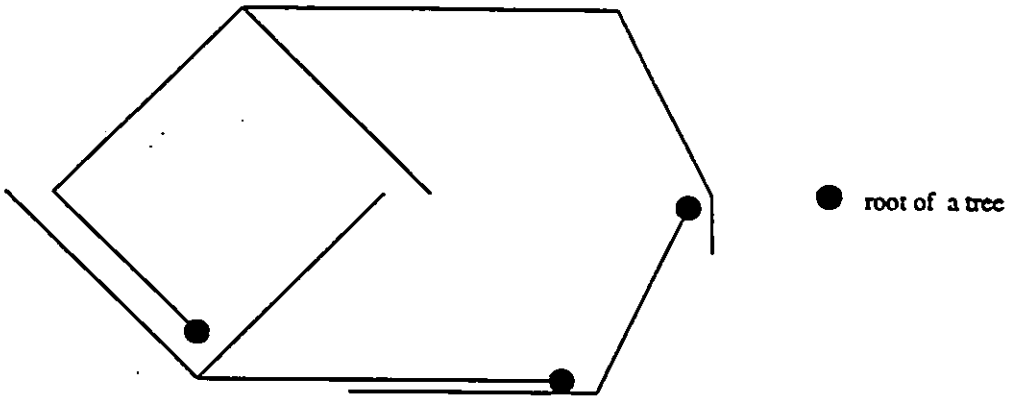

Figure 5.4: A family $\mathcal{T}$ of graftable subtrees.

Definition 20 *A family $\mathcal{T}$ of rooted subtrees of T is called* **graftable** *if for any two subtrees of $\mathcal{T}$ their intersection is empty or contains one of their roots* (figure 5.4).

The following result gives a relation between fraternally orientable graphs and a family $\mathcal{T}$ of graftable subtrees.

Theorem 15 [UrG92] *A graph G is the intersection graph of a family $\mathcal{T}$ of graftable subtrees of an undirected graph if and only if it is fraternally orientable.*

PROOF. Let $v \in V(G)$ be the vertex that represents the subtree $T_v \in \mathcal{T}$. For adjacent vertices u and v in the graph G we orient $u \to v$ if $r(T_v) \in T_u$ and $r(T_u) \notin T_v$. We give any orientation to the edge $u - v$ if $r(T_v) \in T_u$ and $r(T_u) \in T_v$. Given this orientation we can see that if we have three vertices u, v, and w with $u \to w$ and $v \to w$ then $r(T_w) \in T_u \cap T_v$. Therefore the vertices u and v must be adjacent in G.

Conversely let $\vec{G}$ be a fraternal orientation of G. For a vertex $v \in \vec{G}$ we build a subtree T_v with v as root and the outgoing edges of v in $\vec{G}$. Then if two vertices u

and v are adjacent in $\vec{G}$ with $u \rightarrow v$ we have that $v \in T_u$ and from the construction of T_v, $v \in T_v$ therefore $T_u \cap T_v \neq \emptyset$. In this way G is the intersection graph of the family $\mathcal{T}$ of subtrees.

Now we proceed to prove that $\mathcal{T}$ is a family of graftable subtrees. If a vertex $w \in T_u \cap T_v$, from the construction of our trees we have the oriented edges $u \rightarrow w$ and $v \rightarrow w$ since $\vec{G}$ is a fraternal orientation. Then the vertices u and v are adjacent and if $u \rightarrow v$ then $r(T_v) \in T_u$ or if $v \rightarrow u$ then $r(T_u) \in T_v$. □

Corollary 1 *Chordal graphs are fraternally orientable.*

PROOF. Let G be the intersection graph of a family $\mathcal{F} = \{T_1, T_2, \ldots, T_n\}$ of subtrees of a tree T. Choose a point $p \in T$, and call it the root of T. For any subtree $T_i \in \mathcal{F}$ of T let $r(T_i)$ be the closest point of T_i to p, or p if $p \in T_i$. It is easy to show that with this choice of roots $\mathcal{F}$ becomes graftable. □

Now we proceed to prove that circular-arc graphs are fraternally orientable.

Theorem 16 *If G is a circular-arc graph then G is a fraternally orientable graph.*

PROOF. We denote as A_v the arc corresponding to a vertex v. For each arc we mark as the root the first end-point that we find towards the counterclockwise direction and we orient $u, v \in G$ as $u \rightarrow v$ if the root of the arc A_v is contained in the arc A_u. In the case that the root of arc A_v is contained in the arc A_u and vice versa we orient $u - v$ either way. Suppose that we do not obtain a fraternal orientation. Then we have that $u \rightarrow v$ and $w \rightarrow v$ and $u - w \notin E(G)$. Since $u \rightarrow v$, the root of the arc A_v is contained in the arc A_u. Similarly since $w \rightarrow v$ the root of the arc A_v is contained in the arc A_w; therefore the root of the arc A_v is contained in the

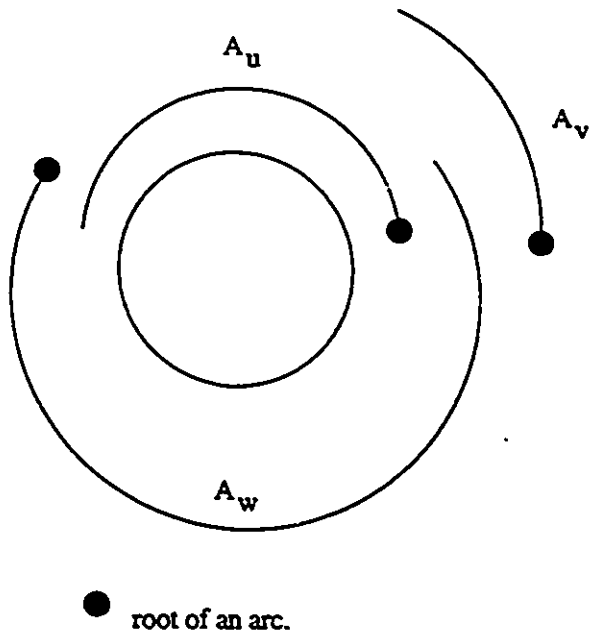

Figure 5.5: Circular-arc graph with arcs A_u, A_v and A_w.

intersection of the arcs A_u and A_w. This contradicts that G is a circular-arc graph (see figure 5.5). □

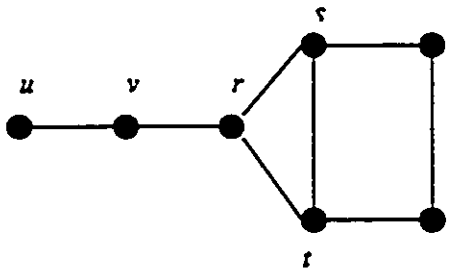

Figure 5.6: Example of a monocycle.

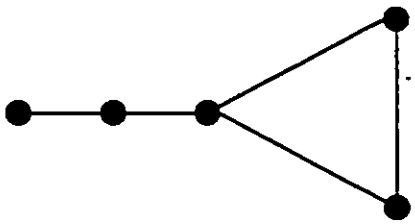

Figure 5.7: Example of a non-monocycle.

Definition 21 *A* **monocycle** $MC(C, P(u,v,r))$ *is a graph having a cycle* C, $|C| \geq 4$, *and a path* $P = v_1, \ldots, v_m$, $m \geq 2$ *such that* $v_1 = u, v_2 = v$ *and* $v_m = r$ *and satisfying:*

1. *the vertex set of* MC *is* $V(MC) = V(C) \cup V(P)$,
2. *the vertex* v_m *is adjacent to two consecutive vertices* s,t *of* C *or it is in* C *and its immediate neighbors in* C *are* s,t,
3. C *and* P *have no triangular chords in* MC *and* v_{m-1} *is distinct from and not adjacent to* s,t; *any other edges are allowed in* MC.

Notice that the graph shown in figure 5.7 is not a monocycle.

Definition 22 *A* **bicycle** *is a graph whose vertex and edge sets are the union of the vertex and edge sets of two monocycles* $MC(C_1, P_1(u,v,r_1))$ *and* $MC_2(C_2, P_2(v,u,r_2))$.

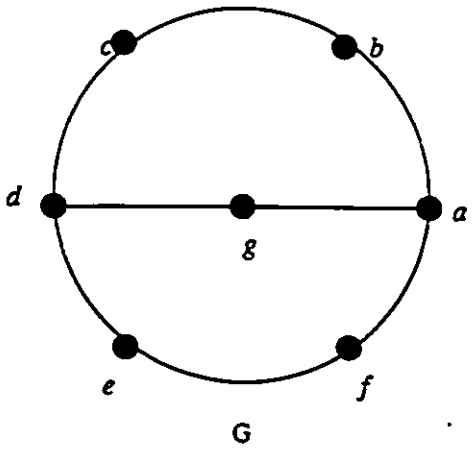

Figure 5.8: A bicycle.

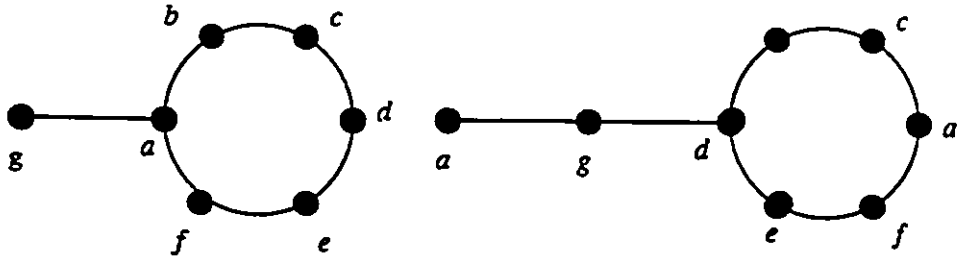

Figure 5.9: Monocycles.

In figure 5.8 we show a graph G, which apparently has no bicycles. However G contains two monocycles (figure 5.9) such that their union generates G.

We can notice that in the definition of a bicycle it is possible to have repetition of vertices and edges. Intuitively speaking we can say that if we "unfold" G we get the graph given in figure 5.10, which is a bicycle.

Theorem 17 *An undirected graph G has a fraternal orientation if and only if it has no bicycles.*

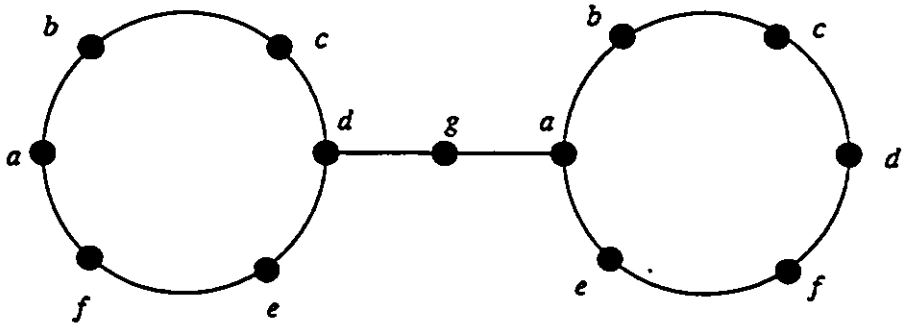

Figure 5.10: A bicycle.

5.1 Algorithm

A polynomial algorithm for finding a fraternal orientation graph if one exists was given in [UrG92]. This algorithm uses the procedure **ITER**$(u \to v, \vec{H}_{u\to v})$, where $\vec{H}_{u\to v}$ denotes the set of edges such that once we orient $u \to v$ an orientation of these edges is uniquely determined.

ITER$(u \to v; \vec{H}_{u\to v})$

1. $\vec{H}_{u\to v} = \{u \to v\}$,
2. **for all** $v - w \in E(G)$ such that $u - w \notin E(G)$ **do**

   if $v - w$ is unoriented **then**

   (a) Orient $v \to w$,

   (b) $\vec{H}_{u\to v} = \vec{H}_{u\to v} \cup \{v \to w\}$,

   (c) ITER$(v \to w, \vec{H}_{v\to w})$

   (d) $\vec{H}_{u\to v} = \vec{H}_{u\to v} \cup \vec{H}_{v\to w}$

   else

   if $v - w$ is oriented as $w \to v$ **then**

   (a) $\vec{H}_{u\to v}$ is not well defined,

   (b) STOP.

   else

   (a) $\vec{H}_{u\to v} = \vec{H}_{u\to v} \cup \vec{H}_{v\to w}$,

3. $\vec{H}_{u\to v}$ is well defined.

If during the execution of $\mathbf{ITER}(u \to v, \vec{H}_{u\to v})$ we try to assign opposing orientations to an edge $v - w$, we say that $\vec{H}_{u\to v}$ is not well defined. If $\vec{H}_{u\to v}$ is not well defined then there exists no fraternal orientation $\vec{G}$ in which $u - v$ is oriented as $u \to v$. Therefore if $\vec{H}_{u\to v}$ and $\vec{H}_{v\to u}$ are both not well defined then there exists no fraternal orientation in G.

It is easy to see that $x \to y \in \vec{H}_{u\to v}$ if and only if G has a walk with no triangular chords from $u - v$ to $x - y$. We can notice that if an edge $x \to y \in \vec{H}_{u\to v}$ then $\vec{H}_{x\to y} \subset \vec{H}_{u\to v}$.

We also have the following result:

Lemma 7 [UrG92] *If an edge is oriented in opposite directions by two sets $\vec{H}_{u\to v}$ and $\vec{H}_{w\to s}$ then $s \to w \in \vec{H}_{u\to v}$.*

PROOF. Suppose that an edge $x \to y \in \vec{H}_{u\to v}$ and $y \to x \in \vec{H}_{w\to s}$. Therefore there exists a walk on G from $u - v$ to $x - y$ with no triangular chords and a walk from $w - s$ to $y - x$ on G with no triangular chords. By inverting the second walk and linking it to the first one at the edge $x - y$ we obtain a walk from $u - v$ to $s - w$ with no triangular chords. Therefore $s \to w \in \vec{H}_{u\to v}$. □

The algorithm frater(G) ([UrG92]) obtains a fraternal orientation on a graph G. Here the set H_{u-v} is the set of undirected edges of the set $\vec{H}_{u\to v}$.

frater(G)

1. Let $F = \emptyset$
2. Let $F' = \emptyset$
3. **while** $F' \neq E$ **do**

(a) Take an unoriented edge $u - v \in E - F'$

(b) ITER($u \rightarrow v, \vec{H}_{u \rightarrow v}$)

(c) **if** $\vec{H}_{u \rightarrow v}$ **is not well defined then**

i) ITER($v \rightarrow u, \vec{H}_{v \rightarrow u}$)

ii) **if** $\vec{H}_{v \rightarrow u}$ **is not well defined then**

There exists no fraternal orientation on G,

STOP.

else

i) $F = F \cup \vec{H}_{v \rightarrow u}$,

ii) $F' = F' \cup H_{v-u}$.

else

i) $F = F \cup \vec{H}_{u \rightarrow v}$,

ii) $F' = F' \cup H_{u-v}$.

4. $\vec{G}(V, F)$ is a fraternal orientation.

Notice that in the algorithm **frater(G)** the edges that are permanently oriented are not deleted as in the algorithm **Rec-comp(G)** (chapter 3), in which in order to obtain a transitive orientation of the edges, the oriented edges are deleted.

Theorem 18 [UrG92] *The algorithm* **frater(G)** *finds a fraternal orientation of an undirected graph G, if one exists.*

PROOF. If the algorithm finishes with $F' = E$ then $\vec{G}(V, F)$ is a fraternal orientation. Since procedure **ITER**($u \rightarrow v, \vec{H}_{u \rightarrow v}$) obtains an orientation on the edges of

$\vec{H}_{u\to v}$, and by lemma 7, every successive execution of ITER($u \to v, \vec{H}_{u\to v}$) can be performed only in the set $E - F'$ of unoriented edges.

Conversely, suppose that G has a fraternal orientation $\vec{G}$. Take any oriented edge $u_1 \to v_1$ of $\vec{G}$ and execute ITER($u_1 \to v_1, \vec{H}_{u_1\to v_1}$) on G. The orientations obtained in the edges of the set $\vec{H}_{u_1\to v_1}$ are the same as $\vec{G}$. Take an edge $u_2 \to v_2 \notin \vec{H}_{u_1\to v_1}$ of $\vec{G}$ and execute ITER($u_2 \to v_2, \vec{H}_{u_2\to v_2}$). In this way we obtain a sequence of sets $\vec{H}_{u_1\to v_1}$, $\vec{H}_{u_1\to v_1}$, ..., $\vec{H}_{u_k\to v_k}$, which are well defined and their edges are oriented in G as in $\vec{G}$. Suppose that the algorithm frater(G) does not find a fraternal orientation on G. Therefore there exists an edge $u - v$ in which both sets $\vec{H}_{u\to v}$ and $\vec{H}_{v\to u}$ are not well defined. This edge $u - v$ is in one of the sets $\vec{H}_{u_i\to v_i}$. Suppose that it is oriented $u \to v$. Then we have that $\vec{H}_{u\to v} \subset \vec{H}_{u_i\to v_i}$ and this contradicts that $\vec{H}_{u_i\to v_i}$ is well defined. □

Now we proceed with the analysis of the algorithms. The procedure ITER($u \to v, \vec{H}_{u\to v}$) without recursive calls takes $O(\Delta)$ time. This generates $O(|E|)$ recursive calls. Therefore ITER($u \to v, \vec{H}_{u\to v}$) takes $O(\Delta|E|)$ time in total. The algorithm frater(G) executes the algorithm ITER($u \to v, \vec{H}_{u\to v}$) $O(|E|)$ times. Thus frater(G) works in $O(\Delta|E|^2)$ time.

To reduce the complexity, the following modification is applied: ITER($u \to v, \vec{H}_{u\to v}$) and ITER($v \to u, \vec{H}_{v\to u}$) are performed in parallel. If one of them ends with a well defined set $\vec{H}_{u\to v}$ or $\vec{H}_{v\to u}$, then stop both. In this way only when a well defined set $\vec{H}_{u\to v}$ or $\vec{H}_{v\to u}$ is obtained is it counted. We also have that in this case two subsequent calls of ITER orient disjoint set of edges. With this modification the algorithm frater(G) works in $O(\Delta|E|)$ time.

5.2 Transitive Reduction

Definition 23 *The* **transitive closure** *of a finite directed graph is obtained by adding a directed edge wherever the original graph contains a path from one vertex to other.*

For any graph, its transitive closure is uniquely determined. Furthermore, the transitive closure of an acyclic graph is itself acyclic.

Definition 24 *The* **transitive reduction** *of a finite directed graph is obtained by removing all edges whose absence do not affect the transitive closure.*

We denote the exterior neighborhood of a vertex v by $N^+(v)$ and the transitive reduction of a graph induced by v and $N^+(v)$ by $TR(N^+(v))$.

Notice that if there is an edge $u \to v$ in digraph $\vec{G}$ and there exists a directed path $\vec{P} = u, v_1, \ldots, v_n, v$ then the edge $u \to v$ does not appear in $TR(N^+(v))$

Lemma 8 *Let $\vec{G}$ be a fraternal orientation. Then for a vertex $v \in \vec{G}$ the transitive orientation of the graph induced by v and $N^+(v)$ is a tree.*

PROOF. Let u be some vertex in $\vec{G}$ such that $u \in TR(N^+(v))$. Suppose that the in-degree of $u > 1$ (figure 5.11).

Then we have $w \to u$ and $y \to u$ for some vertices $w, y \in TR(N^+(v))$. Since $\vec{G}$ is a fraternal orientation then $w - y \in G$. Without loss of generality we take $y \to w$. Then $w \to u$ should not be in $TR(N^+(v))$. This contradicts that we already have a transitive reduction. Therefore the in-degree of $u \leq 1$. Suppose

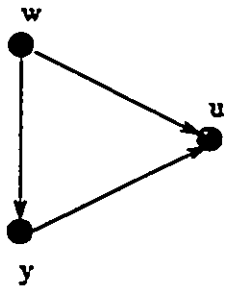

Figure 5.11: Transitive reduction $TR(N^+(v))$.

that $u \in TR(N^+(v))$ and the in-degree of u is equal to zero. Then we have that $TR(N^+(v))$ is not a transitive reduction since u is not reached from v. Let n be the number of vertices in the induced graph obtained from the vertices v and $N^+(v)$. Since for all $u \in TR(N^+(v))$ the in-degree is equal to one, we have a connected graph with $n-1$ edges; therefore $TR(N^+(v))$ is a tree. □

Lemma 9 *Let $\vec{G}$ be an acyclic fraternally oriented graph. If we delete an edge $w \to y$ when we take $TR(N^+(v))$ with $v \in \vec{G}$ then if $w, y \in N^+(u)$ with $u \in \vec{G}$ the edge $w \to y$ is also deleted when we take $TR(N^+(u))$.*

PROOF. Since we delete $w \to y$ when we take $TR(N^+(v))$, this means that there is a directed path $\vec{P} = v_0, \ldots, v_p$ in $TR(N^+(v))$ with $v_0 = w$ and $v_p = y$. Since $w, y \in TR(N^+(u))$ we have $u \to w$, $u \to y$. We have $v_{p-1} \to y$, since $\vec{G}$ is a fraternal orientation. Then we have $u - v_{p-1} \in E(G)$ with orientation $u \to v_{p-1}$, otherwise we obtain a cycle $u, v_0, v_1, \ldots, v_{p-1}$, contradicting that $\vec{G}$ is acyclic.

We apply the same argument to edges $u \to v_{p-1}$ and $v_{p-2} \to v_{p-1}$, and so on. In this way we get that the vertex u is adjacent to all vertices v_i in the directed path $(i = 0, \ldots, p)$. The path $\vec{P}$ is contained in $N^+(u)$ and therefore when we take $TR(N^+(u))$ we have to delete the edge $w \to y$. □

Theorem 19 *Let G be a connected graph and $\vec{G}$ an acyclic fraternal orientation of G. If we take $TR(N^+(v))$ for all $v \in V(G)$ then the resulting graph is a tree.*

PROOF. We take the transitive orientation $TR(N^+(v))$ for all vertices $v \in V(G)$. Suppose we have some vertex v with in-degree greater than one. Then we have two different vertices u, w such that $u \rightarrow v$ and $w \rightarrow v$. Since $\vec{G}$ is a fraternal orientation, then $u - w \in E(G)$. Without loss of generality we assume $u \rightarrow w$ but this contradicts that we take the transitive reduction $TR(N^+(u))$. Therefore all vertices have in-degree smaller than or equal to one. By lemma 9 we know that if we delete an edge $w - y$ when we take a transitive reduction $TR(N^+(v))$ for some vertex $v \in V(G)$ it has to be deleted when a transitive reduction $TR(N^+(u))$ is taken for a vertex $u \in V(G)$ such that $w, y \in N^+(u)$.

Therefore when we take the transitive reduction of all vertices of a graph G, we do not delete edges that are contained in a transitive reduction of some vertex.

Suppose the graph obtained in the transitive reduction $TR(N^+(v))$ for all $v \in V(G)$ has a vertex with in-degree equal to zero. This contradicts the fact that G is a connected graph.

The in-degree of the vertices in the new graph obtained is equal to one. If $|V| = n$ then the new graph obtained has $n - 1$ edges and since G is connected, the new graph obtained is a tree. □

Chapter 6

Hamiltonian Cycles in FOG

In this chapter we present our main result. We study a special case of the Hamiltonian Cycle Problem in Fraternally Orientable Graphs. In the previous chapter we saw that interval graphs, circular-arc graphs and chordal graphs are fraternally orientable. Results obtained for the Hamiltonian Cycle Problem in some of these graphs are the following:

- In [Kei85] Keil showed that a Hamiltonian Cycle could be constructed for interval graphs in time and space $O(|V| + |E|)$. Keil's algorithm has the same complexity as Booth and Lueker's interval graph recognition algorithm [BLe76] so it is optimal in the size of the underlying graph. Given a set of intervals Manacher, Mankus and Smith [Man90] gave an $O(n \log n)$ time algorithm, where n is the number of intervals.

- A linear algorithm to find a Hamiltonian Cycle on circular-arc graphs was proposed by Bonuccelli and Bovet [BBo79]. It was later pointed out that their algorithm was incorrect by [HSU92] and by [Man90] independently. Recently

Chang [Cha93] gives an $O(n)$ time algorithm on circular-arc graphs, where an arc model with the end points of arcs sorted is given and n is the number of arcs.

- In [Ber86] it was proved that the Hamiltonian Cycle Problem for undirected path graphs is equivalent to finding a Hamiltonian Cycle in a bipartite graph with maximum vertex degree 3, which is NP-complete. Hence for chordal graphs the problem is NP-complete. Undirected path graphs are contained in chordal graphs, therefore the problem is NP-complete on chordal graphs.

In the previous chapter we saw that a graph G is chordal if and only if G is a subtree graph. We also proved that subtree graphs are fraternally orientable. Therefore if we can solve the Hamiltonian Cycle Problem for fraternally orientable graphs, since chordal graphs are contained in such graphs, we can solve the Hamiltonian Cycle Problem for chordal graphs. But the Hamiltonian Cycle Problem for chordal graphs is known to be NP-complete. For that reason we restrict ourselves to a special case of fraternally orientable graphs. Our objective is to show that every strongly connected fraternally oriented graph is Hamiltonian. We also give an $O(|E|)$ time algorithm to find a Hamiltonian Cycle in any such graph.

In the rest of this thesis, all orientations are assumed to be fraternal.

In order to present our result we define the following concept:

Definition 25 *Let C_n be a directed cycle and P_m a directed path with vertices $v_1, v_2, \ldots, v_m$. We say that P_m is* **attached** *to C_n if $v_1, v_m \in V(C_n)$ and $v_i \notin V(C_n)$, $i \neq 1, m$.* (Figure 6.1)

The following lemma is fundamental in order to prove our main result.

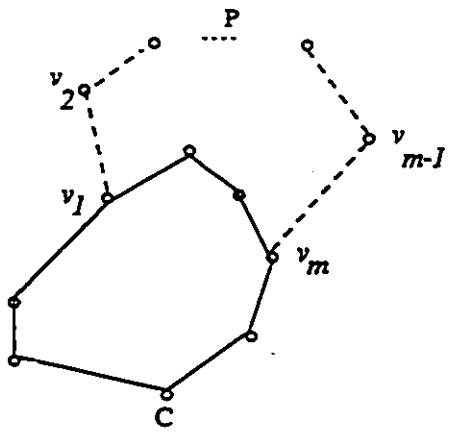

Figure 6.1: A path P attached to a cycle C.

Lemma 10 *Let C_n be a directed cycle in $\vec{G}$ and P_m a path of $\vec{G}$ attached to C_n. Then there is a directed cycle C' of $\vec{G}$ with*

$$V(C') = V(C_n) \cup V(P_m).$$

PROOF. Let $V(C_n) = \{c_1, \ldots, c_n\}$ be the vertex set of the directed cycle C_n and let the vertex set of the path P_m attached to C_n be $V(P_m) = \{v_1, \ldots, v_m\}$. Without loss of generality we assume $v_1 = c_j$ and $v_m = c_k$ with $j < k$.

Our proof proceeds by induction on $|P_m|$ with $m \geq 3$.

Let $m = 3$. We notice that since $v_2 \rightarrow c_k$ then v_2 and c_{k-1}, the predecessor of c_k in C_n, must be adjacent. If the orientation is $v_2 \rightarrow c_{k-1}$ then v_2 and c_{k-2}, the predecessor of c_{k-1} in C_n, must be adjacent, and so on.

Let l be the largest index $j \leq l < k$ in such a way that $c_l \rightarrow v_2$. Then $v_2 \rightarrow c_{l+1}$ and therefore

$$C' = c_1, c_2, \ldots, c_l, v_2, c_{l+1}, \ldots, c_n$$

is the new directed cycle.

Now let P_m be a path attached to C_n with $m > 3$.

We could have either one of the following cases:

1. there is no edge $c_i \rightarrow v_{m-1}, 1 \leq i \leq n$ or
2. there is an edge $c_i \rightarrow v_{m-1}$, $j \leq i \leq k$.

Case 1:

In general, if $v_{m-1} \rightarrow c_i$ for some $c_i \in V(C_n)$ then v_{m-1} and c_{i-1}, the predecessor of c_i in C_n, must be adjacent, since $\vec{G}$ is a fraternally oriented.

In this case, since $v_{m-1} \rightarrow c_k$ then v_{m-1} and c_{k-1}, the predecessor of c_k in C, must be adjacent. By hypothesis $v_{m-1} \rightarrow c_{k-1}$. Again since we have the edge $v_{m-1} \rightarrow c_{k-1}$ then v_{m-1} and c_{k-2}, the predecessor of c_{k-1} in C_n, are adjacent and so on. Thus v_{m-1} is adjacent to all the vertices in C_n.

Therefore we can build a new cycle C' as follows:

$$C' = c_1, c_2, \ldots, c_j, v_2, v_3, \ldots, v_{m-1}, c_{j+1}, \ldots, c_n$$

and C' is the new directed cycle containing $V(P_m) \cup V(C_n)$. Figure 6.2 illustrates this case.

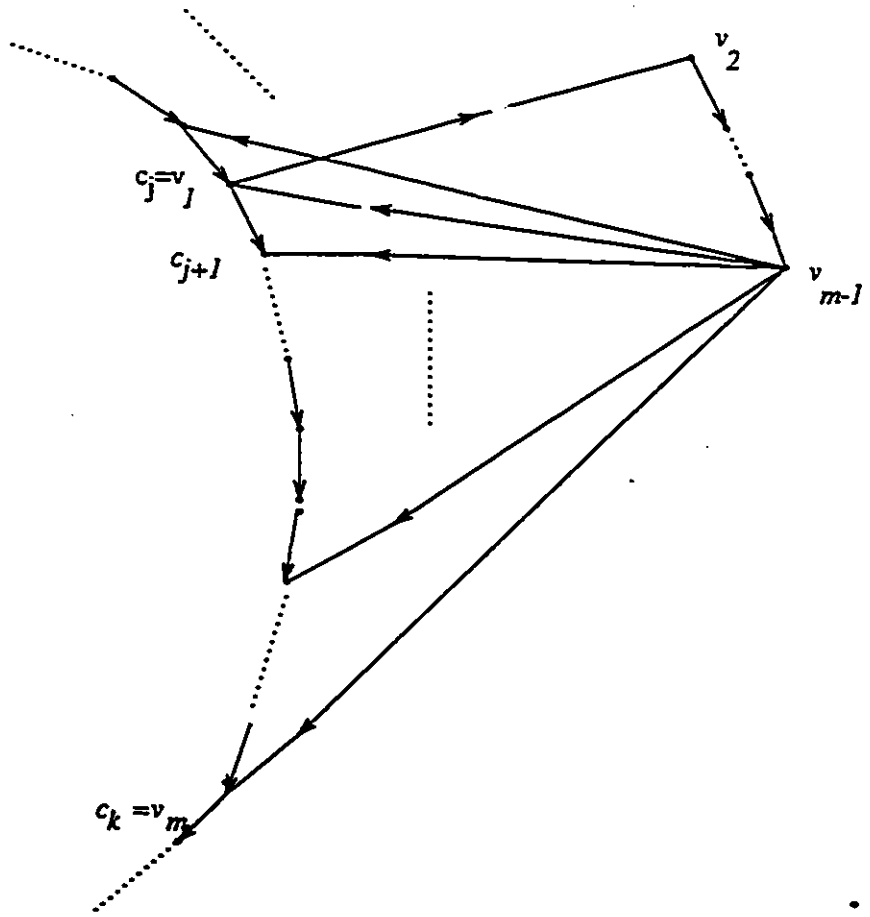

Figure 6.2: Case 1.

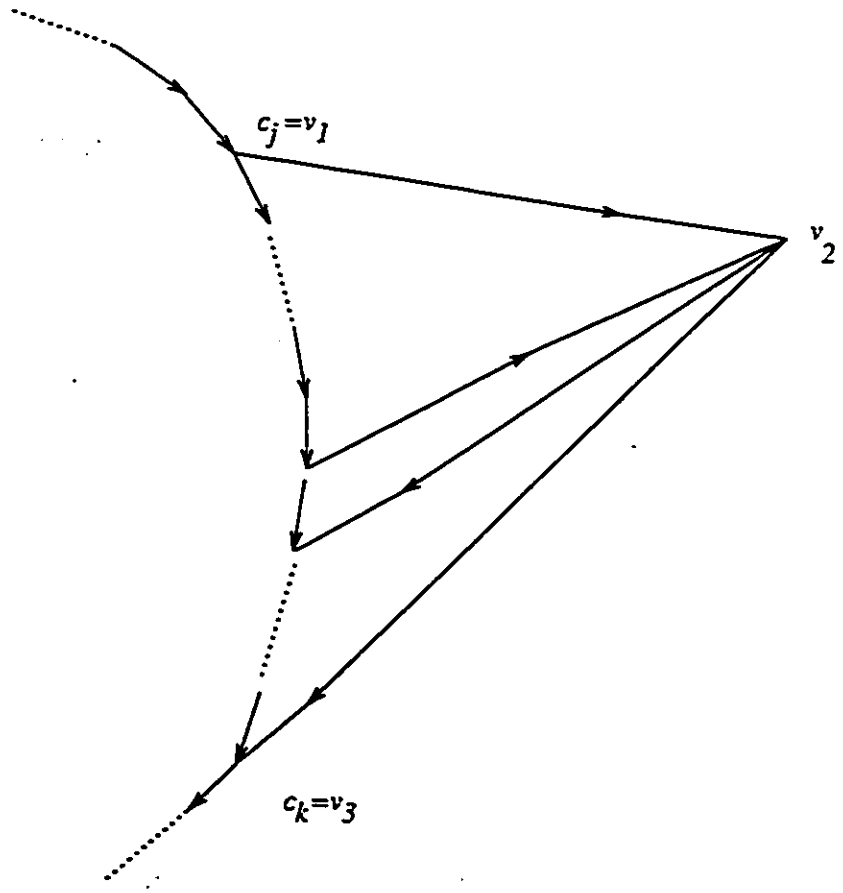

Figure 6.3: Case 2.

Case 2:

Again let l be the largest index $j \leq l < k$ such that $c_l \rightarrow v_{m-1}$, clearly $v_{m-1} \rightarrow c_{l+1}$ and the cycle

$$C_{n+1} = c_1, c_2, \ldots, c_l, v_{m-1}, c_{l+1}, \ldots, c_n$$

is a new directed cycle and the path

$$P_{m-1} = v_1, \ldots, v_{m-2}, v_{m-1}$$

is attached to C_{n+1}. By induction on the size of P_{m-1}, we know that there is a cycle C of $\vec{G}$ with $V(C) = V(C_{n+1}) \cup V(P_{m-1})$ and we take $C' = C$. □

Now we are ready to prove our main result:

Theorem 20 *If a graph $\vec{G}$ is strongly connected and fraternally oriented then $\vec{G}$ is hamiltonian.*

PROOF. Assume that C is the largest directed cycle in $\vec{G}$ and there exists a vertex $v \in \vec{G}$ with $v \notin V(C)$. Since the graph $\vec{G}$ is strongly connected there exist paths P_1 and P_2 such that:

1. $P_1 = u_1, \ldots, u_m$ with $u_1 = c_j$ for some $c_j \in V(C)$, $u_m = v$ and $u_i \notin V(C)$ for all $i \neq 1$.

2. $P_2 = w_1, \ldots, w_{m'}$ with $w_1 = v$ and $w_{m'} = c_k$ for some $c_k \in V(C)$, $w_i \notin V(C)$ for all $i \neq m'$ and $j \neq k$.

If $E(P_1) \cap E(P_2) = \emptyset$ then let the path $P = P_1 \cup P_2$. We can see that P is an attached path to C, and by lemma 10, we can build a new cycle with vertices $V(C) \cup V(P)$, which contradicts the fact that C is the largest cycle in $\vec{G}$.

If $E(P_1) \cap E(P_2) \neq \emptyset$ then let $u_{i_0} \in V(P_1)$, the vertex such that $i_0 = \min_i\{u_i \in V(P_1) \cap V(P_2)\}$. We build the paths $P_1' = u_1, \ldots, u_{i_0}$ and $P_2' = w_j, \ldots, w_m'$, where $w_j = u_{i_0}$ and we take $v = u_{i_0}$. In this way we have that $E(P_1') \cap E(P_2') = \emptyset$ and therefore we can proceed as we mentioned above with $P = P_1' \cup P_2'$ as an attached path to C. □

6.1 Algorithm

Let $\vec{G}$ be a strongly connected fraternally oriented graph.

The algorithm that we present below uses the result given in lemma 10. We try to give an intuitive idea before we present this algorithm.

We proceed first to find a directed cycle C in $\vec{G}$. Then we find a path P attached to C. Now using lemma 10 we can build a new cycle C' in such a way that $V(C') = V(C) \cup V(P)$.

If all the vertices of $\vec{G}$ are contained in C' we finish the process, since C' is a directed Hamiltonian Cycle of $\vec{G}$. Otherwise we repeat the previous procedure until a Hamiltonian Cycle is obtained.

In order to find a path P attached to C we proceed as follows. We begin to create a family of rooted trees with roots contained in C. We grow the rooted trees until some rooted tree $T(v)$ reaches a vertex in C. Thus we find a path P contained in $T(v)$ attached to C. The path P is uniquely determined in $T(v)$ from its root to the vertex that reached C. At this point we obtain a new cycle C' with $V(C') = V(C) \cup V(P)$. We take P out from $T(v)$. Notice that $T(v) - E(P)$ is a collection of rooted trees with roots in C'.

With only this idea in mind we can get an $O(\Delta|E|)$ algorithm.

To improve the complexity of our algorithm we create the following labels:

1. *unprocessed-edge* and *unprocessed-vertex*. At the beginning all vertices and edges are labeled unprocessed; their labels change as our algorithm is executed.

2. Leaves is a list of all leaf vertices of all rooted trees.

3. *active-edge* is an edge label to indicate an edge of some rooted tree reaches a vertex in the cycle C.

4. Active is a list of *active-edges*.

5. *tree-edge* is an edge label to indicate an edge that belongs to some rooted tree.

6. *tree-vertex* is a vertex label to indicate a vertex that belongs to some rooted tree.

7. *sleeping-edge* is an edge label to indicate those edges that join two *tree-vertices*; these vertices could be in the same rooted tree or in different ones.

8. *dead-edge* is an edge label to indicate those edges that are not in the cycle obtained, but they join vertices in the cycle, i.e. diagonals of the cycle.

9. *cycle-edge* is an edge label to indicate those edges that belong to the cycle.

10. *cycle-vertex* is a vertex label to indicate that a vertex belongs to the cycle.

Accordingly at any stage during the execution of our algorithm that we present next, the algorithm keeps track of the following:

1. A directed cycle C.

2. A set of rooted subtrees $\{T_1, \ldots, T_p\}$ with roots in C.

3. The set Leaves containing all the leaves of $T_1, \ldots T_p$.

4. The set Active containing edges joining a *tree-vertex* to a vertex in C.

5. The labels on the edges of $\vec{G}$.

See figures 6.4, 6.5 and 6.6.

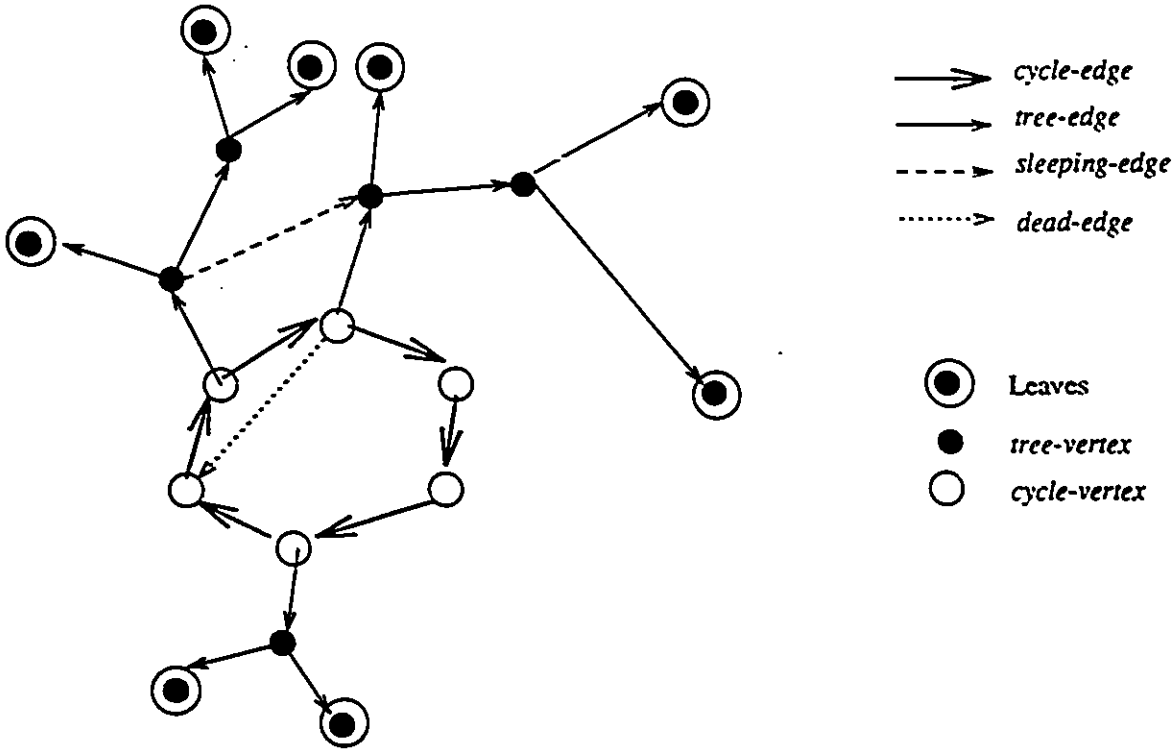

Figure 6.4: The rooted trees are grown.

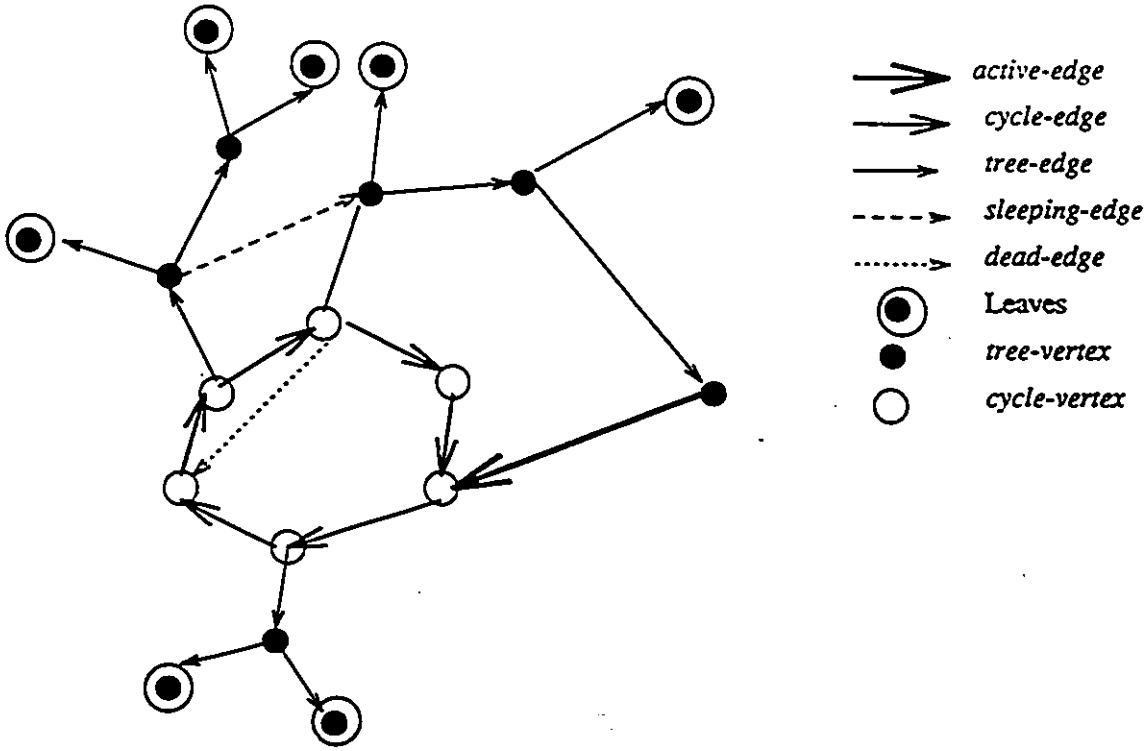

Figure 6.5: An *active-edge* is found.

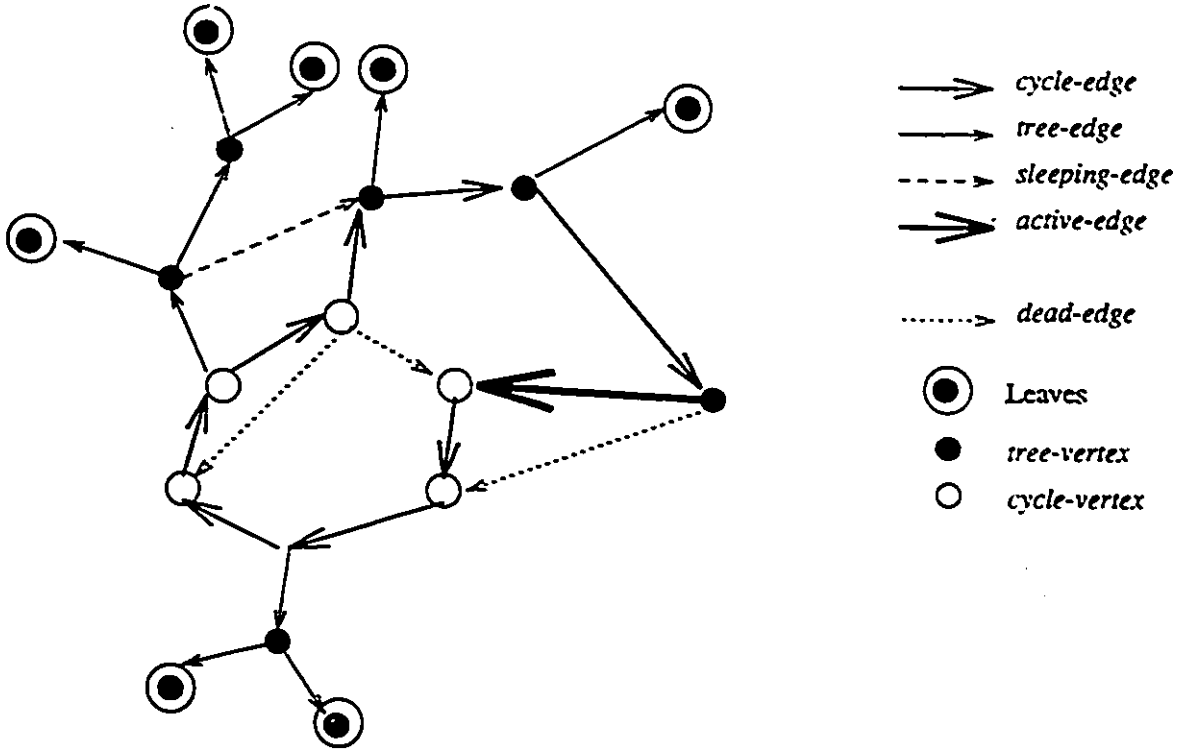

Figure 6.6: The cycle is grown.

We now proceed to present an $O(|E|)$ algorithm that finds a Hamiltonian Cycle in $\vec{G}$.

Algorithm FOG-forest

1. Initialize Leaves $= \emptyset$
2. Initialize Active $= \emptyset$

3. Find a cycle C in $\tilde{G}$.

4. Find a vertex $v \in V(C)$ with $(v \rightarrow v_i)$ and $v_i \notin V(C)$.

5. **if** there is no such vertex v **then** go to step 9.

6. Create a rooted tree $T(v)$ with root v:

   for all $(v \rightarrow v_i)$ **do**

   if $v_i \notin V(C)$ **then**

   (i) Mark $(v \rightarrow v_i)$ as *tree-edge*,

   (ii) Mark the vertex v_i as *tree-vertex*,

   (iii) Leaves = Leaves $\cup\{v_i\}$

   else

   (i) Mark $(v \rightarrow v_i)$ as *dead-edge*.

7. **while** Leaves $\neq \emptyset$ **do**

   (a) Take a vertex $v_j \in$ Leaves

   (b) Update Leaves:

   (i) Leaves = Leaves $-\{v_j\}$,

   (c) **while** $v_k \notin V(C)$ for $(v_j \rightarrow v_k)$ **do**

   if v_k **is not a** *tree-vertex* **then**

   (i) Mark $(v_j \rightarrow v_k)$ as *tree-edge*.

   (ii) Mark v_k as *tree-vertex*

(iii) Leaves = Leaves $\cup\{v_k\}$.

else

(i) Mark $(v_j \rightarrow v_k)$ as *sleeping-edge*

(d) Mark $(v_j \rightarrow v_k)$ as *active-edge*.

(e) **Build-Attached**$(P, v_j \rightarrow v_k)$

(f) **Grow-cycle**(C, P).

(g) **while** Active $\neq \emptyset$ **do**

for each edge $(v_j \rightarrow v_k) \in$ Active **do**

Active = Active $-\{(v_j \rightarrow v_k)\}$.

if $v_j \notin V(C)$ **then**

(i) **Build-Attached**$(P, v_j \rightarrow v_k)$

(ii) **Grow-cycle**(C, P).

else

(i) Mark $(v_j \rightarrow v_k)$ as *dead-edge*.

if $(v_j \rightarrow v_k)$ **is not a** *cycle-edge* **then**

(i) Mark $(v_j \rightarrow v_k)$ as *dead-edge*.

8. **if there is** $v \notin V(C)$ **then go to step 4.**

9. STOP

The subroutine **Build-Attached**$(P, v_j \rightarrow v_k)$, whose parameters are a path P and an edge $(v_j \rightarrow v_k)$, obtains a path P attached to C with last edge $(v_j \rightarrow v_k)$.

6.1.1 Build-Attached$(P, v_j \rightarrow v_k)$

1. Initialize P

   $P = (v_j \rightarrow v_k)$.

2. $w = v_j$.

3. **while** $w \notin V(C)$ **do**

   (a) $P = P \cup \{(parent(w) \rightarrow w)\}$.

   (b) **if** there is an edge $(v_i \rightarrow w) =$ *sleeping-edge* **then**

   Mark $(v_i \rightarrow w)$ as *active-edge*.

   Active = Active $\cup \{(v_i \rightarrow w)\}$.

   (c) $w = parent(w)$

The subroutine **Grow-cycle**(C, P), whose parameters are a cycle C and a path P attached to C, obtains a new cycle with vertex set $V(C) \cup V(P)$.

Let $P = v_1, \ldots, v_m$ and $C = c_1, \ldots, c_n$ and $v_1 = c_j$, $v_m = c_k$, with $j < k$.

6.1.2 Grow-cycle (C, P)

1. **while** $P \neq \emptyset$

if $(v_{m-1} \to c_{j+1})$ **then**

(i) $C = c_1, \ldots, c_j, v_2, \ldots, v_{m-1}, c_{j+1}, \ldots, c_n$

(ii) Mark the edges $\neq (v_{m-1} \to v_m)$ in P as *cycle-edges*

(iii) Mark the vertices of P as *cycle-vertex*.

(iv) Mark $(c_j \to c_{j+1})$ as *dead-edge*

(iv) $P = \emptyset$

else

(a) $i = k - 1$

(b) **while** $(v_{m-1} \to c_i)$ and $i \neq j + 1$ **do**

(i) Mark $(v_{m-1} \to c_i)$ as *dead-edge*.

(ii) $i = i - 1$.

(c) $C = c_1, \ldots, c_i, v_{m-1}, c_{i+1}, \ldots, c_k$

(d) Mark the edges $(c_i \to v_{m-1})$ and $(v_{m-1} \to c_{i+1})$ as *cycle-edges*.

(e) Mark v_{m-1} as *cycle-vertex*.

(f) Mark $(c_i \to c_{i+1})$ as *dead-edge*

(g) $P = v_1, \ldots, v_{m-1}$.

(h) $m = m - 1$.

(i) $k = i + 1$.

Now we give an explanation of the algorithm.

Recall that we use rooted trees with roots contained in the cycle obtained by the algorithm at some point.

First we find a cycle C in $\tilde{G}$. From a vertex $c_{i_0} \in C$ we build a rooted tree $T(c_{i_0})$ with root c_{i_0}. The rooted tree $T(c_{i_0})$ is built with edges $c_{i_0} \rightarrow v_i$ such that $v_i \notin C$. Each vertex v_i is a leaf vertex, therefore we insert it in the Leaves list.

From now on as long as Leaves $\neq \emptyset$ we proceed as follows:

Take a vertex v_i out of Leaves. We grow the subtree containing v_i by adding the exterior edges of v_i to it. The following cases arise:

1. The vertex v_j is a *tree-vertex*, that is $(v_i \rightarrow v_j)$ is an edge that joins two rooted trees. Then we mark $(v_i \rightarrow v_j)$ as *sleeping-edge*.

2. The vertex v_j is not a tree-vertex, nor a cycle vertex. Then we mark $(v_i \rightarrow v_j)$ as *tree-edge*, v_j as *tree-vertex* and we insert v_j in the list Leaves.

3. The vertex $v_j \in V(C)$; then the edge $(v_i \rightarrow v_j)$ is marked as *active-edge*.

As soon as an *active-edge* is found, we build an attached path P by using the subroutine **Build-Attached**. Next we incorporate the vertices of P into C using **Grow-cycle**.

Let $T(v)$ be the rooted subtree containing P. Then deleting the edges of P from $T(v)$, we obtain a new set of rooted subtrees with roots in P since the vertices in P now become vertices of C. At this point it is important to notice that the set of Leaves remains unchanged and access to our new trees can be achieved using their leaves which are already in Leaves. Thus we do not need to store any changes to the set of rooted subtrees other than marking the new roots of these new subtrees. Notice however that this can be accomplished by simply changing the label of the vertices of P from *tree-vertices* to *cycle-vertices*. This can be done when we grow C.

While finding P, it could happen that a *sleeping-edge* $v_a \rightarrow v_b$ is such that $v_b \in P$. In this case, we change the label of $v_a \rightarrow v_b$ to *active-edge* and insert it into Active. This is because once we extend C, the vertices of P will be in C.

While extending C to incorporate the vertices of P, it may happen that an edge $u \rightarrow v$ that belongs to C is dropped out of C. These edges will not be considered any more by our algorithm and are labeled as *dead-edges* (in practice we could think of these edges as deleted from $\vec{G}$).

As long as Active is nonempty, we proceed as follows:

Take out an edge $u \rightarrow v$ from Active. Two cases arise:

1. both ends of $u \rightarrow v$ are in C. In this case we mark $u \rightarrow v$ as *dead-edge*,
2. $u \notin C$. In this case we proceed to find an attached path P to C ending in $u \rightarrow v$ by using Build-Attached and grow C again.

Observation 1. At this point we would like to emphasize that while our algorithm needs to keep track of a family of disjoint rooted subtrees of $\vec{G}$ in order to build paths P attached to C, in practice all we need to build P is to attach to each *tree-vertex* v a pointer indicating where the father of v is.

Moreover access to these trees can be done by keeping track only of the set of their leaves. It now follows that our basic operations that are "grow" and "find an attached path to C" can be performed using only the list Leaves and the pointers to the parents of the *tree-vertices*.

In order to identify the attached paths that were found, we take out an edge from the Active list and we walk backwards on its rooted tree until we find the root.

Theorem 21 *For a strongly connected fraternally oriented graph $\vec{G}$ the algorithm FOG-forest obtains a hamiltonian directed cycle of $\vec{G}$.*

PROOF. Let C be the cycle obtained by the algorithm FOG-forest Suppose there is a vertex $v \notin V(C)$. The algorithm stops in step 4 when we can not find a vertex $v_i \in V(C)$, such that $(v_i \rightarrow v_k)$ with $v_k \notin V(C)$. Then there is no path from the vertices in C to v. This contradicts that $\vec{G}$ is strongly connected. □

We now proceed with the complexity analysis of our algorithm.

During the execution of FOG-forest an edge $(u \rightarrow v)$ could receive the following labels: *tree-edge*, *active-edge*, *sleeping-edge*, *cycle-edge*, and/or *dead-edge*; initially all edges are labeled *unprocessed-edges*.

Notice that edges that are diagonals of a cycle are not used in the construction of new cycles. If we find a diagonal edge we mark it as *dead-edge*.

It is easy to show that the labels on the edges of $\vec{G}$ can change according to the following diagram:

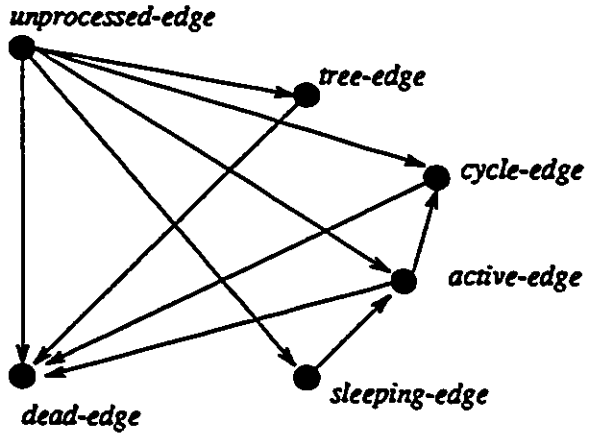

Figure 6.7: edge labels

For example an *unprocessed-edge* $u \rightarrow v$ could change from *unprocessed-edge* to *sleeping-edge*, if the first time we use $u \rightarrow v$ both ends u and v were already labeled as *tree-vertex* (this happens when we try to extend the rooted tree containing u as a leaf). At this point $u \rightarrow v$ will change label when an attached path P that contains v is found; in this case $u \rightarrow v$ will become an *active-edge*. Next when we incorporate the vertices of P into C according to Grow-cycle $u \rightarrow v$ may become a *cycle-edge* or a *dead-edge*.

Similarly we can verify the remaining cases illustrated in figure 6.7. Since the graph shown in figure 6.7 is acyclic and each time an edge is processed its label changes, it follows that every edge in $\vec{G}$ is considered at most 4 times, i.e. the length of the longest directed path in figure 6.7.

Then we can see that our algorithm processes each edge of $\vec{G}$ a constant number of times. Therefore our algorithm FOG-forest takes $O(|E|)$ time.

Bibliography

[Ben59] Benzer S. On the topology of the genetic fine structure, *Proc. Nat. Acad. Sd.* USA 45 (1959), 1607-1620.

[Ber81] Bertossi A.A. The edge Hamiltonian path problem is NP-complete, *Inform. Process. Lett.* 13 (1981) 157-159.

[Ber86] Bertossi A.A. and Bonuccelli M.A. Hamiltonian circuits in interval graph generalizations, *Inform. Process. Lett.* 23 (1986) 195-200.

[BMu76] Bondy J.A. and Murty U.S.R. *Graph Theory with Applications*, North-Holland, New York (1976).

[BBo79] Bonuccelli M.A. and Bovet D.P. Minumum Node Disjoint Path Covering for Circular-Arc Graphs, *Inform. Process. Lett.* 8 (1979) 159-161.

[BLe76] Booth K.S. and Lueker G.S., Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, *J. comput. Syst. Sci.* 13, (1976) 335-379.

[Cha93] Chang M-S., Peng S-L. and Liaw J-L. Deferred Query An efficient Approach for Problems on Interval and Circular-Arc Graphs, *Lecture Notes*

in Computer Science (709), Springer-Verlag (1993), F. Dehne, J.R. Sack, N. Santoro, S. Whitesides (e).

[Cohen] Cohen J., Interaxal graphs and food webs, *The RAND Corporation* D-17696-PR.

[Dam89] Damaschke P. The hamiltonian Circuit problem for circle graphs is NP-complete, *Inform. Process. Lett.* 32 (1989) 1-2.

[EET76] Ehrlich G., Even S. and Tarjan R.E. Intersection Graphs of Curves in the Plane, *J.C.T.* (B) 21, 8-20 (1976).

[Ful65] Fulkerson D.R. and Gross O.A., Incidence matrices and interval graphs , *Pacif. J. Math.* 15 (1965) 835-855.

[GaH92] Galeana H. B_1 and B_2-orientable graphs in kernel theory, *Technical Report (160). Instituto de Matematicas.* Universidad Autonoma de Mexico (UNAM).

[GaH02] Galeana H. A characterization of normal B_1-orientable Perfect Graphs, *Technical Report (248). Instituto de Matematicas.* Universidad Autonoma de Mexico (UNAM).

[GJT76] Garey M.R., Johnson D.S. and Tarjan R.E. Hamiltonian circuit problem is NP-complete, *SIAM J. Comput.* 5 (1976) 704-714.

[Gav74] Gavril F. The Intersection Graphs of Subtrees in Trees Are exactly the Chordal Graphs, *Journal of Combinatorial Theory* (B) 16, (1974), 47-56.

[GHo64] Gilmore P.C. and Hoffman A.J., A characterization of comparability graphs and of interval graphs, *Canad. J. Math.* 16, 539-548, (1964).

[Gol80] Golumbic M.C. *Algorithmic Graph Theory and Perfect Graphs*, Academic press, New York, 1980.

[Gou82] Gouyou-Beauchamps D. The Hamiltonian circuit problem is polynomial for 4-connected planar graphs, *SIAM J. Comput.* 11 (3) (1982) 529-539.

[Had64] Hadwiger H., Debrunner H., and Klee V. *Combinatorial Geometry in the Plane*, Holt, Rinehart and Winston, New York, 1964, 54.

[Haj57] Hajós *Über eine Art von Graphen*, Internat. Math. Nachr. 11 (1957), 65.

[HSU92] Hsu W.L., Shih W.K. and Chern T.C. An $O(n^2 \log n)$ time algorithm for the Hamiltonian Cycle Problem, To appear in *SIAM J. on Compt.*

[IPS82] Itai A., Papadimitriou C.H. and Szwarcfiter J.L. Hamiltonian paths in grid graphs, *SIAM J. Comput.* 11 (4) (1982) 676-686.

[Joh85] Johnson D.S. The NP-complete Column: an Ongoing Guide, *J. of Algorithms* 6 (1985) 434-451.

[Kei85] Keil M. Finding Hamiltonian Circuits in Interval Graphs, *Inform. Process. Lett.* 20 (1985) 201-206.

[Ken69] Kendall D.G. Incidence matrices, interval graphs, and seriation in archeology, *Pacific J. Math.* 28 (1969), 565-570.

[Kle69] Klee V. What are the intersection graphs of arcs in a circle?, *Amer. Math. Monthly* 76 (1969), 810-813.

[Kri76] Krishnamoorthy M.S. An NP-hard problem in bipartite graphs, *SIGACT News* 7 (1) (1976) 26.

[Lek62] Lekkerkerker C.G. and Boland J.C., Representation of a finite graph by a set of intervals on the real line , *Fund. Math.* 51 (1962) 45-64.

[Man90] Manacher G.K., Mankus T.A. and Smith C.J., An optimum $\theta(n \log n)$ Algorithm for finding a Canonical Hamiltonian Path and Canonical Hamiltonian Circuit in a set of Intervals, *Inform. Process. Lett.* 35 (1990) 205-211.

[Mar45] Marczewski E., Sur deux proprietes des classes d'ensembles, *Fund. Math.* 33, 303-307.

[PLE64] Pnueli A., Lempel A. and Even S., Transitive Orientations of Graphs and Identification of Permutation Graphs, *Can. J. Math.* 23 (1971) 160-175.

[Rob69] Roberts F. *Indifference graphs, "Proof Techniques in Graph Theory"*, (F. Harary, ed.), Academic Press, New York, 1969, 139-146.

[Ros70] Rose D.J. . Triangulated graphs and the elimination process, *J. Math. Anal. Appl. 32* (1970) 597-609.

[Shi88] Shibata Y. On the Tree Representation of Chordal Graphs, *Journal of Graph Theory,* 12,3, 421-428 (1988).

[Skr82] Skrien D.J. A relationship between triangulated graphs, comparability graphs, proper interval graphs, proper circular-arc graphs, *J. Graph Theory* 6 (1982) 309-316.

[Tuc71] Tucker A. Matrix characterizations of circular-arc graphs, *Pacific J. Math.* 34 (1970), 501-510.

[Tuc80] Tucker A. An efficient test for circular-arc graphs, *SIAM Journal on Computing* 9 (1980), 1-24.

[Urr80] Urrutia J. *Intersection Graphs of Some Families of Plane Curves*, Ph.D. Thesis, University of Waterloo.

[UrG92] Urrutia J., Gavril F. An Algorithm for fraternal orientation graphs, *Information Processing Letters* 41 (1992) 271-274.

[Wal78] Walter J.R. Representations of Chordal Graphs as Subtrees of a Tree, *Journal of Graph Theory* 2 (1978) 265-267.