University of Alberta

SUBTREE OVERLAP GRAPHS - TOWARDS RECOGNITION

by

Jessica Anne Enright ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2006

Your file Votre référence
ISBN: 978-0-494-22256-0
Our file Notre référence
ISBN: 978-0-494-22256-0

Abstract

Subtree overlap graphs are the overlap graphs of subtrees in a tree. Several related classes and subclasses of the subtree overlap graphs are well-studied with polynomial time recognition algorithms and applications. The subtree overlap graphs present a gap in knowledge. In this work we prove that the subtree overlap graphs are equivalent to the subtree filament graphs, and are therefore the complements of cochordal-mixed graphs. We also show that a number of graph operations, compositions and decompositions preserve the property of being a subtree overlap graph when applied to the known subclasses of subtree overlap graphs. Overall, we develop several tools that may be useful in future work on recognizing or building algorithms for subtree overlap graphs.

Acknowledgements

Firstly, thanks go to my supervisor, Lorna Stewart. Lorna proposed this topic of study, and encouraged and corrected me throughout. This work certainly could not have been done without her. The members of my thesis committee provided very helpful feedback on the document itself. Many grad student friends helped me in innumerable ways - they deserve far more than mere thanks. I'd like to thank Lynn Evans Phillips and Robin Dawes for early inspiration. Finally, of course, thanks to my family, which needs no explanation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Graphs are abstract objects composed of vertices, and edges connecting those vertices. Graph classes are particular sets of graphs, defined based on some restriction. Often, the restriction is based on a structural characterization, or a forbidden induced subgraph. The study of graph classes sometimes allows us to solve problems on these restricted classes that cannot be solved on graphs in general.

In this thesis, we are concerned with classes of graphs defined by methods of representing them. Many methods of representation are possible. Here we work with restricted set overlapping, containment, and disjointness. Our class of graphs are those that are representable as the overlap graphs of subtrees in a tree.

We wish to better understand the properties of these graphs, and work towards their recognition. In this thesis, we first present some motivation and background, and then proceed to refine our understanding of the hierarchy of graph classes related to subtree overlap graphs, and finally describe various operations and transformations and how they relate to the subtree overlap graph class.

1.2 Preliminaries

A set is an unordered collection of elements, in which each element appears only once. Let S and S' be two sets. The *intersection* of S and S' is the set of elements found in both sets, and is denoted $S \cap S'$. The *union* of the two sets is the set composed of all elements in S, and all elements in S'. If all elements in S are also in S', but S' has some element not in S, then we say that S is *contained* in S', or $S \subseteq S'$. Two sets are equal if they have exactly the same elements. For two sets S_1 and S_2, $S_1 - S_2$ denotes the elements in S_1 that are not also in S_2. An element e being in a set S is denoted as $e \in S$. Unless otherwise noted, all sets used here are finite. The *size*, or *cardinality* of a set S is the number of elements it has, denoted $|S|$.

A graph G is composed of the pair (V, E), where V is a set of vertices, and E a set of edges between vertices in V. In general, we will use $n = |V|$ to refer to the number of vertices in a graph, and $m = |E|$ to refer to the number of edges in a graph. Two vertices $u, v \in V$ are *adjacent* if there is some edge between them in E, that is, $(u, v) \in E$. Unless stated otherwise, all graphs in this work are undirected, finite, and have no self-loops or multiple edges.

We will occasionally use directed edges. In this thesis a directed edge from a to b is denoted $(a \rightarrow b)$.

Let v be some vertex in V. v is an *isolated vertex* if v is adjacent to no other vertices in V. The *neighborhood* of v is the set of vertices to which v is adjacent, and is denoted $N(v)$. The *degree* of v is the size of its neighborhood, $|N(v)|$. v is a *universal vertex* if it is adjacent to all other vertices in V.

An *induced subgraph*, $H = (V', E')$ of G is a graph made of some set of vertices $V' \subset V$, and all edges in E between vertices in V'. That is, for any two vertices $u, v \in V'$, such that $(u, v) \in E$ if and only if $(u, v) \in E'$. We denote the subgraph of G induced by the vertex set V' as $G(V')$. A graph class or property is *hereditary* if the fact that the class or property holds for some graph G means that it also holds for all induced subgraphs of G.

A *clique* is a set of pairwise adjacent vertices, and an *independent set* is a set of pairwise non-adjacent vertices. The *size* of a clique or independent set is the number of vertices in it.

Let V be a set of n vertices ordered from 0 to $n-1$. The vertices in V are a *path* of length $n-1$ if and only if $v_0 \in V$ is adjacent to exactly v_1, $v_{n-1} \in V$ is adjacent to exactly v_{n-2}, and $v_i \in V$ where $0 < i < n-1$ is adjacent to exactly vertices v_{i-1} and v_{i+1}. The vertices in V are a *cycle* of length n if and only if each vertex v_i is adjacent to only $v_{(i-1) \bmod n}$, and $v_{(i+1) \bmod n}$. A *tree* is a connected acyclic graph. A star is a tree consisting of one central vertex, and any number of pendant leaves adjacent to exactly the central vertex. A caterpillar is a tree consisting of only a path and leaves attached to vertices on that path.

1.3 Set Representations of Graphs

Let T be a set. Let S be a set of subsets of T. That is, each $s_i \in S$ is such that $s_i \subseteq T$. Given two sets, $s_1 \in S$ and $s_2 \in S$, we are interested in some relationships that could exist between them. If $s_1 \supset s_2$, then we will say that s_1 contains s_2, and s_2 is contained in s_1. If s_1 and s_2 share no elements, that is $s_1 \cap s_2 = \emptyset$, then we will say that s_1 and s_2 are disjoint. We denote this as $s_1 | s_2$. If s_1 and s_2 share any elements, that is, $s_1 \cap s_2 \neq \emptyset$, then s_1 and s_2 intersect. If s_1 and s_2 intersect, but neither contains the other, then we will say that they overlap, and denote this as $s_1 \between s_2$. Recall that if two sets contain exactly the same elements, then they are equal.

The relationships among sets in S can be represented by graphs. A graph G = (V, E) is an overlap graph of the sets in S if there is a bijection from V to S, such that two vertices $v_i, v_j \in V$ are adjacent if $s_i \between s_j$.

A graph G = (V, E) is an intersection graph of the sets in S if there is a bijection from V to S, such that two vertices $v_i, v_j \in V$ are adjacent if s_i intersects s_j.

A graph G = (V, E) is a containment graph of the sets in S if there is a bijection from V to S, such that two vertices $v_i, v_j \in V$ are adjacent if either $s_i \subset s_j$, or $s_j \subset s_i$.

A graph G = (V, E) is a disjointness graph of the sets in S if there is a bijection from V to S, such that each vertex $v \in V$ corresponds to some set $s \in S$, and two vertices $v_i, v_j \in V$ are adjacent if and only if $s_i | s_j$.

For some graph G that is an overlap, containment, disjointness, or intersection graph of sets $s_i \in S$, with the union of the sets T, we say that G has a *representation* $R = (S, T)$, and without loss of generality can dictate that there are no two sets $s_i, s_j \in S$ such that $s_i = s_j$. We justify this as follows. Suppose there is some representation $R = (S, T)$ such that two sets $s_i, s_j \in S$ where s_i equals s_j. We can then add an element p to T that is in only s_i, and any containers of s_i. Since p was only added to sets that already had non-zero intersection, its addition cannot have disrupted disjointness. Since no elements were removed, it cannot have disrupted overlapping. Since for any s_k to which p was added, p was also added to all sets in S containing s_k, containment cannot have been disrupted. If the nature of the sets in S is unrestricted, then every graph has intersection, overlap, and disjointness representations [32]. Only comparability graphs have containment representations [16].

If we restrict the type of set that T is, the type of set that the sets in S can be, or both, we can restrict the class of graphs that is represented. We can define a class of graphs as being the graphs representable by particular relationships between restricted types of sets.

For example, we might restrict T to be a line, and the sets in S to be intervals on T. We can restrict T to be a tree, and the sets in S to be subtrees in T. This work is concerned mainly with the overlap graphs of subtrees in a tree.

If $G = (V, E)$ is a subtree overlap graph, we will say it has a subtree overlap representation $R = (S, T)$, where S is the set of subtrees of T, with exactly one subtree $s \in S$ for each vertex in $v \in V$.

Given an overlap representation $R = (S, T)$ for graph $G = (V, E)$ we say that the edges or non-edges of G are *due to* relationships between sets in S. Let $(v_i, v_j) \in E$ be an edge between two vertices in V. We can say that the (v_i, v_j) edge is due to overlapping in R, since $s_i \between s_j$. Similarly, if $(v_i, v_k) \notin E$ is a non-edge between two vertices in V, we say that (v_i, v_k) is due to containment if $s_i \subset s_k$ or $s_k \subset s_i$, and due to disjointness if $s_i | s_k$.

Let $R = (S, T)$ be a representation such that there are no two subsets $s_i, s_j \in S$ such that $s_i \subset s_j$. We say that R has no containment. The overlap graph G_O, and the intersection graph G_I of the sets in S are the same, as all non-edges of both G_O and G_I are due to disjointness in R, and all edges due to overlapping.

Similarly, if $R = (S, T)$ is a representation in which there are no two $s_i, s_j \in S$ such that $s_i | s_j$, then we say that R has no disjointness, and the overlap graph of the sets in S is equivalent to the containment graph of those sets.

When considering overlap representations, we assume that there are no sets consisting of only one element, as such a set would represent an isolated vertex. Isolated vertices can instead be represented by sets consisting of all the elements in the entire representation.

All sets used in representations will have the *Helly property*. That is:

Definition 1.1. *Set S has the Helly property if, for any subset S', if all sets in S' are pairwise non-disjoint, then the intersection of all sets in S' is non-zero - they all have at least one element in common.*

We will define two notions here that will be used later in this work: isomorphism of representations and internal and external sets.

Two representations $R_1 = (S_1, T_1)$ and $R_2 = (S_2, T_2)$ are isomorphic if the following is true: there is bijection $f : S_1 \rightarrow S_2$ such that for all $s_i, s_j \in S_1$

- $s_i \subset s_j \leftrightarrow f(s_i) \subset f(s_j)$,
- $s_i \supset s_j \leftrightarrow f(s_i) \supset f(s_j)$,
- $s_i \between s_j \leftrightarrow f(s_i) \between f(s_j)$,
- $s_i | s_j \leftrightarrow f(s_i) | f(s_j)$

Internal and External Sets and Vertices

Let $R = (S, T)$ be some representation. Let $s_i \in S$. If there is no set $s_j \in S$ such that $s_j \subset s_i$, then we say that s_i is *internal* in R. If there is no set $s_k \in S$ such that $s_k \supset s_i$, then we say that s_i is *external* in R.

Let ψ be one of the relationships - containment, disjointness, overlapping, or intersection. Let γ be a type of set, for example, an unrestricted set, or a subtree in a tree. Let $G = (V, E)$ be a ψ graph of sets of type γ. Let v_i be a vertex in V. We say that v_i *can be internal* (with respect to ψ) if there is some γ ψ representation $R_1 = (S_1, T_1)$ of G, in which s_i, the element of S_1 corresponding to v_i, is internal. We say that v_i *can be external* if there is some γ ψ representation $R_2 = (S_2, T_2)$ of G, in which s_i, the element of S_1 corresponding to v_i, is external. Unless stated otherwise, in this work we are concerned with overlap representations when discussing whether a vertex can be internal or external.

1.4 A First Look at Subtree Overlap Graphs

A graph $G = (V, E)$ where $V = \{v_1 ... v_n\}$is a subtree overlap graph if and only if there is some tree T and set $S = \{s_1 ... s_n\}$ of subtrees of T such that two vertices v_i and v_j are adjacent if and only if $s_i \between s_j$.

Note that here we have used the subscript convention that v_x and s_x are corresponding, with v_x being a vertex and s_x a subtree. We will continue to use this convention for convenience throughout

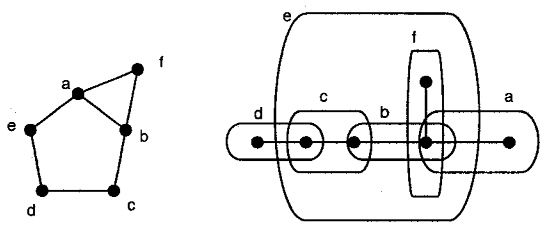


Figure 1.1: An example of a subtree overlap graph on the left, and a corresponding subtree overlap representation on the right. The nodes and thick lines on the right show the underlying tree, with the subtrees indicated and labeled with the same labels as their corresponding vertices on the left.

this work. Occasionally, we will also use the convention of a being some vertex, and s_a being its corresponding subtree.

To avoid confusion, we will use the term *vertex* when referring to elements of the graph G, and *node* when referring to elements of the tree T, though, of course, T is itself a graph, and so has vertices and edges. For an example of a subtree overlap graph and its subtree overlap representation, see Figure 1.1.

Subtree overlap graphs generalize several well-known graph classes. We understand many properties of these subclasses, how to recognize them, and how to solve problems on them. We know of many applications of the known subclasses, for example, they are useful in: scheduling [17], the modelling of genetic structure [3], archaeological dating [17], and data storage ordering [17]. Despite the wide knowledge of the subclasses, we do not know how to recognize subtree overlap graphs, or many of their properties.

It is natural to want to fill in this gap in knowledge. The ability to recognize subtree overlap graphs might allow us to generalize some work done on the subclasses.

The research reported in this thesis was done with an aim toward recognition of subtree overlap graphs. Many tools that could be useful in recognizing or designing algorithms for subtree overlap graphs are presented here. We specifically consider operations and compositions and decompositions that have special relevance to the subclasses.

1.5 Outline and Contributions

In the remaining chapters of this thesis, we present some class hierarchy results, describe subtree overlap representations for a few simple graphs, and then examine a series of graph operations in the context of subtree overlap graphs and the subclasses.

Chapter 2 describes some graph classes discussed in this thesis, and further introduces the subtree overlap graphs.

Chapter 3 shows that the subtree overlap graphs are exactly the subtree filament graphs, and from that gives a proof that subtree overlap graphs are exactly the complements of cochordal mixed graphs. This presents possibilities for future algorithms for constructing subtree overlap representations, given a particular kind of edge partition.

Chapter 4 shows that the consistency of a set representation can be checked in polynomial time. This might have implications for which part of the recognition problem is hard - once we have generated a representation, could we check it? From this point on, we focus primarily on subtree overlap representations.

Chapter 5 describes the few possible non-isomorphic subtree overlap representations for a cycle, and gives an exponential lower bound on the number of possible non-isomorphic subtree overlap representations for a path.

Chapter 6 discusses internal and external subtrees in the known subclasses, as well as in subtree overlap graphs in general. Here, we show that any chordal graph or spider graph can be external, but this is not true for subtree overlap graphs in general.

Chapter 7 investigates a series of compositions and decompositions on the subclasses, as well as subtree overlap graphs in general. We show that some compositions preserve the property of being a subtree overlap graph when applied on some subclasses.

Chapter 8 examines several graph and tree operations, such as adding or removing edges or vertices to or from graph, where the graph is a member of each of the subclasses. A particularly interesting result from this chapter is that adding a new vertex adjacent to an arbitrary neighborhood in a chordal graph preserves subtree overlap graph. The results found in this Chapter, as well as the previous one, are summarized in Table 7.1.

Finally, in Chapter 9, we present some directions for future work, and summarize the work presented here.

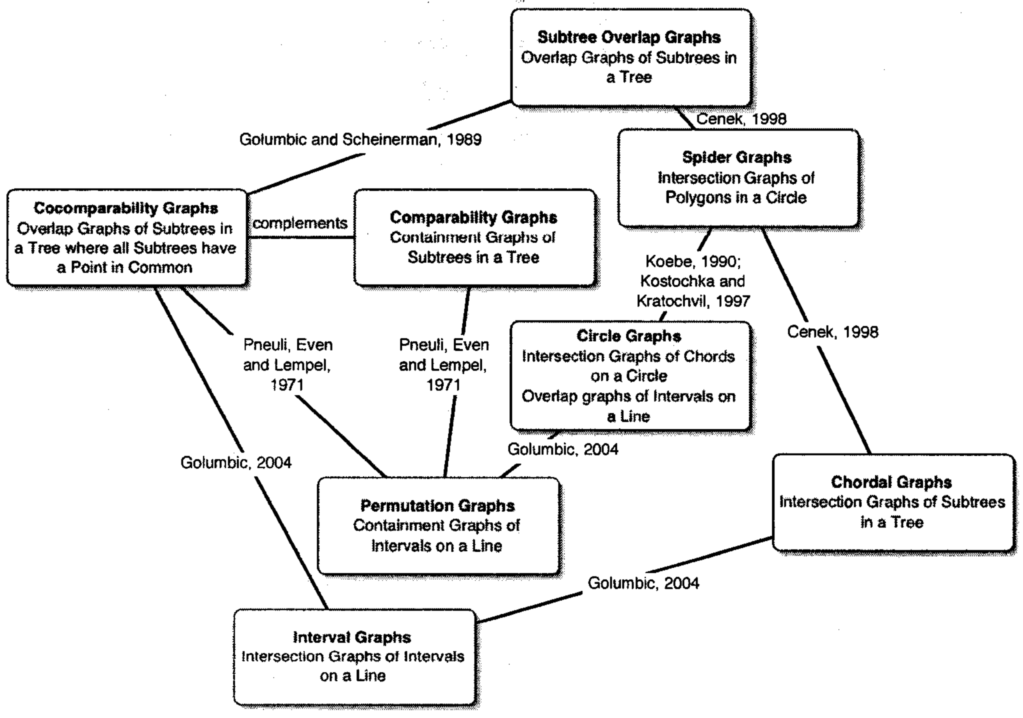
Chapter 2

Graph Classes

2.1 Subclasses

In this section we will present a few subclasses of subtree overlap graphs. Most of these are fairly well-known, with known properties, recognition algorithms and applications.

They are presented not only as background information, but also as motivation for the study of subtree overlap graphs. Figure 2.1 shows an illustration of the subclasses discussed here, and their relationships to each other and to subtree overlap graphs.



Subtree Overlap Graphs
Overlap Graphs of Subtrees in a Tree
Cenek, 1998
Golumbic and Scheinerman, 1989
Spider Graphs
Intersection Graphs of Polygons in a Circle
Cocomparability Graphs
Overlap Graphs of Subtrees in a Tree where all Subtrees have a Point in Common
complements
Comparability Graphs
Containment Graphs of Subtrees in a Tree
Koebe, 1990; Kostochka and Kratochvil, 1997
Cenek, 1998
Pneuli, Even and Lempel, 1971
Pneuli, Even and Lempel, 1971
Circle Graphs
Intersection Graphs of Chords on a Circle
Overlap graphs of Intervals on a Line
Golumbic, 2004
Golumbic, 2004
Chordal Graphs
Intersection Graphs of Subtrees in a Tree
Permutation Graphs
Containment Graphs of Intervals on a Line
Golumbic, 2004
Interval Graphs
Intersection Graphs of Intervals on a Line

Figure 2.1: An illustration of the hierarchy of the class of subtree overlap graphs and some of its subclasses. Below the bold class labels are descriptions of set representation-based characterizations of the classes. Classes lower in the figure are contained in classes higher in the figure, along the lines connecting the classes.

The background information here is by no means exhaustive. Many events in the historical development of these classes are omitted here. For a more thorough accounting we refer the reader to: [17, 31, 5, 1].

2.1.1 Chordal Graphs

A graph is chordal if and only if every cycle of length greater than or equal to four has a chord, or, equivalently, the graph has no chordless cycle of length greater than or equal to four.

Theorem 2.1. *The following are equivalent statements for a graph $G = (V, E)$*

- *G is chordal.*
- *Every induced subgraph of G contains a simplicial vertex. A vertex is simplicial if and only if its neighborhood induces a clique [17].*
- *G is the intersection graph of subtrees in a tree [14].*
- *G is a subtree overlap graph that can be represented without containment [14].*
- *Every minimal cutset of a G is a clique, and every induced subgraph of a G has a clique cutset [8].*

It is known that clique cutset composition and decomposition preserves the property of being chordal. Clique cutset composition is as follows: Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. Let $V_{C1} \subseteq V_1$ and $V_{C2} \subseteq V_2$ be two vertex sets that induce cliques in G_1 and G_2, respectively, such that $|V_{C1}| = |V_{C2}|$. A clique cutset composition identifies each vertex in V_{C1} with exactly one vertex in V_{C2}, producing a larger graph. We will later consider the application of restricted clique cutset decomposition and composition on other known subclasses of subtree overlap graphs, as well as the class of subtree overlap graphs in general.

We know that the chordal graphs are the subtree overlap graphs that have subtree overlap representations with no containment[14]. That is, if $G = (V, E)$ is a chordal graph, it has some subtree overlap representation $R = (T, S)$, for every two non-adjacent vertices $v_1, v_2 \in V, (v_1, v_2) \notin E$, $s_1 | s_2$. Since we know that every chordal graph has some representation as the intersection graph of subtrees in a tree [14], we can modify that intersection representation by adding to each subtree a new leaf that is in no other subtree. Now two subtrees overlap if and only if they intersected in the original intersection representation, and no subtree is contained in any other.

Recall that if there is no containment in a representation, then the overlap and intersection graphs are the same (Section 1.3).

The fastest known recognition algorithms for chordal graphs use the existence of a simplicial vertex, and a lexicographic breadth-first search ordering. This algorithm runs in linear time in the number of vertices plus the number of edges [27]. The idea of some important vertex, or the use of

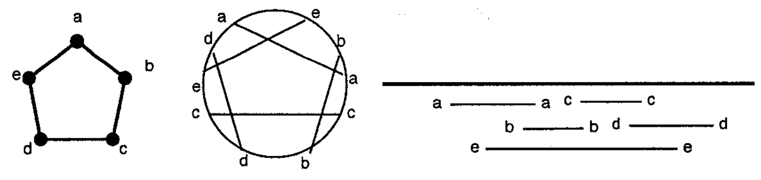


Figure 2.2: An example of a circle graph on the left, its corresponding representation as chords on a circle in the middle, and its representation as overlapping intervals on a line on the right. The chord and interval endpoints are labeled with the same labels as their corresponding vertices on the left.

a vertex ordering may contribute to recognition of subtree overlap graphs, but is not investigated in this work.

2.1.2 Circle Graphs

Circle graphs were introduced by Even and Itai [10]. Circle graphs are the intersection graphs of chords in a circle, or, equivalently, the overlap graphs of intervals on a line [13].

Theorem 2.2. *[13] The circle graphs are exactly the interval overlap graphs.*

Circle graphs can be recognized in polynomial time, as presented in [30], using the join decomposition.

2.1.3 Comparability Graphs

A graph $G = (V, E)$ is a comparability graph if there is a transitive orientation of its edges, that is, there is some orientation of the edges in E such that, for all vertices $u, v, w \in V$, $(u \rightarrow v)$ and $(v \rightarrow w)$ implies that there must also be the edge: $(u \rightarrow w)$ [17].

Comparability graphs are the containment graphs of subtrees in a tree [16]. Golumbic and Scheinerman [16] presented a method for constructing a subtree containment representation of a comparability graph on a star, with every subtree containing the central node. In this containment representation, all non-edges are due to overlapping, and all edges due to containment.

Briefly, their method [16] is as follows: let $G = (V, E)$ be a comparability graph, and F a transitive orientation of its edges. Let T be a star, with a central node q and leaf nodes $l_1, ...l_n$.

For each vertex $v_j \in V$, we create a corresponding subtree s_j composed of q, and each leaf node l_i such that $(v_i \rightarrow v_j) \in F$. This method is lacking one trivial point: each subtree s_j also contains l_j. By this construction, $s_i \subset s_j$ if and only if $(v_i \rightarrow v_j) \in F$. Therefore, $(v_i, v_j) \in E$ if and only if either $s_i \subset s_j$ or $s_i \supset s_j$. G is therefore the containment graph of the subtrees created on star T. An illustration of a simple example of this construction can be found in Figure 2.3.

Comparability graphs can be recognized in polynomial time. McConnell and Spinrad [23] presented a linear-time algorithm for computing a modular decomposition of a graph, and, given that

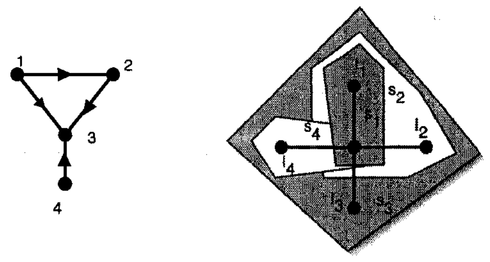


Figure 2.3: A comparability graph shown on the left, with a transitive orientation of its edges indicated by arrows. On the right is a containment representation of the graph, as produced by the method described by Golumbic and Scheinerman [16]. The leaves and subtrees are labeled.

decomposition, assigning a transitive orientation of the edges should one exist. This is used to recognize comparability graphs in $O(n + m \log n)$ time.

2.1.4 Cocomparability Graphs

Cocomparability graphs are the complements of comparability graphs. As described in the previous section, comparability graphs are the containment graphs of subtrees in a star. From this we can see that all cocomparability graphs have subtree overlap representations on stars with no disjointness. Here, all non-edges are due to containment, and all edges due to overlapping.

We also know that any overlap representation with no disjointness represents a cocomparability graph. This can be seen from the fact that any non-edges in the graph represented must be due to containment. Since containment is transitive, these non-edges must have a transitive orientation, and therefore the complement of the graph is a comparability graph. So, every cocomparability graph has a subtree overlap representation on a star with no disjointness, and every subtree overlap representation with no disjointness represents a cocomparability graph. Thus, every graph that has a subtree overlap representation with no disjointness can also be represented as the overlap graph of subtrees on a star.

Recognition of cocomparability graphs can be done in polynomial time, from the recognition algorithm using modular decomposition for comparability graphs. Modular decomposition will later be examined in the context of subtree overlap graphs and the other subtree overlap graph subclasses.

2.1.5 Interval Graphs

Interval graphs were first introduced by Hajös in 1957 [17] and were one of the earlier classes of intersection graphs studied. The interval graphs are the intersection graphs of intervals on a line [31, 17].

Interestingly for this work, interval graphs are the intersection of chordal and cocomparability graphs [17]. Both of these classes are also subclasses of subtree overlap graphs, and are further discussed in this chapter.

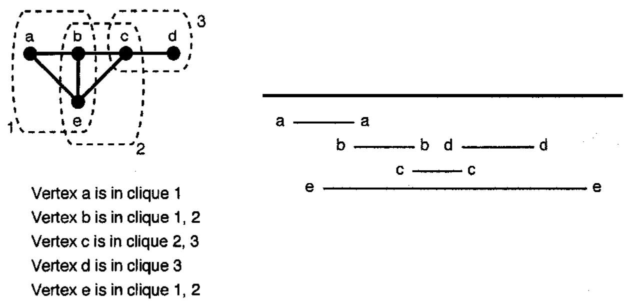

a
b
c
d
e
1
2
3
Vertex a is in clique 1
Vertex b is in clique 1, 2
Vertex c is in clique 2, 3
Vertex d is in clique 3
Vertex e is in clique 1, 2
a a
b b d d
c c
e e

Figure 2.4: An interval graph on the left with labeled maximal cliques, and its representation as the intersection of intervals on a line on the right.

Interval graphs can be recognized in linear time with respect to the sum of the number of vertices and edges [4]. This algorithm exploits a characterization of interval graphs based on cliques. Interval graphs are exactly the graphs with maximal cliques that can be linearly ordered such that each vertex occurs in only consecutive maximal cliques. An example of an interval graph with its representation as intervals on a line and numbered maximal cliques can be found in Figure 2.4.

2.1.6 Permutation Graphs

Let $G = (V, E)$ be some graph, with vertex set $V = \{v_1, v_2 ... v_n\}$. G is a permutation graph if there is some permutation $\pi = [\pi_1, \pi_2 ... \pi_n]$ of the numbers 1 to n such that two vertices in V are adjacent if and only if their indices appear out of order in π. That is, the vertices v_i and v_j, where $i < j$ are adjacent if and only if j appears before i in π.

Theorem 2.3. *The following are equivalent statements for a graph $G = (V, E)$:*

- *G is a permutation graph.*
- *G is the intersection graph of line segments between two parallel lines [31].*
- *G is the containment graph of intervals on a line [9].*
- *G and $\bar{G}$ are comparability graphs [26].*

Permutation graphs can be recognized in polynomial time [29].

2.1.7 Spider Graphs

The class of spider graphs was formally defined by Koebe [21]. The equivalent polygon circle graphs were presented by Kostochka and Kratochvil [22], and their inception credited to a personal communication with Fellows. Further work on spider graphs was done by Čenek [5].

A graph $G = (V, E)$ is a spider graph if there is some bijection between V and a set of sets of endpoints on a circle such that two vertices v_i and v_j are adjacent if and only if their corresponding

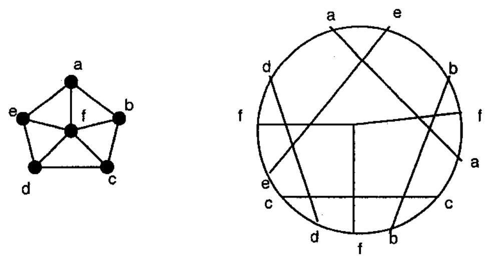


Figure 2.5: An example of a spider graph on the left, and its corresponding representation as sets of endpoints on a circle on the right. The endpoints are labeled with the same labels as their corresponding vertices on the left.

sets of endpoints, c_i and c_j, satisfy the property that there is no chord on the circle that can separate all endpoints in c_i from all endpoints in c_j [21, 5]. An example of a spider graph and its spider representation can be seen in Figure 2.5.

Čenek [5] showed that spider graphs are a subset of subtree overlap graphs. In that proof, she shows a transformation from the circle-based representation to an overlap representation on a caterpillar. Similar to the placement of intervals on a line in the proof of Theorem 2.2, Čenek places subtrees in some order on the caterpillar, based on the order of the endpoints around the circle, with the first subtree placed being the only one to contain a particular node at one end of the caterpillar. Since she starts the endpoint ordering at an arbitrary endpoint, an arbitrary vertex in the graph can have a subtree that is the only one to contain some node in the caterpillar. This fact will be used in Section 6.1.

The class of spider graphs contains several of the graph classes already discussed here. Chordal graphs and circle graphs are both subclasses of spider graphs. However, throughout this thesis, we still sometimes discuss chordal and circle graphs to refine our understanding of when particular operations can be applied.

A polynomial time recognition algorithm for spider graphs is presented in [21]. However, Spinrad [31] has expressed doubts as to the correctness of Koebe's recognition algorithm.

2.2 Subtree Filament Graphs

In [15], Gavril introduced several classes of filament graphs, including interval filament graphs and subtree filament graphs. The interval filament graphs are intersection graphs of interval filaments. Let L be a line, and I a set of intervals on that line. Let P be a plane perpendicular to the plane in which L lies, intersecting that plane exactly at L. For each interval $i_a \in I$, let f_a be a curve connecting the endpoints of i_a in P, such that for all $i_b \in I$, if $i_a | i_b$, then f_a and f_b do not intersect. From the construction, if i_a overlaps i_b then $f_a \cap f_b \neq \emptyset$. Note that, if $i_a \subset i_b$ then f_a and f_b may or may not intersect.

Expanding the idea of an interval filament graph, Gavril [15] defined subtree filament graphs - the intersection graphs of subtree filaments. Let T be a tree, and S a set of subtrees on T. Then let P be a surface perpendicular to the plane in which T lies, intersecting with that plane at exactly T. For each $s_a \in S$, let f_a be a curve in P connecting all endpoints of s_a such that, for all $s_b \in S$, if $s_a | s_b$, then f_a and f_b do not intersect, and if $s_a \between s_b$, then f_a and f_b do intersect. Again, if $s_a \subset s_b$, then f_a and f_b may or may not intersect.

This class contains the subtree overlap graphs. At first glance, it seems that it should be a larger class than the subtree overlap graphs. However, later in this work, in Section 3.1, we will show that the subtree filament graphs are exactly the subtree overlap graphs.

2.3 G-mixed Graphs

In the same work where subtree filament graphs are introduced, Gavril [15] presented the idea of **G**-*mixed* graphs. If **G** is some family of graphs, then graph $G = (V, E)$ is **G**-*mixed* if the edges of G can be partitioned into two sets E_1 and E_2 such that $G' = (V, E_1)$ is in the family **G**, and the edges in E_2 are transitively oriented such that, for any three vertices $u, v, w \in V$, if $(u, v) \in E_1$ and $(w \rightarrow v) \in E_2$, then $(u, w) \in E_1$.

Using these characteristics of **G**-*mixed* graphs, Gavril [15] provided an algorithm for finding maximum weight cliques on **G**-*mixed* graphs. He also shows relationships between some families of filament graphs, and families of **G**-*mixed* graphs. He explicitly proves Theorem 2.4, where the cointerval graphs are the complements of the interval graphs, and the cochordal graphs are the complements of chordal graphs.

Theorem 2.4. *The interval filament graphs are exactly the complements of cointerval-mixed graphs.*

Gavril also stated Theorem 2.5, but omits the proof, instead stating that it is very similar to the proof of Theorem 2.4. We will later explicitly provide this proof, in Section 3.2.

Theorem 2.5. *A graph is a subtree filament graph if and only if its complement is cochordal mixed.*

2.4 Subtree Overlap Graphs

The subtree overlap graphs were defined in the first chapter. A subtree overlap graph $G = (V, E)$, $V = \{v_1 ... v_n\}$ has a representation $R = (S, T)$ where $S = \{s_1 ... s_n\}$ is a set of subtrees on the tree T and v_i and v_j are adjacent if and only if $s_i \between s_j$.

The subtree overlap graphs contain the spider graphs, circle graphs, cocomparability graphs, interval graphs, permutation graphs, and chordal graphs, as discussed in each of the sections pertaining to these classes. The subtree overlap graphs are equivalent to the subtree filament graphs, as discussed in Section 2.2.

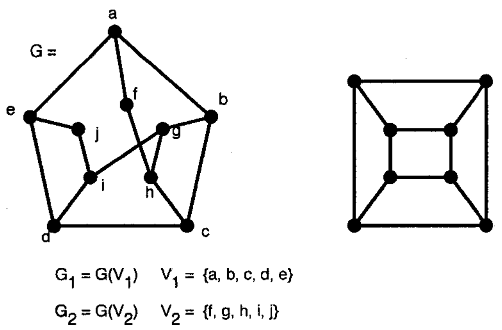

G =
a
b
c
d
e
f
g
h
i
j
$G_1 = G(V_1)$ $V_1 = \{a, b, c, d, e\}$
$G_2 = G(V_2)$ $V_2 = \{f, g, h, i, j\}$

Figure 2.6: Two non-subtree overlap graphs. The cube is shown on the right, and a graph that is non-subtree overlap graph from Theorem 2.6 with the partition required on the left.

Novillo [24] discussed the non-subtree overlap graphs, both showing that the cube is not a subtree overlap graph and establishing a family of non-subtree overlap graphs. The cube and a graph described by Theorem 2.6 can be found in Figure 2.6.

Theorem 2.6. *A graph G is not a subtree overlap graph if it contains two disjoint induced subgraphs G_1 and G_2, where G_1 is a cycle of length at least 5, and G_2 is a connected graph of at least 5 vertices, such that each vertex v in G_1 is adjacent to some vertex u in G_2, where u is adjacent to no other vertices in G_1 [24].*

Čenek [5] studied subtree overlap graphs, refining their position in the hierarchy of all graphs. Čenek shows that they contain the class of spider graphs. In the same work, bounds on the maximum size of a minimal tree in a subtree overlap representation are shown.

As we have seen in the previous sections, many of the subclasses of subtree overlap graphs are well-known. In particular, the chordal graphs, interval graphs, permutation graphs, cocomparability graphs, and circle graphs have many known properties. The chordal graphs and cocomparability graphs are also the subtree intersection and containment graphs, respectively.

If we know so much about the subtree containment and intersection graphs, why do we know so little about the subtree overlap graphs? The subtree overlap graphs represent a gap in knowledge.

Filling in this gap is particularly appealing because some otherwise NP-complete problems are polynomial-time solvable on some subclasses of subtree overlap graphs, as well as because known subclasses have both natural structural characterizations, as well as natural set representation characterizations.

Maximum independent set, maximum clique, and k-colouring can all be solved on chordal graphs [12], as can minimum clique-cover [17]. Maximum independent set [2] and maximum clique [2] can be computed in polynomial time on circle graphs. k-colouring is NP-Complete on circle graphs for k greater than 4, but polynomial for circle graphs of bounded degree. Clique cover on circle graphs is NP-complete [19]. Maximum independent set, maximum clique, and k-colouring of comparability and cocomparability graphs can be computed in polynomial time [17]. Clique cover is polynomial from perfect graphs [18]. Koebe [20] showed that for bounded-degree spider graphs, k-colouring can be solved in polynomial time, and a maximum independent set can be found in time polynomial in the total number of endpoints in the spider representation. From circle graph, clique cover is NP-Complete on spider graphs.

In addition, we know that, if given an overlap representation, a maximum independent set or maximum clique on a subtree overlap graph can be found in polynomial time. [5, 6]

Given that recognition of well-known subclasses is polynomial, we wish to work toward a polynomial-time recognition algorithm for subtree overlap graphs. While this is not achieved in this work, a number of potentially useful tools are developed.

Chapter 3

Equivalent Classes

In this section, we present two proofs of class equivalence. Showing that graph classes are equivalent allows us to refine our understanding of the graph class hierarchy. A better understanding of the class hierarchy could improve our understanding of the properties of subtree overlap graphs.

First, we show that the subtree filament graphs and subtree overlap graphs are equivalent.

We then show that the subtree filament, and therefore subtree overlap, graphs are exactly the complements of cochordal-mixed graphs. This result is stated in [15], where the proof is omitted. We provide a proof.

Finally, we make some investigations into the overlap graphs of paths in trees, and show that if we restrict the sites of overlapping on the paths, then we are reduced to the class of interval overlap graphs, that is, circle graphs.

3.1 Subtree Filament Graphs are Subtree Overlap graphs

Gavril [15] introduced subtree filament graphs. The definition is repeated in Section 1.

We assume a representation of a set of subtree filaments F such that we know the tree T on which the filaments are defined, and for each filament $f_i \in F$, we know the endpoints of f_i on T, and whether f_i intersects each $f_j \in F$.

Before we begin the main proof in this section, we need a few lemmas about subtree overlap representations:

Lemma 3.1. *Let $R = (S, T)$ be a subtree overlap representation and let $S_a \subseteq S$ be such that all elements of S_a contain a common node u of T. Let T' be the tree T with a new leaf p added, where p is adjacent to u. For each $s_i \in S$, let s'_i be s_i if $s_i \notin S_a$, and s_i plus the new node p if $s_i \in S_a$. Now, for all $s_i, s_j \in S$, if $s_i \between s_j$, then $s'_i \between s'_j$ and if $s_i | s_j$ then $s'_i | s'_j$.*

Proof:

Let $s_i, s_j \in S$. If $s_i \between s_j$, then there exist nodes $x \in s_i \cap s_j$, $y \in s_i \backslash s_j$, and $z \in s_j \backslash s_i$. Since adding a new node to one or both of s_i and s_j cannot remove these nodes, we have $x \in s'_i \cap s'_j$, $y \in s'_i \backslash s'_j$, and $z \in s'_j \backslash s'_i$. Therefore $s'_i \between s'_j$. If $s_i | s_j$ then $s_i \cap s_j = \emptyset$ and therefore the new node p is added to at most one of s_i and s_j, thus we have $s'_i | s'_j$.

Lemma 3.2. *Let $R = (S, T)$ be a subtree overlap representation and let $S_a \subseteq S$ be such that all elements of S_a contain a common node, u, of T and every element of S that contains an element of S_a is also in S_a. Let T' be the tree T with a new leaf, p, added where p is adjacent to u. For each $s_i \in S$, let s'_i be s_i if $s_i \notin S_a$, and s_i plus the new node p if $s_i \in S_a$. Now, for all $s_i, s_j \in S$, if $s_i \between s_j$ then $s'_i \between s'_j$, if $s_i | s_j$ then $s'_i | s'_j$, and if $s_i \subset s_j$ then $s'_i \subset s'_j$.*

Proof:

From Lemma 3.1 we know that if $s_i \between s_j$ then $s'_i \between s'_j$, and that if $s_i | s_j$ then $s'_i | s'_j$. It remains to show that if $s_i \subset s_j$ then $s'_i \subset s'_j$. Let us assume that $s_i \subset s_j$. Then we can consider two cases: either $s_i \in S_a$ or $s_i \notin S_a$. If $s_i \notin S_a$, then $s_i = s'_i$, and since $s'_j \supseteq s_j$ then $s'_j \supset s'_i$. If $s_i \in S_a$, then $s_j \in S_a$. Therefore the node p in in both s'_i and s'_j, therefore $s'_i \subset s'_j$. □

Theorem 3.1. *Let $R = (S, T)$ be a subtree overlap representation. We can, without loss of generality, assume that no two subtrees in S share an endpoint (leaf).*

Proof:

As proof, we present a modification that can be made to a subtree overlap representation to eliminate a shared endpoint without altering any overlapping or non-overlapping.

Let $R = (S, T)$ be a subtree overlap representation and $S_a \subset S$ a set of subtrees that all share the endpoint $q \in T = (V, E)$. Let P be a set of $|S_a|$ nodes where $P \cap V = \emptyset$. We then make each $p_i \in P$ adjacent to exactly q, and added to T.

For each s_i in S, let s'_i be s_i if $s_i \notin S_a$, and s_i contains no s_k such that $s_k \in S_a$. Otherwise, s'_i is s_i plus p_i (if $s_i \in S_a$), and each $p_k \in P$ such that $s_k \in s_i$. If any of these new leaves are now leaves of multiple subtrees (*i.e.* a subtree and its containers) we repeat the procedure.

Each leaf $p_i \in P$ was added to a set of subtrees S_i such that every element of S_i contained the point q, and any subtree in S that contained any element of S_i was also in S_i.

Therefore, by Lemma 3.2, if $s_i \between s_j$ then $s'_i \between s'_j$, if $s_i | s_j$ then $s'_i | s'_j$, and if $s_i \subset s_j$ then $s'_i \subset s'_j$. □

Let $T = (V, E)$ be a tree and $V' \subseteq V$. The *subtree induced by V'* on T comprises the subtree of T induced by $V' \cup \{x | x$ is a node on a path in T between two vertices of $V'\}$. For convenience, we will refer to the subtree induced by the endpoints of filament f_i on tree T as $ind(f_i, T)$.

Theorem 3.2. *A graph $G = (V, E)$ is a subtree overlap graph if and only if it is a subtree filament graph.*

Proof:

Every Subtree Overlap Graph is a Subtree Filament Graph

Given a set of subtrees $S = \{s_1 ... s_n\}$ on tree T, we show the construction of a set F of filaments on T such that for all $1 \leq i, j \leq n$ filaments $f_i, f_j \in F$ intersect if and only if $s_i, s_j \in S$ overlap.

This construction occurs in two stages: we initially create the filaments, and then modify them to ensure intersection as required.

Let F be a set of filaments on the subtrees of S such that $f_i | f_j$ if either $s_i | s_j$, or $s_i \subset s_j$. By the definition of subtree filaments, we can construct such filaments - if $s_i \supset s_j$, then f_i is drawn entirely above f_j.

We now modify some filaments to create intersection where it ought to exist. For every pair of filaments f_i and f_j that are non-intersecting, but $s_i \between s_j$, let p be an arbitrary point on T such that $p \in s_i \cap s_j$. Let f_j be the lower of the two filaments above p. We take an point directly above p on f_j and draw it upwards so that it intersects f_i. For each filament f_k that we encounter when drawing the point up, if $s_k \between s_j$, we simply intersect it, otherwise if $s_k \supset s_j$, we draw a point of f_k up as well, such that $f_k | f_j$. We say that f_k was *pushed* up. We call each of these parts of a filament extended upwards a *spike.*

We show that the introduction of the spikes has not caused any two filaments to incorrectly overlap. We do this by contradiction - given that spikes are drawn strictly upward, we need only consider the containment case, and not disjointness.

Assume f_i intersects f_j and $s_i \supset s_j$. It must be that f_j was extended upward in a spike to intersect f_i. f_j can have been drawn up, or it can have been pushed up. It can have been extended upwards to intentionally reach either f_i or to reach some filament f_u above f_i. We therefore consider four cases:

Case 1: f_j was drawn up with the intention of overlapping f_i - this is a contradiction against $s_i \supset s_j$.

Case 2: f_j was drawn upwards to intersect f_u, a filament above f_i - in this case, f_i would have been pushed upwards, and not intersected f_j.

Case 3: Some filament f_k below f_j was drawn up to intersect f_i, and f_j was pushed upwards. In this case, $s_k \between s_i$, $s_j \supset s_k$ - a contradiction with $s_i \supset s_j$, as there must be some node in s_k that is not in s_i, and therefore that same node in s_j that is not in s_i.

Case 4: Some filament f_k below f_j was drawn upwards to intersect some filament f_u above f_i, and f_j was pushed, but f_i was not. In this case, it must be that $s_k \between s_i$, and similarly to the above case, we then have some node in s_k (and therefore s_j) that is not in s_i, a contradiction to $s_i \supset s_j$. □

Every Subtree Filament Graph is a Subtree Overlap Graph

Given filaments $F = \{f_1 ... f_n\}$ on tree T, we show the construction of a tree T' and a set S of subtrees of T such that for all $1 \leq i, j, \leq n$, $s_i, s_j \in S$ overlap if and only if f_i and f_j intersect.

Let T' be a tree with the same structure as T (and for convenience, we will refer to analogous nodes on T and T' as q and q'), but with n additional nodes $P = \{p_1 ... p_n\}$ attached as follows: for $1 \leq i \leq n$, p_i is adjacent to one node q_i' such that q_i is an endpoint of f_i on T.

Now for $1 \leq i \leq n$, s_i consists of: each node q' in T' such that $q \in ind(f_i, T)$, the leaf p_i, and each leaf p_j such that $ind(f_i, T) \supset ind(f_j, T)$ and f_i does not intersect f_j.

We now prove that for every two subtrees $s_i, s_j \in S$, $s_i \between s_j$ if and only if f_i intersects f_j.

We consider several cases. We first consider the cases in which f_i intersects f_j, proving that this implies $s_i \between s_j$, and then the cases in which $f_i | f_j$, showing that this implies that s_i and s_j are either disjoint, or one is contained in the other.

Case 1: f_i intersects f_j and $ind(f_i, T) \between ind(f_j, T)$. There are then the following nodes in T: $q \in ind(f_i, T) \cap ind(f_j, T)$, $r \in ind(f_i, T) \backslash ind(f_j, T)$, $o \in ind(f_j, T) \backslash ind(f_i, T)$. There are therefore the analogous nodes in T' that are in s_i and s_j: $q' \in s_i \cap s_j$, $r' \in s_i \backslash s_j$, $o' \in s_j \backslash s_i$. Therefore $s_i \between s_j$.

Case 2: f_i intersects f_j and $ind(f_i, T) \supset ind(f_j, T)$. In this case, the leaf p_j of T' is in s_j and not s_i. The subtrees s_i and s_j also share nodes, and s_i contains some node not in s_j (from the assumption that no two subtrees have the same endpoints) and therefore $s_i \between s_j$.

Case 3: $f_i | f_j$, and $ind(f_i, T) | ind(f_j, T)$. In this case, s_i and s_j share no nodes, and are disjoint.

Case 4: $f_i | f_j$, and $ind(f_i, T) \supset ind(f_j, T)$. Since $(s_j \backslash P) \subset s_i$, we need only ensure that there is no $p_k \in P$ that is in s_j but not in s_i.

We do this by contradiction: assume that there is a p_k in s_j that is not in s_i. Since $p_j \in s_i$, it must be that $k \neq j$. This means that there is a filament f_k such that $ind(f_k, T) \subset ind(f_j, T) \subset ind(f_i, T)$, where f_k intersects f_i, but not f_j.

Since $ind(f_k, T) \subset ind(f_j, T) \subset ind(f_i, T)$, we know that f_k is below both f_i and f_j at some point in the surface P, and since $f_i | f_j$, f_j is entirely below f_i. To intersect f_i, f_k must extend above f_i at some point. f_k is then above f_j at some point, causing them to intersect, a contradiction.

We have shown that every subtree overlap graph is a subtree filament graph, and *vice versa.* □

3.2 Subtree Overlap Graphs are the complements of Cochordal-Mixed Graphs

In [15], Gavril gave a proof that a graph is an interval-filament graph if and only if its complement is a co-interval mixed graph. He states that a graph is a subtree filament graph if and only if its complement is cochordal mixed, and says that this can be proved by a proof similar to the one for interval filament graphs.

Here, we explicitly provide that proof. We draw very heavily on Gavril [15].

Theorem 3.3. *A graph is a subtree filament graph if and only if its complement is cochordal mixed.*

Proof:

We know from Gavril [15] that the complement of a subtree filament graph is cochordal-mixed. Briefly, the reason for this is as follows: Let $G_F = (V, E)$ be a subtree filament graph, and $R = (F, T)$ a filament representation, where F is a set of filaments on tree T. Let S be the subtrees on T that are induced by the endpoints of the filaments in F. For every edge (v_i, v_j) in $\overline{G}_F = (V, \overline{E})$ (or non-edge in G_F), either $s_i \subset s_j$, $s_j \subset s_i$, or $s_i|s_j$. This leads us to a partition of $\bar{E}$ into the disjoint sets $E_1 = \{(v_i, v_j)$ such that $s_i|s_j\}$ and $E_2 = \{(v_i \rightarrow v_j)$ such that $s_i \subset s_j\}$. E_2 is transitive because it is based on containment, and E_1 is cochordal because its edges are due to disjointness. Finally, if v_i, v_j, and v_k are three vertices in V such that $(v_k \rightarrow v_j) \in E_2$ and $(v_i, v_j) \in E_1$, then it must be that $s_k \subset s_j$ and $s_i|s_j$, and therefore $s_k|s_i$ and $(v_i, v_k) \in E_1$. $\overline{G}_F$ is therefore cochordal-mixed [15].

It remains to show that if a graph's complement is cochordal mixed, then the graph is a subtree filament graph. We will do this by constructing a set of filaments on a subtree, and showing that the graph is the intersection graph of these filaments.

Let $G = (V, E)$ be a graph. Let $\bar{G} = (V, E_1 \cup E_2)$ be a cochordal mixed graph, $G_a = (V, E_2)$ a transitive graph, and $G_b = (V, E_1)$ the complement of the chordal graph $G_c = (V, \bar{E}_1)$. As discussed in Chapter 2, we know that G_c is the intersection graph of some set of subtrees in a tree T. Let S be such a set of subtrees of tree T. Two subtrees s_i, $s_j \in S$ have non-empty intersection if and only if $(v_i, v_j) \in \bar{E}_1$, which we can restate as $(v_i, v_j) \notin E_1$.

Let v_i and v_j be two vertices in V such that $(v_i, v_j) \in \bar{E}_1$, and $(v_i \rightarrow v_j) \in E_2$, and $s_i \between s_j$. Assume that there is some third vertex v_k such that $(v_i, v_k) \in \bar{E}_1$ and $(v_j, v_k) \notin \bar{E}_1$. Since $(v_j, v_k) \notin \bar{E}_1$ implies $(v_j, v_k) \in E_1$, and $(v_i \rightarrow v_j) \in E_2$, then from the definition of $\mathbf{G} - mixed$ graphs provided by Gavril [15] and given in Section 2.3, we have that $(v_i, v_k) \in E_1$.

This contradicts $(v_i, v_k) \in \bar{E}_1$. Therefore, every vertex adjacent to v_i in G_c is also adjacent to v_j. Thus, we can expand s_j to contain s_i, preserving disjointness, and so we can assume that for two adjacent (in G_c) vertices v_i and v_j, if $(v_i \rightarrow v_j) \in E_2$, then $s_i \subset s_j$.

So, this means that for two vertices v_i and v_j in V, if $(v_i, v_j) \in E$, then s_i and s_j are non-disjoint, and if $(v_i \rightarrow v_j) \in E_2$, then $s_i \subset s_j$. If $(v_i, v_j) \notin E$ and $(v_i \rightarrow v_j), (v_i \leftarrow v_j) \notin E_2$ then $s_i|s_j$. If $(v_j \rightarrow v_i) \in E_2$ then $s_i \supset s_j$.

Now we will create and transform filaments above T. We add above each $s_i \in S$ a filament f_i connecting the endpoints of s_i, such that for two subtrees s_i and s_j, if $s_i|s_j$, then f_i and f_j do not intersect and if $s_i \supset s_j$, then f_i is completely above f_j.

Now we alter the filaments by adding spikes to them in order to represent the edges that are not currently represented by filament intersection. Let P be a directed traversal ordering on G_a, such that for no two subtrees $s_i \subset s_j$ does v_j occur before v_i in the ordering.

Now, we iterate through P. For each f_i, where $i < |P|$, we consider each subtree $f_j, i < j < |P|$. If v_i is adjacent to v_j in G, and therefore s_i and s_j are not disjoint, and f_i does not intersect f_j yet, then we stretch some point of f_i, directly above a point q of T such that $q \in s_i \cap s_j$, upwards away from the plane of T to intersect f_j. For every filament f_k encountered on the way up such that $(v_i \rightarrow v_k) \in E_2$, we also stretch that filament upwards such that there is no intersection of f_i and f_k, nor between any two such f_k. However, there will now be intersection between f_i and f_j, and between all the f_k and f_j.

We now show by contradiction that for every f_k stretched in this way, v_k and v_j are adjacent in G. Assume that $(v_k, v_j) \notin E$. Since s_k and s_j are non-disjoint, we know that $(v_j, v_k) \in \bar{E}_1$, and so $(v_j, v_k) \notin E_1$. Combining this with the fact that (v_j, v_k) must be in $E_1 \cup E_2$, we have that $(v_j, v_k) \in E_2$. Since during the construction of the spike we encountered f_k below f_j, we know that $s_k \subset s_j$, and therefore $(v_k \rightarrow v_j) \in E_2$. Since E_2 is transitive, and $(v_i \rightarrow v_k) \in E_2$, we have $(v_i \rightarrow v_j) \in E_2$ - this is a contradiction with these vertices being adjacent in G. Therefore $(v_k, v_j) \in E$.

We have completed our construction of the subtree filaments. We show that G is the intersection graph of these filaments.

Let v_i and v_j be two vertices adjacent in $G = (V, E)$. Since these vertices are adjacent in G_c, s_i and s_j intersect. If $s_i \between s_j$, then by the spike construction f_i intersects f_j,. If $s_i \subset s_j$, a spike was added to f_i so that f_i intersects f_j.

Let v_i and v_j be two vertices non-adjacent in $G = (V, E)$. We have that $(v_i, v_j) \in E_1 \cup E_2$. We will consider two cases: $(v_i, v_j) \in E_2$ or $(v_i, v_j) \in E_1$.

If $(v_i, v_j) \in E_1$, then s_i and s_j are disjoint, and therefore f_i and f_j do not intersect.

If $(v_i \rightarrow v_j) \in E_2$, then $s_i \subset s_j$. We will show by contradiction that f_i and f_j do not intersect. Assume that f_i and f_j intersect. This intersection must have been produced by a spike. Since we would not have directly created a spike in f_i to f_j, there must be some filament f_k below f_i that was stretched upwards, and pushed v_i upwards. f_k can have been stretched upwards to intersect either f_j, or some f_l above f_j.

Therefore we have f_i such that v_i and v_k are not adjacent in G, $(v_k \rightarrow v_i) \in E_2$, and either $(v_j, v_k) \in E$ or $(v_k, v_l) \in E, (v_j \rightarrow v_l) \in E_2$. However, in either case, by the transitivity of E_2, if $(v_k \rightarrow v_i) \in E_2$ and $(v_i \rightarrow v_j) \in E_2$, then $(v_k \rightarrow v_j) \in E_2$ and (v_k, v_j) could not be in E, a contradiction.

Therefore, two filaments f_i and f_j intersect if and only if v_i and v_j are adjacent in G, and so G is the intersection graph of the filaments created. □.

Corollary 3.1. *As a Corollary of Theorems 3.2 and 3.3, we have that the following are equivalent statements about a graph G:*

- *G is a subtree overlap graph*

- *G is a subtree filament graph*
- *G is the complement of a cochordal mixed graph*

The proofs of Theorems 3.2 and 3.3 imply that, if we were given a graph G, and a partition of the edges of $\bar{G}$ into a transitive graph G_a and a cochordal graph G_b, then we can produce both a subtree overlap representation, and a subtree filament representation for G. We first obtain a subtree intersection model for $\bar{G}_b$ by an algorithm presented in [14]. Using this, and the construction in the proof of Theorem 3.3, we produce a set of filaments F of which G is the intersection graph. We then use the construction from the proof of Theorem 3.2 to create, from F, subtrees of which G is the overlap graph. The time complexity of constructing the subtree intersection model using the algorithm described originally in [14] is $O(n^4)$. However, the limiting step in that algorithm was the identification of a series of simplicial vertices. This can now be done in linear time [27], so the limiting step here is the creation of the filaments and subtrees. Adding spikes to the filaments takes $O(n^2)$ time due to the nested iteration through P. Creating a subtree overlap representation from the filaments takes $O(n^2)$ due to the creation of the new leaves - for each filament, its intersection with each other filament may need to be checked. The time complexity of the creation of filaments and then subtrees is $O(n^2)$, where n is the number of vertices in G. The overall time complexity is therefore $O(n^2)$.

3.3 Restricted Paths in a Tree

We now turn our attention to the overlap graphs of paths in a tree. We will show that if the paths are restricted in their overlapping in a certain way then the graphs representable are exactly the circle graphs.

First, we must define a few new notions. Let s be some subtree in a subtree overlap representation $R = (S, T)$. A *boundary node* is defined with respect to a particular subtree. A node p is a boundary node of s if that node is either a leaf of T, or has some neighbor node that is not in s. A *non-leaf boundary node* is a boundary node that is not a leaf of the subtree. A *divergence node* is any node in T with degree greater than 2. For an illustration of these definitions, see Figure 3.1.

The overlap graphs of intervals on a line (or subpaths of a path) are the circle graphs [13]. For convenience of direct comparison to subtrees and trees, we will discuss these graphs as overlap graphs of subpaths on a path here, and not as intervals on a line.

Observation 3.1. *In an overlap representation of subpaths of a path, those subpaths of a path have no non-leaf boundary nodes.*

This observation can be made by noting that subpaths on a path have only two endpoints, and all nodes between those are in the subpath. For any of the non-endpoint nodes to be boundary nodes, they would have to have a neighbor not on the subpath between the two endpoints, and the underlying structure could not be a path.

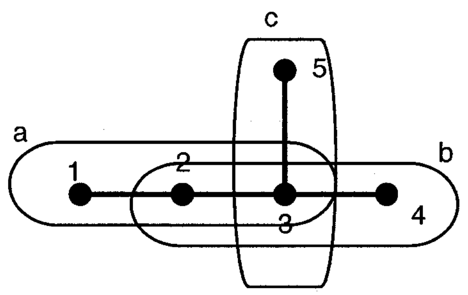


Figure 3.1: A subtree overlap representation with labeled subtrees and nodes. Nodes 1, 5, and 4 are leaves of the underlying tree. Nodes 1 and 3 are leaf boundary nodes of subtree a. Subtree a has no non-leaf boundary nodes. Nodes 2, 3 and 4 are boundary nodes of subtree b. Node 3 is a non-leaf boundary node of subtree b. Nodes 5 and 3 are boundary nodes of subtree c. Node 3 is a divergence node.

3.3.1 Subpaths in a Tree with no Non-Leaf Boundary Nodes

Let $G = (V, E)$ be an overlap graph of subpaths in a tree T such that no subpath has any non-leaf boundary nodes.

Observation 3.2. *In an overlap representation of subpaths in a tree in which no subpath has a non-leaf boundary node, no subpath may contain any divergence node d and more than one neighbour of d.*

That is, for any subpath that contains a divergence node, that divergence node is an endpoint of that subpath. This can be seen by contradiction. No subpath can contain any more than two neighbours of any node, or else it would not be a subpath. Assume that some subpath contains a divergence node, as well as two of its neighbors. There must then be at least one neighbor of the divergence node that is not in the subpath. The divergence node is then a non-leaf boundary node of that subpath, forbidden in this representation of G.

Observation 3.3. *The set of subpaths containing only nodes on a particular minimal segment of the tree connecting two divergence nodes, or a divergence node and a leaf in the tree, are a set of subpaths on a path. The corresponding vertices in G therefore induce a circle graph.*

This follows from the fact that any such segment must be a path, and from Observation 3.2.

Let there be some divergence node d in the tree. Let W and X be two paths in the tree that connect d to two different divergence nodes, or leaves. Let Q and R be sets that contain the subpaths containing nodes from W and X, respectively. Let Q_d and R_d be the subsets of Q and R, respectively, that contain the members of Q and R that contain d. H is a subgraph of G induced by the vertices corresponding to subpaths in Q. I is a subgraph of G induced by vertices corresponding to subpaths in R. H and I are circle graphs, from Observation 3.3. H_d is a subgraph of H (and G) induced by vertices corresponding to subpaths in Q_d. I_d is a subgraph of I (and G) induced by vertices corresponding to subpaths in Q_d.

I_d and H_d are completely joined in G. Since no subpath in Q_d can contain, or be contained in, any subpath in R_d, but all share the node d, every subpath in Q_d must overlap every subpath in R_d, every vertex in I_d must be adjacent to every vertex in H_d, and there are no other edges between H and I. We present a transformation from this representation on a tree to a representation on a path.

Given a divergence node d with a neighborhood N of size $|N|$, with at least $|N| - 1$ of the branches having no divergence nodes of their own, we can collapse these $|N| - 1$ flat branches into one branch with no divergence nodes while preserving overlapping of paths.

First, pick some order of the flat branches. Place the entirety of branch 1 (except d) onto the edge between d and the first node in branch 2, with all of branch 1 being contained in exactly the paths of branch 2 that contained d. No overlapping has been introduced, though the overlapping of paths from branch 1 that contained d with the paths from branch 2 that also did so has been destroyed. Now, repeat this process with branches 2 and 3, 3 and 4, through $|N| - 2$ and $|N| - 1$.

Once all of these branches have been nested inside each other, we have a path with no branching on it. There is no overlapping between subpaths that were formerly on different branches, only containment and disjointness. Now it remains to restore the overlapping of the subtrees from former different branches that contain d. Note that d now only has degree two, and so is not a divergence node at all. Between d and its neighbor node that was not on any of the branches collapsed, add $|N| - 1$ nodes in a path, such that the paths containing d from branch 1 contain all the new nodes, the paths containing d from branch 2 contain all but the farthest of the new nodes, the paths containing d from branch 3 contain all but the farthest 2 of the new nodes, etc. That is, if the new nodes are labeled 1 through $|N| - 1$ (with node $|N| - 1$ being the closest to d), then the paths from the former branch i contain nodes $i, i + 1 ... |N| - 1$.

If we assign an orientation to the resulting structure such that the new nodes are to the left, and the flattened branches to the right, then the overlapping can be proven as follows: for each pair of former branches, branch i and branch j, where $i < j$, let the sets of paths containing d from these former branches be called i_d and j_d respectively. Since each path in i_d was inserted (at some nested level) into the edge between d and the first non-d node in each of the paths in j_d, then all of the paths in j_d extend farther to the right than any path in i_d. Since each of the paths in i_d contains a new node to the left of d that is not contained in any path in j_d, then all of the paths in i_d extend farther to the left than any path in j_d. Therefore, no path in i_d contains, or is contained in j_d, but all paths in i_d and j_d contain d, so none are disjoint. Therefore, every path in i_d overlaps every path in j_d, just as they did before the branch collapsing operation.

Theorem 3.4. *Repeated application of the transformation outlined above on subpaths in a tree with no non-leaf boundary nodes, the overlap graph of which is G, results in a set of subpaths in a path, the overlap graph of which is G. The overlap graphs of subpaths in a tree with no non-leaf boundary nodes are therefore exactly the circle graphs.*

Proof:

Given that a single application of the transformation to a set of subpaths in a tree with no non-leaf boundary nodes preserves overlapping as well as the transformation's preconditions, repeated application will also preserve overlapping. At any point, the transformation can be legitimately applied to any divergence node closest to the leaves.

Since the transformation can always be applied to this type of representation, each application destroys a divergence node, and there were a finite number of divergence nodes at the beginning, the repeated application of the transformation will eventually destroy all divergence nodes. At this point we have a path as the underlying tree. As overlapping has been preserved, the overlap graph of the subpaths on the path is the same as the overlap graph of the subpaths on the original tree. □

Chapter 4

Consistency of a Representation

We are interested in recognizing subtree overlap graphs. One approach to the recognition problem is to view it as two sub-problems - creating a representation if possible, and then checking to see if the representation created is valid. It is useful to know which phase of this problem is the difficult one. Here, we consider the task of checking a particular format of representation information.

4.1 Consistency of General Representations

Let $S = \{s_1, ..., s_n\}$ be a set of sets, no two of which are equal. The *relationship matrix* for S is an $n \times n$ matrix M with entries:

$$M_{ij} = \begin{cases} \between & if \quad s_i \between s_j \\ | & if \quad s_i | s_j \\ \supset & if \quad s_i \supset s_j \\ \subset & if \quad s_i \subset s_j \end{cases}$$

Recall from Chapter 2 that we do not allow sets to be equal.

No information about the elements in the sets themselves is included in M, only information about their relationships with regards to overlapping, containment, and disjointness. If M is the relationship matrix for the set of sets S, we say that S *realizes* M. We say that a relationship matrix M is *realizable* if there exists some set S of sets that realizes M.

Given M, we can construct an overlap, disjointness, containment, or intersection graph of the realizing set of sets S, if S exists, without knowing S itself. Therefore, each relationship matrix also represents overlap, disjointness, containment, and intersection graphs.

We can then pose the question: given a relationship matrix M, can we check in polynomial time if M is realizable? We answer this question by presenting here a method for constructing sets to realize M, if M can be realized.

Theorem 4.1. *Relationship matrix M is realizable if and only if Algorithm 1 produces a set of sets that realizes M.*

Proof:

Algorithm 1: Builds a set of sets S that realizes relationship matrix M, if possible
Input: A $n \times n$ relationship matrix M
Output: true if M can be realized, false if not
CANREALIZE(M)

```
$S \leftarrow \emptyset$
foreach $i, 1 \leq i \leq n$
    $s_i \leftarrow \emptyset$
    $S \leftarrow S \cup \{s_i\}$
#COMMENT - Overlap Phase begins here
foreach $s_i$, $s_j \in S$
    if $M_{i,j} = \between$
        Create three new elements $a$, $b$ and $c$
        $s_i \leftarrow s_i \cup \{a, b\}$
        $s_j \leftarrow s_j \cup \{b, c\}$
#COMMENT - No-overlap Phase begins here
# This is to ensure that sets that overlap no other sets are not empty
To each subtree $s_j$ such that $s_j = \emptyset$ add a unique new element
#COMMENT - Containment Phase begins here
Let $D$ be an ordering of the sets in $S$, such that there is no set $s_i$ preceding any set
$s_j$ in $D$, where $M_{ij} = \supset$
foreach $s_i \in D$
    Add to $s_i$ all elements of every set $s_j$, where $M_{ij} = \supset$
#COMMENT - Consistency Phase begins here
foreach $i, j \in 1, 2..n$
    if $M_{i,j} = \supset$ and $\neg(s_i \supset s_j)$
        return false
    else if $M_{i,j} = \subset$ and $\neg(s_i \subset s_j)$
        return false
    else if $M_{i,j} = \between$ and $\neg(s_i \between s_j)$
        return false
    else if $M_{i,j} = |$ and $\neg(s_1 | s_j)$
        return false
return true
```

First, if the sets satisfy the relationships, then clearly the relationships are consistent. To show that if the sets do not satisfy the relationships, then the relationships are inconsistent, we will examine cases.

Case 1. The Contains Check Fails

Subcase 1.1: $M_{ij} = \supset$ and $s_i | s_j$

It is not possible for two sets that should be in a containment relationship to be disjoint, as the outer one was expanded during the containment phase to include the elements in the inner one.

Subcase 1.2: $M_{ij} = \supset$ and $(s_i \subset s_j$ or $s_i \between s_j)$

s_i cannot have occurred before s_j in the D ordering in the containment phase. At some point in the containment phase, all elements of s_j were added to s_i. s_j cannot have been changed after that, so s_i must contain s_j. This case therefore cannot occur.

Case 2: The Overlapping Check Fails

Subcase 2.1: $M_{ij} = \between$ *and* $s_i | s_j$

This case cannot occur. During the overlap phase, for every pair of i and j such that $M_{ij} = \between$, a common element was added to s_i and s_j. These two sets therefore share at least one element, and cannot be disjoint.

Subcase 2.2: $M_{ij} = \between$ *and* $s_i \supset s_j$

During the overlap phase, three elements, a, b, and c were created for this pair of sets. a was added only to s_i, c only to s_j, and b to both. a and c were not added to any other sets during the overlapping phase. For c to be an element of s_i, it would have had to be added during the containment phase. Only sets s_k, for which $M_{k,j} = \supset$ would have gained c, or sets containing containers of s_j. Since $M_{i,j} \neq \supset$, s_i must contain some container of s_j. This is then an inconsistency in M, since containment is transitive.

Case 3. The Disjointness Check Fails

$M_{ij} = |$ *and* $s_i \cap s_j \neq \emptyset$

Since no element in common was added to these two sets during the overlap phase, and single unique elements in the no-overlap phase were added to only one set, heir shared elements can only be due to expansion during the containment phase. If s_i contains some set non-disjoint from s_j, then the sets cannot be disjoint. M is inconsistent.

We have shown that if the sets produced by Algorithm 1 do not realizable M, then M is inconsistent with respect to set properties. By definition, if the sets realize M, then M is consistent. Therefore M is consistent if and only if the sets produced by this algorithm realize it. □

While we have produced sets that realize the matrix M if such sets exist, these sets are likely not a smallest realization of M, where the smallest representation has the fewest elements in the union of all sets created. While new elements are created only in the overlap phase, three new elements are created for every overlapping pair. The sets in S can be seen as a representation of the overlap, containment, disjointness, and intersection graphs represented by M.

4.2 Consistency of Subtree Overlap Representations

We have shown that we can check the consistency of a relationship matrix, where the sets described by the relationship matrix are unrestricted. However, we are interested primarily in subtree overlap graphs.

We would therefore like to resolve two questions: given a relationship matrix M, is the overlap graph of M a subtree overlap graph? More importantly, given a relationship matrix M, can M be realized by a set of subtrees in a tree? We have not resolved either of these questions.

	a	b	c	d
a	-	≬	|	≬
b	≬	-	≬	|
c	|	≬	-	≬
d	≬	- |	≬	-

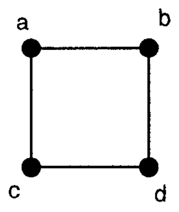


Figure 4.1: A relationship matrix and its overlap graph.

It is important to note that these questions are not equivalent - the overlap graph of a relationship matrix may be a subtree overlap graph, without that particular matrix providing a valid subtree overlap representation. As an example, consider the possible relationship matrices of which the overlap graph is a cycle. A matrix with no containment can produce the four-cycle as an overlap graph (Figure 4.1), but cannot itself be satisfied by subtrees in a tree.

Chapter 5

Representations of a Few Simple Graphs

There are a few simple graphs for which we would like to know all the representations. We are particularly interested in graphs with very few or unique subtree overlap representations. Such graphs could be ultimately useful in the recognition problem - in constructing a representation they might be components of a graph that restrict the number of possible representations. Here we look at cycles, which have few non-isomorphic representations, and paths, which have many.

5.1 Cycles

Firstly, we use a lemma due to Rosgen [28]:

Lemma 5.1. *[28] Let $G = (V, E)$ be an overlap graph, and $R = (S, T)$ an overlap representation of G. Let $v, u \in V$ be two non-adjacent vertices. Let V_2 be $V - \{v\} - N(v)$. Let $A_u \subseteq V_2$ be the vertex set of the connected component of the graph $G_2 = (V_2, E)$ in which u appears. If, in R, $s_v \supset s_u$, then $s_v \supset s_i$ for all s_i corresponding to $v_i \in A_u$.*

We can now proceed to show that:

Lemma 5.2. *Let G be a chordless cycle of arbitrary size greater then three. G has exactly three non-isomorphic set overlap representations.*

Proof:

Let $C = (V, E)$ be a chordless cycle of arbitrary size. In any overlap representation of C, by Lemma 5.1, for any set s_i corresponding to some vertex $v_i \in V$ such that s_i contains some set s_k corresponding to some other vertex, $v_k \in V$, s_i also contains all other sets in the subtree overlap representation of C that it does not overlap.

Let $v_i \in V$ and $v_j \in V$ be neighbors, and $v_k \in V$ be some vertex non-adjacent to v_i. Let the set corresponding to v_i, s_i, contain all sets except those overlapping it. s_k must be disjoint from all but its overlappers, and s_i, which contains s_k. If it were not, it would contain s_i - a contradiction. However, s_j may contain all sets except its overlappers, as s_i is an overlapper of s_j. At most two overlapping sets may contain all but their overlapping sets - in any group of three or more, at least two are non-overlapping. There are therefore three options for representation: no set may contain any other, one set may contain others, or two overlapping ones may contain others. An illustration of the three possible representations of a cycle can be found in Figure 5.1. □

Lemma 5.3. *Let $G = (V, E)$ be a chordless cycle of arbitrary length greater than three. G has exactly two representations as the overlap graph of subtrees of a tree, both of which can be expressed as intervals on a line.*

Proof:

Of the three non-isomorphic representations proven to exist in Lemma 5.2, one cannot be produced by subtrees in a tree - the one with no containment. A realization of that representation with subtrees would result in a cycle in the underlying tree. The other two can be realized by intervals on a line, as shown in Figure 5.1.

While we have shown that there are only a few non-isomorphic overlap representations of the cycle, and only two non-isomorphic representation of a cycle as intervals on a line, or subtrees in a tree, note that there are many possible actual representations. There are many sets of sets that can realize those representations, though many of those sets of sets will be isomorphic. □

5.2 Paths

The path might appear to be an even simpler graph than the cycle. However, it turns out that, while the cycle has few non-isomorphic representations, the path has many. While we haven't determined the precise number of non-isomorphic representations of a path of length n, we have an exponential lower bound, as described in the rest of this section.

Let $P_k = (V, E)$ be a path of length k, and $R = (S, T)$ a subtree overlap representation of P_k. Let us assign left and right sides to P_k for convenience, numbering each vertex in V. Let the indices of vertices (and corresponding subtrees) increase from left to right. That is, v_1 is adjacent to exactly v_2, v_i is adjacent to exactly v_{i-1} and v_{i+1} for all $1 < i < k$, and v_k is adjacent to only v_{k-1}.

Theorem 5.1. *There are at least 2^{k-2} non-isomorphic subtree overlap representations of P_k.*

Proof:

Due to Lemma 5.1, if some subtree s_i contains some subtree s_j on this path, it must also contain all subtrees in the "direction" of s_j along the path. If we assign the path left and right sides, then if s_j is to the left of s_i, s_i must contain all subtrees to its left, except for those it overlaps. Any subtree s_l to the left of s_i cannot contain any subtrees to its right, as then it would then have to contain s_i. This is a conflict, as $s_i \supset s_l$.

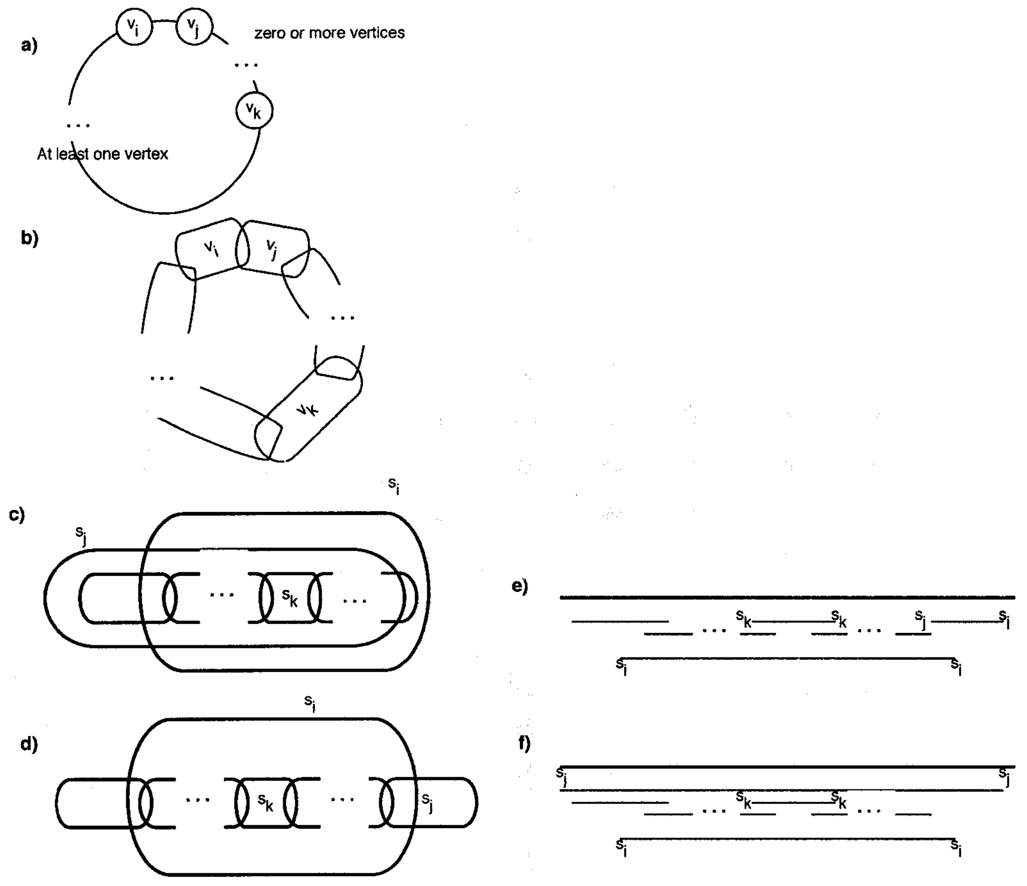


Figure 5.1: A cycle of arbitrary size with a few labeled vertices (a), its three possible representations as overlapping sets (b, c, d), and the realization of c and d as intervals on a line (e, f). The ellipses indicate an arbitrary number of additional intervals, sets, or vertices as appropriate.

Therefore, if $s_i \supset s_j$ are two subtrees in a subtree overlap representation of P_k such that $j < i$, then there can be no subtree $s_p, p < j, s_p \supset s_l$ such that $l > p$.

For the purposes of establishing a lower bound, let us only consider the number of representations in which containment must always go from right to left - that is, $s_i \supset s_j \rightarrow i > j$.

We then wish to know how many possible minimal non-isomorphic subtree overlap representations of P_k there are, given the directional containment restriction.

We will show how to construct, for any k of at least three, a set of 2^{k-2} pairwise non-isomorphic subtree overlap representations for P_k. Throughout this proof we assume that the numbering and directional contains restriction are in effect.

First, consider the path P_3. There are two non-isomorphic representations of P_3: either s_3 may contain s_1 or not. These two can be seen in Figure 5.2.

Now consider a path P_k. Assume there are 2^{k-3} representations of the path P_{k-1}, or P_k without v_k, and that in all of them s_{k-1} has some node not in any other subtree. We can assume this because, since v_k is not contained in any other subtree (due to the directional contains constraint), a leaf can be added to the representation that is only in v_k.

For each of the 2^{k-3} representations of P_{k-1}, we can construct two representations of P_k. For convenience, we will refer to the subtrees on the first and second representations with 1 and 2 superscripts, respectively. We construct the representations as follows: Let $R_{k-1} = (S_{k-1}, T_{k-1})$ be a representation of P_{k-1}. Let T_1 be equal to T_{k-1}. Let T_2 be equal to T_{k-1} with an additional leaf p_2 adjacent to only the node q_2 that is found only in s^2_{k-1}. Let S^1 and S^2 be copies of the subtrees in S_{k-1} on T_1 and T_2, respectively. Let q_1 be the node in only s^2_{k-1} in T_1. Let the subtree s^1_k be $T_1 - q_1$. Let s^2_k be q_2 and p_2. Two representations of P_k are then $R_1 = ((S^1 \cup \{s^1_k\}), T_1)$ and $R_2 = ((S^2 \cup \{s^2_k\}), T_2)$.

R_1 and R_2 are non-isomorphic: in R_1, s_k contains other subtrees, and in R_2 it contains no others. Given the assumption that all 2^{n-3} representations of P_{k-1} are pairwise non-isomorphic, there are 2^{n-2} representations of P_k

An abstract illustration of the construction of R_1 and R_2 from R_{k-1} is shown in Figure 5.3.

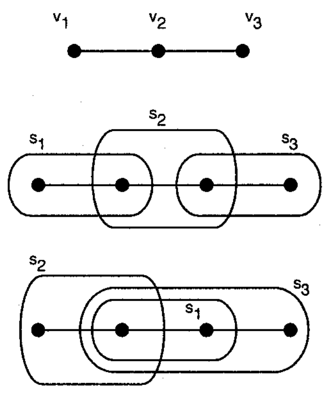


Figure 5.2: An illustration of a path of three vertices, and two non-isomorphic subtree overlap representations of a path of three vertices.

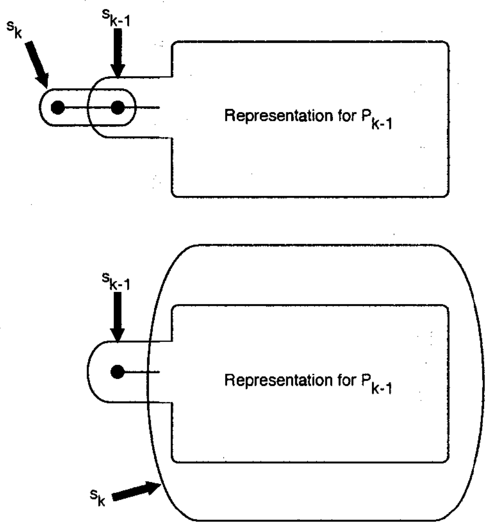


Figure 5.3: An illustration of two non-isomorphic subtree overlap representations of a path of k vertices given a representation of a path of $k - 1$ vertices.

Chapter 6

Internal and External Subtrees

Not every subtree is external in every subtree overlap representation. Some subtrees can never be external in any subtree overlap representation. Understanding which vertices in a subtree overlap graph can be external or internal in some valid subtree overlap representation is relevant not only to eventually constructing subtree overlap representations, but also to the application of decompositions to be discussed in future chapters. In this section, we consider the problem of internal and external subtrees in some of the subclasses of subtree overlap graphs. Specifically, we pose the question: Given a subtree overlap graph $G = (V, E)$, and a vertex $v_i \in V$, does there exist a subtree overlap representation for G in which s_i is internal? That is, is v_i a vertex that can be internal? Similarly, we wish to know if v_i can be external.

6.1 Spider Graphs

Lemma 6.1. *Any vertex in a spider graph can be external.*

Proof:

As discussed in the introduction, there is a transformation from a spider representation to an overlap representation of subtrees in a caterpillar. This transformation involves iterating through the endpoints on the spider representation. The starting point is arbitrary, and the subtree corresponding to the set of endpoints that has the first endpoint encountered is placed first on the caterpillar. The subtree placed first on the caterpillar is the only one to contain the node at the extreme end of the caterpillar's spine. This subtree is therefore not contained in any other. Since this subtree is arbitrary, any vertex in a spider graph can be external. □

It is currently unknown if an arbitrary vertex of a spider graph can be internal.

6.2 Chordal Graphs

Chordal graphs have subtree overlap representations without containment [14]. Therefore there is a subtree overlap representation for every chordal graph in which every subtree is both external and internal.

6.3 Cocomparability Graphs

6.3.1 External Subtrees of Cocomparability Graphs

Let G be some cocomparability graph. First, observe that any subtree that corresponds to a vertex that in $\bar{G}$ can be a source or a sink is internal in some representation of G without disjointness, and external in some other representation of G without disjointness. We know this from the construction method for a containment representation of a comparability graph presented by Golumbic and Scheinermen [16] and discussed in Chapter 2. However, there are some vertices in cocomparability graphs that can never be sources or sinks in a transitive orientation of the complement [25]. An example of such a vertex, taken from Olariu [25] can be found in Figure 6.1.

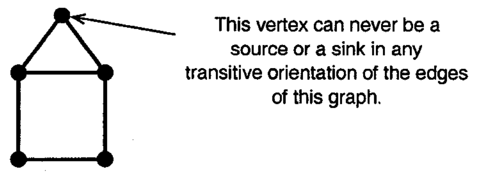

This vertex can never be a source or a sink in any transitive orientation of the edges of this graph.

Figure 6.1: A comparability graph, with a labeled vertex that is neither a source nor a sink in any transitive orientation of its edges, as shown in [25].

However, it seems reasonable that with some transformation to introduce disjointness in the representation, any subtree might be made external. We therefore make the following conjecture, and leave it for future work:

Conjecture 6.1. *Let $G = (V, E)$ be a cocomparability graph, and $v \in V$ an arbitrary vertex in V. Then v can be external.*

6.3.2 Internal Subtrees of Cocomparability Graphs

Theorem 6.1. *Let $G = (V, E)$ be a cocomparability graph, and $v_{internal} \in V$ an arbitrary vertex in V. Then $v_{internal}$ can be internal.*

Proof:

Let $G = (V, E)$ be a cocomparability graph, and $R = (S, T)$ a subtree overlap representation of G such that T is a star, and there is no disjointness in R. Let $v_{internal}$ be an arbitrary vertex of G.

Let $S_d \in S$ be the set of subtrees that are contained in $s_{internal}$. Let $S_o \in S$ be the set of subtrees that overlap $s_{internal}$. Let $S_c \in S$ be the set of subtrees that contain $s_{internal}$.

Let T_1 be a star isomorphic to T. Let the tree T_2 be isomorphic to T, but with the following leaves: let P be a set of $|S_d|$ leaves, with each leaf p_d corresponding to a subtree $s_d \in S_d$. Let p_d be adjacent to any node $q_2 \in T_2$ such that $q \in s_d$ in R, and q is not the central node of T. For convenience, we will refer to the nodes on T_1 and T_2 that correspond to the node q on T as q_1 and q_2, respectively. Let there be an edge between the central nodes of T_1 and T_2.

We define a set S' of subtrees on the union of T_1 and T_2. Each subtree in S' is defined as follows:

s'_i such that $s_i \in S_d$ comprises exactly every node $q_2 \in T_2$ such that $q \in s_i$ in R, and every leaf p_j such that $i = j$ or $s_i \supset s_j$ in R.

s'_i such that $s_i \in (S - S_d - \{s_{internal}\})$ comprises exactly every node $q_1 \in T_1$ such that $q \in s_i$ in R, and every node of T_2 except each leaf p_j such that $s_i \between s_j$ in R.

$s'_{internal}$ comprises exactly every node q_1 on T_1 such that $q \in s_{internal}$ in R.

It remains to prove that for all $1 \leq i, j \leq n, s'_i \between s'_j$ if and only if $s_i \between s_j$. We do this by examination of several cases.

Case 1: $s'_i \in (S_o \cup S_c)$ and $s'_j \in S_d$. s'_i contains all nodes of T_2 except each leaf p_k such that $s_k \between s_i$. Therefore s'_i contains s'_j unless s'_j contains some leaf p_k such that $s_k \between s_i$. The subtree s'_j will contain such a leaf if and only if either $s_j \between s_i$, or some subtree $s_l \subset s_j$ is such that $s_l \between s_i$. However, this implies that $s_j \between s_i$ - since s_i certainly contains some node not in s_j, they shared at least the central node in the original representation, and s_j contains the node in s_l that is not in s_i.

Therefore, s'_j contains a leaf that is not in s'_i (and therefore $s'_i \between s'_j$) if and only if $s_i \between s_j$.

Case 2: $s'_i \in (S_o \cup S_c)$ and $s'_j \in (S_o \cup S_c)$. If $s_i \between s_j$, then $s'_i \between s'_j$, as the nodes q, r, o, where $q \in s_i, q \notin s_j, r \in s_i, r \in s_j, o \notin s_i, o \in s_j$ must exist in T, and therefore q_1, r_1, o_1 must exist in T_1, such that $q_1 \in s'_i, q_1 \notin s'_j, r_1 \in s'_i, r_1 \in s'_j, o_1 \notin s'_i, o_1 \in s'_j$.

If $s_i \supset s_j$, then all nodes on T_1 that are in s'_j are also in s'_i, and we need consider only the nodes on T_2. Let us assume that there is some leaf $p_k \in T_2$ that is in only s'_j, and not s'_i, and show a contradiction. There is therefore some subtree $s_k \in S_d$ such that $s_k \between s_i$, but $s_k \subset s_j$. This is a contradiction: if s_k overlaps s_i, then it must have some node not shared with s_i, a contradiction to the transitivity of containment.

Case 3: $s'_i \in S_d$ and $s'_j \in S_d$. If $s_i \between s_j$, then $s'_i \between s'_j$, as the nodes q, r, o, where $q \in s_i, q \notin s_j, r \in s_i, r \in s_j, o \notin s_i, o \in s_j$ must exist in T, and therefore q_2, r_2, o_2 must exist in T_2, such that $q_2 \in s'_i, q_2 \notin s'_j, r_2 \in s'_i, r_2 \in s'_j, o_2 \notin s'_i, o_2 \in s'_j$.

If $s_i \supset s_j$, then by the construction, every node of $T_2 - P$ in s'_j is also in s'_i, and every leaf $p_k \in P$ that is in s'_j is also in s'_i.

Case 4: Consider $s'_{internal}$ and $s'_i \in (S_o \cup S_c)$ If $s_i \between s_{internal}$, then $s'_i \between s'_{internal}$, as the nodes q, r, o, where $q \in s_i, q \notin s_{internal}, r \in s_i, r \in s_{internal} o \notin s_i, o \in s_{internal}$ must exist in T, and therefore q_1, r_1, o_1 must exist in T_1, such that
$q_1 \in s'_i, q_1 \notin s'_{internal}, r_1 \in s'_i, r_1 \in s'_{internal}, o_1 \notin s'_i, o_1 \in s'_{internal}$.

If $s_i \supset s_{internal}$, then by the construction every node $q_1 \in T_1$ that is in $s'_{internal}$ is also in s'_i, because in R, every node q in T that is in $s_{internal}$ is also in s_i.

Case 5: Consider $s'_{internal}$ and $s'_i \in (S_d)$ Here, we know that $s_{internal} \supset s_i$. Since every node in $s'_{internal}$ is in T_1, and every node in s'_i is in T_2, and T_1 and T_2 are disjoint, $s'_{internal}|s'_i$. □

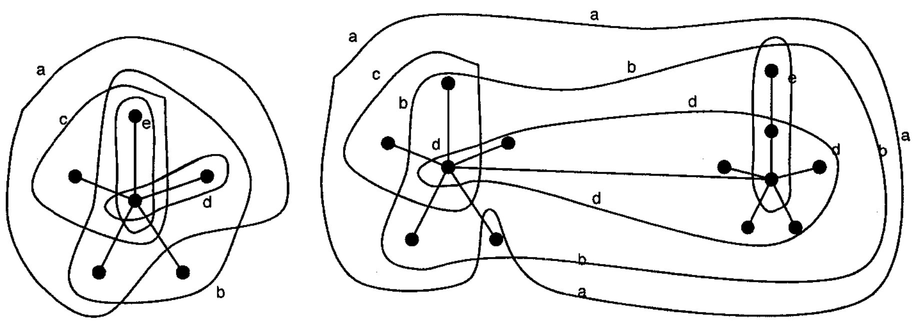


Figure 6.2: An example of the construction to allow subtree c to be internal. The original star representation is shown first, and then the constructed version.

6.4 External Trees in Subtree Overlap Graphs

Lemma 6.2. *There exists a subtree overlap graph with a vertex that can not be external.*

Here we present a vertex that can never be external.

Let G be the graph shown in Figure 6.3.

We know that in a subtree overlap representation of a cycle at least one subtree must contain all others except its neighbors on that cycle (proof of Lemma 5.2). Consider a subtree overlap representation for G, and let s_o be the containing subtree from C_1. Let V_{do} be the vertices on C_1 that are not neighbors of v_o. For every subtree $s_{do} \in S_{do}$, $s_{do} \subset s_o$.

We show that for any choice of v_o the subtree s_c, which corresponds to the vertex c must, for every representation of G, be contained in some other subtree.

We consider three cases:

Neither c nor a neighbor of c is v_o

In this case, whichever vertex is v_o has a corresponding subtree s_o, which must contain s_c. For an illustration that a valid subtree overlap representation can be constructed in this way, see Figure 6.4.

b is v_o

Assume that $v_o = b$ and s_c is not contained in any other subtree. We first show that s_a and s_e, corresponding to vertices a and e, must be disjoint in this representation: s_a cannot contain s_e, as then it would also contain s_c. s_e cannot contain any other subtrees corresponding to vertices on C_1.

There must be some subtree s_z corresponding to a vertex on C_2 that contains all subtrees on C_2 except the subtrees corresponding to the neighbors of v_z. Since $s_a|s_e$, s_z cannot be s_a nor s_e. Thus,

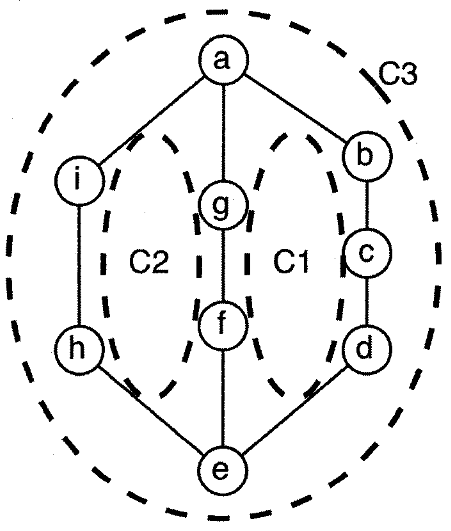


Figure 6.3: A subtree overlap graph. The dotted ovals with labels indicate named cycles, for ease of reference.

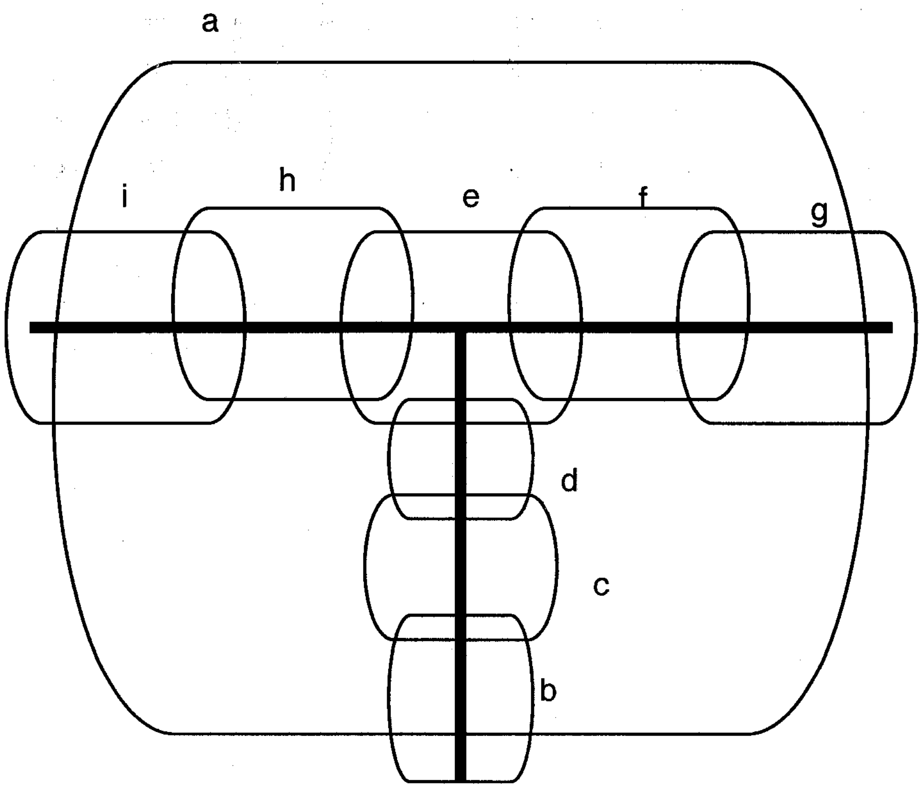


Figure 6.4: A subtree overlap representation of the graph in Figure 6.3. The black line is the underlying tree. The labelled outlined shapes indicate subtrees.

some subtree s_z corresponding to some vertex on C_2 must contain both s_e and s_a. However, by Lemma 5.1, s_z must also contain s_c. s_c would then be contained.

c is v_o

Assume that $v_o = c$, and s_c is not contained in any other subtree. As neither a nor e are neighbors of c, s_a and s_e, corresponding to vertices a and e, must be disjoint in this representation.

There must be some subtree s_z corresponding to a vertex on C_2 that contains all subtrees on C_2 except the subtrees corresponding to the neighbors of v_z. Since $s_a | s_e$, s_z cannot be s_a nor s_e. Thus, some subtree s_z corresponding to some vertex on C_2 must contain both s_e and s_a. However, by Lemma 5.1, s_z must also contain s_c. s_c would then be contained.

The subtree corresponding to vertex c must always be contained in some other subtree in every valid subtree overlap representation of G. $\square$

It is currently unknown if every vertex in a subtree overlap graph can be internal.

Chapter 7

Compositions and Decompositions

Compositions are methods of building up a graph by combining two other graphs in some way. Decompositions are the reverse - breaking a graph down into smaller pieces. Decomposing a graph can allow algorithms to work on smaller parts of it, or may even iteratively break a graph down into small, recognizable parts with particular properties.

We are particularly interested in composition and decomposition because they could contribute to a recognition algorithm for subtree overlap graphs. Knowing which compositions and decompositions preserve what properties allows us to consider them for future use in algorithms acting on subtree overlap graphs.

We consider the join composition, the modular composition, and a single vertex cutset composition because these have special relevance to some known subclasses of subtree overlap graphs. The join decompositon is used in recognition of circle graphs [11]. The modular decomposition is used in recognition of cocomparability graphs [23]. The single vertex cutset is a very simple and restricted case of clique cutset decomposition, known to preserve the property of being chordal [8].

The results in this chapter, as well as the chapter on graph operations, are summarized in Table 7.1.

Table 7.1: A table showing which operations and compositions preserve the property of being a subtree overlap graph when applied on listed classes of graphs. The classes are listed across the top row of the table, the operations and compositions in the first column. References to Theorems or citations proving that a given operation does or does not necessarily preserve the property of being a subtree overlap graph on a particular class are given.

	Chordal	Cocomparability	Circle	Spider	Subtree Overlap
Join Composition	Not necessarily - Thm 7.1	Partially - Thm 7.1	Yes - [11]	Not necessarily - Thm 7.1	Not necessarily - Thm 7.1
Modular Composition	Not necessarily - Thm 7.5	Yes - [23]	Yes - 7.4	Not necessarily - Thm 7.5	Not necessarily - Thm 7.5
Single Vertex Cutset	Yes - Thm 7.2	Partially - Thm 7.2	Yes - Thm 7.2	Yes - Thm 7.2	Not necessarily - Thm 7.3
Vertex Addition	Yes - Thm 8.1	Not necessarily - Thm 8.4	Not necessarily - Thm 8.6	Not necessarily - Thm 8.6	Not necessarily - Thm 8.6
Edge Addition	Yes - Thm 8.2	Not necessarily - Thm 8.3	Not necessarily - Thm 8.5	Not necessarily - Thm 8.5	Not necessarily - Thm 8.5
Edge Removal	Yes - Thm 8.2	Partially - Thm 8.3	Not necessarily - Thm 8.5	Not necessarily - Thm 8.5	Not necessarily - Thm 8.5
Edge Subdivision	Yes - Thm 8.2	Partially - Thm 8.3	Not necessarily - Thm 8.4	Not necessarily - Thm 8.4	Not necessarily - Thm 8.4

7.1 Join Composition

The join decomposition was described by Cunningham and Edmonds [7]. It can be applied on a graph $G = (V, E)$ if G can be partitioned into four parts V_1, V_2, V_3 and V_4 such that there are no edges between V_1 and V_3 or V_4, nor any edges between V_4 and V_2, and every vertex in V_2 is adjacent to every vertex in V_3. While V_1 and V_4 may be empty, $(V_1 \cup V_2)$ and $(V_4 \cup V_3)$ must both have at least two elements.

The decomposition consists of disconnecting V_2 (and therefore V_1) from V_3 (and therefore V_4), and adding a vertex adjacent to exactly all vertices in V_2, and another vertex adjacent to exactly all vertices in V_3. We call these added vertices *placeholder vertices*. An illustration can be found in Figure 7.1.

The join composition is, informally, just the reverse of the decomposition. Two graphs G_1 and G_2 can be joined by identifying a placeholder vertex on each. Let v_{p1} and v_{p2} be the two placeholder vertices. We then remove v_{p1} and v_{p2}, and make every member of the former $N(v_{p1})$ adjacent to every member of the former $N(v_{p2})$.

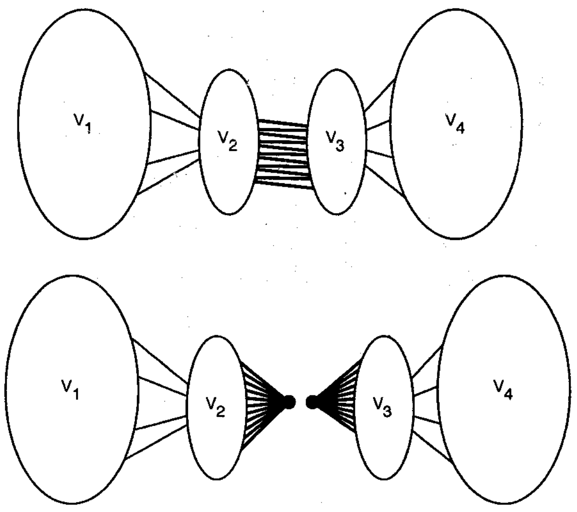


Figure 7.1: An illustration of the join decomposition.

7.1.1 Complete Joining Two Noncomparability Graphs

Let G be the subtree overlap graph in Figure 7.2.A. $G - g$ is non-cocomparability. Any graph for which there exists a subtree overlap representation with no disjoint subtrees is a cocomparability graph. Since $G - g$ is non-cocomparability, any subtree overlap representation must have at least two disjoint subtrees.

Consider what happens if we take two copies of $G - g$ (we will call them G_1 and G_2) and completely join them, producing a graph G'. If G' is a subtree overlap graph, then it has a subtree

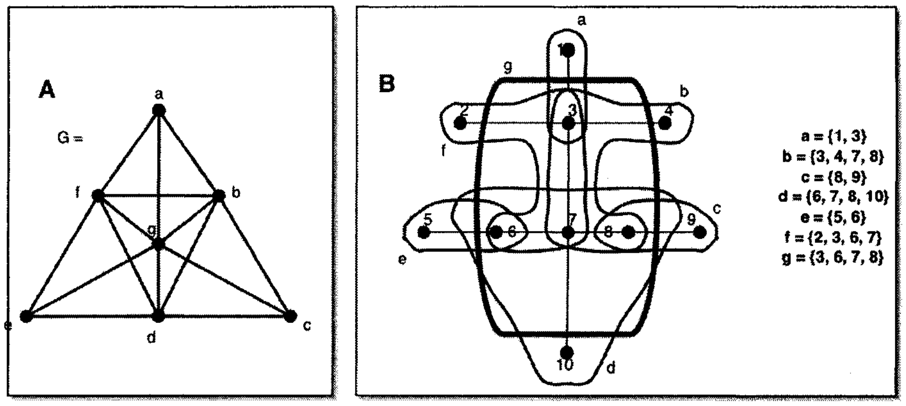


Figure 7.2: A subtree overlap graph in part **A**, and its subtree overlap representation in **B**. The letter labels for the vertices in **A** correspond to subtree labels in **B**. The underlying tree nodes are labeled with numbers, and the nodes in each letter-labeled subtree are listed on the right for clarity.

overlap representation $R' = (S', T')$, where S' is the set of subtrees representing both the vertices from G_1, and the vertices from G_2.

There are two subtrees from G_1, s_{1A} and s_{1B}, that must be disjoint. There are two subtrees from G_2, s_{2A} and s_{2B}, that must be disjoint. As G_1 and G_2 have been completely joined, there must be pairwise intersection between s_{2A} and s_{2B}, and s_{1A} and s_{1B}. That is, there must be some point *b* in the tree that is in both s_{1A} and s_{2A}, as well as a point a that is in the intersection of s_{1A} and s_{2B}, a point *c* that is in the intersection of s_{1B} and s_{2A}, and a point *d* that is in the intersection of s_{1B} and s_{2B}.

So, to summarize, the points *a*, *b*, *c*, and *d* on the tree T are defined as follows:

$$a \in (s_{1A} \cap s_{2B})$$

$$b \in (s_{1A} \cap s_{2A})$$

$$c \in (s_{1B} \cap s_{2A})$$

$$d \in (s_{1B} \cap s_{2B})$$

Recall that none of the four subtrees may be contained in any other, and that s_{1A} and s_{1B} must be disjoint, as must s_{2A} and s_{2B}. This forces *a*, *b*, *c*, and *d* to be distinct. If any two were not, those two would be forcing an intersection or containment between s_{1A} and s_{1B} or s_{2A} and s_{2B}. An illustration can be seen in Figure 7.3.

So, then, there is a cycle in the tree, which is forbidden. The cycle can be found as follows:

Let $p_{ab} \in s_{1A}$ be a path from a to b. Let $p_{bc} \in s_{2A}$ be a path from b to c. Let $p_{cd} \in s_{1B}$ be a path from c to d. Let $p_{da} \in s_{2B}$ be a path from d to a. Let G_c be the union of p_{ab}, p_{bc}, p_{cd}, and p_{da}. G_c must contain a chordless cycle of length greater than or equal to four.

Since the complete join of G_1 and G_2, called G', must contain a cycle in its overlap representation, it is not a subtree overlap graph.

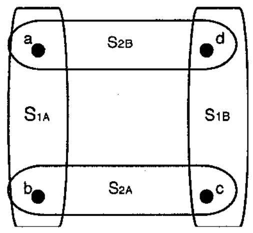


Figure 7.3: A bad cycle that must appear in any subtree overlap representation of G' - two copies of the graph in Figure 7.2 joined by the complete join composition

Since the noncocomparability graph G is a subtree overlap graph, chordal graph, and spider graph (Figure 7.4), we have Theorem 7.1:

Theorem 7.1. *The complete join composition of chordal graphs, spider graphs, or subtree overlap graphs does not necessarily preserve the property of being a subtree overlap graph.*

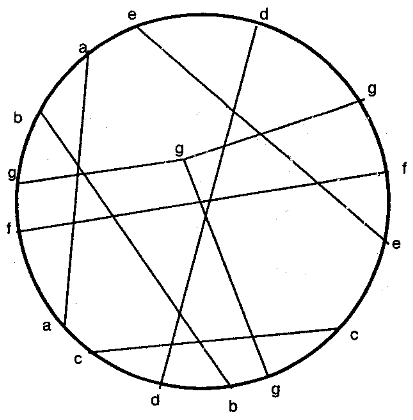


Figure 7.4: A spider representation of the subtree overlap graph in Figure 7.2

7.1.2 Circle Graphs

The join decomposition and composition preserve the property of being a circle graph, and therefore the join composition on circle graphs preserves the property of being a subtree overlap graph [11]. As discussed above, the join decomposition was used by Gabor *et al.* [11] in polynomial time recognition of circle graphs.

7.1.3 Cocomparability Graphs

Lemma 7.1. *The join composition of two cocomparability graphs G_1 and G_2 preserves the property of being a subtree overlap graph if some vertex adjacent to the placeholder vertex in each graph is either a sink or a source in some transitive orientation of the edges of each of $\bar{G}_1$ and $\bar{G}_2$.*

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two cocomparability graphs with no isolated vertices. Let G' be the graph produced by applying the join composition to G_1 and G_2. Let $R_1 = (S_1, T_1)$ and $R_2 = (S_2, T_2)$ be subtree overlap representations for G_1 and G_2, respectively, where T_1 and T_2 are stars, and in each representation some subtree overlapping the subtree of the placeholder vertex is external. We know that there are two such representations from [16]. S_1 and S_2 are sets of subtrees on the stars T_1 and T_2. Let $s_{p1} \in S_1$ and $s_{p2} \in S_2$ be the subtrees corresponding to the placeholder vertices. Let $S_{h1} \subset S_1$ and $S_{h2} \subset S_2$ be the sets of subtrees adjacent to the placeholder vertices, and the ones that will therefore be completely joined in the composition.

We present a method of transforming R_1 and R_2 into a representation for G'. Let n_c be the central node of T_1.

Observation 7.1. *There is no $s_b \in (S_1 - S_{h1})$ such that for $s_i, s_j \in S_{h1}$ $s_i \subset s_b \subset s_j$.*

The justification for this is as follows by contradiction: s_b does not overlap s_{p1}, but $s_i, s_j \between s_{p1}$. Therefore, s_b can neither contain, nor be contained in s_{p1}, implying that $s_b \between s_{p1}$ - a contradiction. □

First, we remove the placeholder subtrees from R_1 and R_2

Add to n_c a leaf p that is in only the subtrees in S_{h1}. We know this cannot disrupt overlapping because no subtree in S_{h1} can be contained in any subtree not in S_{h1}, from Observation 7.1 and the fact that we have transformed R_1 such that some subtree in S_{h1} is external.

We know that there are no three subtrees s_a, s_b, and s_c in S_1 (or, symmetrically in S_2) such that s_a and s_b are in S_{h1}, s_c is not in S_{h1}, and $s_a \subset s_c \subset s_b$. We can see this by contradiction - if this containment situation were to exist, then s_c would overlap s_{p1}, a contradiction to s_c not being in S_{h1}.

We transform R_2 in the same way, adding a leaf q.

We then identify p and q.

In this final representation, R_3, made by transforming and joining R_1 and R_2, all subtrees in S_{h1} overlap all subtrees in S_{h2}, the placeholder subtrees have been removed, and no other overlapping has been disrupted. We therefore have a representation for G', proving that G' is a subtree overlap graph. □

If Conjecture 6.1 is true, then this proof can be modified to show that the join composition on two cocomparability graphs will always preserve the property of being a subtree overlap graph.

7.2 Single Vertex Cutset Composition

The single vertex cutset composition is a restricted version of the clique cutset composition. The composition consists of joining two graphs by identifying two vertices - one from each graph. The decomposition can be applied when there is a single vertex cutset.

We consider it here because of its simplicity, and because the clique cutset decomposition can be applied to chordal graphs. The treatment of the single vertex cutset composition here might be a first step towards an eventual understanding of the clique cutset composition on the subclasses of subtree overlap graphs.

7.2.1 Single Vertex Composition and External Subtrees

Theorem 7.2. *Let G_1 and G_2 be subtree overlap graphs. If at least one of G_1 or G_2 is a chordal graph or spider graph, or at least one of G_1 or G_2 is a cocomparability graph G_c, and the vertex to be identified in that graph is a source or sink in some transitive orientation of the edges of $\bar{G}_c$, then the single vertex composition on G_1 and G_2 will preserve the property of being a subtree overlap graph.*

Proof:

Let v_a and v_b be the vertices from G_1 and G_2 to be identified in the composition, producing G'.

If s_a is external in some representation $R_1 = (T_1, S_1)$ of G_1, then we can create a subtree overlap representation $R' = (T', S')$ for G'. Let $R_2 = (T_2, S_2)$ be a subtree overlap representation of G_2. Let $p \in s_a$ be a node on T_1. Let $q \in s_b$ be a node on T_2. To produce R', we add an edge between p and q, add all of T_1 to any subtree in S_2 that contains s_b, and union the two subtrees s_a and s_b into one, which we shall call s_{ab}. We now have a representation in which overlapping within S_1 has not changed, overlapping within S_2 has not changed, every member of S_2 either contains or is disjoint from every member of S_1, and s_{ab} overlaps all subtrees that were previously overlapped by either s_a or s_b. We therefore have a representation of G'.

Since we know that any vertex in a chordal or spider graph can be external, and the ones that are sources or sinks in some orientation of the edges of the complement can be external in a cocomparability graph, then we know that if G_1 is any of these, then s_a can be external in some representation, and therefore R' can be created. □.

If Conjecture 6.1 is true, then this proof will also imply that the single vertex composition on two subtree overlap graphs, one of which is a cocomparability graph, will always preserve the property of being a subtree overlap graph.

Theorem 7.3. *The single vertex composition on subtree overlap graphs does not necessarily preserve the property of being a subtree overlap graph.*

Proof:

Let v_a and v_b be the vertices from G_1 and G_2 that we will identify in the composition. Let G' be the graph resulting from the composition.

If neither v_a nor v_b can ever be external in a subtree overlap representation, then identifying v_a and v_b will cause a representational impossibility: there will be some subtree corresponding to a vertex from G_1 that must contain all subtrees corresponding to vertices from G_2, and *vice versa* - a contradiction. There would therefore be no valid subtree overlap representation of G'.

We know that there is a subtree overlap graph with a vertex with a corresponding subtree that can never be external in any representation, from Lemma 6.2. If G_1 and G_2 are both the graph shown in Figure 6.3, and v_a and v_b are the vertices that can never correspond to external subtrees, then we have the above situation. There is no valid subtree overlap representation for two copies of the graph in Figure 6.3 with the c vertices from the two copies identified. □

7.3 Modular Composition

Modular decomposition is used in the recognition of comparability graphs, and can therefore be used in the recognition of cocomparability graphs. We know that this composition preserves the property of being a cocomparability graph [23], and thus we need only consider spider graphs, chordal graphs, and subtree overlap graphs.

A *module* in a graph is a set of vertices with identical neighborhoods - every vertex not in the module is either adjacent to all vertices in the module, or no vertices in the module. In modular decomposition, a module is replaced with a single vertex. In modular composition, a single vertex is replaced with a module.

Theorem 7.4. *The modular composition on circle graphs preserves the property of being a subtree overlap graph.*

Proof:

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two circle graphs. Let $v_m \in V_1$ be a vertex in G_1 that we will replace with G_2.

G_1 and G_2 have representations as subtrees on a path. Let $R_2 = (S_2, T_2)$ be such a representation of G_2. Let $R_1 = (S_1, T_1)$ be such a representation of G_1, with s_m at one extreme of the path T_1. Such a path exists by Lemma 6.1

Let p be the endpoint node of s_m that is not a leaf of T_2. Let q be the neighboring node of p that is not in s_m.

Note that subtree s_i overlaps s_m if and only if it contains both p and q.

We now remove s_m from S_1. We place, on the edge between p and q, all of T_2. That is, p and q are no longer adjacent, p is adjacent to one former leaf of T_2, and q is adjacent to the other former leaf of T_2. All nodes of the former T_2 are added to all subtrees from S_1 containing both p and q.

This has not changed the relationships between subtrees in S_1, except for the removal of s_m. This has not changed the relationships between the subtrees in S_2, as the entire tree on which they exist has been added, intact, to T_1. Currently, no subtree in S_2 overlaps any subtree in S_1. We want every subtree in S_2 to overlap all subtrees from S_1 that currently contain the subtrees from S_1.

We therefore add to each node r on the former T_2 a leaf node w that is in only subtrees from S_2 that contain r, and no subtrees from S_1. Every subtree in S_2 overlaps exactly the subtrees in S_1 that contain both p and q, and therefore exactly the ones that previously overlapped s_m. □

Theorem 7.5. *The modular composition on spider graphs, chordal graphs, and therefore subtree overlap graphs, does not necessarily preserve the property of being a subtree overlap graph.*

Proof:

The graph G as shown in Figure 7.2 is both a spider (Figure 7.1.1) and a chordal graph. Replacing g with a copy of $G - g$ results in a non-subtree overlap graph, as it is equivalent to completely joining two copies of G, which we know produces a non-subtree overlap graph from the proof of Theorem 7.1. By extension, the modular composition on subtree overlap graphs does not necessarily preserve the property of being a subtree overlap graph, and the modular decomposition on subtree overlap graphs, spider graphs, and chordal graphs, does not necessarily preserve the property of not being a subtree overlap graph. □

Chapter 8

Graph Operations

This chapter explores some basic tree and graph operations with regards to the preservation of being a subtree overlap graph. Removing and adding vertices and edges are the most basic possible atomic actions on a graph. Understanding the impact of these basic operations on the subtree overlap graphs with respect to which preserve inclusion in the class could contribute to a recognition algorithm for subtree overlap graphs.

8.1 Adding and Removing Edges and Vertices

Vertices and edges are the stuff that graphs are made of. The consequences of adding or removing edges or vertices from the known subclasses of subtree overlap graphs could be critical to understanding the construction of subtree overlap graphs, and from there perhaps their recognition.

In this section, we will examine whether the addition and removal of vertices and edges from graphs of various subclasses preserves the property of being a subtree overlap graph.

8.1.1 Interactions between edge and vertex removal and addition

Before we start on the particular subclasses, we can make an observation on the relationship between edge operations and vertex operations in general.

Lemma 8.1. *Let **F** be some hereditary class of graphs. Let **P** be some property that all members of **F** have. Since **F** is hereditary, removing a vertex from any member of **F** preserves **P**. If adding a vertex adjacent to arbitrary vertices in any member of **F** preserves **P**, then adding or removing any edge to or from any member of **F** also preserves **P**. Conversely, if adding or removing an arbitrary edge from a member of **F** does not necessarily preserve **P**, then adding or removing an arbitrary vertex to or from a member of **F** does not necessarily preserve **P**.*

Proof:

Let $G = (V, E)$ be some member of **F**. Let v_a be some vertex in V. Let N be the original neighborhood of v_a. We can consider adding or removing an edge as an operation composed of removing a vertex, and then re-adding it with a different neighborhood.

Let H be the graph induced on G by the vertex set $V - v_a$. We know that H is a member of **F**.

If we can add a new vertex to any neighborhood of any member of **F** and preserve **F**, then we can re-add v_a to the graph with either one extra neighbor, or one neighbor fewer, depending on whether we want to add or remove an edge, and preserve **P**.□

8.1.2 Chordal Graphs

Here we will start with a general theorem, and then restate it in the more restricted single-vertex case.

Let $G_c = (V_c, E_c)$ be a chordal graph, and $G = (V, E)$ a subtree overlap graph. Let $V_z \subset V$ be a set of vertices such that for some subtree overlap representation of G, $R = (S, T)$, there is a node $q \in T$ that is in every subtree $s_z \in S_z$, and there are no subtrees $s_a, s_b \in S_z$, $s_c \notin S_z, s_c \in S$ such that $s_a \subset s_c \subset s_b$. Let $V_y \subset V_c$ be an arbitrary subset of the vertices in V_c. Let $E_B = \{(v_i, v_j)|v_i \in V_z, v_j \in V_y$. For sets and graphs so defined, we have the following:

Theorem 8.1. *The graph $G_2 = (V \cup V_c, E \cup E_c \cup E_B)$ is a subtree overlap graph.*

Proof:

Firstly, note that we have completely joined the vertices in V_z and V_y, thus attaching a chordal graph, and a subtree overlap graph.

Let $R_c = (S_c, T_c)$ be a subtree overlap representation of G_c with no containment. Let $S_x \subset S$ be the set of subtrees that contain some subtree in S_z, but are not themselves in S_z. Let p be a node of T shared between all subtrees in S_z. We can perform the following transformation on R and R_c to produce R_2, a subtree overlap representation of G_2.

Transformation

First, we add an edge between p and an arbitrary node of T_c. Expand every subtree in $(S_z \cup S_x)$ to contain all nodes in T_c.

To each subtree $s_i \in S_y$, add a new leaf node q_i adjacent to an arbitrary node in s_i such that q_i is contained in exactly s_i and every member of S_x.

Now we must prove that this transformation has produced exactly the required overlapping. To do this, we will consider pairs of subtrees.

Let $s_i, s_j \in S_z$. The nodes added to s_i are exactly those added to s_j, therefore since s_i and s_j were not previously disjoint, their relationship is preserved. An identical argument can be used for $s_i, s_j \in S_x$.

Let $s_i, s_j \in S_c$. Since neither subtree is contained in the other in R_c (that is, before the transformation), adding leaves to the tree cannot have disrupted their relationship (Lemma 3.1).

Let $s_i \in S_z$, $s_j \in S_x$. Before the transformation, either $s_i \between s_j$ or $s_i \subset s_j$, by the definitions of S_z and S_x. Since every node added to s_i was also added to s_j, their relationship is preserved.

Let $s_i \in S_z$, $s_j \in S_y$. s_i contains some node not in s_j. The intersection of s_i and s_j is non-empty, and s_j contains the leaf q_j while s_i does not. Therefore $s_i \between s_j$.

Let $s_i \in S_z$, $s_j \in S_c$, $s_j \notin S_y$. In this case, no new leaf was added to s_j, and therefore $s_i \supset s_j$.

Let $s_i \in S_x$ and $s_j \in S_c$. All nodes in T_c, as well as the new leaves, are in s_i, therefore $s_i \supset s_j$

Let $s_i \in S$ but not in S_x or S_z, and $s_j \in S_c$. As no nodes have been added to s_j, $s_j | s_i$.

Let $s_i \in (S_x \cup S_z)$, $s_j \in S$ but not in $(S_x \cup S_z)$. Since prior to the transformation s_j did not contain s_i, s_i has been expanded, and s_j has not been changed, their relationship has not been disrupted.

We have that all relationships within S and S_c are maintained, all subtrees in S_x contain all subtrees in S_c, all subtrees in S but not in S_z or S_x are disjoint from all subtrees in S_c, and all subtrees in S_z overlap all subtrees in S_y. We therefore have a subtree overlap representation for G_2. □

This may seem at first to be of no use, since we cannot identify if an arbitrary set of vertices in a subtree overlap graph can have subtrees with some point in common in a subtree overlap representation, as well as obey the containment rule, and therefore serve as a V_z in the transformation.

However, there are some particular cases in which we know a set of vertices can have subtrees that all have some point in common. If $G = (V, E)$ is a cocomparability graph and $V_z = V$, then we can perform this operation, since there is a representation in which all subtrees share an point in common, and there are no subtrees outside S_z to break containment rules.

In addition, if V_z is a clique in G, then we can apply the operation. We know that, due to the Helly property, all subtrees in S_z must share some node in common, and since there are no two nodes $s_i, s_j \in S_z$ such that $s_i \subset s_j$, and so there cannot be any third vertex s_k such that $s_i \subset s_k \subset s_j$.

A single vertex is a subcase of both the cocomparability and clique cases, and so we can always add a single vertex to an arbitrary neighborhood of a chordal graph, producing a subtree overlap graph.

Theorem 8.2. *Adding or removing edges that are all incident at a single vertex to or from a chordal graph results in a subtree overlap graph.*

Proof:

Given that chordal is a hereditary class, and arbitrary vertex addition preserves the property of being a subtree overlap graph, we have from Lemma 8.1 that adding or removing arbitrary edges to or from a chordal graph preserves the property of being a subtree overlap graph. □

8.1.3 Cocomparability Graphs

Theorem 8.3. *Adding an edge to a permutation (and therefore cocomparability, circle, or spider) graph may not preserve the property of being a subtree overlap graph, but removing an edge from a cocomparability graph does, if one of the vertices at which the edge is incident is a source or a sink in some transitive orientation of the edges of the complement.*

Proof:

Adding an Edge

Let G be the graph shown in Figure 8.1. G is a permutation (and therefore cocomparability, circle, and spider) graph, as shown by the interval containment representation in Figure 8.2. From Novillo [24] (Theorem 2.6), we know that G with an edge added between j and f is not a subtree overlap graph.

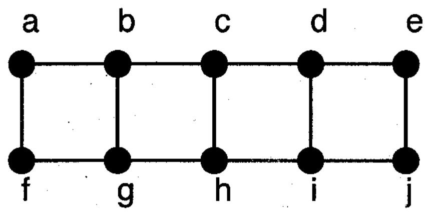


Figure 8.1: A simple subtree overlap graph G.

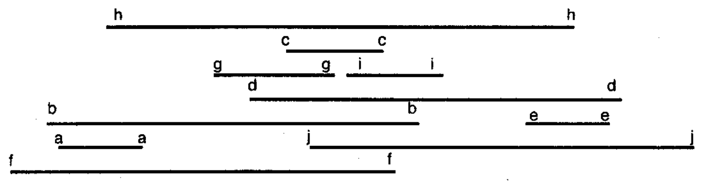


Figure 8.2: The graph in Figure 8.1, is the containment graph of the intervals shown here, proving that it is a permutation graph. The intervals are labeled with the vertices to which they correspond in the graph.

Removing an Edge

Let $G = (V, E)$ be a cocomparability graph.

Let (v_a, v_b) be an arbitrary edge in E, such that v_a is a source or a sink in some transitive orientation of the edges of $\bar{G}$. From [16], as discussed in Chapter 2, we know that there is a subtree overlap representation $R = (S, T)$ of G in which T is a star, and s_a is external. As we have done throughout this thesis, we assume that there are no isolated vertices in G.

We show that we can build a representation $R' = (S', T')$ of the graph $G' = (V, (E - \{(v_a, v_b)\}))$, and therefore G' is a subtree overlap graph and removal of such an edge in a cocomparability graph always results in a subtree overlap graph.

We define some relevant sets, describe the building of the representation, and then prove its correctness.

Let S_I be the subtrees in S contained in s_b, and S_o be the subtrees other than s_a and s_b that overlap s_a or s_b but are not in S_I. Recall that there are no subtrees containing s_a. For any subtree $s_j \in S$, where s_j is not s_a or s_b, s_j is in exactly one of S_o, or S_c, since they are mutually exclusive classes, and there is no disjointness in R. These classes are defined with respect to the original representation of G.

First, we perform the construction, as described in the proof of Theorem 6.1, to produce a representation in which s_b internal. We know, from the proof of Theorem 6.1, that overlapping has been preserved up to this point. Recall that after this construction we have the stars T_1 and T_2 connected by an edge between their central nodes. We call the construction at this point Stage 2, and the tree made of the conjoined T_1 and T_2 we call T'. We will refer to the subtrees and sets of subtrees on the new representation from this point on with a prime superscript. Thus, s'_i is a subtree on the new representation, s_i is its corresponding subtree on the original representation.

Next, for each subtree s'_k such that s_k overlaps both s_a and s_b but $s_k \notin S_I$, we add a leaf p'_k adjacent to the central node of T_1 that is contained in exactly s'_k and any subtree containing s'_k previous to the addition of the leaf. We add this leaf also to T'. As shown in the proof of Lemma 3.2, adding such leaves cannot disrupt overlapping. Finally, we expand s'_b to contain all nodes in s'_a. We call this final representation Stage 3.

We must show that for every subtree $s'_i \in S' - \{s'_a, s'_b\}$, $s'_i \between s'_a$ if and only if $s_a \between s_i$ - we consider this in several cases.

Let $s_i \in S_I$. In this case, no nodes in s'_b were also in s'_i at Stage 2, and the addition of those nodes to s'_a cannot have disrupted the relationship between s'_a and s'_i between Stages 2 and 3.

Let $s_i \in S_o$. In this case, if s'_i overlapped s'_b as well at Stage 2, then a leaf p'_i was added that is in only s'_i and its containers - thus $s'_i \between s'_a$ at Stage 3. If $s'_i \subset s'_a$ at Stage 2, then the growth of s'_a cannot have affected their relationship, and $s'_i \subset s'_a$ at Stage 3.

All overlapping except that between s_a and s_b has been preserved, and thus we have a representation $R' = (S', T')$ for G' at Stage 3. □

Theorem 8.4. *Adding a vertex adjacent to an arbitrary set of vertices in a cocomparablity graph does not necessarily preserve the property of being a subtree overlap graph.*

Proof:

Cocomparability is hereditary, and adding an edge does not necessarily preserve the property of being a subtree overlap graph, therefore from Lemma 8.1 adding a vertex with arbitrary neighborhood does not necessarily preserve the property of being subtree overlap graph. □

8.1.4 Circle and Spider Graphs

Theorem 8.5. *Adding or removing an arbitrary edge from a circle (and therefore spider) graph does not necessarily preserve the property of being a subtree overlap graph.*

Proof:

We know from Theorem 8.3 that removing an edge may not preserve the property of being a subtree overlap graph. Find in Figure 8.3 the graphs G_1, and G'. G_1 is a circle graph - its circle representation is also shown in Figure 8.3. However, from Novillo [24], G' is not a subtree overlap graph. G' can be produced by adding edge (j, e) to G_1. □

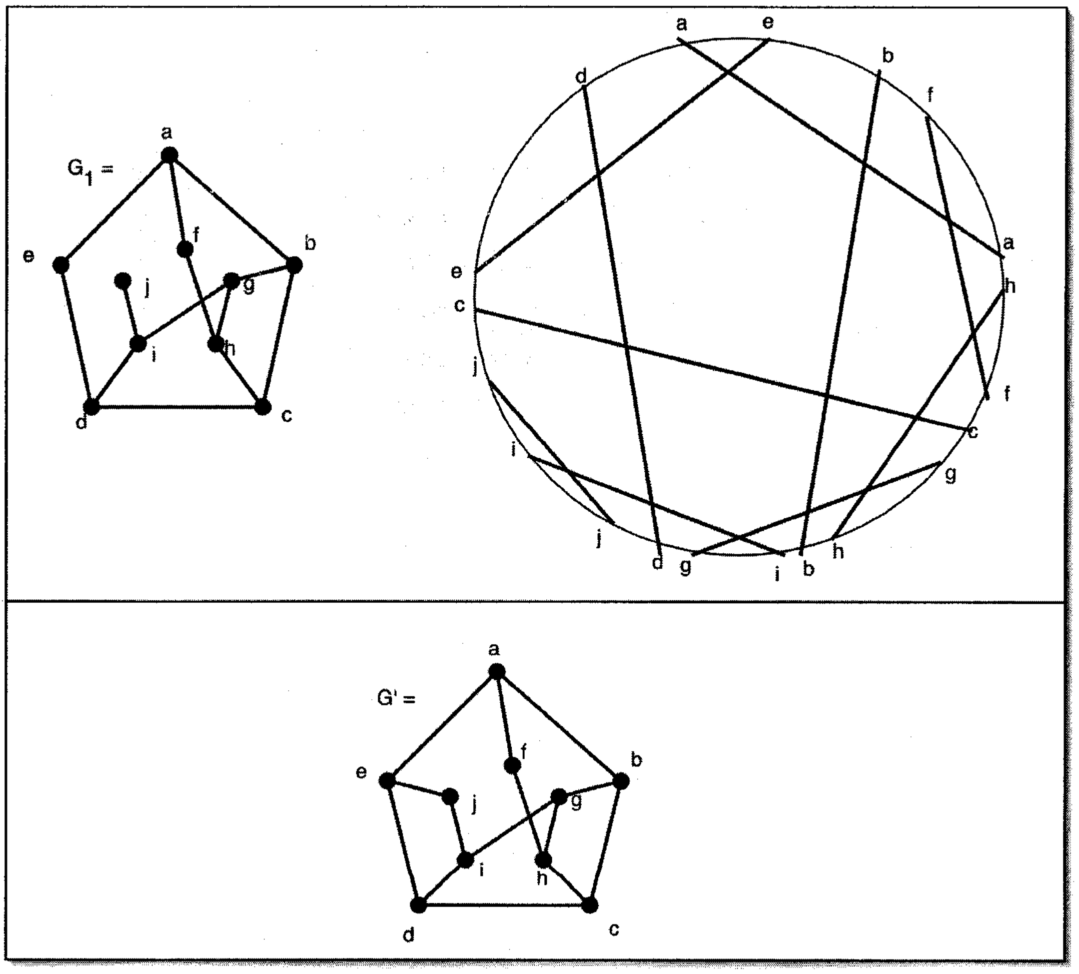


Figure 8.3: Two graphs, one of which (G_1) is a circle graph - the circle representation is also shown. The second graph, G' is not a subtree overlap graph.

Theorem 8.6. *Adding a vertex adjacent to an arbitrary neighborhood to a circle (and therefore spider) graph does not necessarily preserve the property of being a subtree overlap graph.*

Proof:

The class of circle graphs is hereditary, adding or removing edges from a circle graph does not necessarily preserve the property of being subtree overlap graph, and so from Lemma 8.1, adding a vertex with arbitrary neighborhood to a circle graph does not necessarily preserve the property of being a subtree overlap graph. □.

8.2 Subdividing Edges

Here we consider whether subdividing edges in each of the subclasses preserves the property of being a subtree overlap graph. Subdivision of an edge can be defined as follows: let $G = (V, E)$ be a graph, with $v_a, v_b \in V$ and $(v_a, v_b) \in E$. To subdivide (v_a, v_b), we add a new vertex v_c to V, remove the edge (v_a, v_b) from E, and add to E the edges (v_a, v_c) and (v_b, v_c). An illustration of this can be found in Figure 8.2.

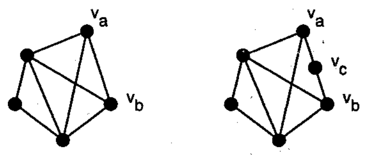


Figure 8.4: An illustration of the subdivision of the edge (v_a, v_b) in a graph with the new vertex v_c. The original graph is on the left, the subdivided version is on the right.

8.2.1 Subdividing Edges in a Chordal Graph

Lemma 8.2. *Subdividing an edge in a chordal graph always produces a subtree overlap graph.*

Proof:

Let $G = (V, E)$ be a chordal graph. Let $e \in E$ be the edge we wish to subdivide. Let $v_a, v_b \in V$ be the vertices at which e is incident. Let $S_N = N(v_b) - \{v_a\}$. Observe that subdividing e with v_c is equivalent to removing v_b, adding v_c as a leaf to v_a, and then adding v_b adjacent to $S_N \cup \{v_c\}$.

$H = G - \{v_b\}$ is a chordal graph. We can add a single vertex, v_c adjacent to only v_a to produce H', still a chordal graph. Then, we can add v_b to H', adjacent to the vertices in $S_N \cup \{v_c\}$. Since H' is chordal, by Theorem 8.1, the result is a subtree overlap graph. □

8.2.2 Subdividing Edges in a Cocomparability Graph

Lemma 8.3. *Let $G = (V, E)$ be a cocomparability graph. Subdividing an edge $e = (v_a, v_b) \in E$, such that v_a is a source or sink in some transitive orientation of the edges of $\bar{G}$, results in a subtree overlap graph.*

Proof:

Here we draw on the proof of Theorem 8.3. Let $G = (V, E)$ be a cocomparability graph, with a representation on a star $R = (S, T)$. Let v_a and v_b be two adjacent vertices in V, and s_a and s_b their corresponding subtrees in S. If the transformation to remove the overlap between s_a and s_b is applied, we have a valid representation of G in which $s_a \supset s_b$.

We know that there is no container of s_b that is contained in s_a, since there could not have been in the the original representation without the edge removed and no subtree would have been moved into that position during the transformation. There is therefore some node q in s_a and s_b in the final representation that is in only s_a, s_b, mutual containers, mutual overlappers, and subtrees that contain s_b and overlap s_a.

We can therefore add to q a leaf that will be contained in all containers and overlappers of s_a or s_b that contain q. The new subtree will be q and the new leaf - overlapping only s_a and s_b. We have a representation of G with the edge between v_a and v_b subdivided. □

8.2.3 Subdividing Edges in a Spider Graph

Lemma 8.4. *Subdividing an edge in a circle (and therefore spider) graph does not necessarily preserve the property of being a subtree overlap graph.*

Proof:

We know that removing an edge in a circle graph does not necessarily preserve the property of being a subtree overlap graph by Theorem 8.5. Let $G = (V, E)$ be some circle graph. Let J be a non-subtree overlap graph produced by removing some edge $(v_a, v_b) \in E$ from G. Let J' be the graph produced from G be subdividing the edge $(v_a, v_b) \in E$. J' has J as an induced subgraph, and is therefore not a subtree overlap graph. □.

8.3 Complementing

Here we show that taking the complement of a graph from the well-known subclasses of subtree overlap graphs does not necessarily preserve the property of being a subtree overlap graph.

Let G be the graph shown in Figure 8.5. As the complement of G is two completely joined non-cocomparability graphs, it is not a subtree overlap graph, as discussed in Section 7.1.1. As G is a chordal, spider, and circle graph (Figure 8.6), we know that taking the complement of a graph from any of these classes does not necessarily produce a subtree overlap graph. The comparability graphs are not a subclass of subtree overlap graphs [5], and therefore the complement of a cocomparability graph is not always a subtree overlap graph.

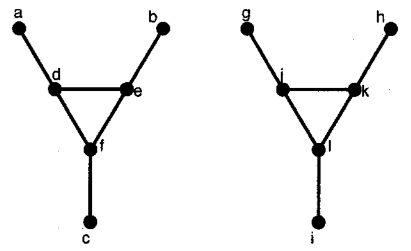


Figure 8.5: A subtree overlap graph G

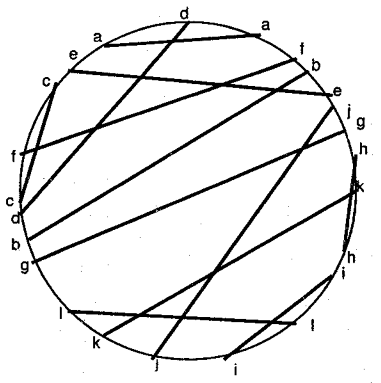


Figure 8.6: A representation for the graph in Figure 8.5, showing that it is a circle, and therefore spider, graph.

Chapter 9

Conclusion

Here we will summarize the work presented in this thesis, discuss some directions for future work, and then make some concluding remarks.

9.1 Summary and Future Work

Through the work done here, we have discovered some properties of the class of subtree overlap graphs and its subclasses.

Subtree overlap graphs are equivalent to the subtree filament graphs, and therefore the complements of cochordal-mixed graphs. This ties in to previously done work, and allows for possible future work. Given that the subtree overlap graphs are the complements of cochordal-mixed graphs, and a constructive proof of this provided, we could produce a subtree overlap representation given some pieces of information on edge partitioning of the cochordal-mixed graph. This has promise for resolving recognition of subtree overlap graphs.

The overlap graphs of paths in trees with no non-leaf boundary nodes are equivalent to the overlap graphs of intervals on a line and therefore the circle graphs. This emphasizes the importance of the restriction of the nature of overlapping allowed, in addition to restriction of the type of overlapping sets themselves in defining an overlap graph class.

Some very simple graphs - the cycles - have few non-isomorphic subtree overlap representations, whereas other very simple graphs - such as the paths - have many non-isomorphic subtree overlap representations. In a scheme to produce a representation, we might pay special attention to the cycles in a graph.

Various compositions and operations can be applied to graphs in the subclasses of subtree overlap graphs, and produce a graph that is still a subtree overlap graph. With these compositions and operations in hand, we might be able to contribute to a method of recognizing subtree overlap graphs by building them up from smaller parts, or breaking them down into smaller graphs for faster running of a recognition algorithm. If Conjecture 6.1 is true, we will have further results on these operations and decompositions as applied to cocomparability graphs.

9.2 Concluding Remarks

The work presented here is perhaps best viewed as the development of a toolbox that could contribute to recognition, or other, algorithms for subtree overlap graphs. However, some results presented are interesting in their own right. The equivalence of subtree overlap and subtree filament graphs contributes to our understanding of the graph hierarchy. The work on relationship matrices as a method of specifying parts of a representation allows us to check the consistency of a set of set relationships, and in combination with the result that subtree overlap graphs are cochordal mixed presents possibilities for future work. The work presented here contributes to the understanding of subtree overlap graphs, and their relationship with graphs in subclasses of subtree overlap graphs.

Bibliography

[1] Jorge L. Ramirex Alfonsin and Bruce A. Reed. *Perfect Graphs*. John Wiley and Sons, Ltd., 2001.

[2] Alberto Apostolico, Mikhail J. Atallah, and Susanne E. Hambrusch. New clique and independent set algorithms for circle graphs. *Discrete Appl. Math.*, 36(1):1–24, 1992.

[3] S. Benzer. On the topology of the genetic fine structure. In *Proceedings of the National Academy of Science*, volume 45, pages 1607–1620, 1959.

[4] Kellogg S. Booth and George S. Lueker. Linear algorithms to recognize interval graphs and test for the consecutive ones property. In *STOC '75: Proceedings of seventh annual ACM symposium on Theory of computing*, pages 255–265, New York, NY, USA, 1975. ACM Press.

[5] Eowyn Čenek. Subtree overlap graphs and the maximum independent set problem. Master's thesis, University of Alberta, Department of Computing Science, 1998.

[6] Eowyn Čenek and Lorna Stewart. Maximum independent set and maximum clique algorithms for overlap graphs. *Discrete Appl. Math.*, 131(1):77–91, 2003.

[7] W. H. Cunningham and J. Edmonds. A combinatorial decomposition theory. *Canadian Journal of Mathematics*, 32:734–765, 1980.

[8] G. A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg*, 25:71–76, 1961.

[9] B. Dushnik and E.W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63:600–610, 1941.

[10] S. Even and A. Itai. Queues, stacks, and graphs. *Proc. International Symp. on Theory of Machines and Computations*, pages 71–86, 1971.

[11] Csaba P. Gabor, Kenneth J. Supowit, and Wen-Lian Hsu. Recognizing circle graphs in polynomial time. *J. ACM*, 36(3):435–473, 1989.

[12] Fanica Gavril. Algorithms for minimum colouring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal of Computing*, 1(2):180–187, 1972.

[13] Fanica Gavril. Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks*, 3:261–273, 1973.

[14] Fanica Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory (B)*, 16:47–56, 1974.

[15] Fanica Gavril. Maximum weight independent sets and cliques in intersection graphs of filaments. *Inf. Process. Lett.*, 73(5-6):181–188, 2000.

[16] M. C. Golumbic and E. R. Scheinerman. Containment graphs, posets and related classes of graphs. *Annual New York Academy of Science*, 55:192–204, 1989.

[17] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland, 2004.

[18] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. Polynomial algorithms for perfect graphs. In Claude Berge and Vašek Chvátal, editors, *Topics on perfect graphs*, volume 21 of *Annals of Discrete Mathematics*, pages 325–356. North-Holland, 1984.

[19] Mark Keil and Lorna Stewart. Approximating the minimum clique cover and other hard problems in subtree filament graphs. *Discrete Applied Mathematics*, 145(14):1983–1995, 2006.

[20] Manfred Koebe. Colouring of spider graphs. In *Topics in Combinatorics and Graph Theory*, pages 435–442. Physics-Verlag Heidelberg, 1990.

[21] Manfred Koebe. Spider graphs - a new class of intersection graphs. Master's thesis, Ernst-Moritz-Arndt-Universitaet, Sektion Mathematik, 1990.

[22] Alexandr Kostochka and Jan Kratochvil. Covering and coloring polygon-circle graphs. *Discrete Math.*, 163(1-3):299–305, 1997.

[23] Ross M. McConnell and Jeremy P. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 536–545, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[24] Diego Novillo. Overlap graphs of subtrees in a tree, 1994.

[25] S. Olariu. On sources in comparability graphs, with applications. *Discrete Math.*, 110(1-3):289–292, 1992.

[26] A. Pneuli, S. Even, and A. Lempel. Transitive orientation of graphs and identification of permutation graphs. *Canadian Journal of Mathematics*, 23:160–175, 1971.

[27] Donald J. Rose, R. Endre Tarjan, and George S. Leuker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal of Computing*, 5(2):266–283, 1976.

[28] William Rosgen. Set representations of graphs. Master's thesis, University of Alberta, Department of Computing Science, 2005.

[29] Jeremy Spinrad. On comparability and permutation graphs. *SIAM Journal of Computing*, 14:658–670, 1985.

[30] Jeremy Spinrad. Recognition of circle graphs. *Journal of Algorithms*, 16:264–282, 1994.

[31] Jeremy P. Spinrad. *Efficient Graph Representations*. AMS, Providence, the fields institute monographs 19th edition, 2003.

[32] Edward Szpilrajn-Marczewski. Sur deux proprietes des classes densembles. *Fundamenta Mathematicae*, 33:303–307, 1945.