DISSERTATION

MODULAR DECOMPOSITION OF k-HYPERGRAPHS

Submitted by

Sripriya Venkataraman

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2006

COLORADO STATE UNIVERSITY

May 9, 2006

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY SRIPRIYA VENKATARAMAN ENTITLED MODULAR DECOMPOSITION OF K-HYPERGRAPHS BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Adviser

Department Head

ABSTRACT OF DISSERTATION

MODULAR DECOMPOSITION OF k-HYPERGRAPHS

A *module* is a subset of vertices in a graph wherein each vertex outside the module is either adjacent to all or none of the vertices inside it. The *modular decomposition tree* of a graph gives an $O(n)$-space representation of the modules in a graph. The notion of modules in graphs has been extended to hypergraphs by Bonizzoni and Della Vedova [6]. Bonizzoni and Della Vedova also provide an algorithm to compute the decomposition tree with a time-bound of $O(n^{3k-5})$, where k is the maximum size of hyperedges in the hypergraph. In this thesis, I provide a new time-bound to compute the decomposition tree in hypergraphs of $O(k!n^{k-2}m \log n)$, where m is the number of edges. The new algorithm takes advantage of sparseness in the hypergraph. Even in dense hypergraph, where there are $O(n^k)$ edges, the new time-bound is $O(k!n^{2k-2} \log n)$ which is still better than $O(n^{3k-5})$ for k greater than or equal to four.

Sripriya Venkataraman
Department of Mathematics
Colorado State University
Fort Collins, Colorado 80523
Summer 2006

ACKNOWLEDGMENTS

I would like to thank Dr. Ross Mc Connell who was more than a teacher, guide and source of encouragement. He was a great listener and truly a friend.

I would like to thank my committee members Dr. Robert Liebler, Dr. Alexander Hulpke and Dr. V. Chandrashekar for their cooperation, encouragement and understanding. I would also like to thank Bryan Elder and Irma Woollen for helping me with graduate school formalities. Thanks also to Sylvia, Stefan and Abdullah.

My deepest thanks to my grandfather Prof S. Lakshminarayanan, who instilled in me the love of mathematics and the concept of eternal learning. At the age of 90, he displayed his passion for learning by sitting down with me and solving my College Physics exercises.

My parents motivated me and completely supported me through my several years in college. My in-laws have always believed in me and wanted me to succeed. My relatives were very understanding of my time at work and helped me justify this time away from my family.

My close friends Subbu and Ramya spent innumerable weekends baby sitting Ananya (for free) so that I could work. A million thanks to Sriram and Nandu for listening to me and reinstilling the reason for this pursuit. Neelsh and Sandhya were always there, ready with the perfect pep-talk when I needed it. I have been lucky finding all these and more friends that supported my efforts by being family to my family.

My loving, caring, sensitive husband whom I took for granted, and who responded at all times, made my struggle through college worth it.

DEDICATION

To Ananya,

my loving, understanding, 3 year old daughter.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Introduction to the problem

A module in a graph $G = (V, E)$ is a set X of vertices such that every vertex in $(V - X)$ is either adjacent to all vertices in X or non-adjacent to all of them. Any vertex outside

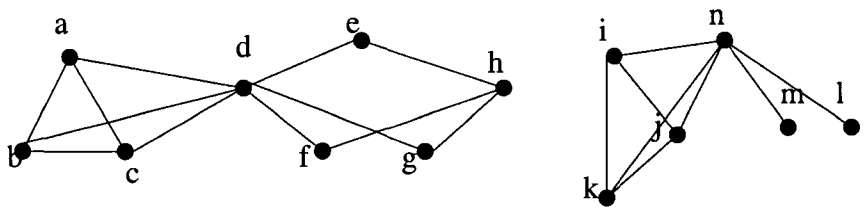

Figure 1.1: $\{a, b, c, d, e, f, g, h\}$ and $\{i, j, k, l, m, n\}$ are the connected components of this graph

a module either has edges to all vertices in the module or has no edge to any vertex inside the module. Hence modules are also invariant under taking complement. The vertex set V and singleton vertices of a graph are modules in any graph.

The connected components of a graph are always modules in a graph. However, they may not be the only modules in a graph. If G is a clique then every subset of vertices of V is a module. Hence there can be an exponential number of modules in a graph. In spite of this existence of numerous modules, there is an $O(n)$, implicit, tree-like representation of all the modules in a graph. The root of this tree is the vertex set V and its leaf nodes are the vertices of V.

If X and Y are two disjoint modules and if some vertex of Y is a neighbor of a vertex

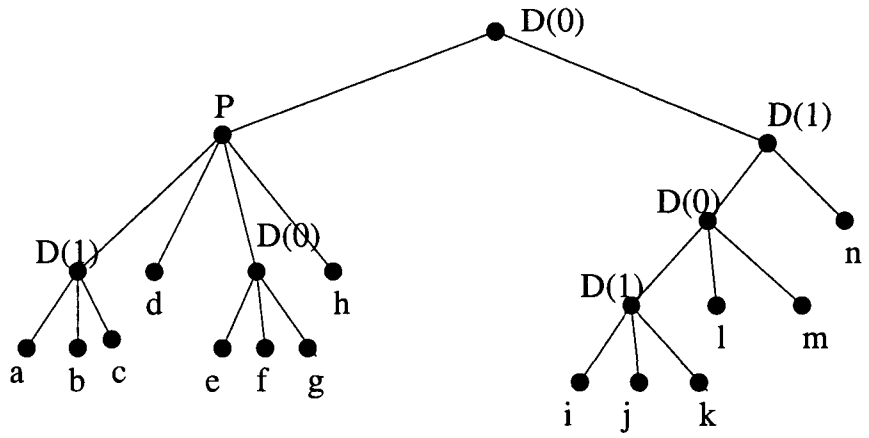

Figure 1.2: P indicates the node is prime, D indicates it is a degenerate node

in X, then all vertices of Y are neighbors of all vertices of X. Let $\mathcal{Q}$ be a non-trivial partition of V such that each member of $\mathcal{Q}$ is a module. Since all the vertices of a module have the same relation to all the vertices of another module, we can replace each of these modules by a representative vertex. This observation gives rise to a quotient graph. This graph represents the adjacencies between members of $\mathcal{Q}$.

A set is a module in the quotient graph if and only if the union of all its elements in the parent graph is a module. The subgraphs induced by the members of $\mathcal{Q}$ record the relationships in G, that are not captured by the quotient. Any module contained in X is a module in $G|X$ if and only if it is a module in G. Further simplification can often be obtained by decomposing the factors and the quotient recursively. Modular decomposition is a unique, canonical method that implicitly represents all possible ways to decompose the graph into quotients and factors.

In the tree representation every node is a module. Modules that are not nodes are always unions of children. A remarkable property of modules is that if the union of any non-trivial subset of children is a module, then all possible unions of children are also modules. Accordingly, the nodes of the tree are labeled Prime or Degenerate (denoted as P or D).

McConnell and Spinrad[34, 30] have given the first linear-time algorithm to compute the modular decomposition tree in graphs. The linear-time modular decomposition of

undirected graphs gives a linear-time bound for computing the transitive orientation of undirected graphs [35]. Some NP-hard combinatorial problems like computing the maximum weighted clique, the maximum independent set and the minimum number of cliques required to cover all the vertices can be solved in polynomial time for restricted classes of graphs using their modular decomposition. The tree representation is also used to classify graphs for which such solutions are possible.

Modular decomposition was first described in the 1960's by Sabidussi [22] and Gallai [50]. Möhring and Rademacher [39, 38] have surveyed the topic. Kelly [14] gives a history of the idea. James Stanton and Cowan gave an $O(n^4)$ algorithm to compute the decomposition tree in undirected graphs [28]. Various $O(n^3)$ algorithms to construct the decomposition tree have come up since then. Buer and Möhring [24], Cunningham [9], Golumbic [40], Habib and Maurer [41] and Steiner [49] present some of them. Muller and Spinrad [42], Ehrenfeucht, Gabow, McConnell and Sullivan [2] and McConnell [31] give an $O(n^2)$ algorithm for undirected graphs. Linear-time algorithm to compute the decomposition tree in undirected graphs has been given by [35, 30, 8, 16].

There is also a generalization of modules and modular decomposition to directed graphs and there have been algorithms of various complexities in the case of directed graphs. Maurer[1977] gives an $O(n^4)$ algorithm for the general case of directed graphs. The time bound for a the directed case of graphs $O(n^2)$ [2, 31], or $O(m \log n)$ [18] were followed by a $O(n+m)$ algorithm by McConnell and Montgolfier to compute the decomposition tree in the directed case [33].

The notion of modules in graphs has been extended to k-hypergraphs by Bonizonni and Della Vedova[6]. This notion is what this thesis is built upon. Bonizzoni and Della Vedova [6] also present an $O(n^{3k-5})$ algorithm to compute the Modular decomposition tree in k-hypergraphs .

1.2 Thesis Outline

This section captures the significance of each chapter of this thesis. Chapter 2 deals with some definitions and terminology used in this thesis. Chapter 3 provides an overview of properties of modules in undirected graphs. Chapter 4 gives details of the properties of modules in graphs that gives rise to the modular decomposition tree. Chapter 5 presents some classes of perfect graphs, where modular decomposition has found several applications. Chapter 6 presents the relation between transitive orientation and modular decomposition. This chapter also presents some previous approaches to compute the decomposition tree in graphs. Chapter 7 presents a discussion of modules in k-hypergraphs and the properties of the decomposition tree. Chapter 8 summarizes the new approach to compute the decomposition tree and Chapter 9 elaborates on the thesis result. Chapter 10 concludes this thesis and conjectures a future improvement in the time bound.

Chapter 2

Notation and Terminology

The following are some of the notational conventions and terms adopted by this dissertation.

The **difference** between two sets X and Y is denoted by $X - Y$ or $Y - X$, where $X - Y = \{u \in X, \quad u \notin Y\}$. Sets X and Y are said to **properly overlap** if $X \cap Y \neq \phi, X - Y \neq \phi$ and $Y - X \neq \phi$. $X \Delta Y$ is used to denote the **symmetric difference** of the sets X and Y, $X \Delta Y = (X - Y) \cup (Y - X)$.

In this dissertation, $X - y$ denotes $X - \{y\}$ and $X + y$ denotes $X + \{y\}$. A k-set is a set of size k where k is a positive integer. A hypergraph is usually denoted as $H = (V, E)$, where V is a finite set called vertices and E is a collection of subsets of V called hyperedges of H. A hyperedge can be of any size greater than or equal to two. If the size of the hyperedges in a hypergraph does not exceed k then it is called a k-hypergraph; such a hypergraph is said to have **dimension** k. An undirected graph is just the special case of a 2-hypergraph.

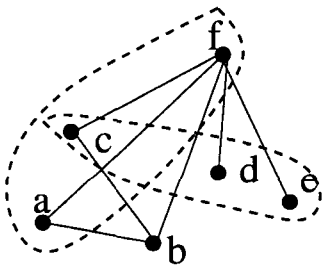

Figure 2.1: Example of a 3-hypergraph

Let n denote the number of vertices and m denote the number of edges in a k-hypergraph. Let s denote the sum of cardinalities of hyperedges in a k-hypergraph, clearly s less than or equal to mk.

The complement of a k-hypergraph is denoted as $\bar{H}$. $\bar{H}$ has the same vertex set as H . The edges of $\bar{H}$ are all possible k-hyperedges that are not contained in H.

A **partition** $\mathcal{Q}$ of a set V is a collection of disjoint sets whose union is set V. A set S is called a **system of representatives** if it contains exactly one vertex from each set in the partition $\mathcal{P}$.

$H|X$ denotes the k-hypergraph of H restricted to X. The vertices of $H|X$ are the elements of X and an edge e belongs to $H|X$ if $e \in E$ and all its vertices are in X *i.e* $H|X = (X, E \cap 2^X)$. This hypergraph is also called an **induced hypergraph** .

In a graph G, a vertex x is said to be **adjacent** or a **neighbor** of a vertex y, if $(x, y) \in E$. The notion of adjacency and a neighbor in any k-hypergraph will be formally defined in the next chapter.

If $G = (V, E)$ is a graph then the subgraph induced by the set of vertices $A \subseteq V$ is the graph $G_A = (A, E_A)$ where $E_A = \{(x, y) \in E : x \in A \text{ and } y \in A\}$.

A 2-structure [1] is a triple $G = (V, E, p)$, where V is a finite vertex set, p is a positive integer and $E : V \times V \mapsto \{1, \ldots, p\}$ is a coloring function. A 2-structure is **symmetric** if $E(x, y) = E(y, x) \ \forall \ x, y \in V$. When $p = 2$, if one of the color classes is interpreted as the edges and the other as a non-edge, the 2-structure denotes a digraph. A symmetric 2-structure can be interpreted as a graph.

Chapter 3

Modules in Undirected Graphs

Definition 3.0.1. *A vertex $y \in V - X$ is said to be a* ***neighbor*** *of a set X in G, if and only if for every $x \in X$, the set $\{y, x\} \in E$. A vertex $y \in V - X$, is said to be a* ***non-neighbor*** *of a set X in G, if there is no $x \in X$ such that the set $\{y, x\} \in E$. A vertex $y \in V - X$ is said to be a* ***spoiler*** *for a set X, if y is neither a neighbor nor a non-neighbor of the set X.*

Definition 3.0.2. *A set $X \subseteq V$ is said to be a* ***module*** *if for every vertex $y \in V - X$, y is either a neighbor of X or is a non-neighbor of X.*

If a set X is a module then every vertex in $V - X$ has a uniform relationship to vertices in X. Hence X can be replaced with a representative vertex which can represent the adjacency relationships of X's vertices with its outsiders.

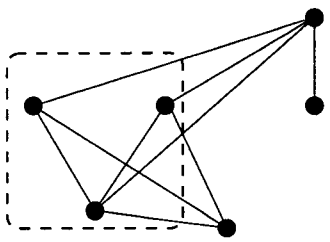

Figure 3.1: Example of a module in a graph

A set X is not a module if there is a vertex in $V - X$ that does not have a uniform relationship with vertices in X. So, in order to prove that a set X is not a module, it is sufficient to identify a spoiler vertex of X.

The following theorems have already been formalized and proved in Möhring and Radermacher's paper[39]. These theorems are presented here again only for contextual review.

Theorem 3.0.3. *[**Existence of quotient**] If X and Y are disjoint modules in G then either every vertex in X is adjacent to every vertex in Y, or no vertex in X is adjacent to any vertex in Y.*

Proof. If there is at least one edge (x, y) such that $x \in X$ and $y \in Y$, then since Y is a module x is adjacent to all vertices in Y. Hence $\{x\} \times Y \in E$. Let $z \in Y$ be an arbitrary vertex. We know that $(z, x) \in E$. Since X is a module this implies $z \times \{X\} \in E$. Since z was arbitrary this concludes that all edges of the form $X \times Y \in E$. □

Definition 3.0.4. *A **congruence partition** is a partition $\mathcal{Q}$ of V such that every set $X_i \in \mathcal{Q}$ is a module.*

The adjacency relationships of the member of $\mathcal{Q}$ is described by the graph called the **quotient graph** $G/\mathcal{Q}$.

Theorem 3.0.5. *If $\mathcal{Q}$ is a congruence partition and if S is a system of representatives from each set in the partition then $G/\mathcal{Q}$ is isomorphic to $G|S$.*

Proof. The proof follows from Theorem 3.0.3. □

The quotient graph can be obtained by replacing each X_i by a representative element from it (Theorem 3.0.5). The relationship between the representative elements will give the edges that go between the sets.

Let $\mathcal{T}$ denote the modular decomposition tree of the undirected graph $G = (V, E)$. Let $\mathcal{Q}$ be a congruence partition of the graph G.

Theorem 3.0.6. *[**Quotient Rule in graphs**] If $\mathcal{Q}$ is a congruence partition of an undirected graph G, then the union of sets in $\mathcal{Q}$ is a module in G if and only if the corresponding set of nodes of $G/\mathcal{Q}$ is a module in $G/\mathcal{Q}$*

In order to reconstruct the graph G, the edges in each of the subgraphs $G|X_i$ need to be known. Given the partition $\mathcal{Q}$ the quotient graph along with the induced subgraphs gives the complete representation of the original graph G. The following is an example.

Theorem 3.0.7. ***[Autonomous rule in graphs]*** *If X is a module of an undirected graph G, then the modules of G that are subsets of X are given by the modules of $G|X$. The strong modules of G that are proper subsets of X are given by the strong modules of $G|X$.*

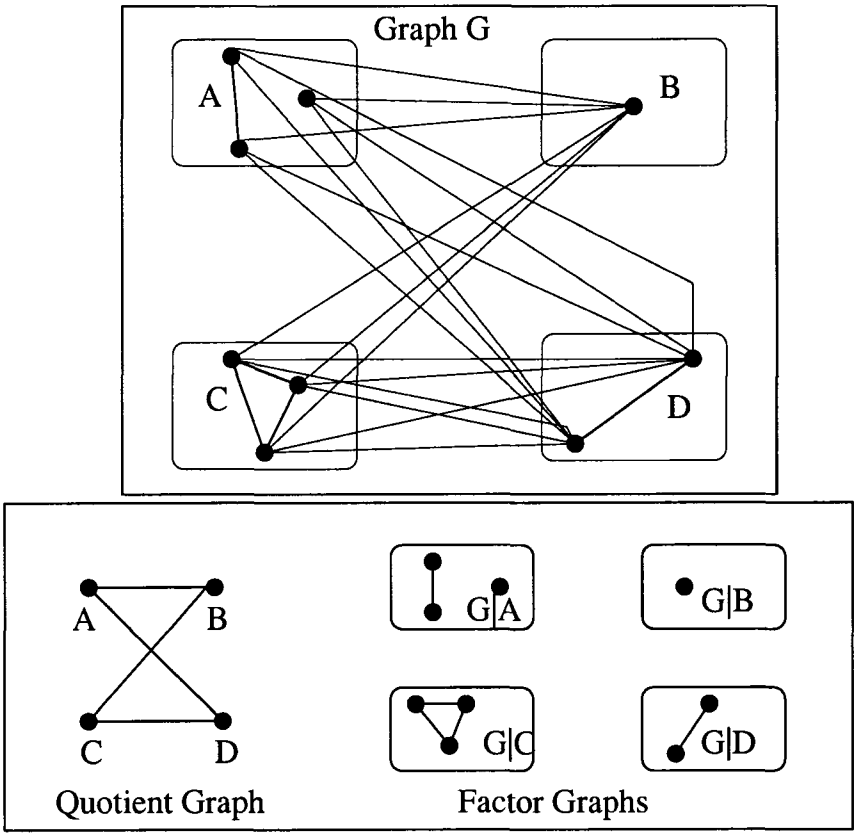

Figure 3.2: The information given by the quotient graph and the factor graphs are enough to reconstruct G

Chapter 4

Modular Decomposition Tree - Properties of the Tree

In the following section we will take a look at properties of modules that give it an O(n)-space representation.

4.1 Partitive Set Family

Definition 4.1.1 ([39]). *Let $\mathcal{F}$ be a family of subsets of a finite set U. $\mathcal{F}$ is a* ***partitive set family*** *if it satisfies the following properties.*

1. *All members of $\mathcal{F}$ are non-empty.*
2. *$U \in \mathcal{F}$.*
3. *$\{x\} \in \mathcal{F}$ for each $x \in U$.*
4. *If X and Y are overlapping members of $\mathcal{F}$, then $X \cup Y$, $X \cap Y$ and $X \Delta Y$ all belong to $\mathcal{F}$.*

Definition 4.1.2. *The sets that do not overlap with any other member of a partitive set family $\mathcal{F}$ are called* ***strong members*** *[39].*

Theorem 4.1.3 ([39]). *If $\mathcal{F}'$ denotes the strong members of $\mathcal{F}$ then the transitive reduction of the containment relation imposes a unique tree on $\mathcal{F}'$. Every member of $\mathcal{F}$ is either a node of the tree or a union of siblings in the tree. Each member of the tree is one of the following types:*

- ***degenerate:*** *the union of every subfamily of its children is a member of* $\mathcal{F}$

- ***prime:*** *non-trivial unions of any subfamily of its children is not a member of* $\mathcal{F}$.

The tree $\mathcal{F}'$ is called the **decomposition tree** of $\mathcal{F}$. In a decomposition tree, a node that has two children may be labeled prime or degenerate. To avoid this ambiguity this thesis deems these nodes degenerate.

4.2 Modular Decomposition Tree

Let $\mathcal{F}$ be the family of modules in a graph $G = (V, E)$. For $\mathcal{F}$ to be a partitive set family, $\mathcal{F}$ has to satisfy Definition 4.1.1. The justification that properties 1, 2 and 3 of Definition 4.1.1 are satisfied is addressed below. The set V is a module since there is

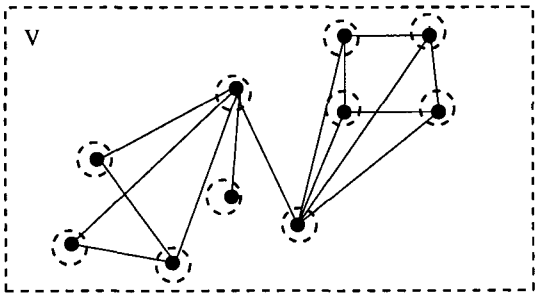

Figure 4.1: Singleton vertices and the vertex set V are trivial modules in any graph

no vertex outside it that can spoil it. Any singleton vertex is a module since any vertex outside this sees the only vertex inside the set the same way. V and its singleton subsets are **trivial modules** and are modules in every graph. A **prime graph** is a graph that has only trivial modules. To prove that a module is a partitive set family, what remains to be proved is :

If X *and* Y *are overlapping members of* $\mathcal{F}$*, then*

- $X \cup Y$

- $X \cap Y$ *and*

- $X \Delta Y$ *all belong to* $\mathcal{F}$.

Theorem 4.2.1 below provides this proof. The theorems following Theorem 4.2.1 are relevant to constructing a Modular Decomposition Tree. They have already been formalized in Möhring and Radermacher's paper [39]. These theorems are presented here again only for contextual review.

Theorem 4.2.1. *If X and Y are modules in G and if X and Y properly overlap one another then $X \cap Y$, $X \cup Y$, $X - Y$, $Y - X$ and the symmetric difference $X \Delta Y$ are also modules in G.*

Proof. Consider the following arbitrary vertices $x \in X - Y$, $y \in Y - X$, $z \in X \cap Y$ and $u \in V - (X \cup Y)$.

- $X \cup Y$: $(x, u) \in E$ if and only if $(z, u) \in E$, since x and z belong to the module X and u is outside X. If $(z, u) \in E$ exists then $(y, u) \in E$, since $y, z \in Y$ and Y is a module. Hence x, y, and z are not distinguished by any vertex in $V - (X \cup Y)$. $X \cup Y$ is a module, observe that $X \cap Y$, $X - Y$ and $Y - X$ are also not distinguished by any vertex in $V - (X \cup Y)$.

- $X \cap Y$: For $X \cap Y$ to be a module it is necessary that $X \cap Y$ not be distinguished by any vertex in $X - Y$ or $Y - X$. We can first show that $X \cap Y$ is not distinguished by vertices in $X - Y$. If $(x, z) \in E$ then $\{x\} \times Y \in E$ since $z \in Y$ and x is outside the module Y. With a similar argument we can show that vertices in $X \cap Y$ is not distinguished by any vertex in $Y - X$.

- $X \Delta Y$: We have shown earlier that $X - Y$ and $Y - X$ are not distinguished by any vertex in $V - (X \cup Y)$. It remains to show that the sets are not distinguished by any vertex in $X \cap Y$. Suppose $(z, x) \in E$. We need to show that z has edges with all vertices in $X - Y$ and $Y - X$. $z \in Y$ and $x \in (V - Y)$ and since Y is a module $Y \times \{x\} \in E$, since $Y - X \subseteq Y$ this implies $(Y - X) \times \{x\} \in E$. For $y \in (Y - X)$, since $x \in X$ and X is a module $\{y\} \times X \in E$, and since $(X - Y) \subseteq X$ and $(X \cap Y) \subseteq X$, $\{y\} \times (X - Y) \in E$ and $\{y\} \times (X \cap Y) \in E$. $z \in X \cap Y$ implies z has edges with all vertices in $Y - X$.

Consider arbitrary vertex $x' \in X - Y$, $x' \neq x$ to conclude this proof we need to show $(z, x') \in E$. From the argument above we know $y \times X - Y \in E$ which implies $(y, x') \in E$, since $y \in Y$ and since x' is a vertex outside a module Y this implies $\{x'\} \times Y \in E$ in particular $\{x'\} \times (X \cap Y) \in E$ and we can conclude $(x', z) \in E$.

- $X - Y$ and $Y - X$: $X - Y = X \cap (X \Delta Y)$. X and $X \Delta Y$ are modules which properly overlap, hence their intersection is a module which implies $X - Y$ is a module. $Y - X = Y \cap (X \Delta Y)$ hence from a similar argument $Y - X$ is a module.

□

By Definition 4.1.1, $\mathcal{F}$ is a partitive set family. Similar to the notion of **strong members** in a partitive set family [Section 4.1], the notion of a **strong module** is very useful in the context of constructing the modular decomposition tree. Strong modules are the strong members of the family $\mathcal{F}$ defined above.

Definition 4.2.2. *A strong module does not overlap with any other module. If X is a strong module and Y is some other module then either $X \subseteq Y$ or $Y \subseteq X$ or $X \cap Y = \phi$.*

A **weak module** is a module that is not strong. Since strong modules are the strong members of the family, owing to Theorem 4.1.3, it is intuitive that strong modules in a graph have a tree representation. The root of the tree is the node V and its leaves are the singleton vertices. The intermediate nodes of the tree are strong modules. It is worth noting that not all unions of siblings in the tree are weak modules. To identify the weak modules in the tree it is useful to label each strong module in the modular decomposition tree as **degenerate** or **prime**[Theorem 4.1.3].

Theorem 4.2.3 ([39]). *If X is a strong module of G and if T represents the modular decomposition tree of G then one of the following cases applies for X.*

- *X is degenerate : if the union of any subfamily of $children_T(X)$ is a member of $\mathcal{F}$*
- *X is prime: no union of any nontrivial subfamily of $children_T(X)$ is a member of $\mathcal{F}$*

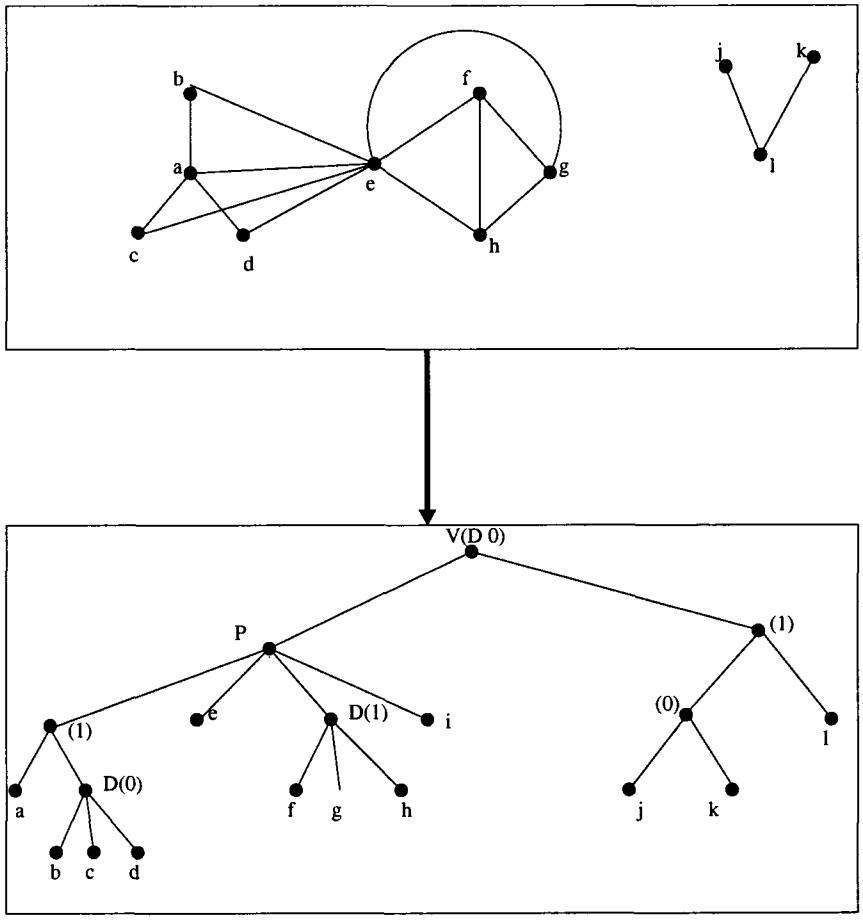

Figure 4.2: A modular decomposition tree where the leaves are vertices and intermediate nodes are strong modules. A node is labeled P or D depending on whether it is prime or degenerate. The figure above, labels them 1-node or 0-node to indicate the edges between them.

A degenerate node can further be classified as either a 1-node(**series node**) which means $X/\text{children}_T(X)$ is a complete graph or a 0-node (**parallel node**), which means $X/\text{children}_T(X)$ is edgeless.

Lemma 4.2.4 ([39]). *A 1-node cannot have a child that is a 1-node and a 0-node cannot have a child that is a 0-node*

Chapter 5

Perfect Graphs

A graph is perfect if the size of its maximum clique is equal to the size of its minimum coloring. Perfect graphs are dealt with extensively in the book by Golumbic [21]. This is a very large class of graphs encompassing many sub-classes of graphs that come up in many combinatorial problems. Characterization of perfect graphs was known to be difficult until the affirmative answer to the then-perfect-graph-conjecture, which is, a graph is perfect if and only if its complement is perfect [19]. The proof of this conjecture was given by Lovasz [29]. Golumbic [21] narrates that, in 1972, Lovasz [29] proved the Perfect Graph Theorem just ahead of Fulkerson who was working on it for a while. Fulkerson completed his proof a few hours after he learn't that Lovasz had proved it. Lovasz's proof makes extensive use of modular quotients. Finding the maximum clique and minimum coloring are NP-hard problems in graphs, but polynomial-time in perfect graphs. The Strong Perfect Graph Theorem that Berge conjectured in 1960, indicated that a graph is perfect if and only if neither the graph nor its complement contains an odd cycle of length at least five. After more than forty years, the Strong Perfect Graph Theorem was proved in 2002 by Chudnovsky, Robertson, Seymour and Thomas. Since then, many proofs on perfect graphs that were quiet complicated before became rather trivial. The following sections are a brief overview of some classes of perfect graphs.

5.1 Chordal graphs or Triangulated graphs

A chord in a simple cycle is an edge that is not an edge of the cycle, but whose end points are both vertices in a cycle. An undirected graph G is called triangulated if every cycle of length strictly greater than three possesses a chord. Triangulated graphs are also called chordal graphs. G does not contain any induced subgraphs isomorphic to C_n for n greater than three. Triangulation is a hereditary property inherited by all the induced subgraphs of G.

Definition 5.1.1. *[21]A vertex $v \in G$ is called **simplicial** if its adjacency set Adj(v) induces a complete subgraph of G, that is, Adj(v) is a clique.*

Dirac [13] and later, Lekkerkerker and Boland [27], proved that a triangulated graph always has a simplicial vertex. Triangulation is a hereditary property. Fulkerson and Gross [11], taking advantage of these properties, suggested an iterative procedure to recognize triangulated graphs by recognizing simplicial vertices and eliminating them one at a time. If the graph is triangulated, the algorithm terminates when the graph has no more vertices. If the graph is not triangulated, the procedure terminates in a graph for which there is no simplicial vertex.

Definition 5.1.2. *[21] Let $G = (V, E)$ be an undirected graph and let $\sigma = [v_1, v_2, \ldots v_n]$ be an ordering of the vertices. We say that σ is a perfect elimination scheme if each v_i is a simplicial vertex of the induced subgraphs $G_{\{v_i, \ldots v_n\}}$. In other words, each set $X_i = \{v_j \in Adj(v_i) | j > i\}$ is complete.*

Definition 5.1.3. *[21] A subset $S \subseteq V$ is a vertex separator for non-adjacent vertices a and b if the removal of S from the graph separates a and b into distinct connected components.*

Lemma 5.1.4. *[21] Let G be an undirected graph then G is triangulated if and only if every minimal vertex separator induces a complete subgraph of G.*

Proof. If G is triangulated, then every simple cycle C of length greater than four has a chord. If G is complete, then this lemma is trivial. Let a and b be two non-adjacent

vertices in G and let S be its minimal vertex separator. If S contains only a vertex x then it is a trivial complete subgraph of G. Let x and y be two arbitrary vertices in S. We will show that x and y are adjacent. This would imply that any two vertices of S are adjacent and hence will conclude that S is complete.

Let G_a and G_b denote the connected components containing the vertices a and b respectively. S is minimal implies that each vertex in S has to be adjacent to at least one vertex in G_a and at least one vertex in G_b. Since G_a and G_b are connected components there is at least a path each from x to y in the components G_a and G_b. Let P_a be the shortest path from x to y in the component G_a and let P_b denote the shortest path from y to x in the component G_b. Both P_a and P_b will contain at least one vertex each from G_a and G_b respectively. $\{x, P_a, y, P_b\}$ is a cycle of length at least four. The graph is triangulated implies the cycle should have a chord. There are no edges between vertices in P_a and vertices in P_b since S is a separator. There is no edge from vertices in P_a (other than the ones in the path) to x or y since our algorithm chose the shortest path. Since the graph is triangulated we can conclude that x and y have to be adjacent. This implies the subgraph induced by the minimal vertex separator is complete.

If every minimal vertex separator is complete then the graph is triangulated. That is we need to show that any cycle of length greater than four has a chord. Let a and b be two vertices of a graph that belong to a cycle of length greater than four. If a and b are adjacent then (a, b) is the chord. Suppose there is a cycle $(a, x_1, b, y_1, \ldots y_k, a)$, $k \geq 1$, the minimal separator of $a - b$ contains x_1 and at least one of the y_i $(1 \leq i \leq k)$. From our assumption, we know that x and y_i have to be adjacent which implies that there is a chord in every cycle of length greater than or equal to four. This concludes that the graph is triangulated. □

Lemma 5.1.5. *[13] Every triangulated graph $G = (V, E)$ has a simplicial vertex. Moreover, if G is not a clique, then it has two non-adjacent simplicial vertices.*

Proof. In a clique every vertex is a simplicial vertex, in which case, the lemma is trivial. Our induction hypothesis is that the lemma is true for all graphs of fewer vertices than

G. Since G is not a clique there exists a non-adjacent pair a, b. Let S be a minimal vertex separator for this pair. Let G_a, G_b denote the connected components containing a and b respectively.

The graph $G_a \cup S$ is a graph which contains at least one vertex less than the graph G. By our induction hypothesis, this graph has two non-adjacent simplicial vertices. Since S is a clique, at least one of the simplicial vertices has to be contained in G_a. Let $x_a \in G_a$ be the simplicial vertex in $G_a \cup S$. Since none of the vertices in G_b is adjacent to x_a, this vertex continues to be simplicial in G.

With a similar argument we can show that G_b contains at least one of the two simplicial vertices of the graph $G_b \cup S$. Once again, since none of the vertices in G_a is adjacent to x_b, x_b continues to be simplicial in G. Since S is their separator, x_a and x_b are non-adjacent in G. Hence we have two non-adjacent simplicial vertices when the graph G is not a clique. □

Lemma 5.1.6. *[21] Let G be an undirected graph. The following statements are equivalent:*

1. *G is triangulated*
2. *G has a perfect vertex elimination scheme. Moreover, any simplicial vertex can start a perfect scheme.*
3. *Every minimal vertex separator induces a complete subgraph of G.*

Proof. The equivalence of one and three follows from Lemma 5.1.4. From Lemma 5.1.5, we know that every triangulated graph has a simplicial vertex. Given a triangulated graph we can eliminate a simplicial vertex. Since being triangulated is a hereditary property, the remaining graph has to be triangulated. Hence triangulated graphs have a perfect elimination scheme.

Suppose there is a perfect elimination scheme for a graph. Let C be a cycle in this graph. Let v be a vertex with the smallest index in the perfect elimination scheme. We

know that the vertices that v is adjacent to in C have to be adjacent to each other. v is adjacent to at least two vertices in C, and hence we get a chord in C. □

Given an undirected graph, this algorithm produces an ordering of vertices that is a perfect elimination scheme. As a part of this algorithm, we maintain a label value and an order number for each vertex in the graph.

Algorithm for Perfect Elimination scheme [21]

Initially set the label of the vertices as zero.

Set the order number of all the vertices to be 0

For $i = n$ to 1

Choose a vertex v with the largest order number, if there is a tie choose one among the tied ones arbitrarily

for each vertex $w \in Adj(v)$

if (label(w)=0) then add i to order(w)

label(v) = i

Output the ordering of vertices as v_1 to v_n as a perfect elimination scheme

Lemma 5.1.7. *[21] An undirected graph $G = (V, E)$ is triangulated if and only if the ordering σ produced by the above algorithm is a perfect elimination scheme.*

Proof. Suppose the result is true for graphs fewer than n vertices. It is sufficient to show that the last vertex in the scheme is a simplicial vertex in G. Suppose x is not simplicial then it has at least one pair of neighbors x_1, x_2 that are non-adjacent. Pick x_2 such that it represents the vertex with largest index for which x is a neighbor but x_1 is not a neighbor.

Since x_1 occurs ahead of x there should be a vertex x_3 that is adjacent to x_1 and not adjacent to x. If x_3 adjacent to x_2 then we would have a cycle of length four with no chord. Hence x_3 only has an edge with x_1.

x_2 is ahead of x_1 in the ordering hence there is a vertex that is adjacent to x_2 but not adjacent to x_1. Let x_4 denote the vertex with largest index. It cannot be adjacent to x or x_3 since then there would a chord less cycle of length greater than three, a violation to G be triangulated. Hence we have the following ordering of vertices [21]

1. $(x, x_i) \in E$ for $i = 1, 2$
2. $(x_i, x_j) \in E$ for $|i - j| = 2$
3. In the perfect elimination ordering x_i occurs before x_j if $i > j$
4. x_j is the vertex with the largest index such that $(x_{j-2}, x_j) \in E$ but $(x_{j-3}, x_j) \notin E$

Suppose we have a series of vertices $\{x, x_2, \ldots x_k\}$, now since x_{k-1} is ahead of x_{k-2} in the series there should be a vertex x_{k+1} which is adjacent to x_{k-1} and not adjacent to x_{k-2}. Choose the vertex with the largest possible index $k+1$. x_{k+1} cannot be a neighbor of x_{k-3} since it would violate the condition that x_{k-1} is the vertex with the largest index that is a neighbor of x_{k-3} and not a neighbor of x_{k-2}. x_{k+1} cannot have any of the vertices in the set $\{x, x_1, \ldots x_{k-4}, x_k\}$ since it would violate the chordality condition of the triangulated graph.

This inductive procedure continues indefinitely. This is a contradiction since the graph is finite. Hence the assumption that x is not simplicial is incorrect.

The converse of this result follows from the equivalence in Lemma 5.1.6. □

Lemma 5.1.8. *[21] Let S be a vertex separator of a connected undirected graph $G = (V, E)$, and let $G_{A_1}, G_{A_2}, \ldots G_{A_t}$ be the connected components of G_{V-S} . If S is a clique then*

$$\chi(G) = \max_i \chi(G_{S+A_i}) \text{ and } \omega(G) = \max_i \omega(G_{S+A_i})$$

Corollary 5.1.9. *[21] Let S be a separating set of a connected digraph $G = (V, E)$, and let $G_{A_1}, G_{A_2}, \ldots G_{A_t}$ be the connected components of G_{V-S}. If S is a clique, and if each subgraph G_{S+A_i} is perfect then G is perfect.*

Proof. This follows directly from lemma above. □

Lemma 5.1.10. *[4] [21] Triangulated graphs are perfect*

Proof. There are two ways to show that triangulated graphs are perfect. The first is using the perfect elimination scheme (Lemma 5.1.7) the second is using the Corollary 5.1.9 above.

If a graph is triangulated then we know that the algorithm will produce a perfect elimination scheme. If $v_1, v_2, \ldots v_n$ are vertices in this scheme then we know that in the graph $G|\{v_i, \ldots v_n\}$ the vertex v_i is simplicial. We will color starting with the vertex v_n. We give v_n the color one and move to the next contiguous vertex v_{n-1}. At vertex v_i, we check the colors of its neighbors and give a smallest integer value unique from its neighbors. If there is a clique of size k then this algorithm ensures that all the colors from one to k are used. Since the size of the maximum clique is equal to the minimum number of colors used, triangulated graphs are perfect.

Another way of proving this lemma is using Corollary 5.1.9. Our induction hypothesis is that the lemma is true for triangulated graphs having fewer than n vertices. If G is a clique, there is no proof required. If G is not a clique, we can find non-adjacent vertices a and b. The minimal separator of these vertices is a clique. The connected components G_a and G_b which contain a and b respectively are perfect because of our induction hypothesis. Hence using Corollary 5.1.9 we can conclude that G is perfect. □

5.2 Comparability graphs

A directed acyclic graph is **transitive** if, whenever (a, b) and (b, c) are directed edges of the directed acyclic graph, (a, c) is also a directed edge. A transitive directed acyclic graph is a graphical representation of a partial order (poset) relation. A graph is a **comparability graph** if orientations can be assigned to its edges so that the resulting digraph is transitive. Comparability graphs are undirected graphs which can be transitively oriented. A transitive orientation of a comparability graph can be found in linear

time [34]. Every induced sub-digraph of a transitive digraph is transitive, so the class of comparability graphs is hereditary.

An undirected graph is a special case of a symmetric directed graph, where each undirected edges (a, b) is represented by two directed edges (a, b) and (b, a). Let $G = (V, E)$ be this representation. Finding a transitive orientation consists of selection $F \subset E$ such that (V, F) is a transitive digraph, and is such that $(a, b) \in F$ if and only if $(b, a) \notin F$. An edge (a, b) is said to be **incompatible** with the edge (b, c) if either $c = a$ or $c \neq a$ and (c, a) does not belong to the graph G.

The **incompatability graph** of a directed graph G is a graph whose vertices are the edges in the graph G. There is an edge between two vertices if the corresponding edges in G are incompatible. The incompatability graph and its relation to modules in a graph will be dealt with in the next chapter.

The proofs of the following two lemmas are dealt with in detail in [10].

Lemma 5.2.1. *[21, 50] An undirected graph G is a comparability graph if and only if its incompatibility graph is bipartite*

Lemma 5.2.2. *[10]When G fails to be a comparability graph, its incompatibility graph has an odd cycle of length O(n).*

Given an undirected graph there is a linear-time algorithm that computes the transitive orientation of the graph. If the given graph is not a comparability graph then this algorithm produces a transitive orientation that has a transitivity violation or an incompatible pair. Since an incompatible pair is not a certificate for recognizing comparability graphs, the algorithm detailed in [10] details a procedure on how to turn this incompatible pair into a certificate. The certificate is an odd cycle in the incompatibility graph.

Lemma 5.2.3. *Comparability graphs are perfect graph.*

Proof. Comparability graphs have a transitive orientation. If the input graph is a comparability graph then [30] finds an acyclic orientation that will be transitive. We can

label each vertex with the length of the longest directed path originating at the vertex in the resulting directed acyclic graph. Since the digraph is transitive, the longest path is a clique in the original graph. Since the size of a clique is a lower bound on the number of colors in a proper coloring, the coloring is a certificate that the clique is the maximum size and the clique is a certificate that the proper coloring is a minimal one.

□

We can do the labeling in linear-time by performing a depth-first search on the digraph. We label each vertex as it finishes during a depth-first search of the graph. Each neighbor of the vertex is already labeled at that point, so the vertex can be labeled with one plus the maximum of the labels of its neighbor.

Bipartite graphs are comparability graphs. If G is bipartite and if X and Y are the two sets in the partition then we can orient the edges from X to Y.

5.3 Interval graphs

Let $\mathcal{F}$ be a family of non-empty sets. The **intersection graph** of $\mathcal{F}$ is obtained by representing each set in the family by a vertex. Two vertices are connected if and only if their corresponding sets intersect. The intersection graph of a family of intervals on a linearly ordered set (like the real line) is called an **interval graph**. A graph is a circular-arc graphs if it is the intersection graph of the arcs in a circle. Interval graphs and circular-arc graphs arise in scheduling problems and other combinatorial problems. It was shown by Seymour Benzer [44], in 1959, that the set of intersections of a large number of fragments of genetic material in a graph is an interval graph. A stronger result, which is being able to represent a graph in terms of intervals or arcs, helps in solving a large number of combinatorial problems like finding maximum independent sets and cliques [7] [52]. Fulkerson and Gross [11] solved this problem in interval graph with an algorithm of $O(n^4)$ order. Booth and Lueker used PQ-trees to improve the time bound to linear [45]. PQ-trees and modular decomposition tree belong to class of families called partitive set families. The family of sets that belongs to a member of a

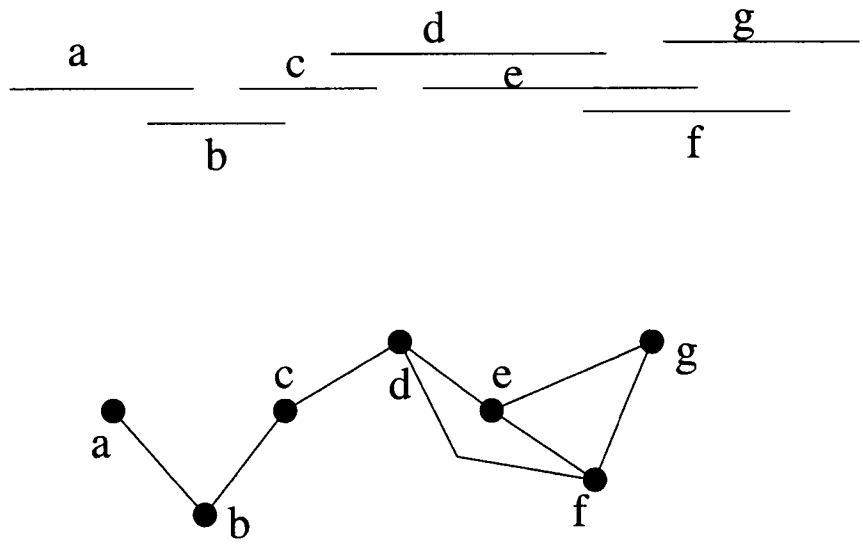

Figure 5.1: Intersection graph and the corresponding interval graph.

partitive set family can be quite large but can have a compact recursive representation in the form of a tree. Interval graphs are a special case of circular-arc graphs. Algorithm for recognizing a circular-arc graph and providing a certificate in the form of a set of arcs that realize it was initially conjectured to be NP-complete by Booth in his PhD thesis. Tucker gave an $O(n^3)$ algorithm to disprove the conjecture. This was followed by Hsu's $O(nm)$ algorithm [52] and Eschen and Spinrad's $O(n^2)$ algorithm. The first linear-time algorithm for circular-arc graph was given by McConnell [32]. This algorithm is based on modular decomposition tree, transitive orientation of comparability graph, and algorithms on permutation graphs and interval graphs.

The intersection graph of a family of intervals on a linearly ordered set(like the real line) is called an interval graph. Formally, let $I_1, I_2, \ldots I_n \in R$ be a set of intervals. Then the corresponding interval graph is $G = (V, E)$ where $V = \{I_1, I_2, \ldots I_n\}$ and $(I_\alpha, I_\beta) \in E \iff I_\alpha \cap I_\beta \neq \phi$. That is, the nodes of the graph are the intervals and there is an edge corresponding to each pair of intersecting intervals.

Interval graphs have many interesting properties. One of them is that interval graph is a hereditary property (that is an induced subgraph of an interval graph is an interval graph). Interval graphs are useful in modeling resource allocation problems in operations

research. Each interval represents a request for a resource for a specific period of time.

Lemma 5.3.1. *[21]Interval graphs are chordal graphs and hence perfect.*

Proof. To prove that interval graphs cannot have chordless cycles of length greater than three. Suppose there is a 4-cycle formed by the intersection of the intervals I_a, I_b, I_c, I_d in that order. Let $I_a = (a_1, a_2)$, $I_b = (b_1, b_2)$, $I_c = (c_1, c_2)$ and $I_d = (d_1, d_2)$. All these points are on the real line. If interval I_a intersects I_b then this means I_b can be contained in I_a or overlaps with I_a. Since I_b intersects with I_c which does not intersect with I_a this implies I_b is not contained in I_a. Without loss of generality let I_b starts just before the right end point of I_a that is $b_1 < a_2$ but $b_1 > a_1$. I_c intersects with I_b but does not intersect with I_a (that is $c_1 > a_2$). I_c is either contained in (a_2, b_2) or it overlaps the right end point of I_b. Since I_c intersects with I_d which does not have any intersection with I_b this implies $c_1 < b_2$ and $c_2 > b_2$. Now I_d should intersect with I_c but does not intersect with I_b this implies $d_1 < c_2, d_1 > b_2$. We already know that $a_2 < b_2$ this implies interval I_d is such that $d_1 > a_2$. This implies I_d does not intersect I_a which implies there is no chordless cycle in the interval graph. □

Lemma 5.3.2. *[21] The complement of an interval graph is a comparability graph*

Proof. Given G an interval graph and its representation by intervals. Let I_a and I_b denote two intervals represented by vertices a and b. If I_a and I_b do not intersect then I_a either lies entirely to the left of I_b or to entirely right of I_b. If I_a is to the left of I_b then orient the edge (a, b) from a to b in the complement. This algorithm will produce a transitive orientation in the complement of an interval graph G. □

Definition 5.3.3. *[10] Three independent vertices x, y, z of a graph G are an* ***asteroidal triple (AT)*** *of G if, between each pair of these vertices there is a path that contains no neighbors of the third.*

Lemma 5.3.4. *[21][10] Let G be an undirected graph. The following statements are equivalent*

1. *G is an interval graph*

2. *G is chordal graph and is AT-free*

3. *the maximal cliques of G can be linearly ordered such that, for every vertex $x \in G$, the maximal cliques containing G occur consecutively*

A certificate for a graph being chordal can be checked in linear-time using the linear time algorithm for the perfect elimination scheme. In [10] a linear-time method is given for checking whether a chordal graph is AT free.

5.4 Permutation graphs

A **permutation graph** is the graph of inversions in a permutation. That is, each vertex correspond to an element of the ground set of a permutation, and two vertices are adjacent if and only if the permutation reverses the relative order of the two corresponding elements. Let $\{v_1, v_2, \ldots v_n\}$ be an ordering of vertices in a graph G. Let $\{v_{\pi(1)}, v_{\pi(2)}, \ldots v_{\pi(n)}\}$ be another linear ordering of vertices. G consists of all the vertices and there is an edge (v_i, v_j) if and only if the relative ordering of v_i and v_j in other words v_i is before v_j in exactly one of the orderings.

Lemma 5.4.1. *[21] A graph is a permutation graph if and only if both G and its complement $\bar{G}$ are comparability graphs.*

There is a linear time algorithm for recognizing permutation graph as given in [10] . This algorithm produces a permutation model if the graph is a member of the class. A forbidden structure given by Gallai serves as a certificate that a graph is not a comparability graph.

A permutation graph is a graph that is a comparability graph, and whose complement is also a comparability graph. The union of a transitive orientation of a permutation graph and a transitive orientation of its complement gives a linear order on the vertices. Reversing the orientations in the graph and again taking its union with its complement

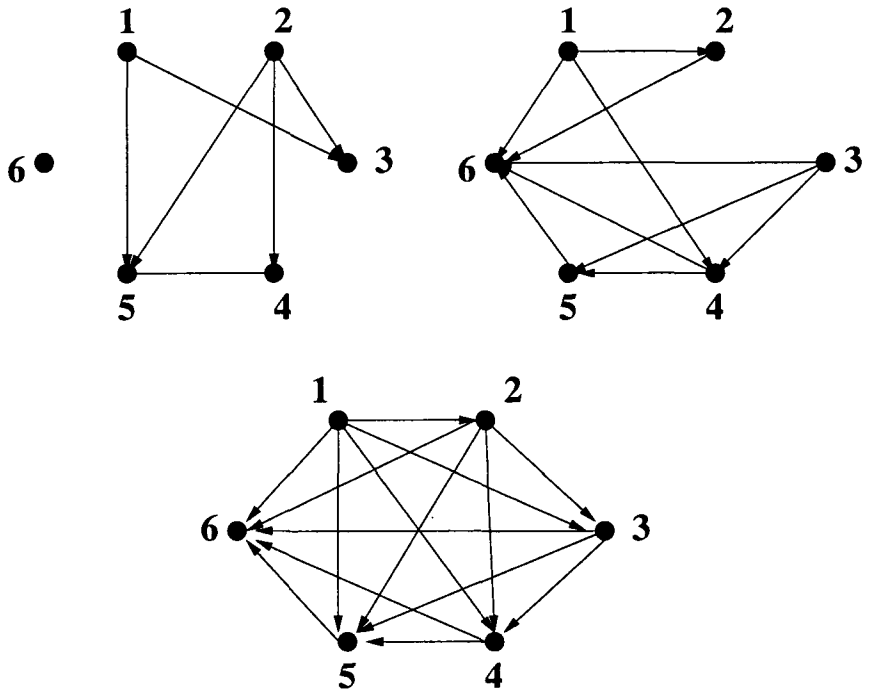

Figure 5.2: The transitive orientation of the graph and its complement gives the linear order $\{1, 2, 3, 4, 5, 6\}$.

gives another linear order. Two vertices are adjacent in the graph if and only if their relative order is the same in both of these orderings. This permutation realizer gives a way to represent permutation graph with two orderings of the vertices. There is a linear-time algorithm to certify whether a graph is a permutation graph [10]. If G is a permutation graph then the algorithm gives a transitive orientation of the graph and its complement in linear-time in the size of the graph. The certificate of correctness can be checked by comparing the two linear orders with the edges in the given permutation graph.

Lemma 5.4.2. *[10] Given an incompatible pair in the orientation of a graph G produced by the transitive orientation algorithm of [30], it takes $O(n + m)$ time to find an odd cycle of length $O(n)$ in G's incompatibility graph, and given an incompatible pair in an orientation of $\bar{G}$ by the algorithm, it takes $O(n + m)$ time to find an odd cycle of length $O(n)$ in $\bar{G}$'s incompatibility graph.*

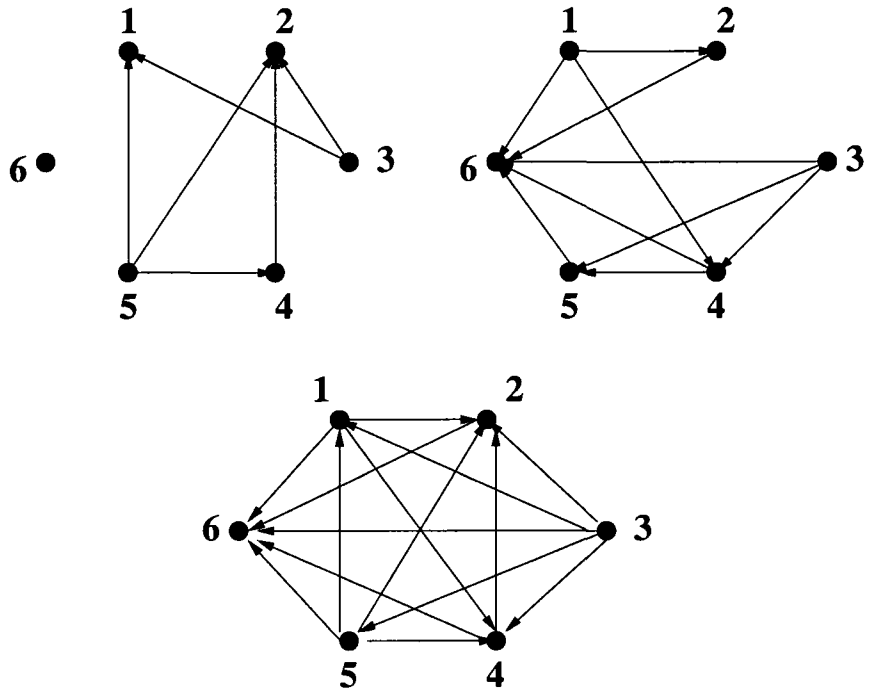

Figure 5.3: The transitive orientation of the reverse of the graph and its complement gives the linear order $\{3, 5, 1, 4, 2, 6\}$.

If the graph is not a permutation graph then there is a incompatible pair in G or $\bar{G}$. This is found when we attempt to check whether the two linear orders produced by the algorithm match the original graph. If the matching fails it points our finger to an incompatible pair. An incompatible pair is not a certificate that can be verified in linear-time. In [10] there is a linear time algorithm that produces a certificate in the form of an odd cycle in its incompatibility graph using the incompatible pair.

5.5 Cographs

The following are characterizations of cographs

- singleton node is a cograph,
- cographs are closed under the operations of union and complement,
- they that do not have an induced path on four vertices
- the decomposition tree of cographs does not contain prime nodes

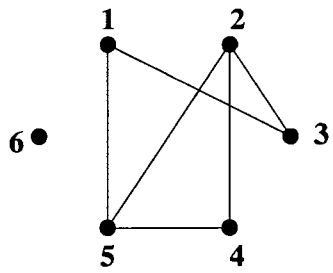

Figure 5.4: The graph in this figure is a permutation graph representing the linear orders $\{1, 2, 3, 4, 5, 6\}$ and $\{3, 5, 1, 4, 2, 6\}$.

Cographs are perfect since they form a proper subset of the permutation graphs. Cographs are hereditary. There is a linear-time algorithm for recognition of cographs [12] and this algorithm uses its modular decomposition tree. The decomposition tree of a cograph has no prime nodes in it. Using the decomposition tree it is possible to design very fast polynomial time algorithms for problems including chromatic number, clique determination, clustering, minimum weight domination, isomorphism, minimum fill-in and Hamiltonicity.

Transitive orientation figure heavily in the definition of perfect graphs and algorithms that operate on them. In the next chapter we describe the relationship between transitive orientation and modular decomposition.

Chapter 6

Some Known algorithms for Modular Decomposition in graphs

In Gallai's paper [50] a close relationship between the transitive orientation problem and modular decomposition of a graph is shown. Using the modular decomposition, a transitive orientation, if it exists can be found in $O(|V|+|E|)$ time [30]. The modules of a graph are an example of *partitive set family* [37, 36]. All partitive set families have a compact representation by means of a tree; the modular decomposition is just an example of it when the set family is the modules of a graph. The PQ-tree is another example of this phenomenon. One of the most significant applications of PQ-trees is finding planar embeddings of planar graphs. Booth and Lueker used PQ-trees to develop and algorithm for determining whether a family of sets has the consecutive ones property [45]. Booth and Lueker's algorithm gave a linear-time algorithm for determining whether a given graph is an interval graph, and, if so, finding such a set family for it. Other partitive families have played a role in linear-time bounds for recognizing circular-arc graphs [32], $O(n + m \log n)$ bounds for recognizing probe interval graphs [43].

Modules occur in the literature under various names. As closed sets [20], stable sets [46] or partitive sets [21, 51] in the theory of comparability graphs, as clumps [3, 5] in graph theory, as job-modules [26] in scheduling theory, as modules [47], autonomous sets [25, 24] or simplifiable subnetworks [48] in network analysis and reliability theory. Modular decomposition is also sometimes referred to as substitution decomposition [37] or X-join decomposition [41].

This chapter will describe the relation between transitive orientation and modular decomposition and also present some approaches used to find modular decomposition in graphs [24, 31, 2, 17].

6.1 Connection between transitive orientation and modular decomposition

An undirected graph $G = (V, E)$ can be considered to be a special case of a symmetric directed graph. In G we replace every undirected edge $(x, y) \in E$ by a directed pair $\{(x, y), (y, x)\}$. A transitive orientation of this graph will be selecting a set $F \subset E$ such that (V, F) is a transitive digraph, and $(x, y) \in F$ if and only if $(y, x) \notin F$.

Definition 6.1.1. *An edge (x, y) is said to be **incompatible** with the edge (y, z) if either*

- *$z = x$ or*
- *$z \neq x$ and (z, x) does not belong to the graph G.*

Definition 6.1.2. *Let $\mathcal{I}(G)$ denote the **incompatability graph** of a directed graph G. The vertices of this graph are the edges in the graph G. There is an edge between two vertices in $\mathcal{I}(G)$ if the corresponding edges in G are incompatible.*

The proofs of the following two lemmas are dealt with in detail in [21, 50, 10].

Lemma 6.1.3. *[21, 50, 10] An undirected graph G is a comparability graph only if its incompatibility graph is bipartite.*

Proof. If a graph is a comparability graph then this means it has a transitive orientation. We will show that if a graph has a transitive orientation then its incompatability graph is bipartite. First we need to show that there are two independent sets of the graph. Since the graph has a transitive orientation there is an orientation of the edges that satisfies the transitive property. Let F be such that (V, F) is a transitive digraph. F cannot contain an incompatible pair as described in Definition 6.1.1. This implies the collection of edges is such that in the incompatability graph there are no edges between them. Hence this is an independent set of the incompatability graph.

The reverse of a transitive orientation is also an orientation. This implies there is another collection of independent vertices in the incompatability graph. This covers all the edges in G hence the incompatability graph is bipartite. □

Lemma 6.1.4. *[21, 50, 10] If the incompatability graph of a graph G is bipartite then G is a comparability graph.*

The extensive proof of the lemma above is given in [21, 50, 10].

We know that a graph is bipartite if and only if it does not contain any odd cycle. From the above two lemmas we can see that a graph is a comparability graph if and only if the incompatability graph contains no odd cycle. This gives an easy to verify certificate that a graph is not a comparability graph; supply an odd cycle in the incompatability graph.

A connected component in an incompatability graph is connected in the graph. The connected component of an incompatability graph tells us which edge forces another and the number of ways to transitively orient the edges of a graph. If the incompatability graph $\mathcal{I}(G)$ is connected then there is only one way to bipartition it and there are exactly two ways to transitively orient G.

If the given graph G is disconnected then we can see that there is more than one bipartition and this gives a larger number of transitive orientation of the graph. For example if G_1 and G_2 are connected components of a graph G. We can choose from any one of the two transitive orientations of G_1 and G_2 respectively to find the transitive orientation of G.

Critical from this is the fact that if $\mathcal{I}(G)$ is disconnected then we have a couple of ways in which we can compute the transitive orientation of the graph. If it is disconnected we can reverse some of the disconnected components. Hence there can be more than one bipartition class and more than two transitive orientations.

Definition 6.1.5. *The connected components of $\mathcal{I}(G)$ are called the **color classes** of G.*

Let $\mathcal{C}(G)$ denote the collection of all color classes of a graph G.

Definition 6.1.6. *If $C \in \mathcal{C}(G)$ is a color class, we let $C_S = \{x|(x,y)$ or $(y,x) \in C\}$ denote the corresponding* ***support*** *of the color class. C_S is the set of vertices incident to the edges in C.*

Lemma 6.1.7. *If C_S is a support of a color class of G, then C_S is a module in G.*

Proof. If C_S is not a module in G then there exists a spoiler $z \in V - C_S$. Let x and y be two vertices in C_S such that x is adjacent to z and y is non-adjacent to z. A connected component of $\mathcal{I}(G)$ is connected in G. Since x, y belong to a connected component in $\mathcal{I}(G)$, the vertices are also connected in G. Consider a path from x to y consisting of vertices in C_S. Somewhere along this path there is pair of adjacent vertices $\{a, b\}$ that is spoilt by z. Without loss of generality assume $(b, z) \notin E$. Then the edges (a, b) and (a, z) are incompatible and will force z into C_S, a contradiction to our assumption that $z \in V - C_S$. Hence supports are modules. □

Lemma 6.1.8. *[50, 24] If there is a connected component in $\mathcal{I}(G)$ that contains (a, b) and (c, d) then there is no module in G that contains both end points of one of them and does not contain both end points of the other.*

The main insight behind the proof is the following. Suppose there is a module in G that contains (a, b) but does not contain (c, d). Since (a, b) and (c, d) belong to the same connected component of $\mathcal{I}(G)$ there is a path between the two edges in G. Let $\{a = x_1, b = x_2, \ldots x_{r-1} = c, x_r = d\}$ denote a collection of vertices such that (x_i, x_{i+1}) is an edge for $i = 1, \ldots r-1$. Somewhere along this path there is a vertex x_j that is not a member of the module. Consider the previous edge (x_{j-2}, x_{j-1}). Since x_j is forced into the support by the two vertices it is adjacent to exactly one of $\{x_{j-1}, x_{j-2}\}$. This makes x_j a spoiler of $\{x_{j-1}, x_{j-2}\}$, a contradiction to our assumption that x_{j-1} and x_{j-2} are members of a module in G. Hence our assumption that a module in G does not contain all the vertices in the support that contains the edge (a, b) is incorrect.

From the above two results we can make the following observations of a prime node and a degenerate 1-node in a decomposition tree.

Lemma 6.1.9. *Suppose* $\{X_1, X_2, \ldots X_r\}$ *are children of a prime node* X. *Suppose a support* C_S *contains an edge that goes between* X_i *and* X_j *for any* $i \neq j$ *and* $1 \leq i, j \leq r$ *then* $C_S = X$.

Proof. Let $(x_i, x_j) \in E$ be such that $x_i \in X_i$ and $x_j \in X_j$. If C_S contains an edge (x_i, x_j) then from Lemma 6.1.8 we know that C_S contains all the vertices in X_i and X_j. Since X is prime the smallest module containing x_i and x_j is X. Hence from Lemma 6.1.7 we can conclude that C_S should contain all the vertices in X.

Since X is a module. C_S cannot contain more vertices than the ones in X because that would contradict Lemma 6.1.8. □

Observation 6.1.10. *If* G *is a complete graph then* $\mathcal{I}(G)$ *is an independent set.*

Lemma 6.1.11. *Suppose* $\{Y_1, Y_2, \ldots Y_s\}$ *are children of a degeneate* 1*-node* Y. *The supports of color classes contained in* Y *but not its children will be unions of pairs of children of* Y.

Proof. Let $(y_i, y_j) \in E$ be such that $y_i \in Y_i$ and $y_j \in Y_j$. This edge will force a support that contains the vertices in Y_i and Y_j (by Lemma 6.1.8). Since the union is a module, it does not contradict Lemma 6.1.7. When we consider three children of a degenerate 1-node we are looking at a complete graph. From Observation 6.1.10 we know that vertices of a complete graph dont force one another hence there are no additional vertices in this support.

Since Y is a degenerate 1-node there is an edge between each pair of children of a degenerate 1-node hence $\mathcal{I}(G)$ will contain all possible unions of pairs of children of Y. □

6.2 Buer and Möhring[1983] - A Recursive Approach

Buer and Möhring in 1983 presented an algorithm to compute the modules which they termed as autonomous sets in $O(n^3)$ time occupying $O(n^2)$ space. This approach is based on the relation between color classes and modules described in the previous section. In the previous section we have seen how modules are related to color classes. We also saw how we can compute the transitive orientation of the graph given the modular decomposition tree. In this recursive approach color classes are uses to guide the modular decompoition algorithm.

Suppose all the nodes are prime then there is one support for each node of the decomposition tree. The root is prime and by Lemma 6.1.9 the supports are exactly the nodes of the tree. It is easy to find the color classes in $O(n^3)$ time and assemble the supports in the tree.

Suppose instead they are all degenerate then at every node either the subgraph is disconnected or its complement is disconnected. The time taken to compute the connected component of a graph or its complement is $O(n^2)$. There are at most n levels hence the run time is $O(n^3)$.

Buer and Möhring's algorithm is a combination of the above two tricks. At each node one of the tricks or the other is used depending on the node is degenerate or not.

6.3 Incremental Approach in undirected graphs

[31] details an incremental approach to find modular decomposition tree of a 2-structure. Since graphs are special cases of 2-structures this incremental approach, which happens to be an $O(n^2)$ algorithm, applies to graphs as well.

The principle of this approach is based on the idea of mathematical induction. It is visibly trivial to find the modular decomposition of a graph with just one node. For any graph, supposing the modular decomposition tree of a structure g is known, a structure g' can be obtained from g by adding one node. Although adding a new node spoils a few modules, this algorithm describes a procedure to construct the tree for g' from the

tree of g.

6.4 Divide and Conquer Algorithm

A simple divide and conquer algorithm for computing the tree decomposition of a two-structure is presented in [2]. The algorithm runs in $O(n^2)$ where n is the number of nodes in a two-structure. Since graphs are a specific case of a two-structure the restriction of this algorithm to graphs gives the modular decomposition tree of a graphs.

As seen in the previous chapter the modules of any graph G has a tree representation $\mathcal{T}$. For a given vertex $v \in V$ we can see that some nodes of $\mathcal{T}$ will contain the vertex v and others do not contain the vertex v.

Definition 6.4.1. *If v is a vertex of G, let **MaxModules**(G, v) denote the set of maximal modules of G that do not contain v. That is, $X \in$ MaxModules(G, v) if X is a module of G that does not contain v, and every proper superset of X that is a module of G contains v.*

It is intuitive from Figure 6.1 that each child of a prime ancestor of v, excluding the child that contains v is a member of Maxmodules(G, v).

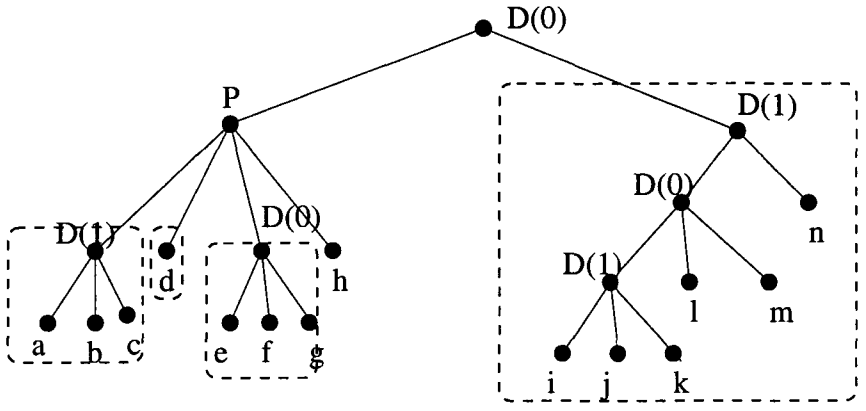

Figure 6.1: MaxModules(G, h) for a vertex $h \in V$, when a node is a child of a prime node each of its siblings is a member of MaxModules(G, h)

The union of all children of a degenerate ancestor of v, excluding the child that contains v, (Figure 6.2) is a member of Maxmodules(G, m), and that, other than $\{m\}$,

there are no other members of Maxmodules(G, m). Therefore, each child of a prime node

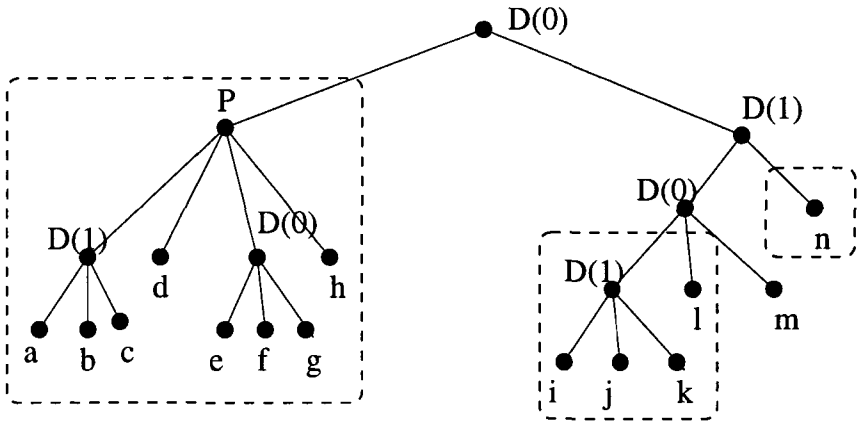

Figure 6.2: MaxModules(G, m) for a vertex $m \in V$, when a node is a child of a degenerate node the union of all its siblings is a member of MaxModules(G, m)

is a member of MaxModules(G, v) for some $v \in V$, and for every child of a degenerate node, the union of all of its siblings appears as a member of MaxModules(G, v) for some $v \in V$.

In [2] we see a simple divide and conquer algorithm for computing the tree decomposition of a two-structure. The algorithm runs in $O(n^2)$ where n is the number of nodes in a two-structure. Since graphs are a specific case of a two-structure the restriction of this algorithm to graphs gives the modular decomposition tree of a graphs. Let $\mathcal{M}(G, v)$ denote the collection of modules that contain $\{v\}$.

1. MaxModules(G, v) is a partition of V in which every partition class is a module. Therefore, the modular quotient $G' = G/\text{MaxModules}(G, v)$ is defined.

2. Since this quotient reduces all modules that don't contain v to points, all nontrivial modules contain $\{v\}$ in G. Therefore, all nontrivial modules of G' are members of $\mathcal{M}(G', \{v\})$.

3. If two members of $\mathcal{M}(G', \{v\})$ overlap, then their symmetric difference is a nontrivial module of G' that does not contain $\{v\}$, a contradiction. Therefore, the members of $\mathcal{M}(G', \{v\})$ are all strong, hence exactly the set of ancestors of $\{v\}$ in

the modular decomposition of G'.

4. By the quotient rule, the inverse images of members of $\mathcal{M}(G', \{v\})$ in G are exactly the ancestors of v in G. This observation reduces the problem of finding the ancestors of v in G to that of finding $\mathcal{M}(G', \{v\})$.

5. By the definition of MaxModules(G, v), each missing subtree of the modular decomposition of G is contained a member of MaxModules(G, v), so the rest of the modular decomposition tree can be found by recursion on the subgraphs induced by members of MaxModules(G, v).

6.4.1 Maximal modules that do not contain the vertex v

Definition 6.4.2. *Let X be a module of a graph G. Then X is a maximal module excluding v if and only if $v \notin X$ and for each module $Y \in G$ such that $X \subset Y$, $v \in Y$.*

Lemma 6.4.3. *[15] Let $G = (V, E)$ be a hypergraph then MaxModules(G, v) forms a partition of the set $V - v$.*

Proof. In order to prove the lemma it needs to be shown that the members are disjoint and that their union forms the set $V - v$. If $X \in$ MaxModules(G, v) then there can be no other module in MaxModules(G, v) that contains X and does not contain v. Let X and Y be members of Maxmodules(G, v) that do not contain v. By the definition of MaxModules, neither is a subset of the other. Therefore, they are either disjoint or they properly overlap. Suppose they properly overlap. By the properties of overlapping modules given by Theorem 4.2.1, $X \cup Y$ is a module. Additionally it can be seen that $X \cup Y$ does not contain v. This is a contradiction to X and Y being maximal modules. We conclude that they are disjoint.

Every singleton vertex in $V - v$ is a maximal module that does not contain v hence a subset of some member of MaxModules(G, v). MaxModules(G, v) forms a partition of the set $V - v$. □

6.4.2 An Algorithm to compute the decomposition tree from a set of modules that contain v

A single call to MaxModules(G, v) will result in a congruence partition. Let $G = G/\mathcal{Q}$, that is G' is the quotient graph.

Theorem 6.4.4. *In G' all modules are strong and contain v.*

Proof. The sets in $\mathcal{Q}$ are vertices in G'. The quotient reduced all modules that do not contain v to vertices in G'. Hence any non-trivial module in G' will contain v. Suppose X and Y properly overlap in G', then by overlap Lemma 4.2.1 $X \Delta Y$ is a module in G'. $X \Delta Y$ is a non-trivial module in G' that does not contain v, which is a contradiction. □

A call to the procedure $\mathcal{M}(G', v)$ will give the decomposition tree $\mathcal{T}'$. Computing modular decomposition of each set $X \in \mathcal{Q}$ recursively will then give the required tree $\mathcal{T}$.

6.4.3 Forcing relation in graphs to find $\mathcal{M}(G, v)$ in an undirected graph

Let $G = (V, E)$ be an undirected graph, to find $\mathcal{M}(G, v)$ for a vertex $v \in V$ a forcing relation technique has been used. Consider a new directed graph $F(G, v) = (V', E')$ represent the directed forcing graph. The vertices of F are all the vertices in G, *i.e* $V' = V$.

Definition 6.4.5. *A vertex z distinguishing a set $\{w, x\}$ in G means that exactly one of the following two edges $\{(w, z), (x, z)\}$ exists in E.*

If x and z are vertices in V' and if z distinguishes $\{v, x\}$ in G then $(z, x) \in E'$. The directed edge indicates that any module that contains x and v should contain the vertex z.

Lemma 6.4.6. *If z distinguishes a set X, then we can find $Y \subseteq X$ of size two that z distinguishes.*

Proof. If z distinguishes X then it has edges to a proper subset of X, let Y' represent this proper subset. Construct Y such that it has one vertex from Y' and one vertex from $X - Y'$. It is clearly seen that Y is the required set of size two that z distinguishes. □

Lemma 6.4.7. *If z distinguishes a pair $\{w, x\}$ then it distinguishes exactly one of $\{v, x\}$ or $\{v, w\}$*

Proof. If z distinguishes $\{x, w\}$ in G then z has an edge to exactly one of x and w. Without loss of generality suppose it has an edge with x, *i.e* $(z, x) \in E$.

1. *If $(z, v) \in E$ then z distinguishes the pair $\{v, w\}$*

2. *If $(z, v) \notin E$ then x distinguishes the pair $\{v, x\}$*

□

Definition 6.4.8. *For $X' \in F(G, v)$, let X represent the union of singleton vertices in X'.*

Lemma 6.4.9. *X is a module containing v in G if and only if X' has no incoming edges in $F(G, v)$*

Proof. Suppose X' has an incoming edge, this means that there is a vertex $z \in V - X$ that distinguishes a pair $\{v, x\}$ where $v, x \in X$. That is z has an edge to exactly one of v and x in G. $\{v, x\} \subseteq X$ and $z \in V - X$ this implies X is not a module in G.

Suppose X contains v but is not a module in G this means there is a vertex $z \in V - X$ such that z distinguishes a pair $\{v, x\} \in X$ [Lemma 6.4.6, Lemma 6.4.7]. This implies $(z, x) \in E'$. Since $x \in X'$ and $z \in V' - X'$ this means there is an incoming edge to the set X', a contradiction. □

Corollary 6.4.10. *If all the modules containing v are strong then the strongly connected components of $F(G, v)$ has a unique topological sort.*

6.5 Ehrenfeucht et al's Recursive Algorithm

[17] elucidates that if there exist modules that do not contain a vertex v, then they are among the neighbor vertices of v or among the non-neighbor vertices of v. Let $N(v)$ be the subgraph induced by the neighbors of the vertex v and let $\bar{N}(v)$ denote the subgraph induced by the non-neighbors of the vertex v. The algorithm recurses on the size of

the graph, suppose $\mathcal{T}_1$ and $\mathcal{T}_2$ denote the modular decomposition tree of the subgraphs $N(v)$ and $\bar{N}(v)$ respectively. The maximal modules of G that do not contain v can be obtained from the nodes of the two trees and the edges that go between the two sets. To find the modules that contain the vertex v the algorithm uses a equivalence relation on the nodes of the subgraphs and defines blocks on the subgraph. Blocks are basically the smallest sets in the partition that can combine to form module with v. The algorithm then uses a forcing relation (Section6.4.3) to find which blocks force others in order to find the nested $\mathcal{M}(G, v)$.

The insight for finding modules that contain the vertex v is if a subset of vertices in $N(v)$ form a module with v then these nodes should have the same relation as v to other nodes in $N(v)$. Since v is neighbor of all vertices in $N(v)$, they should all be neighbors of the vertices in $N(v)$. Similarly for a subset of vertices in $\bar{N}(v)$ to be in a module with v they should not have an edge to any other vertex in $\bar{N}(v)$ A block in $\mathcal{B} \in N(v)$ forces a block in $\mathcal{B}' \in \bar{N}(v)$ if there is at least one edge between the two. Similarly $\mathcal{B}'$ forces $\mathcal{B}$ if there is at least one non-edge between the two blocks. The reason behind this is similar, if they are contained in a module that contain v then they should have the same relation to vertices in $N(v)$ and $\bar{N}(v)$.

Chapter 7

Modules in a k-Hypergraph

In graphs (2-hypergraphs) two vertices are neighbors if there is an edge that contains the two vertices. In k-hypergraphs since the hyperedges can be of size greater than two the idea of undirected edges extends to sets of size greater than two. The first part of this section defines neighbors and non-neighbors in k-hypergraphs. The second part of this section presents the generalization of the notion of modules in a graph as defined in [6] and shows that the family of modules in a k-hypergraph is also a partitive set family.

Definition 7.0.1. *A set $Y \subseteq V - X$ is said to be a **neighbor** of a set X in H if and only if for every non-empty set $S \subseteq X$ of size lesser than or equal to $k - |Y|$, set $Y \cup S \in E$. A set $Y \subseteq V - X$, is said to be a **non-neighbor** of a set X in H, if and only if for every non-empty subset $S \subseteq X$ the set $Y \cup S \notin E$. A set $Y \subseteq V - X$ is said to be a **spoiler** for a set X, if Y is neither a neighbor nor a non-neighbor of the set X.*

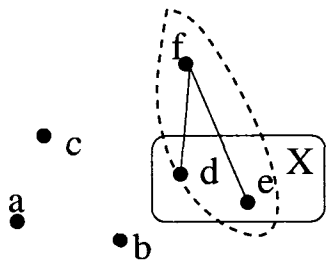

Figure 7.1: Vertex f is a neighbor of the set $X = \{d, e\}$ since it has all possible edges with X. The sets $\{\{f, d\}, \{f, e\}, \{f, d, e\}\}$ exist as edges. If none of these sets exists as an edge then f is a non-neighbor

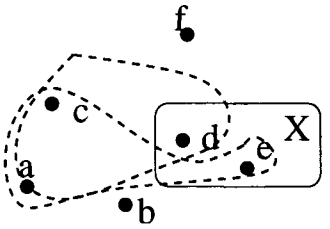

Figure 7.2: The sets of the form $\{\{a, c, d\}, \{a, c, e\}\}$ exists as edges, hence $\{a, c\}$ is a neighbor of the set $\{d, e\}$. If both these sets did not exist as edges then the set $\{a, c\}$ is a non-neighbor of $\{d, e\}$.

For any set X, $Y \subseteq V - X$ of size greater than or equal to k is a non-neighbor of X. If a set Y is a neighbor of a set X, then there can be non-empty subsets of Y that are neighbors or non-neighbors of X.

Definition 7.0.2. *A set $Y \subseteq V - X$ is said to be a **j-neighbor** of a set X if for all $Z \subseteq X$, size of Z less than or equal to j, $Y \cup Z \in E$. In other words, if for every subset of X of size lesser than or equal to j, Y has an edge with that subset.*

A set $Y \subseteq V - X$ is a 2-neighbor of a set X if $Y \cup \{u, w\} \in E$ for all pairs of vertices $u, w \in X$. In other words, Y has edges with all possible pairs of vertices in X. An

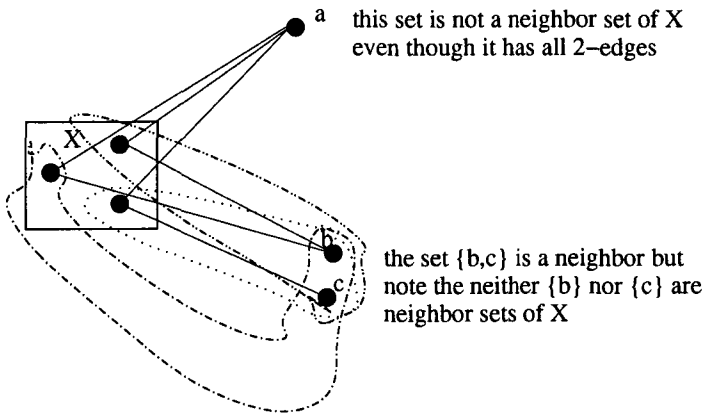

Figure 7.3: Neighbor and Non-neighbor sets in a k-hypergraph

equivalent definition for a set Y to be a neighbor of a set X is that Y is a j-neighbor of X for all $j = 1 \ldots k - |Y|$.

Definition 7.0.3. *[6] A set $X \subseteq V$ is said to be a* ***module*** *if for every subset $Y \subseteq V - X$, Y is either a neighbor of X or a non-neighbor of X.*

Note that only sets Y of size less than k need to be considered.

Observation 7.0.4. *In a k-hypergraph any set of size greater than or equal to k in $V - X$ cannot be a spoiler set for the set X.*

Let $\mathcal{T}$ denote the modular decomposition tree of the k-hypergraph $H = (V, E)$. An algorithm to find the modular decomposition tree of k-hypergraphs exists [6]. The theorems that follow have been proved by Bonizzoni and Della Vedova [6]. They are restated here only for contextual review.

Definition 7.0.5. *Let X and Y be disjoint subsets of H. X and Y are* ***adjacent*** *if for any non-empty subset $Z \subseteq X$, $W \subseteq Y$; if size of $W \cup Z$ does not exceed k then $W \cup Z$ is an edge of H.*

Definition 7.0.6. *Let X and Y be disjoint subsets of H. X and Y are* ***non-adjacent*** *if there is no non-empty subset $Z \subseteq X$, $W \subseteq Y$ such that $W \cup Z$ is an edge of H.*

Theorem 7.0.7. ***[Existence of quotient]*** *If X and Y are disjoint modules in H then either X is adjacent to Y or X is non-adjacent to Y.*

Proof. If no subset of X has an edge with any subset of Y then X and Y are non-neighbors. Let $X_1 \subseteq X$ have an edge with the set $Y_1 \subseteq Y$. Since Y is a module, X_1 should have edges with all possible subsets of Y. If $Y_2 \subseteq Y$ is such a set, then since Y_2 is outside X and since it has an edge with a particular subset in X, Y_2 should have edges with all subsets of X. □

If $\mathcal{Q}$ is a congruence partition of H, then $H/\mathcal{Q}$ denotes the quotient k-hypergraph. Vertices of the quotient hypergraph are members of $\mathcal{Q}$ and the edges represent the adjacency relation between them. The adjacency relation is well defined by Theorem 7.0.7.

Theorem 7.0.8. *If $\mathcal{Q}$ is a congruence partition of H and if S is a system of representatives from each set in the partition then $H/\mathcal{Q}$ is isomorphic to H/S.*

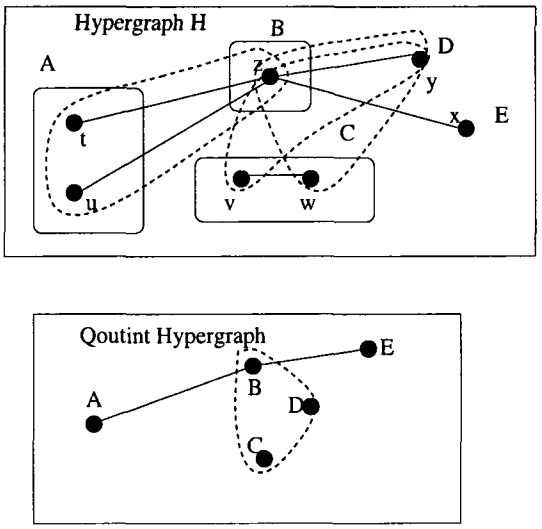

Figure 7.4: Quotient in a k-hypergraph

Observe that the Theorems 3.0.6 and 3.0.7 for graphs generalize to k-hypergraphs.

Theorem 7.0.9. *[**Quotient rule in k-hypergraphs**] If $\mathcal{Q}$ is a congruence partition of a k-hypergraph H, then the union of sets in $\mathcal{Q}$ is a module in H if and only if the corresponding set of nodes of $H/\mathcal{Q}$ is a module in $H/\mathcal{Q}$*

Theorem 7.0.10. *[**Autonomous rule in k-hypergraphs**] If X is a module in a k-hypergraph H, then the modules of H that are subsets of X are given by the modules of $H|X$. The strong modules of H that are proper subsets of X are given by the strong modules of $H|X$.*

If $\mathcal{F}$ is defined to be a family of modules in a k-hypergraph. For $\mathcal{F}$ to be a partitive set family, $\mathcal{F}$ has to satisfy the properties listed in Definition 4.1.1. It can be seen that V and the singleton vertices of H are trivial modules in any k-hypergraph. Theorem 7.0.11 below concludes that the family of modules in a k-hypergraph is a partitive set family.

Theorem 7.0.11. *[6] [**Overlap Rule in k-hypergraphs**] If X, Y are modules in H and if X and Y overlap then $X \cap Y$, $X \cup Y$, $X - Y$, $Y - X$ and $X \Delta Y$ are also modules in H.*

Chapter 8

Summary of New Result

Definition 8.0.12. *If v is a vertex of H, let **MaxModules**(H, v) denote the set of maximal modules of H that do not contain v. That is, $X \in$ MaxModules(H, v) if X is a module of H that does not contain v, and every proper superset of X that is a module of H contains v.*

Definition 8.0.13. *Let X be a module of a hypergraph H. Then X is a maximal module excluding v if and only if $v \notin X$ and for each module $Y \in H$ such that $X \subset Y$, $v \in Y$.*

Lemma 8.0.14. *[6] Let $H = (V, E)$ be a hypergraph then MaxModules(H, v) forms a partition of the set $V - v$.*

Proof. In order to prove the lemma it needs to be shown that the members are disjoint and that their union forms the set $V - v$. If $X \in$ MaxModules(H, v) then there can be no other module in MaxModules(H, v) that contains X and does not contain v. Let X and Y be members of Maxmodules(H, v) that do not contain v. By the definition of MaxModules, neither is a subset of the other. Therefore, they are either disjoint or they properly overlap. Suppose they properly overlap. By the properties of overlapping modules given by Theorem 7.0.11, $X \cup Y$ is a module. Additionally it can be seen that $X \cup Y$ does not contain v. This is a contradiction to X and Y being maximal modules. We conclude that they are disjoint.

Every singleton vertex in $V - v$ is a maximal module that does not contain v hence a subset of some member of MaxModules(H, v). MaxModules(H, v) forms a partition of

the set $V - v$. □

Bonizzoni and Della Vedova [6] developed an algorithm for modular decomposition that finds MaxModules(H, v) for each $v \in V$, and obtain an O(n^{3k-5}) bound to compute the tree $\mathcal{T}$. Our improvement, which gives an $O(k!n^{k-2}m \log n)$, where m is the number of edges, is obtained with an algorithm that follows a strategy that is somewhat similar to that of Bonizzoni and Della Vedova in that it begins with one call to MaxModules(H,v) for each $v \in V$. The algorithm then takes an inventory of the sets found and infers the modular decomposition of H from them, using the following observations. The children of prime nodes are those sets in the inventory that do not properly overlap with any other, and the children of degenerate nodes are found by intersecting members of each group of properly overlapping sets in the inventory.

The new result of this thesis is a faster algorithm for finding MaxModules(H, v)that gives a bound of $O(k!n^{k-3}m \log n)$. A consequence of this is that the modular decomposition can be found by using Bonizzoni's algorithm, but replacing the MaxModules procedure with the new algorithm. Replacing the procedure used in [6] for finding MaxModules gives a new time bound of $O(k!n^{k-2}m \log n)$ for finding the modular decomposition of a hypergraph, When the hypergraph is sparse, this is clearly superior than the previous bound, which is unaffected by how sparse the hypergraph is. In a dense hypergraph, there are $O(n^k)$ edges, and this bound is $O(k!n^{2k-2} \log n)$,

8.1 A Reduction of Finding $\mathcal{T}$ to Finding Maxmodules(H, v)

In this section, we show how an improvement we obtain for finding MaxModules results in an immediate improvement in the time bound to find the modular decomposition from $O(n^{3k-5})$ to $O(k!n^{k-2}m \log n)$.

Lemma 8.1.1. *[2] [6] Let $\mathcal{T}$ be the decomposition tree of the hypergraph $H = (V, E)$. Let X be a strong module which is a proper ancestor of v in $\mathcal{T}$ and let Y be a child of X that contains v. If X is a degenerate node then the union of all children of X except Y*

will be in MaxModules(H, v). If X is prime then each of the children of X other than Y is a maximal module of H excluding v.

To make sure that finding the modular decomposition from the inventory of sets produced by all calls to Maxmodules(H, v) for all $v \in V$ is not a bottleneck when we improve the time bound for finding Maxmodules(H, v), we derive the following:

Lemma 8.1.2. *Given Maxmodules(H, v) for all $v \in V$, it takes time proportional to the sum of cardinalities of their member sets to find the modular decomposition tree of H.*

Proof. By radix sorting using vertex number as secondary key and set number as primary key, we get each set into the form of a sorted list of vertices. Radix sorting the sets using standard string algorithms for ordering strings in lexical order, we can group identical sets together, and then eliminate duplicate sets in time linear in the sum of their sizes.

Let the **overlap graph** of the inventory be the graph whose vertices are the sets in the inventory, and whose edges are the pairs of properly overlapping sets. Dahlhaus [15] gives an algorithm for finding the connected components of the overlap graph of a family of sets in time linear in the sum of cardinalities of the sets.

Because each no prime node overlaps any other module, each prime node is an isolated vertex in the overlap graph. These are recognized after running Dahlhaus's algorithm. Since the union of all children of a degenerate node except the one that contains v is a member of MaxModules(H, v), if a degenerate node has j children, every union of $(j-1)$ children will appear in the inventory. Since these sets are overlapping, they form a connected component in the overlap graph whose union is the parent, D, which is recognized by Dahlhaus's algorithm. For each union X of $(j-1)$ children, $D - X$ identifies one of the children of the degenerate node. These can all be found using Dahlhaus's algorithm. □

Figure 8.1: MaxModules of a component of a tree

8.2 Better Time Bound for Computing MaxModules(H, v)

Bonizzoni and Della Vedova[6] details a procedure to compute MaxModules(H, v) for a 3-hypergraph and then uses the above procedure to compute the tree $\mathcal{T}$ for a 3-hypergraph. [6] also presents a procedure to compute MaxModules(H, v) for a k-hypergraph in $O(n^{3k-6})$ by recursing on the dimension(k) of the hypergraph. With n calls to this procedure $\mathcal{T}$ for a k-hypergraph can be computed. The overall run time to compute $\mathcal{T}$ for a k-hypergraph is $O(n^{3k-5})$. The first main result of this thesis is a better time bound to compute $\mathcal{T}$ in a k-hypergraph. The proof of the Theorem 8.2.1 will be discussed in the next chapter.

Theorem 8.2.1. *For any vertex v in k-hypergraph H, there exists an $O(k!n^{k-3}m \log n)$*

Figure 8.2: A component of the overlap graph. The connected components corresponds to children of a degenerate node. The isolated nodes of the overlap graph are the prime nodes in the tree

to compute MaxModules(H, v) .

Corollary 8.2.2. *It takes $O(k!n^{k-2}m \log n)$ time to compute $\mathcal{T}$ for a k-hypergraph.*

Proof. Multiplying the time for finding Maxmodules(H, v) by the n choices of v gives the total time for all calls to MaxModules. The sum of cardinalities of the sets they produce cannot be greater than this time. The result then follows from Lemma 8.1.2. □

Chapter 9

New bounds for finding the modular decomposition of a hypergraph

With k being the dimension and n the number of vertices, the run time for computing the modular decomposition tree in Bonizonni and Della Vedova's paper [6] is $O(n^{3k-5})$. In this chapter, I give an algorithm that has a new time bound of $O(k!n^{k-2}m\log n)$, where m is the number of edges. In the case of a sparse hypergraph, this is clearly superior than the bound in [6]. In a dense hypergraph, where there are $\Theta(n^k)$ edges, this algorithm runs in $O(k!n^{2k-2}\log n)$ time, which is still better than $O(n^{3k-5})$ for $k \geq 4$.

In the previous chapter, we have seen that given MaxModules for all vertices v, it is possible to find the modular decomposition tree of the hypergraph. Bonizonni and Della Vedova reduced the problem to finding MaxModules(H, v) i.e. the maximal modules that do not contain a vertex v. The key is an improvement in the time bound for finding MaxModules from $O(n^{3k-6})$ to $O(k!n^{k-3}m\log n)$.

9.1 A strategy for finding the maximal modules that do not contain a vertex

Definition 9.1.1. *A **refinement** of a partition $\mathcal{Q}$ is the result of selecting zero or more partition classes, and for each selected class X, replacing $X \in \mathcal{Q}$ with the members of a partition of X.*

The strategy for modules that do not contain v is to start off with a partition $\mathcal{Q} = \{\{v\}, V - v\}$, observe that the modules that do not contain v have to be contained in the partition class $V - v$. I refine the partition by splitting classes, while maintaining the **integrity invariant** that each module not containing v is inside a partition class. I continue using the strategy that allows further refinement as long as some partition classes are not modules, so I halt when all partition classes are modules. The invariant at this point implies that all partition classes other than $\{v\}$ are the maximal modules that don't contain v, namely, $MaxModules(H, v)$.

Definition 9.1.2. *Let H=(V,E) be an arbitrary hypergraph and let $u \in V$. Let $H_u = (V, E')$ denote the subgraph where E' is the subset of edges of H that are incident to u. A set $X \subseteq V - u$ is* **split** *by u if X fails to be a module in H_u.*

Definition 9.1.3. *The degree of a vertex u in a k-hypergraph H, denoted $d(u)$, is the number of edges incident to the vertex u.*

Definition 9.1.4. *An edge e is said to* ***straddle*** *a set $X \subseteq V - X$ if e intersects both X and $V - X$.*

Lemma 9.1.5. *A set $X \subseteq V$ is a module of H if and only if it is not split by any vertex in $V - X$.*

Proof. If X is not a module of H, then there exist $U \subseteq V - X$, and $Y, Z \subseteq X$ of size at most $k - |U|$ such that $U \cup Y$ is an edge and $U \cup Z$ is not. For any $u \in U$, X fails to be a module in H_u.

Suppose X is a module of H. Let $u \in V - X$, $U \subseteq V - X$ such that $u \in U$, and $Y, Z \subseteq X$ such that Y and Z are of size at most $k - |U|$. Either both of $U \cup Y$ and $U \cup Z$ are edge of H or neither is, since X is a module. Since both of these sets contain u, they they are either both edges of H_u or neither is. Since U, Y, Z are arbitrary, X is a module of H_u. □

Definition 9.1.6. *Let $\mathcal{Q}$ be a partition of V. By $\Pi(u, Q)$, let us denote the refinement*

of $\mathcal{Q}$ consisting of the class C that contains u and the maximal modules of H_u that are subsets of $\mathcal{Q} - C$.

Lemma 9.1.7. *If $\mathcal{Q}$ satisfies the integrity invariant then so does $\Pi(u, \mathcal{Q})$.*

Proof. Immediate from Lemma 9.1.5. □

Moreover if u splits a member of $\mathcal{Q}$ then $\Pi(u, \mathcal{Q})$ is a proper refinement. This gives us a strategy for finding MaxModules if $\mathcal{Q}$ contains classes split by u, then replace $\mathcal{Q}$ by $\Pi(u, \mathcal{Q})$. $\Pi(u, \mathcal{Q})$ is a proper refinement of $\mathcal{Q}$, and the procedure must halt by the time $\mathcal{Q}$ has n members. When it halts, no class is split by any vertex, so every class is a module by lemma 9.1.5. The integrity invariant implies that $\mathcal{Q}$ consists of $\{v\}$ and the MaxModules(H, v).

For the sparse model we maintain a list of incident edges with every vertex in the hypergraph. The classes in the partition $\mathcal{Q}$ are maintained as a collection of doubly linked lists. Each vertex points to the head of the list it is contained in. When two vertices in the list point to the same head we know that these two vertices belong to the same class in the partition.

Lemma 9.1.8. *[23] Let $S \subseteq V$ and let $r = |S|$. Suppose that every member of $\mathcal{Q}$ is labeled with its cardinality. It takes $O(r)$ time to replace all Y in $\mathcal{Q}$ with $Y \cap S$ and $Y - S$, label these two classes with their cardinalities, and eliminate any of these sets that are empty.*

Proof. For each class Y that contains members of S, we create a twin class that can store the common members between Y and S. The number of members in $\mathcal{Q}$ may not be $O(r)$, so we need to be careful not to touch classes that don't intersect with S. The first time a class Y is accessed, we move the class to a collection of touched classes and create a twin class that receives the member of $Y \cap S$. If a vertex $w \in S$ belongs to class Y in the partition then move w to Y's twin class. When we are done, we make the twin a new member of $\mathcal{Q}$, and retain the diminished class $Y - S$ as a member unless it is empty.

In this procedure we spend O(1) time for each vertex in S. Hence the new partition can be computed in O(r) time. With every class $Y \in \mathcal{Q}$, we maintain its cardinality. When we remove $Y \cap S$ element we update the cardinality of $Y - S$ and $Y \cap S$ in O(1) time. Hence the cardinality of the classes is also updated in O(r) time. □

Definition 9.1.9. *Let $H = (V, E)$ be a hypergraph of dimension k and let $X \subseteq V$. Then X is a* ***clique*** *of H if and only if every subset of X of size at most k is an edge of E. A set $X \subseteq V$, is called an* ***independent*** *set of H if and only if no subset of X is a hyperedge of H.*

9.2 Finding the modular decomposition of a 3-hypergraph

Let $H = (V, E)$ be a 3-hypergraph. Let $\mathcal{T}$ denote the modular decomposition tree of H. If the hypergraph is disconnected then the root will be a degenerate node whose children are the connected components of H. Given a connected 3-hypergraph with n' vertices and m' edges the algorithm described in this section computes its tree in $O(n'm' \log n')$. The tree $\mathcal{T}$ of H can be obtained from the trees of the different components by making their roots children of a new degenerate node, in $O(n)$ time, for a total of $O(n+nm \log n)$ time. We assume that $m > 0$, so this is $O(nm \log n)$.

To find the connected components of the 3-hypergraph a graph $G' = (V, E')$ is constructed. G' contains all the vertices in V and all the 2-edges in E. For each 3-edge $(x_1, x_2, x_3) \in E$, we add the 2-edges $(x_1, x_2), (x_2, x_3)$ to E'. The connected component of G' can be found by running the depth first search algorithm on G'. These connected components will be the connected components of the hypergraph H.

We have reduced the problem to that of finding the modular decomposition of a connected hypergraph, so henceforth, we assume that $n = O(m)$.

The following definition of underlying graph is related to that defined in Bonizonni and Della Vedova [6].

Definition 9.2.1. *Let $H = (V, E)$ be a 3-hypergraph, and let $\mathcal{Q}$ be a partition of V. Let u be a vertex of H, with $u \in C$, for $C \in \mathcal{Q}$. The underlying graph of H with respect to u*

and $\mathcal{Q}$, denoted as $G_H(u, \mathcal{Q})$ is the subgraph consisting of those edges incident to u and straddling C, and those vertices incident to these edges.

Definition 9.2.2. *Let $H = (V, E)$ be a 3-hypergraph, and let $\mathcal{Q}$ be a partition of V. Let u be a vertex of H, with $u \in C$, for $C \in \mathcal{Q}$. Let $W = \{w : (u, w) \in E \text{ and } w \notin C\}$.*

Lemma 9.2.3. *[6] Let $H = (V, E)$ be a 3-hypergraph, let $\mathcal{Q}$ be a partition of V and $u \in V$. Let C be the class of $\mathcal{Q}$ to which u belongs. Let X be a subset of V that is disjoint from C. Then X is a module in H_u if and only if X is a module of the underlying graph $G_H(u, \mathcal{Q})$ and either :*

- *X is a clique of $G_H(u, \mathcal{Q})$ and each vertex $x \in X$ is a member of W, or*
- *X is an independent set in $G_H(u, \mathcal{Q})$ and no vertex $x \in X$ is a member of W*

The following procedure, which we will call a *pivot on vertex u*, computes $\Pi(u, \mathcal{Q})$ by lemma 9.2.3:

VertexPivot$(u, \mathcal{Q})$

Input : A vertex u of the hypergraph $H = (V, E)$ of dimension three and a partition $\mathcal{Q}$ of vertices V

Let C denote the class in $\mathcal{Q}$ that u belongs to.

Let $\mathcal{T}$ denote the modular decomposition tree of $G_H(u, \mathcal{Q})$.

Mark all the vertices that belong to W.

Let $\mathcal{P}$ be the finest possible partition of the vertices of $G_H(u, \mathcal{Q})$, with each vertex in a one-element partition class.

During a postorder traversal of the tree $\mathcal{T}$ merging into a single in $\mathcal{P}$ all those leaf children that satisfy Lemma 9.2.3.

If X is a maximal module in H_u and for each class $Y \in \mathcal{Q} - C$ that contains vertices of X, replace Y with classes $Y \cap X$ and $Y - X$. Eliminate $Y - X$ from Q if it is empty.

Unmark vertices that occur in W

Return $\mathcal{Q}$

After a VertexPivot$(u, \mathcal{Q})$ operation, except for the class C containing u, each class is a maximal module in the subgraph of edges incident to u and contained in a single class before the pivot. Therefore, the procedure returns $\Pi(u, \mathcal{Q})$.

Lemma 9.2.4. *The run time of VertexPivot$(u, \mathcal{Q})$ is $O(d(u))$.*

Proof. The VertexPivot procedure starts with finding the edges incident to u that straddle the class C. From the straddlers, $G_H(u, \mathcal{Q})$ and W can be constructed. Subsequently, the decomposition tree of the underlying graph needs to be found. From the decomposition tree, the modules in the sub-hypergraph H_u are found.
Straddlers are recognized as those edges that have at least one vertex pointing to a head of a doubly linked list other than the one that u points to. Each edge incident to u is of size $O(1)$, so, it takes $O(d(u))$ time to scan the edges incident to u and to construct the graph $G_H(u, \mathcal{Q})$ and the set W.

There are at most $d(u)$ edges in the underlying graph. There are at most two vertices for every edge in the graph hence the size of the underlying graph if $O(d(u))$.

By McConnell and Spinrad's algorithm [30], the time taken to compute the modular decomposition tree of the graph is linear in its size. For a graph with n' vertices and m' edges the run time is $O(n' + m')$. Hence for computing the decomposition tree for $G_H(u, \mathcal{Q})$, the time taken is $O(d(u))$.

The postorder traversal takes time proportional to the number of nodes of the tree, which is at most twice the number of leaves, which are vertices of $G_H(u, Q)$, which are $O(d(u))$. Therefore, the number of nodes in the tree is $O(d(u))$. The maximal modules X of H_u obtained from the traversal of the tree are then used to compute $\Pi(u, \mathcal{Q})$. For each module X of H_u, the time taken for refinement of the partition is $O(|X|)$ (lemma 9.1.8). Since the sum of sizes of the vertices in such sets X does not

exceed $O(d(u))$, the run time for this last part of the algorithm is also bounded by $O(d(u))$.

Hence the run time for the procedure VertexPivot$(u, \mathcal{Q})$ is $O(d(u))$. □

The key to my new time bound for finding Maxmodules is the use of a sparse representation along with my strategy for choosing the sequence of pivots. Sometimes, we may have to pivot more than once on a vertex u. The first time, u is in a class C and the pivot splits classes that are subsets of $V - C$. The pivot does not split C, since u is internal to it. Later, C may be split by a pivot on a vertex outside of C into C_1 and C_2, where C_1 contains u. It can now be the case that u splits C_2, forcing us to perform another pivot on u to find out how it splits up C_2.

The goal of my strategy for selecting them is to avoid pivoting more than $\log_2 n$ times on any one vertex. My algorithm proceeds by iteratively selecting a partition class and performing pivots on all of the vertices in the class. It remains to describe how it selects the sequence of classes. I adapt and generalize a procedure given in [30] for transitive orientation of comparability graphs. Let us say that a partition class X is **ripe** if it is at most half of the size of the partition class that contained its members the last time they were used as pivots.

Lemma 9.2.5. *When there are no more ripe classes, a largest partition class is a member of MaxModules(H, v).*

Proof. Let us suppose X is a largest partition class in the partition $\mathcal{Q}$. When there are no ripe classes, we need to prove that $X \in$ MaxModules(H, v). Consider a vertex $w \in V - X$. It suffices to show that w has been used as a pivot since the last time it was in a common class with members of X: if this is the case then X must be a module with respect to w since it was not split by a pivot on w, and since w is arbitrary, X is a module with respect to all vertices in $V - X$ and hence a module of H.

Since X is the largest class in the partition, the class containing w is at most as large as X. The last class W' that contained both w and x was at least twice as large.

Therefore, w would have been in a ripe class. Since it was split from W' and is currently not ripe class, w has been used as a pivot. □

Lemma 9.2.6. *If $X \in \mathcal{Q}$ is a member of MaxModules, then after a single pivot on any $u \in X$, no member of the resulting refinement of $\mathcal{Q}$ is split by any vertex in X.*

Proof. Since the class X is a module, the vertices in X have the same adjacency relationship to all the vertices in $V - X$. The result is trivial when X has only one vertex. Let us consider the case when there are at least two vertices.
Suppose we split $\mathcal{Q}$ with a vertex $w \in X$, then the classes $Y \in \Pi(w, \mathcal{Q})$ are maximal modules in H_w. Consider vertices $u, w \in X$, because the vertices in X have the same adjacency relationship the sub-hypergraph $H_w|\{V - X \cup \{w\}\}$ is the same as the sub-hypergraph $H_u|\{V - X \cup \{u\}\}$. Hence u does not split any members of $\Pi(w, \mathcal{Q})$. Since $u \in X$ was arbitrarily chosen, the lemma follows. □

When there are no remaining ripe vertices, my algorithm selects a largest member X of $\mathcal{Q}$, performs a pivot on a member u of X, and then removes X from $\mathcal{Q}$ and inserts it to a list of known members of MaxModules. By the lemma, the remaining members can be found by recursion on the remaining partition $\mathcal{Q} - \{X\}$ of $V - X$. We pivot on the class X before we remove it from the partition as a member of MaxModules(H, v). The new pivot might make a few classes ripe. After a pivot on the largest class X, every other class is a module in the subgraph of edges incident to X. Therefore X can be removed from $\mathcal{Q}$ after a pivot on X and remaining maximal modules found recursively.

Lemma 9.2.7. *Let $H = (V, E)$ be a 3-hypergraph, for any vertex $v \in V$, the run time to compute MaxModules(H, v) is $O(m \log n)$*

Proof. In order to recognize ripe classes, we maintain a label LastPivotSize(X) with each partition class X. LastPivotSize(X) gives the size of the class that X belonged to when it was last pivoted on. When $|X| \leq \frac{1}{2}$ LastPivotSize$(|X|)$, then X is a ripe class. Let Y be a class that gets split into classes $\{Y_1, \ldots Y_r\}$. For each class Y_i, we update LastPivotSize(Y_i)= LastPivotSize(Y).

A list of all the classes that are ripe is maintained. When a class becomes ripe it is added to this list and when a ripe class from this list is pivoted on it is removed from the list. If a ripe class belonging to this class is split, the new split classes replace the existing class. If an unripe class is split, we check current cardinality and for each split class whose size is less than or equal to half of its LastPivotSize, the class is categorized as ripe.

The operation VertexPivot$(u, \mathcal{Q})$ is called only when the class that u belongs to is ripe. Each time a vertex is used as a pivot, it is in a partition class that is at most half the size as the one that contained it the previous time it was used as a pivot. It can be used at most $\log_2 n$ times as a pivot. Each pivot on a vertex u takes $O(d(u))$ time, so in a call to MaxModule, the time to perform all pivots is $O(m \log n)$.

Consider Hasse diagram of the containment relation of a set of n elements, this is a tree T with n leaves where each internal node has at least two children. There are $O(n)$ nodes in this tree.

When the list of ripe classes is empty the largest class can be found by extracting a largest set from the heap, and when a class splits, it can be deleted from the heap and the classes it split into can be inserted into it. Each node of the Hasse diagram is inserted exactly once and is removed by an extract-max or deletion. Each of these operations takes $O(\log n)$ time on a heap. Since there are $O(n)$ of them, the heap operations take a total of $O(n \log n)$ time. run time of MaxModules$(H, v) = O(n \log n + m \log n)$. For a connected 3-hypergraph $n = O(m)$, so the time to compute MaxModules(H, v) is $O(m \log n)$. □

The run time of MaxModules(H, v) in the dense case is $O(n^3 \log n)$. In Bonizonni and Della Vedova's paper the run time is $O(n^3)$, hence the new bound is better when the graph is sparse.

Lemma 9.2.8. *The modular decomposition tree $\mathcal{T}$ of a 3-hypergraph is $O(nm \log n)$.*

Proof. The run time to compute MaxModules for all vertices $v \in V$ is $O(nm \log n)$. From lemma 8.1.2 we can conclude that the time taken to compute the modular decomposition of a 3-hypegraph is $O(nm \log n)$. □

9.3 Finding the modular decomposition of an arbitrary hypergraph

I want to show that the time taken to construct $\mathcal{T}$ is $O(k!n^{k-2}m \log n)$, for k greater than or equal to three. For the base case $k = 3$; I have already show that the time taken to construct the tree is $O(nmlogn) = O(3!nm \log n)$. Let $H = (V, E)$ be a k-hypergraph (k greater than or equal to four)) . Let $\mathcal{T}$ denote the modular decomposition tree of H. Our induction hypothesis is that the time taken to compute the decomposition tree of any hypergraph of dimension $(k-1)$ is $O((k-1)!n^{k-3}m \log n)$.

The following procedure, which we will again call *pivot on vertex* u, computes $\Pi(u, \mathcal{Q})$ by lemma 9.2.3. Notice that this is the same procedure that was used in 3-hypergraph.

VertexPivot$(u, \mathcal{Q})$

Input : A vertex u of the hypergraph $H = (V, E)$ of dimension k and a partition $\mathcal{Q}$ of vertices V

Let C denote the class in $\mathcal{Q}$ that u belongs to.

Let $\mathcal{T}$ denote the modular decomposition tree of the hypergraph $G_H(u, \mathcal{Q})$ which is of dimension $k-1$.

Mark all the vertices that belong to W.

Let $\mathcal{Q}$ be the finest possible partition of the vertices of $G_H(u, \mathcal{Q})$, with each vertex in a one-element partition class.

During a postorder traversal, at each degenerate node of the tree $\mathcal{T}$, merge into a single set in $\mathcal{Q}$ all those leaf children that satisfy Lemma 9.2.3.

If X is a maximal module in H_u and for each class $Y \in \mathcal{Q} - C$ that contains vertices of

X, replace Y with classes $Y \cap X$ and $Y - X$. Eliminate $Y - X$ from Q if it is empty. Unmark vertices that occur in W

Return $\mathcal{Q}$

Lemma 9.3.1. *The run time over all calls to VertexPivot with u as a parameter is $O((k-1)!n^{k-3}d(u)\log n)$.*

Proof. To compute MaxModules(H, v) for each vertex v there are several calls to VertexPivot with vertex u as parameter. In a call to VertexPivot, time is spent on finding the straddlers incident to u in the given partition. Given straddlers, the decomposition tree of the underlying hypergraph is used to compute $\Pi(u, \mathcal{Q})$. These procedures are repeated every time a vertex u is pivoted on.

Each edge incident to u is of size $O(k)$, so, it takes $O(kd(u))$ time to scan the edges incident to u and compute the straddlers of u in a given partition. The proof of the ripe set rule lemmas ?? is general to k-hypergraphs, hence a vertex u is chosen to be pivoted on at most $\log n$ times. Hence the run time to compute straddlers over all calls to VertexPivot with u as parameter is $O(kd(u)\log n)$.

Suppose $d'(u)$ are the number of straddling edges in a call to VertexPivot. There are at most $d'(u)$ edges in the underlying graph. The number of vertices in G_H cannot exceed n the number of vertices in H. From our induction hypothesis, the time taken to compute the modular decomposition tree of the $(k-1)$-hypergraph with n' vertices and m' edges is $O((k-1)!n^{k-3}m'\log n')$. Hence for computing the decomposition tree for $G_H(u, \mathcal{Q})$, the time taken is $O(k-1)!n^{k-3}d'(u)\log n)$. The postorder traversal takes time proportional to the number of nodes of the tree, which is at most twice the number of leaves, which are vertices of $G_H(u, \mathcal{Q})$, which are $O(n)$. Therefore, the number of nodes in the tree is $O(n)$. The maximal modules X of H_u obtained from the traversal of the tree are then used to compute $\Pi(u, \mathcal{Q})$. For each module X of H_u, the time taken for refinement of the partition is $O(|X|)$ (lemma 9.1.8). Since the sum of sizes of the vertices in such sets X does not exceed $O(n)$, the run time for this last part of the algorithm is bounded by $O(n)$. The run time of one call to VertexPivot with

$d'(u)$ edges is $O((k-1)!n^{k-3}d'(u)\log n)$. An edge that straddles the set C and is incident to u will not play a role in any other calls to VertexPivot with u as parameter. Hence the run time over all calls to procedure VertexPivot with u as parameter is $O(kd(u)\log n + (k-1)!n^{k-3}d(u)\log n) = O((k-1)!n^{k-3}d(u)\log n)$. □

When $k = 3$ the time taken to compute the straddlers was $O(d(u)\log n)$ while that to compute the the modules in the underlying graph was $O(d(u))$. For $k > 3$ the time taken to compute the straddlers is bounded by the time taken to compute the modules in the underlying it is $O(kd(u)\log n)$ in comparison with $O((k-1)!n^{k-3}d(u)\log n)$.

Lemma 9.3.2. *Let $H = (V, E)$ be a k-hypergraph, for any vertex $v \in V$, the run time to compute MaxModules(H, v) is $O(k!n^{k-3}m\log n)$*

Proof. An edge in the hypergraph is incident to at most k vertices, hence the run time over all calls to VertexPivot in a call to MaxModules is $O(k!n^{k-3}m\log n)$. The time taken to compute the largest set in a call to MaxModules is $O(n\log n)$ using the same heap data structure that was used in the case of 3-hypergraph. Hence the run time for a call to MaxModules is $O(k!n^{k-3}m\log n)$. □

Theorem 9.3.3. *Time taken to compute the decomposition tree of a k-hypergraph, where $k \geq 3$ is $O(k!n^{k-2}m\log n)$*

Proof. The run time to compute MaxModules for all vertices $v \in V$ is $O(k!n^{k-2}m\log n)$. From lemma 8.1.2 we can conclude that the time taken to compute the modular decomposition of a k-hypegraph is $O(k!n^{k-2}m\log n)$. □

Chapter 10

Conclusions and Future Work

I believe that I can improve the run time further by using the new strategy based on the divide and conquer algorithm [2]. In chapter six we have seen the description of the divide and conquer algorithm in [2] specific to the case of graphs. The algorithm runs in $O(n^2)$ where n is the number of nodes in a two-structure. The following observation gives a further generalization of that procedure to k-hypergraphs. MaxModules(H, v) is a partition of V (by Lemma 8.0.14) in which every partition class is a module. Therefore, the modular quotient $H' = H/\text{MaxModules}(H, v)$ is well defined in hypergraphs. In this quotient all modules that do not contain v are reduced to points, hence all its non-trivial modules contain $\{v\}$ in H'. This implies all the non-trivial modules of H' are members of $\mathcal{M}(H', \{v\})$.

Lemma 10.0.4. *The members of $\mathcal{M}(H', \{v\})$ are all strong, hence exactly the set of ancestors of $\{v\}$ in the modular decomposition of H'.*

Proof. If two members of $\mathcal{M}(H', \{v\})$ overlap, then their symmetric difference is a non-trivial module of H' that does not contain $\{v\}$, a contradiction. □

By the quotient rule, Lemma 7.0.9 the inverse images of members of $\mathcal{M}(H', \{v\})$ in H are exactly the ancestors of v in H. This observation reduces the problem of finding the ancestors of v in H to that of finding $\mathcal{M}(H', \{v\})$. From the definition of MaxModules(H, v), each missing subtree of the modular decomposition of H is contained in a member of MaxModules(H, v), so the rest of the modular decomposition

tree can be found by recursion on the subgraphs induced by members of MaxModules(H, v).

The advantage of this is sparse use of the MaxModules procedure, since it requires only a single call to MaxModules(H, v) before generating recursive calls, instead of n calls to it. Let $\mathcal{Q}$ denote the sets in MaxModules(H, v). From Lemma 8.0.14 $\mathcal{Q}$ is a congruence partition of H implying that $H/\mathcal{Q}$ (Theorem 7.0.8) is defined. Theorem 7.0.9 shows that all the modules of the quotient graph $H/\mathcal{Q}$ are strong and contain $\{v\}$(Theorem 10.0.4).

The main new obstacle the new approach poses is the development of an efficient procedure for finding $\mathcal{M}(H, \{v\})$, which I conjecture as part of the thesis.

There are known methods to compute the $\mathcal{M}(H, \{v\})$ in the case of a 2-hypergraph. Here I conjecture an algorithm to compute $\mathcal{M}(H, \{v\})$ in the case of a 3-hypergraph. In Bonizonni's paper [6] the time taken to compute the modular decomposition of a 3-hypergraph is $O(n^4)$. Applying this new conjecture if it is true to compute $\mathcal{T}$ improves Bonizonni's time bound by a factor of n.

Conjecture 10.0.5. *Computing $\mathcal{M}(H, \{v\})$ for a vertex v in a 3-hypergraph takes $O(n^3)$ time.*

A corollary to this would be that the time taken to compute modular decomposition tree for a 3-hypergraph is $O(n^3)$. We have seen in the previous chapter that the time taken to compute MaxModules(H, v) is $O(n^3)$. Let $\mathcal{Q}$ represent the congruence partition MaxModules(H, v). Let $H' = H/\mathcal{Q}$ represent the quotient hypergraph. Let n' represent the number of vertices in H'. If Conjecture 10.0.5 is true then we know that the time taken to compute the decomposition tree of H' is $O(n'^3)$. The remaining part of the algorithm will be recursing on the leaf nodes of the tree $\mathcal{T}'$. Since the sum of the size of vertices in each of the leaf nodes does not exceed n the run time to compute the decomposition tree of $\mathcal{T}$ is $O(n^3)$.

This is the starting point for k-hypergraphs. The whole strategy for finding $\mathcal{M}(H, v)$ can be generalized to k-hypergraphs and hence the run time for computing the

decomposition tree in k-hypergraphs might be improved further.

REFERENCES

[1] A.Ehrenfeucht and G.Rozenberg. Theory of 2-structures: Part I clans, basic subclasses and morphisms. *Theoretical Computer Science*, 3(70):277–303, 1990.

[2] R.M. McConnell A.Ehrenfeucht, H.N.Gabow and S.J. Sullivan. An $O(n^2)$ divide-and-conquer algorithm for the prime tree decomposition of two-structures and modular decomposition of graphs. *Journal of Algorithms*, 16:283–294, 1994.

[3] M. Ashbacher. A homomorphism theorem for finite graphs. *Proc. Amer. Math. Soc.*, 54:468.

[4] Claude Berge. Les problemes de colorations en theorie des graphs. *Publ. Inst. Statist. Univ. Paris*, 9:123–160, 1960.

[5] A. Blass. Extremal graph theory. *Academic Press*, 1978.

[6] Paola Bonizzoni and Gianluca Della Vedova. An algorithm for the modular decomposition of hypergraphs. *Journal of algorithms*, 1995.

[7] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[8] A. Cournier and M. Habib. A new linear time algorithm for modular decomposition. 19^{th} *International Commoquium CAAP 94*, 787:68–84, 1994.

[9] W. Cunningham. Decomposition of directed graphs. *SIAM J. Algorithms and Discrete Math.*, 3:214–228, 1982.

[10] Kurt Mehlhorn Deiter Kratsch, R.M.McConnell and J.P.Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. *SIAM Journal of Computing.*

[11] D.Fulkerson and O.Gross. Incidence matrices and interval graphs. *Pacific J. Math*, 15:835–855, 1965.

[12] Y.Perl D.G. Corneil and L.Stewart. A linear recognition algorithm for cographs. *SIAM J. Computing*, 14:926–934, 1985.

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

[14] D.Kelly. Comparability graphs. In I.Rival, editor, *Graphs and Order*, pages 3–40. D.Riedal, Boston.

[15] E.Dahlhaus. Parallel algorithms for hierarchical clustering, and applications to split decomposition and partity graph recognition. *Journal of Algorithms*, 36:205–240, 2000.

[16] R.M.McConnell E.Dahlhaus, J.Gustedt. Efficient and practical algorithms for sequential decomposition. *Journal of Algorithms*, 41:360–387, 2001.

[17] A. Ehrenfeucht, H.N. Gabow, R.M. McConnell, and S.J. Sullivan. An $O(n^2)$ divide-and-conquer algorithm for the prime tree decomposition of two-structures and modular decomposition of graphs. *Journal of Algorithms*, 16:283–294, 1994.

[18] Jens Gustedt Elias Dahlhaus and Ross M.McConnell. Partially complemented representation of digraphs. *Discrete Mathematics and Theoretical Computer Science*, 5(1):147–168, 2002.

[19] D. R. Fulkerson. Blocking and anti-blocking pairs of polyhedra. *Math. Program.*, 1:168–194, 1971.

[20] T. Gallai. Transitiv orientierbare graphen. *Acta Math. Acad. Sci.*, 18:25–66, 1967.

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

[22] G.Sabidussi. Graph derivatives. *Math. Z*, 76:385–401, 1961.

[23] M. Habib, R.M. McConnell, C. Paul, and L.Viennot. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234:59–84, 2000.

[24] H.Buer, , and R.Möhring. A fast algorithm for the decomposition of graphs and posets. *Math. Oper. Res.*, 3:170–184, 1983.

[25] R. Kaerkes. Netzplan theorie. *Oper. Res. Verfahren*, 27:1–65.

[26] E.L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Ann. Discrete Math.*, 2:75–90, 1978.

[27] C.G. Lekkerkerker and J.Ch. Boland. Representation of a finite graph by a set of intervals on the real line. *Fund. Math*, 51:45–64, 1962.

[28] R.G.Stanton L.O.James and D.D. Cowan. Graph decomposition for undirected graphs. *3rd South-Eastern Conf. Combinatorics, Graph Theory and Computing*, pages 281–290, 1972.

[29] L. Lovasz. Normal hypergraphs and the perfect graph conjecture. *Disc. Math.*, 2:253–267, 1972.

[30] R. M. McConnell and J. P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.

[31] R.M. McConnell. An $O(n^2)$ incremental algorithm for modular decomposition of graphs and two-structures. *Algorithmica*, 14:209–227, 1995.

[32] R.M. McConnell. Linear time recognition of circular-arc graphs. *IEEE FOCS*, 42:386, 2001.

[33] R.M. McConnell and F. de Montgolfier. Linear-time modular decomposition of directed graphs. *Discrete Applied Mathematics*, 145(2):189–209, 2005.

[34] R.M. McConnell and J.P. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 5:536–545, 1994.

[35] R.M. McConnell and J.P. Spinrad. Linear-time transitive orientation. *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 8:19–25, 1997.

[36] M.Chein, M.Habib, and M.C.Maurer. Partitive hypergraphs. *Discrete Mathematics*, 37:35–50, 1981.

[37] R.H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In I. Rival, editor, *Graphs and Order*, pages 41–101. D. Reidel, Boston, 1985.

[38] R.H. Möhring. Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and boolean functions. *Annals of operations research*, 6:195–225, 1985.

[39] R.H. Möhring and F.J. Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. *Annals of Discrete mathematics*, 19:257–356, 1984.

[40] M.Golumbic. Comparability graphs and a new matroid. *J. Combin. Theory Ser.B*, 22:68–90, 1977.

[41] M.Habib and M. Maurer. On the X-join decomposition for undirected graphs. *Discrete Appl. Discrete Math*, 1:201–207, 1979.

[42] John H. Muller and Jeremy Spinrad. Incremental modular decomposition. *J. ACM*, 36(1):1–19, 1989.

[43] R.M.McConnell and J.P.Spinrad. Construction of probe interval models. *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 866–875, 2002.

[44] S.Benzer. On the topology of the genetic fine structure. *Proc. Nat. Acad. Sci., U.S.A*, 45:1607–1620, 1959.

[45] S.Booth and S.Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379, 1976.

[46] L.N. Shevrin and N.D Filippov. Partially ordered sets and their comparability graphs. *Siberian Math. J.*, 11:497–509, 1970.

[47] A.W. Shogan. Modular decomposition in stochastic transportation networks. *Technical Report UCB/ERL M 78/86, Electronics Research Laboratory*, 1978.

[48] Jr. Sielken, R.L. and H.O. Hartley. A new statistical approach to procject schedling. *In Decision Information, C.P.Tsokos and R.M.Thrall, eds.*, pages 153–184, 1979.

[49] G. Steiner. Machine scheduling with precedence constraints. *Ph.D Dissertation*, 3, 1982.

[50] Gallai Tibor. Transitiv orientierbare graphen. *Acta Math. Acad. Sci-Hungar.*, 18:25–66, 1967.

[51] Jr. Moore J.I. Jr. Trotter, W.T. and D.P. Sumner. The dimension of a comparability graph. *Proc. Amer. Math. Soc.*, 60:35–38.

[52] W.L.Hsu. O(mn) algorithms for the recognition and isomorphis probelems on circular-arc graphs. *SIAM J. Computing*, 24:411–439, 1995.