Thesis

An Algorithm for Modular Decomposition Based on Multiplexes

Submitted by

Pritish Chamania

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2015

Abstract

An Algorithm for Modular Decomposition Based on Multiplexes

Modular decomposition is instrumental in the the design of algorithms for solving many important graph theory problems. It has been applied towards developing recognition algorithms for many important perfect graph families. It also forms the basis of a number of efficient algorithms for solving combinatorial optimization problems on graphs. There are a number of efficient algorithms proposed in literature for computing the modular decomposition. Here we explore an $O(n^3)$ modular decomposition algorithm based on the theory of transitive orientation. The algorithm highlights how the problem of finding a transitive orientation is intimately related to that of finding the modular decomposition.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Ross McConnell, whose continuous support, expertise, understanding and patience, added immensely to my graduate experience.
I would like to thank other members of my committee Dr. Wim Bohm and Dr. Alexander Hulpke for the assistance and encouragement they provided.
I am grateful to Sharon Van Gorder, the administrative assistant in the computer science department for assisting me in many different ways.
Finally I would like to extend my deepest gratitude to my parents without whose love, support and understanding I could never have completed this thesis.

This thesis is typset in LaTeX using a document class designed by Leif Anderson.

Table of Contents

List of Figures

CHAPTER 1

Introduction

A *module* is a set X of vertices of a graph $G = (V, E)$ where every vertex in X has identical neighbors in the set $V \setminus X$. That is for each $y \in V \setminus X$, y is either adjacent to every member of X or to none. As an example, in figure 1.1 the vertex set $\{2, 3\}$ is a module. The set V and its all one element subsets are called *trivial modules* of G. A graph that only has trivial modules is called *prime*.

Consider two disjoint modules X and Y, we claim that either all elements of X are adjacent to all elements of Y, or no element of X is adjacent to any element of Y. If no member of X is adjacent to any member of Y, then the claim is true. Alternatively suppose $x \in X$ has a neighbor $y \in Y$, then as X is a module, all elements of X are adjacent to y. Furthermore, as Y is a module, every vertex of Y is adjacent to every vertex of X.

We can thus describe the relationship between two disjoint modules by saying that the two modules are either neighbors or non-neighbors. Let a *modular partition* be a partition of the vertices of a graph such that every partition class is a module. For a modular partition P

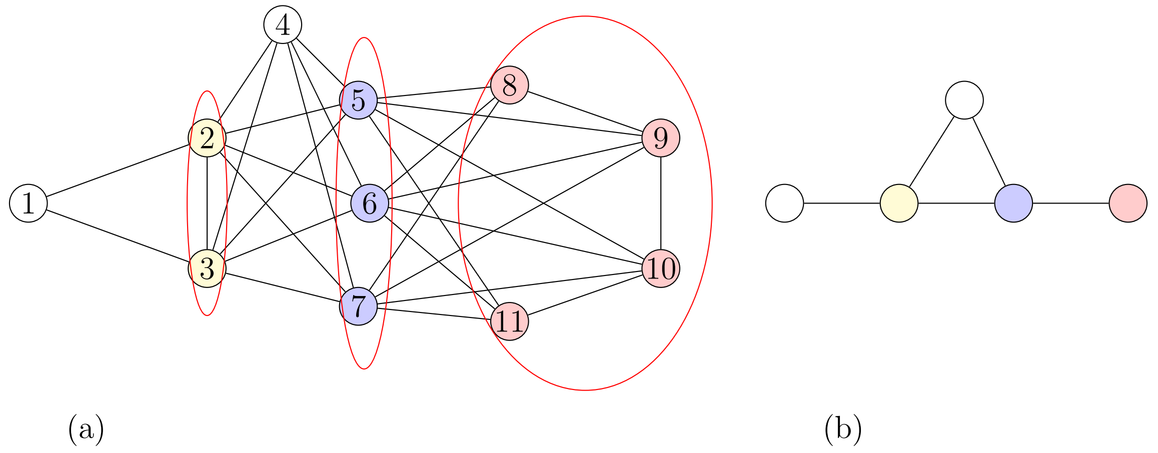

FIGURE 1.1. (a) Graph G with the modules color coded (b) The corresponding quotient graph G/P

the relationships between the partition classes define a *quotient graph* denoted G/P, whose vertices are members of P and whose edges describe the neighbor and non-neighbor relations between the members of P. Figure 1.1 (a)shows a graph G with vertices color coded, where vertices of the same color belong to the same partition class and (b) shows its corresponding quotient graph.

From the definition of a module it follows that each connected component of a graph is a module. The quotient defined by a modular partition P, such that every member of P is a connected component is an independent graph. Further, every union of connected components of a graph is also a module. Also observe that a module in the graph complement $\bar{G}$ is also a module in the graph G, and conversely a module in G is a module in $\bar{G}$, hence the operation of taking complement is module invariant. If P is a modular partition of the vertices of G, such that every member of P is a connected component in $\bar{G}$, then G/P is a complete graph.

The quotient graph captures the adjacencies between the members of a modular partition P. As a module may contain another module, partition classes of P may be recursively decomposed further into modules and quotients. There are many possible ways to do this. The *modular decomposition* is a unique canonical way to decompose a graph which subsumes all others.

In optimization theory, a large number of scheduling problems are often modeled as graph coloring problems. A *graph coloring* is simply an assignment of colors to vertices, such that no two adjacent vertices have the same colors. The lower bound on the number of colors needed is defined as the *chromatic number* of the graph. For example, in the graph shown in figure 1.2, the chromatic number is 3. Observe that a complete graph G with n vertices has

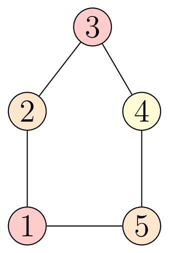

FIGURE 1.2. A graph with chromatic number 3

chromatic number n. A *clique* is a set of vertices in a graph such that every pair of vertices in the set are connected by an edge. For any graph G, it must be the case that the chromatic number is greater than or equal to the size of the largest clique of graph G. The size of the largest clique is termed as the *clique number*. *Perfect graphs* [1] are a special graph class where the chromatic number is exactly equal to the clique number. Perfect graphs were introduced by Berge in the 1960s. These graphs have nice algorithmic properties. A number of NP-hard optimization problems such as maximum stable set, maximum clique, minimum coloring and minimum clique cover on general graphs can be solved in polynomial time for perfect graphs. This theoretical result was first shown in a paper [2] by Grotschel, Lovasz and Schrijver. Since then, many algorithms which use modular decomposition as a first step to solve these problems on perfect graphs in linear time have been proposed in literature.

Modular decomposition is also an important ingredient of many recognition algorithms for certain classes of perfect graphs. The general strategy adopted by algorithms based on modular decomposition is to recursively decompose the given graph into subgraphs until no further decompositions are possible on the obtained graphs. These graphs are prime. Then the problem is solved on the prime graphs, and these partial solutions are then combined recursively to find the solution for the input graph. This approach was pivotal to the design of linear time recognition algorithms for comparability graphs, permutation graphs and interval graphs.

It is not surprising then that the problem of computing the modular decomposition of a graph has received considerable attention. The theory of modular decomposition was first described in a paper by Gallei. Since then a number of algorithms with different complexities have been proposed for computing the modular decomposition. The first linear time algorithm was developed by McConnell and Spinrad in 1994 [3]. Here in this report we explore an $O(n^3)$ modular decomposition algorithm based on a theory of transitive orientation developed by Golumbic. [4].

The paper is organized as follows. Section 2 presents general graph theory preliminaries and notation. This is followed by a review of the theories of modular decomposition and transitive orientation. The next section details the relationship between transitive orientation graph classes and modules. These results lead to an efficient modular decomposition algorithm. We end by describing a modular decomposition based recognition algorithm for interval graphs.

CHAPTER 2

Preliminaries

Let us recall some of the formal definitions in graph theory.

A *directed graph* G is a nonempty set of vertices and edges, formally described as an ordered pair (V, E), where V or $V(G)$ is a finite set and E or $E(G)$ is a subset of $V \times V$. If (u, v) is an edge, then v is *adjacent* to u, and a *neighbor* of u. The set of vertices which are adjacent to a vertex u denoted $N(u)$ form the *open neighborhood* of u. The *closed neighborhood* of u denoted $N[u]$ is given as the set $u \cup N(u)$. If (u, v) is an edge, then (u, v) is *incident* to u and v.

An *undirected graph* is the special case of a symmetric directed graph, that is, where $(v, u) \in E$ whenever $(u, v) \in E$. An *undirected edge* is the pair $\{(u, v), (v, u)\}$; this is denoted $\hat{uv}$.

The *degree* of a vertex v represented $deg(v)$ is the number of graph edges incident to v.

An *oriented* graph is a directed graph having no symmetric pair of directed edges.

An *orientation* of an undirected graph G is an assignment of exactly one direction to each of the edges of G.

Except when otherwise stated, we will henceforth assume that a graph is undirected.

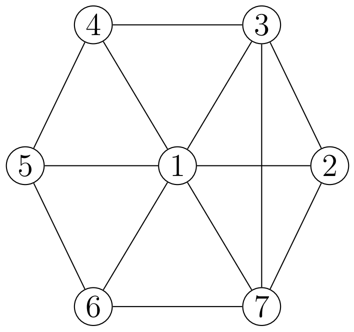

Figure 2.1. A graph $G(V, E)$ $V = \{1, 2, 3, 4, 5, 6, 7\}$ $E = \{\hat{12}, \hat{13}, \hat{14}, \hat{15}, \hat{16}, \hat{17}, \hat{23}, \hat{34}, \hat{45}, \hat{56}, \hat{67}, \hat{72}, \hat{37}\}$

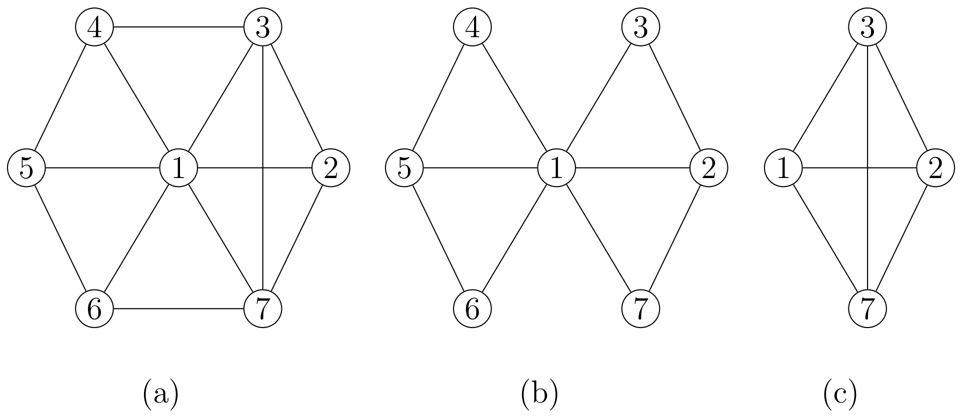

FIGURE 2.2. (b) is a sub-graph and (c) is an induced sub-graph of the graph shown in (a)

For a graph G(V,E), a *sub-graph* is a graph H(V',E') with V' a nonempty subset of V and E' a subset of $E \cap (V' \times V')$. For nonempty $V' \subseteq V$, the sub-graph $G[V']$ of G *induced* by V' is the graph (V', E') such that, for any $u, v \in V'$, (u,v) belongs to E' iff (u,v) belongs to E. Induced sub-graphs can be obtained by deleting vertices and all the incident edges from G.

A subset A $A \subseteq V$ is a *clique* if it induces a complete subgraph. A set defined with respect to some property is *maximal* if it is not contained in a larger set satisfying the property, for example a *maximal clique* is a clique that is not a subset of a larger clique. A *maximum clique* of graph G is a clique of maximum possible size for G. A maximum clique is a maximal clique, but the converse is not always true.

A *clique cover* of size k is a partition of the vertices $V = A_1 + A_2 + .. + A_k$ such that each A_i is a clique. The size of the smallest possible clique cover of G is called the *clique cover number* of G.

An *independent set* is a subset A of the vertices V of a graph such that no two vertices in A are adjacent.

A sequence of distinct vertices $[v_0, v_1, v_2, v_3, .., v_k]$ is a *path* from v_0 to v_k provided that $v_{i-}\hat{_1}v_i \in E$ for $i = 1, 2, ...k$

A sequence of vertices $[v_0, v_1, v_2, v_3, .., v_k, v_0]$ is a *cycle* provided that $v_0, v_1, v_2, v_3, .., v_k$ are distinct, $v_{i-}\hat{_1}v_i \in E$ for $i = 1, 2, ...k$ and $v_k\hat{v}_0 \in E$

Finally, A predicate P about G is *hereditary* if P holds true for G and every induced subgraph of G.

CHAPTER 3

Modular Decomposition

Definition 3.0.1. Given graph $G(V, E)$. For $U \subseteq V$, a vertex $\{v | v \in V \text{ and } v \notin U\}$ *distinguishes* U if it has both a neighbor and a non neighbor in U. A module in a graph is a non-empty subset of vertices not distinguished by any vertex.

Definition 3.0.2. A graph in which every module is trivial is called *prime*.

Definition 3.0.3. Two sets *overlap* if they have a non-empty intersection but neither is a subset of the other.

Lemma 3.0.1. *if M and M' are overlapping modules of a graph G, then the following are also modules of G.*

(1) $M \cap M'$

(2) $M \cup M'$

(3) $M \triangle M'$

(4) $M \setminus M'$

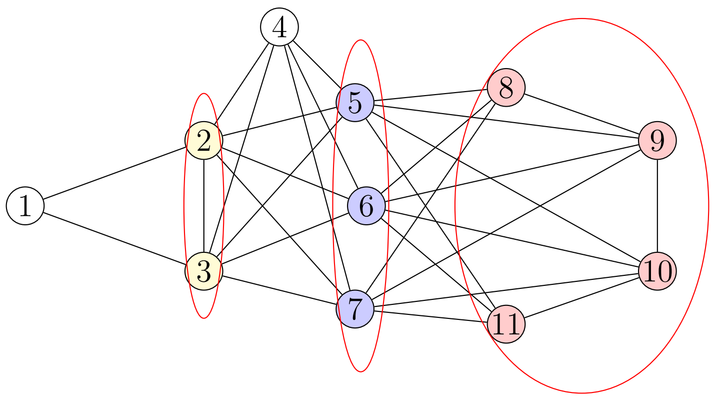

Figure 3.1. A graph with the modules color coded

(5) $M' \setminus M$

PROOF. (1) Let x be a vertex outside M. As M is a module, x does not distinguish M and hence does not distinguish $M \cap M'$. Similarly, as M' is a module, no vertex outside M' distinguishes $M \cap M'$.

(2) Let x be a vertex outside $M \cup M'$. Further, suppose x is adjacent some vertex in M. Since M is a module, x is adjacent to all vertices in M. This implies that x is adjacent to at least some vertices in M'. As M' is a module, this implies that x is adjacent all elements in M'. Similarly we can reason that if x is not adjacent to any members of M, then it is not adjacent to any vertex in M' as well.

(3) No vertex outside $M \cup M'$ distinguishes $M \triangle M'$. Suppose no vertex in $M \cap M'$ is adjacent to any vertex in $M \triangle M'$, then the claim is true. Alternatively, if vertex $x \in M \cap M'$ is adjacent to some vertex in $M \triangle M'$, then without loss of generality, assume this vertex v is in $M \setminus M'$. As M' is a module, v is adjacent to all members of M'. Next, since M is a module, all vertices of $M' \setminus M$ are adjacent to all vertices in M. Similarly, we can conclude that all vertices of $M \setminus M'$ are adjacent to all vertices in M'. Hence, no vertex in $M \cap M'$ distinguishes $M \triangle M'$.

(4) $M \triangle M'$ and M are two modules which overlap, hence their intersection which is $M \setminus M'$ is also a module.

(5) By symmetry.

□

Consider a modular partition $\{X_1, \ldots, X_k\}$ of a graph G. The induced subgraphs of each partition class X_i are called *factors*. Any induced subgraph of a set of vertices obtained by selecting any one vertex per partition class, is isomorphic to G/P. Hence, the quotient graph specifies all the edges of G which do not have both ends in a partition class. Thus,the quotient graph along with the factors are a complete specification of G.

LEMMA 3.0.2 ([5]). *If X is a module of G, then the modules of G that are subsets of X are the modules of G/X.*

LEMMA 3.0.3 ([5]). *Let P be a modular partition of a graph G. Then $X \subseteq P$ is a module of G/P iff $\cup_{M \in X} M$ is a module of G.*

DEFINITION 3.0.4. A module U in G is proper if $U \neq V(G)$. A module M is called a *strong* module, if for any module $M' \neq M$ of G, either $M' \cap M = \emptyset$ or one module is included into the other.

Although the number of modules of a graph can be exponential, the number of strong modules is $O(n)$. One strong module may be contained in another, and the order of the strong modules with respect to set inclusion forms the *modular decomposition tree* T. The root of the tree is V, and the leaves are the single element subsets of V. Every other strong module is the least common ancestor of its one element subsets. Let us call any module of a graph G which is not strong, a *weak* module of G. As the strong modules do not overlap any other modules, it follows that every weak module in the graph is a union of siblings in the T. However, the tree described so far does not specify all the modules of G. The following theorem describes a labeling scheme for the tree, which then allows one to determine all modules of G from T.

THEOREM 3.0.4. *There is a labeling of each strong module in the modular decomposition tree as degenerate or prime such that $Y \in X$ is a module if and only if it is either a strong module or a union of children of a strong module that is labeled degenerate.*

The next is the modular decomposition theorem, which describes the structure of the modular decomposition of G.

Theorem 3.0.5. *[6] For any graph G one of the following three cases must occur.*

(1) *G is disconnected, with components $G_1, G_2, ..., G_k$. Each union of the sets $V(G_i)$ is a module of G, and the other modules of G are precisely all the modules of individual components of G_i.*

(2) *The complement of G is disconnected, with components $H_1, H_2, ..., H_l$. Each union of the sets $V(H_j)$ is a module of G, and the other modules of G are precisely all the modules of individual subgraphs $\bar{H}_j$.*

(3) *Both G and its complement are connected. There is a partition $S_1, S_2, ..., S_r$ of $V(G)$(which can be computed in linear time), such that all the modules of G are precisely all the modules of individual subgraphs induced by the sets $S_t, t = 1, ...,$ plus the module $V(G)$.*

The theorem leads to the following recursive algorithm for computing the modular decomposition tree of a graph.

Algorithm 1 Recursive Modular Decomposition

```
1: procedure MD(G)
2:     If G has one vertex, v, then return v.
3:     Let r be a new internal node in the modular decomposition tree representing G.
4:     If G is disconnected, then for each connected component, C, make MD(C), a child of
       r. Finally mark r as degenerate.
5:     Else if the complement of G is disconnected, then for every connected component of
       the complement, C, make MD(C), the child of r. Finally mark r as degenerate.
6:     Else for every maximal module X, make MD(G[X]) a child of r. Finally mark r as
       prime.
7:     return r.
```

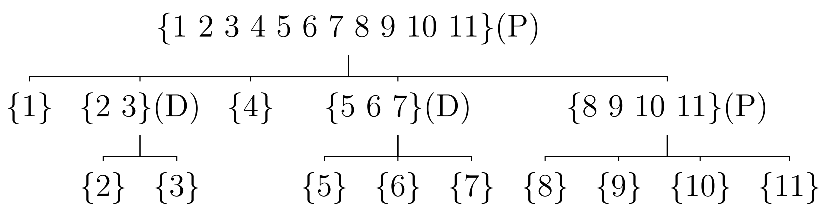

FIGURE 3.2. The modular decomposition tree of graph shown in figure 3.1. The nodes are labeled prime(P) and degenerate(D). Every node of the tree and every union of children of a degenerate node is a module.

LEMMA 3.0.6. *If G and its complement are both connected, then the maximal modules, induce a partition of the vertices of G, hence the modular decomposition tree is well defined and unique.*

Consider a node t of the modular decomposition tree , and let $M(t)$ be the module corresponding to t. We denote by $G(t)$ the quotient graph of all the children of t.

LEMMA 3.0.7. *Let G be a graph, $T(G)$ its modular decomposition tree. Then for any internal node t that belongs to $T(G)$, $G(t)$ is an edgeless graph or a complete graph if t is a degenerate node, and is a prime graph if t is a prime node.*

PROOF. The first case is trivial. For the third, as U is a prime node, the children are all maximal modules and this partition is unique. Therefore, the modules in the quotient are all singletons. □

Given the modular decomposition tree for a graph G and the set of quotient graphs for the nodes, G can be reconstructed inductively by reconstructing the factors from the bottom up, combining factors with the quotient at each level.

CHAPTER 4

Transitive Orientation Theory

An orientation (V, F) obtained by assigning directions to each edge of an undirected graph is *transitive* if:

$xy \in F$ and $yz \in F$ implies $xz \in F$

Let us try to realize a transitive orientation of the four cycle shown in figure 4.1.

Arbitrarily orienting the edge $\{a, b\}$ from a to b forces us to orient edge $\{b, c\}$ from c to b and edge $\{a, d\}$ from a to d. Gallai formalized this idea by defining the Γ relation. We next review the formal model of transitive orientation developed by Golumbic [4].

Definition 4.0.5. The binary relation Γ is defined on the edges of an undirected graph $G(V, E)$ as follows: ab Γ $a'b'$ iff either $a = a'$ and $bb' \notin E$, or $b = b'$ and $aa' \notin E$. This relation is symmetric and reflexive. The equivalent classes of its transitive closure Γ^* are called implication classes and form a partition of the edges.

Lemma 4.0.8. *Two edges are in the same equivalence class if ab Γ^* cd, and hence there exists a Γ chain from ab to cd of the form* $ab = a_0b_0\Gamma a_1b_1\Gamma..\Gamma a_kb_k = cd$

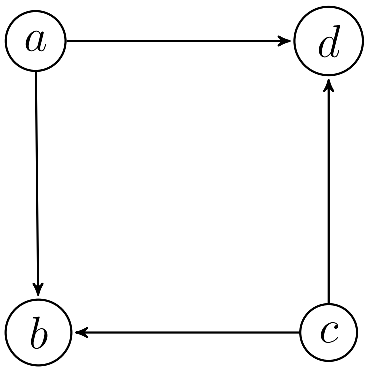

Figure 4.1. A four cycle

Let ab denote a directed edge and $\hat{ab}$ denote an undirected edge. For an edge ab, edge ba is called the inverse edge. A set of directed edges I^{-1} is the inverse of a set I iff for any edge $ab \in I$, $ba \in I^{-1}$. If I is an implication class then so is I^{-1}. The set $\hat{I} = I \cup I^{-1}$ is known as a *color class.*

LEMMA 4.0.9. *Denote $I[G]$ as the collection of all implication classes of G. For $I \in I[G]$, we have either $I \cap I^{-1} = \emptyset$ or $I = I^{-1} = \hat{I}$*

For example in the graph shown in figure 4.2 we have,

$$I_1 = \{12, 13\} \quad I_1^{-1} = \{21, 31\}$$

$$I_2 = \{14, 24, 34, 54, 56, 57\} \quad I_2^{-1} = \{41, 42, 43, 45, 65, 75\}$$

$$I_3 = \{67\} \quad I_3^{-1} = \{76\}$$

For the example in the graph shown in figure 4.3 we have,

$$I = I^{-1} = \hat{I} = \{12, 23, 34, 45, 51, 21, 32, 43, 54, 15\}$$

DEFINITION 4.0.6. $G(V, E)$ be an undirected graph. A complete subgraph (V_s, S) on $r + 1$ vertices is called a *simplex* of rank r if each undirected edge of S is contained in a different color class of G. The *multiplex* generated by a simplex S of rank r is defined to be the undirected subgraph (V_m, M) with $M = \cup C$ where the union is over all color classes C satisfying $C \cap S \neq \emptyset$.

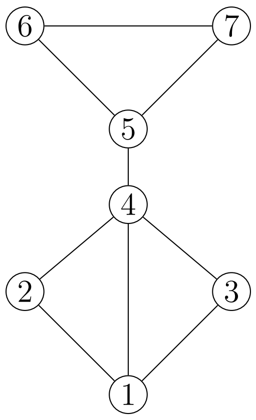

FIGURE 4.2. Graph G

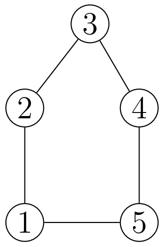

FIGURE 4.3. A five cycle

LEMMA 4.0.10 ([4]). *A multiplex M is maximal iff the simplex S generating M is a maximal simplex.*

THEOREM 4.0.11 ([4]). *If M_1 and M_2 are maximal multiplexes of an undirected graph G, then either $M_1 \cap M_2 = \emptyset$ or $M_1 = M_2$*

THEOREM 4.0.12 ([4]). *Let $G(V, E)$ be an undirected graph and let $E = M_1 + M_2 + M_k$, where each M_i is a maximal multiplex .*

(1) *If F is a transitive orientation of G, then $F \cap M_i$ is a transitive orientation of M_i.*
(2) *If $F_1, ., F_k$ are transitive orientations of $M_1, ., M_k$ respectively, then $F_1 + F_2 + ... + F_k$ is a transitive orientation of G.*

The above theorem states that the maximal multiplexes partition the edges and act independently with respect to transitive orientation.

CHAPTER 5

Modular Decomposition Algorithm

First we consider the problem of listing all the maximal multiplexes of an undirected graph G. Assume G contains a tri-colored triangle T.

Lemma 5.0.13 ([4]). *Let(V_s,S) be a simplex of an undirected graph $G = (V, E)$ generating a multiplex M. If G contains a tri-colored triangle on vertices a,b,c such that $ab \notin M$ but bc $\in M$, then we may adjoin the vertex a to (V_s,S) to obtain the larger simplex (V_t,T) containing (V_s,S), where $V_t = V_s \cup \{a\}$, $T = S \cup \{\hat{ad} \| d \in V_s\}$*

Let us say that any two edges are *cousins* of each other if they are in the same color class. By lemma 5.0.12 all three edges of T must be in the same maximal multiplex and must have cousins in any simplex generating this maximal multiplex M.

Theorem 5.0.14 (Golumbic 1977). *Let S be a simplex contained in a multiplex M. There exists a simplex S_m generating M such that $S \subseteq S_m$.*

Proof. Now if S is of the same rank as the multiplex, then the claim is true. Suppose the theorem is true for all simplex with rank greater than S's rank. Let V be the multiplex generated by S. Further, let U be any simplex generating M. All edges of S have cousins in U. Thus we have $V \subseteq M$. Since U is connected, there exists a tri-colored triangle abc in U where $ab \in V$ and $bc \notin V$, thus by lemma 5.0.12 S can be extended to obtain a simplex T of rank = rank $S + 1$. Then by induction we have the result that there exists a simplex S_m generating M such that $S \subseteq S_m$. □

This theorem implies that T itself is a part of some simplex generating M. Let us call this simplex S_t. Suppose another tri-colored triangle T_1 is contained in G where at least one of the edges of T_1 has the same color class as an edge of T. Again by the appplication of lemma 5.0.12, we can see that T_1 is also contained in M. Further, all edges of T_1 have cousins in S_t. Thus it can be seen, that all triangles with intersecting color classes must be contained in the same maximal multiplex, and hence their edges must have cousins in any simplex generating this multiplex. The algorithm proceeds by computing the color class for each edge of graph G. Then we detect all the tri-colored triangles. Finally we compute the unions of color classes of all intersecting tri-colored triangles. Each union determines the color classes of all the edges of a simplex generating a maximal multiplex. The remaining color classes (whose edges are not in any tri-colored triangle) are each in a multiplex of rank 1.

Consider an undirected graph G with n vertices and m edges (each symmetric pair of edges is counted as one). A naive algorithm with running time $O(n^3)$ may be implemented for listing all the triangles of G. However, let us work on getting a tighter bound. We claim that the number of triangles in any graph is $O(m^{\frac{3}{2}})$. To see this, consider the following argument. Let the *big neighbors* of a vertex v be the neighbors whose degrees are at least as large as v's, and let the *big degree* of v be the number of big neighbors of v.

Since the sum of the degrees of all the vertices of any graph is $2m$, for the case where a vertex v of a graph has k big neighbors , we get the inequality:

$$k.k <= 2m \tag{1}$$

$$k = O(m^{\frac{1}{2}}) \tag{2}$$

For each triangle of G, choose a vertex of maximum degree in the triangle and charge the triangle to the opposite edge. This implies that each edge is charged $O(m^{\frac{1}{2}})$ times. Next there being m edges each charged $O(m^{\frac{1}{2}})$ times, we can conclude that the number of triangles in a graph is $O(m^{\frac{3}{2}})$. Working on similar lines, Thomas Schank developed algorithm Forward [7] with running time $O(m^{\frac{3}{2}})$ for listing all the triangles in a graph. The algorithm accepts as input a list of vertices ordered by degree. Since the degree of each vertex lies between 0 and $n-1$, radix sort may be used to order the vertices in $O(n)$ time.

Algorithm 2 Algorithm Forward

```
    Input: ordered list (high degree first) of vertices (1,...,n); Adjacencies Adj(v)
1: for $v \in V$ do
2:     $A(v) \leftarrow \emptyset$
3: for $s \in (1, ......, n)$ do
4:     for $t \in Adj(s)$ do
5:         if $s < t$ then
6:             for all $v \in A(s) \cap A(t)$ do
7:                 Output triangle $\{v, s, t\}$
8:             $A(t) \leftarrow A(t) \cup s$;
```

Borrowing ideas from algorithm Forward, we develop an algorithm Implication Classes to list all the implication classes of an undirected graph also in $O(m^{\frac{3}{2}})$ time. The algorithm constructs the Γ graph X of G, whose vertices are the edges of graph G, and two vertices x and y of X are connected by an edge iff $x \Gamma y$ in G. Note that in the algorithm we assume that it takes $O(1)$ time to tell whether there is an edge between two vertices of G.

THEOREM 5.0.15. *The connected components of graph X output by algorithm Implication Classes represent the implication classes of the input undirected graph.*

Algorithm 3 Algorithm Implication Classes

```
Input: ordered list (high degree first) of vertices (1,...,n); Adjacencies Adj(v)
Empty Γ graph X;
Output: Graph X with edges set;
1: for $s \in (1, ......, n)$ do
2:     for $t \in Adj(s)$ do
3:         Add node st to X
4: for $v \in V$ do
5:     $A(v) \leftarrow \emptyset$
6: for $s \in (1, ......, n)$ do
7:     for $t \in Adj(s)$ do
8:         if $s < t$ then
9:             for all $v \in A(s) \triangle A(t)$ do
10:                if v is adjacent to s then
11:                    Add edges vs-ts and sv-st to X
12:                else
13:                    Add edges vt-st and ts-tv to X
14:            $A(t) \leftarrow A(t) \cup s$;
```

Once we have the triangles list of G, with each edge color coded, the list can be traversed to detect all tri-colored triangles. The algorithm Multiplex with $O(m^{\frac{3}{2}})$ running time computes the maximal multiplexes of G , when given as input the list of tri-colored triangles of G.

Algorithm 4 Algorithm Multiplex

```
Input: List of tri-colored triangles (L) of an undirected graph G
$L = x_i y_j z_k \quad where 1 \leq (x_i, y_j, z_k) \leq k$ ;(G has k color classes)
Graph C with nodes (1,...,k) and no edges;
Output: Graph C with edges set;
1: for all $xyz \in L$ do
2:     create edges $xy, yz, zx$ in C if they do not exist
```

THEOREM 5.0.16. *The connected components of graph C outputted by algorithm multiplex represent the maximal multiplexes of the input undirected graph G.*

5.1. Computing the Modular Decomposition from the Maximal Multiplexes

Let us begin to explore the relationship between modules and multiplexes. For a set of edges C, define the *vertex set* of C, denoted V_C, as the union of all the endpoints of all the edges in C.

Lemma 5.1.1. *The vertex set of each color class in a graph is a module.*

Proof. Consider a color class C. Suppose V_C is not a module. Then there is a vertex $v \notin V_C$ which distinguishes V_C. Let $x \in V_C$ be a vertex which is adjacent to v and $y \in V_C$ be a vertex not adjacent to v. As a color classes is connected, there is a path from y to x of size $n \geq 2$. Denote this path as $\{i_1, ..., i_n\}$. Suppose i_k is the first vertex in the path which is a neighbor of v, then we have $i_k i_{k-1} \Gamma i_k v$, which implies that $v \in V_C$. Hence we arrive at a contradiction. □

Lemma 5.1.2. *If both end points of an undirected edge in color class C are inside a module M, then $V_C \subseteq M$.*

Proof. Suppose there is a vertex v such that $v \in V_C$ and $v \notin M$. Then v distinguishes M, a contradiction. □

Lemma 5.1.3. *For all maximal multiplex M, V_M is a strong module.*

Proof. Suppose this is not the case then consider a maximal multiplex M_1 whose vertex set overlaps a module M. It is easy to see that V_{M_1} is a module and that there is a tri-colored triangle, with a vertex in $V_{M_1} \setminus M$, a vertex in $M \setminus V_{M_1}$ and a vertex in $V_{M_1} \cap V_M$. Furthermore, this triangle has a side in M_1 and a side not in M_1, which then implies that M_1 is not maximal, which is contrary to our assumption. □

Lemma 5.1.4. *Let M be a strong module and $\hat{ab}$ be an undirected edge such that $a, b \in M$. if M is the maximal multiplex containing $\hat{ab}$, then $V_M \subseteq M$.*

Morvan et al.[8] develop these ideas and also present the following theorem.

Theorem 5.1.5. *The canonical decomposition tree of any undirected graph $G = (V, E)$ verifies the following properties.*

(1) *Every non parallel interior node is a strong module V_M where M is a maximal multiplex.*
(2) *A node corresponding to a module M is an ancestor of a node corresponding to a module N if and only if M covers N, or equivalently $V_N \subseteq V_M$.*

The preceding theorem implies that for any undirected graph G, the list of maximal multiplexes of G and the list of maximal multiplexes of the graph complement $\bar{G}$ can be used to construct the modular decomposition of G. The maximal multiplexes of G may be determined using the algorithm Multiplex. For finding the maximal multiplexes of $\bar{G}$, a simple $O(nm)$ algorithm may be devised to first compute the implication classes of the Γ relation of $\bar{G}$. For each edge in G, we find a list L of vertices not adjacent to either of the end points of the edge. For an edge $\hat{ab}$ and a vertex v in L, $av\Gamma bv$ and $va\Gamma vb$ in $\bar{G}$. After the maximal multiplexes for G and the complement graph have been determined, it is fairly simple to compute an ordering of the strong modules with respect to set inclusion in $O(n^3)$ running time, which then leads to the modular decomposition algorithm.

CHAPTER 6

Interval Graph Recognition

Interval graphs arise in modeling overlapping intervals of a line. Each interval is represented as a vertex in the graph and two vertices are adjacent if the corresponding intervals overlap. Here we consider the problem of recognizing interval graphs. First we look at methods to solve the problem on prime interval graphs. Solution to a prime interval graph can then be used to construct a solution to a general interval graph using the modular decomposition tree as a guide.

In 1959, an American biologist Seymour Benzer was working on problems of genetic structure, specifically he worked with mutations of a virus that infect bacteria. He looked into a large number of mutated strains of a bacteriophage T4. Normally virus T4 kills bacteria. However, mutated forms of T4 (where a continuous interval is deleted from an important gene) are unable to kill the bacteria. Suppose a bacteria is infected with two different mutants at the same time, the question then arises is whether the infected bacteria will survive. Surprisingly, as Benzer observed, a pair of mutated strains were able to kill the bacteria in some cases. The explanation was that in some cases these mutated strains were able to recombine and produce as offspring, a non-mutated form of T4. He expressed his observations in the form of a matrix, part of which is shown in the following figure. The rows and columns labels are the different mutated strains. Matrix element value of 0 indicates that the corresponding mutants in the row,column were able to recombine to produce undamaged offspring. He showed that a graph where each vertex is a mutant, and where there is an edge between mutant pairs iff the corresponding row,col value is one, is an interval graph. This gave strong support to the hypothesis that the genes are arranged in a

Strain	184	215	250	C33
184	1	1	1	1
215	1	1	0	0
250	1	0	1	1
C33	1	0	1	1

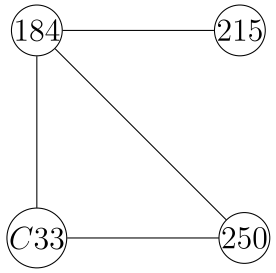

FIGURE 6.1. A matrix representing a subset of Benzer's data and the corresponding graph, whose vertices are viral strains and whose edges connect strains that couldn't recombine to produce undamaged offspring.

linear sequence on the chromosomes. If two mutants had overlapping intervals deleted, they were unable to recombine and recover the entire undamaged genome, thus were not able to kill the bacteria.

Many more applications of interval graphs have come up since then. These graphs have been applied to solving problems in a variety of areas from biology to chip design. Here we explore the problem of recognizing interval graphs. The algorithm for interval graph recognition involves using the modular recognition procedure described in the previous sections. Also here we explore relations between chordal graphs, which are an important graph class, and interval graphs.

6.0.1. Preliminaries on Interval graphs and Chordal graphs.

DEFINITION 6.0.1. A graph G = (V, E) is called an interval graph if there exists a set $I_v|v \in V$ of real intervals such that $I_u \cap I_v \neq \emptyset$ if and only if $(u, v) \in E$. The set of intervals is known as the *interval model* of graph G.

DEFINITION 6.0.2. A simple Graph G is *triangulated* if every cycle of length greater than three has an edge joining two non-consecutive vertices of the cycle. The edge is called a *chord*, and triangulated graphs are also called *chordal* graphs.

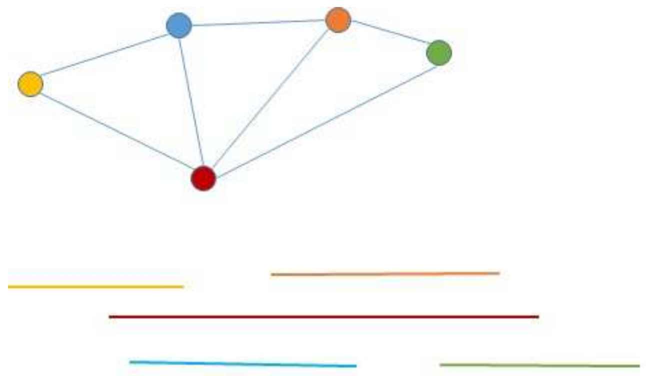

FIGURE 6.2. An interval graph and the corresponding interval model

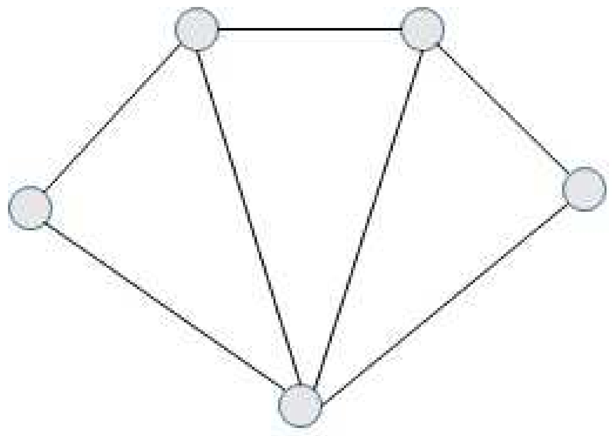

FIGURE 6.3. A chordal graph.

We can see that chordal graphs are hereditary, as deleting some vertices of a chordal graph still results in a graph which is triangulated.

LEMMA 6.0.6. *All Interval graphs are Chordal*

DEFINITION 6.0.3. A vertex v in graph G is called *simplicial* if and only if the subgraph induced by $N[v]$ is a complete graph. A graph of n vertices has a *perfect elimination ordering* if and only if the vertices of G can be ordered $\{v_1, v_2, \ldots, v_n\}$ such that each vertex v_i is

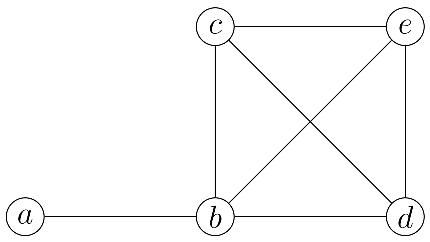

FIGURE 6.4. {a, b, c, d, e} is a perfect elimination order

simplicial in the subgraph induced by $\{v_i, \ldots, v_n\}$ In figure 6.4 {a, b, c, d, e} is a perfect elimination order for the graph.

DEFINITION 6.0.4. For graph $G(V, E)$, $S \subset V$ is an a-b separator for nonadjacent vertices a,b, if a and b are in distinct connected components of $G[V - S]$. An a-b separator S is minimal for a,b if no proper subset of S is an a-b separator.

THEOREM 6.0.7. *Every minimal a-b separator of a chordal graph G induces a complete subgraph of G.*

PROOF. Assume graph $G(V, E)$ is chordal. If G is complete then the claim is true. Alternatively, suppose S is a minimal x-y separator for two nonadjacent vertices x,y of G. Let X and Y be the connected components of $G[V - S]$ which contain x and y respectively. Since S is minimal, every element of S has one neighbor in X and one neighbor in Y. Let $\{a, b\}$ be an arbitrary pair of vertices in S, paths P_x from a to b with internal vertices in X and P_y from b to a with internal vertices in Y exist. Select the paths to be of minimum length. Then the vertices of P_x and P_y forms a cycle whose length is greater than 3. As G is chordal this cycle must have a chord. Consider an internal vertex of P_x and an internal vertex of P_y. No edge exists between any such pair of vertices. Furthermore, as the path lengths are the smallest, for P_x no edge exists between the internal elements of this path that are not consecutive. Similar is the case for P_y. Therefore, the only possible chord is the edge $\{a, b\}$. This implies that there is an edge between any pair of vertices in S. □

THEOREM 6.0.8. *Every chordal graph $G = (V, E)$ has a simplicial vertex. Moreover, if G is not a clique then it has two nonadjacent simplicial vertices.*

PROOF. if G is complete then every vertex is simplicial. For the case when G is not complete the proof follows by induction on the number of vertices. Let x, y be two non-adjacent vertices of G. Suppose the theorem is true for all graphs with fewer vertices than G. Then, let S be a minimal x-y separator of x and y. Let X and Y be the connected components of $G[V - S]$ which contain x and y respectively. By the inductive hypothesis, either the induced subgraph $G[X \cup S]$ has two nonadjacent simplicial vertices or $G[X \cup S]$ is a complete subgraph. If $G[X \cup S]$ is not a complete subgraph then one of the non adjacent simplicial vertex must be in $G[X]$ as $G[S]$ is complete. A simplicial vertex of $G[X \cup S]$ is also a simplicial vertex of G as $G[X \cup Y \cup S] = G$ and there are no edges between elements of X and Y. Similarly it can be shown that Y contains a simplicial vertex. Thus we have the result that if G is chordal then either it has two non-adjacent simplicial vertices or it is a complete graph. □

THEOREM 6.0.9. *A graph G is chordal if and only if G has a perfect vertex elimination ordering.*

PROOF. Find a simplicial vertex x in G. By theorem 6.0.8 x must exist. Remove the vertex from the graph and repeat the procedure until all the vertices are consumed. The order in which the vertices were removed is the perfect elimination order. Conversely, let C be a cycle of G and v be the vertex of C which is leftmost in the perfect elimination order.

Since x has at least two neighbors in C, the eventual simpliciality of x guarantees a chord in C. $\square$

The number of maximal cliques in a graph is generally greater than $O(n)$. Consider a complete bipartite graph, which is a bipartite graph where every vertex of the first set is connected to every vertex of the second set. It is easy to see that the number of maximal cliques is $O(n^2)$ in such a graph. Similarly we can see that a complete tripartite graph denoted $K_{r,s,t}$,where r is the number of vertices in the first set, s is the number of vertices in the second set and t the number of vertices in the third set, can have $O(n^3)$ cliques. A complete multipartite graph $K_{3,3,...,}$ has 3^n maximal cliques. For the case of chordal graph, let us establish an upper bound on the number of maximal cliques.

For a chordal graph $G = (V, E)$, let $X(v)$ denote all the neighbors of a vertex $v \in V$ which are to the right of it in a perfect elimination ordering. We claim that all maximal cliques in a chordal graph are of the form $\{k \cup X(k)\}$. The set $\{k \cup X(k)\}$ is a clique, this is implied by the definition of a perfect elimination ordering. Further, if w is the leftmost vertex of a maximal clique C, in the perfect elimination ordering, then C is of the form $w \cup X(w)$. Thus, all the maximal cliques of G are included in the set $\{v_i \cup X(v_i)\}$ where $1 \leq i \leq n$. This leads to the following result.

Proposition 6.0.10. *Every chordal graph has at most n maximal cliques, where n is the number of vertices in the graph.*

6.0.2. Recognition of Interval graphs. We saw earlier that a chordal graph has two non-adjacent simplicial vertices. Further, as chordal graphs are hereditary, it is possible to make any vertex v as the last in a perfect elimination ordering. The algorithm is simple, find a simplicial vertex which is not v, delete it and recurse on the updated graph. The

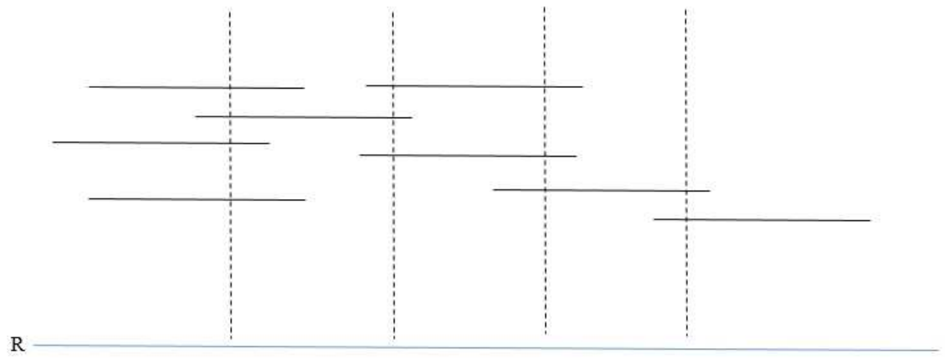

FIGURE 6.5. The interval model is a collection of intervals which are subsets of the real line. The intervals are shown one above another for clarity. Intervals which form a maximal clique are marked by a dashed line.

algorithm described above is $O(nm)$ as one must first find a simplicial vertex in each iteration. Also note, that if G is a clique, then every possible ordering of the vertices is a perfect elimination order. Many algorithms for computing a perfect elimination ordering of chordal graphs in linear time have been devised. They exploit the fact that any vertex may be the last vertex in a perfect elimination order, and proceed to construct the ordering in reverse. The general strategy is to first select any vertex v and put it in position n in the order, then select a neighbor of v and put it in position $n-1$ and so on. Lex-BFS is a linear time algorithm based on this strategy which was introduced by Rose and Tarjan in 1976 [9]. The algorithm accepts a $G(V, E)$ and a vertex $v \in V$ as input and outputs a linear ordering of the vertices. When the input graph is a chordal graph, the Lex-BFS ordering is in fact a reversed perfect elimination ordering.

As seen in figure 6.5, all maximal cliques of an interval graph can be represented by distinct points on the real line (the point where the dashed lines touch the real line). Thus, interval graphs seem to have a natural ordering of the maximal cliques. This observation motivated the following theorem by Gilmore and Hoffmann.

THEOREM 6.0.11. *[10] An undirected graph G is an interval graph if and only if the maximal cliques of G can be linearly ordered such that, for every vertex v in G, the maximal cliques containing the vertex v occur consecutively.*

Let us call such a linear ordering of the maximal cliques an *interval ordering.* Generally an interval graph may have many interval orderings. However, prime interval graphs have a unique interval ordering[11].

DEFINITION 6.0.5. A maximal clique C is an *outer* clique if there is an interval ordering of the maximal cliques such that C is either the first or the last clique.

THEOREM 6.0.12. *[12] Let $G = (V, E)$ be an arbitrary graph with associated Lex-BFS ordering $\lambda = [v_1, ..., v_n]$, and let G_i denote the subgraph of G induced by $V_i = (v_1, ..., v_i)$. Then G is an interval graph iff, for each $i = 1 \ldots n-1$, G_i is an interval graph and $N(v_{i+1}) \cup v_i$ is contained in an outer clique of G_i. In this case, v_{i+1} is a simplicial vertex of G_{i+1}.*

So the Lex-BFS traverses the maximal cliques of G in some order and stops with the vertices from an outer clique. All the maximal cliques of a chordal graph can also be computed with Lex-BFS in linear time. Given the set of maximal cliques and a vertex from an outer clique, an algorithm developed by Hsu[11] may be used to find the interval ordering of a prime interval graph in linear time.

LEMMA 6.0.13. *Let S be a module in a chordal graph $G = (V, E)$. Either S is a clique or $N(S)$ is a clique.*

PROOF. Suppose neither S nor $N(S)$ is a clique. Then S has two non adjacent vertices, call them x, y. Similarly $N(S)$ has two non adjacent vertices, call them a, b. We now see that $\{x, a, y, b, x\}$ is a four cycle with no chord, a contradiction. $\square$

Moving on to the problem of identifying interval graphs. Consider a module S of a chordal graph G. If we replace all the vertices of S by a single marker vertex V_s and set the adjacencies of this vertex to be the adjacencies of any vertex in S, then the resultant graph G_2 is still an interval graph. Further, we can separately compute the interval model for S and G_2. To obtain an interval model of G, we can replace the interval of V_s with the interval representation of S. This observation enables us to build an interval model for any interval graph from its modular decomposition tree. We start with the modular decomposition tree. First we find the interval models for all quotient graphs, for the prime case we use Hsu's algorithm. Then we combine these partial models and build the interval model in a bottom-up fashion. This is a recurring theme seen in many graph algorithms. For example, a similar approach has been used to find the transitive orientation of comparability graphs. First a solution to the problem is designed for the prime case, then this solution is used to find a solution to the general case using the modular decomposition tree as a guide.

Bibliography

[1] J. L. Ramírez-Alfonsín and B. A. Reed, *Perfect graphs*, vol. 44. Wiley, 2001.

[2] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981.

[3] R. M. McConnell and J. Spinrad, "Linear-time modular decomposition and efficient transitive orientation of comparability graphs.," in *SODA*, vol. 94, pp. 536–545, 1994.

[4] M. C. Golumbic, *Algorithmic graph theory and perfect graphs*, vol. 57. Elsevier, 2004.

[5] R. M. McConnell and J. Spinrad, "Ordered vertex partitioning.," *Discrete Mathematics & Theoretical Computer Science*, vol. 4, no. 1, pp. 45–60, 2000.

[6] P. Hell, M. Hermann, and M. M. Nevisi, "Counting partitions of graphs," in *Algorithms and Computation*, pp. 227–236, Springer, 2012.

[7] T. Schank and D. Wagner, "Finding, counting and listing all triangles in large graphs, an experimental study," in *Experimental and Efficient Algorithms*, pp. 606–609, Springer, 2005.

[8] M. Morvan and L. Viennot, "From parallel comparability graph recognition and modular decomposition," in *13th Symposium on Theoretical Aspects of Computer Science (STACS)*, vol. 1046, pp. 169–180, Springer Berlin/Heidelberg, 1996.

[9] D. J. Rose, R. E. Tarjan, and G. S. Lueker, "Algorithmic aspects of vertex elimination on graphs," *SIAM Journal on computing*, vol. 5, no. 2, pp. 266–283, 1976.

[10] P. C. Gilmore and A. J. Hoffman, "A characterization of comparability graphs and of interval graphs," *Canad. J. Math*, vol. 16, no. 539-548, p. 4, 1964.

[11] W.-L. Hsu and T.-H. Ma, "Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs," *SIAM Journal on Computing*, vol. 28, no. 3, pp. 1004–1020, 1998.

[12] N. Korte and R. H. Möhring, "A simple linear-time algorithm to recognize interval graphs," in *Graph-Theoretic Concepts in Computer Science*, pp. 1–16, Springer, 1987.