Decomposition by complete minimum separators and applications

Romain Pogorelcnik

N° d'ordre : 2301
EDSPIC : 590

UNIVERSITÉ BLAISE PASCAL - CLERMONT II

Ecole Doctorale
SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

THÈSE

présentée par

Romain Pogorelcnik

pour obtenir le grade de

Docteur d'Université

Spécialité : INFORMATIQUE

préparée au laboratoire **LIMOS**

Décomposition par séparateurs minimaux complets et applications

Soutenue publiquement le 4 décembre 2012
devant le jury composé de :

Remerciements

Je tiens tout d'abord à remercier ma directrice de thèse, Anne Berry, pour m'avoir conseillé, encouragé et soutenu tout au long de la thèse avec patience et disponibilité.
Je remercie Alain Denise et Irena Rusu pour l'intérêt qu'ils ont porté à ce travail et pour les remarques constructives et les différentes corrections qu'ils ont pu apporter.
Je remercie Vassilis Giakoumakis et Alain Quillot pour avoir accepté de participer à mon jury de thèse.
Je remercie mes collègues et amis en particulier Jean-Christophe, Cédric, Frédéric, Olivier, Jan, Hélène pour tous les moments de détente que l'on a pu passer ensemble et pour leurs conseils.
Enfin je tiens à remercier mon épouse pour m'avoir toujours encouragé et pour le soutien moral qu'elle a su m'apporter.

Résumé

Nous avons utilisé la décomposition par séparateurs minimaux complets.

Pour décomposer un graphe G, il est nécessaire de trouver les séparateurs minimaux dans le graphe triangulé H correspondant. Dans ce contexte, nos premiers efforts se sont tournés vers la détection de séparateurs minimaux dans un graphe triangulé.

Nous avons défini une structure, que nous avons nommée 'atom tree'. Cette dernière est inspirée du 'clique tree' et permet d'obtenir et de représenter les atomes qui sont les produits de la décomposition.

Lors de la manipulation de données à l'aide de treillis de Galois, nous avons remarqué que la décomposition par séparateurs minimaux permettait une approche de type 'Diviser pour régner' pour les treillis de Galois.

La détection des gènes fusionnés, qui est une étape importante pour la compréhension de l'évolution des espèces, nous a permis d'appliquer nos algorithmes de détection de séparateurs minimaux complets, qui nous a permis de détecter et regrouper de manière efficace les gènes fusionnés.

Une autre application biologique fut la détection de familles de gènes d'intérêts à partir de données de niveaux d'expression de gènes. La structure de 'l'atom tree' nous a permis d'avoir un bon outils de visualisation et de gérer des volumes de données importantes.

Mots-clés : décomposition par séparateurs minimaux complets, treillis de Galois, réseaux de similarités, réseaux de co-expression, gènes fusionnés, graphe triangulé.

Abstract

We worked on clique minimal separator decomposition.

In order to compute this decomposition on a graph G we need to compute the minimal separators of its triangulation H. In this context, the first efforts were on finding a clique minimal separators in a chordal graph.

We defined a structure called atom tree inspired from the clique tree to compute and represent the final products of the decomposition, called atoms.

The purpose of this thesis was to apply this technique on biological data. While we were manipulating this data using Galois lattices, we noticed that the clique minimal separator decomposition allows a divide and conquer approach on Galois lattices.

One biological application of this thesis was the detection of fused genes which are important evolutionary events. Using algorithms we produced in the course of along our work we implemented a program called MosaicFinder that allows an efficient detection of this fusion event and their pooling.

Another biological application was the extraction of genes of interest using expression level data. The atom tree structure allowed us to have a good visualization of the data and to be able to compute large datasets.

Keywords: clique minimal separator decomposition, Galois lattice, similarity network, expression level network, fused genes, microarray, chordal graph.

Contents

Introduction

The central tool of this thesis is a decomposition called *clique minimal separator decomposition*. A separator is a set of vertices whose removal disconnects a previously connected graph. A separator is a clique if all its vertices are pairwise adjacent.

The decomposition consists in finding a separator, breaking up the graph into a set of subgraphs inside which the separator is copied, and reproducing this step until there is no more subgraph containing clique separators. These subgraphs are called *Atoms*.

In 1985 Tarjan, used clique separator decomposition to solve, by a divide and conquer approach, hard problems such as vertex colouring, maximum clique and graph perfection testing. He proposed an $O(nm)$ time algorithm to do this.

One major problem of the algorithm Tarjan proposed is the fact that it leads to a non unique decomposition. Leimer in 1993 optimized this decomposition using only clique *minimal* separators, which leads to a unique decomposition.

There are still two drawbacks to this decomposition:

- First, not every graph has a clique separator.
- Second, the subgraphs obtained are not disjoint, which means that if the overlap is large, little is to be gained by a Divide-and-Conquer approach.

However recent research on data mining and microarray have modelized the data by a graph and have used clique minimal separator decomposition [10, 24]. The graphs obtained almost always have a clique separator. Moreover clique minimal separators are often important loci since they reflect maximum interaction. Finally, the subgraph overlaps can be an advantage in applications. For example, when searching for clusters in microarray, it is interesting to allow a gene to interact with different groups of genes, which is not the case with classical gene clustering.

For those reasons we decided to enhance this technique in order to apply it to different problematics.

Before implementing, we examined some theoretical algorithmic issues. We started by studying clique separator decomposition extensively and published a survey [9]:

An introduction to clique minimal separator decomposition.
A. Berry, R. Pogorelcnik, and G. Simonet.
Algorithms, 3:197–215, 2010.

This explains the decomposition in detail and contains detailed algorithms which are destined to help practicians with the implementation. This work is described in Chapter 2.

Tarjan [39] used the property that a clique separator of a graph G is a clique separator of any minimal triangulation of G. Thus, finding the clique separators of a graph can be done by first computing a minimal triangulation, then finding the minimal separators of this triangulation. The second step of our work was to examine how we could generate the minimal separators of a chordal graph. This gave birth to paper [7] described in Chapter 3:

A simple algorithm to generate the minimal separators and the maximal cliques of a chordal graph.
A. Berry and R. Pogorelcnik.
Information Processing Letters, 111(11):508 – 511, 2011.

Once we had a simple and efficient algorithm to detect clique minimal separators in a chordal graph, we investigated a way to represent the clique minimal separators of a minimal triangulation H of the input graph G, in order to efficiently test the completeness of the separators of G. In this process, we used the article of Blair and Peyton [12] which was a real breakthrough for us. In this article, they precisely define the *clique tree* which represents a chordal graph in a compact way, nodes represent cliques and there is an edge between two nodes if the intersection between them is a clique minimal separator. We define an *atom tree* of a graph as a generalization of the clique tree of a chordal graph: its nodes are the atoms obtained by clique minimal separator decomposition, and its edges correspond to the clique minimal separators of the graph.

Once an atom tree was defined, we developed an algorithm to compute it using clique tree of a minimal triangulation. For that we proved that merging cliques of the clique tree linked by an edge representing a non clique separator of the original graph leads to an atom tree.

In addition, we worked on algorithms allowing a faster merging process based on the atom tree structure and properties.

The implementation of the triangulation and the clique tree gave rise to an extension of the Blair and Peyton algorithm to compute the triangulation of a graph and one of its clique trees in a single pass. This in turn led to compute an atom tree of a graph in a single pass. The main advantages of this algorithm is the fact it is easy to implement and for our biological purposes it allowed us to investigate atoms as they are constructed, thus gaining time for large datasets. This work is described in Chapter 4, and led to a submitted article:

Efficiently computing clique separator decomposition using an atom tree.
A. Berry , R. Pogorelcnik, G. Simonet.
Submitted to Discrete Applied Mathematics.

During the thesis, we used Galois lattices to represent biological informations to add some knowledge on clusters produced by clique minimal separator decomposition on microarray data and for data mining purposes. This work which is not exposed here prompted us to apply clique minimal separator decomposition to the bipartite graph corresponding to the binary relation of a Galois lattice. We applied clique minimal separator decomposition to decompose this bipartite graph that has the particularity to have separators of size one or two which can be found in a linear time. Using this process we obtain atoms which correspond to smaller relations. This work, described in Chapter 5, was presented in an article [8] and was selected for submission in the special issue of AMAI:

Vertical decomposition of a lattice using clique separators.
A. Berry, R. Pogorelcnik, and A. Sigayret.
The 8th International Conference on Concept Lattices and
their Applications (CLA11), 2011.
Submitted to the special issue of AMAI.

The review written at the beginning of this thesis, attracted the attention of a bioinformatic team composed by Pierre-Alain Jachiet, Eric Bapteste and Philippe Lopez, who contacted us to work on *fused gene* detection. Fused genes are important evolutionary events. They yield valuable information to infer protein interactions and functions. Fused genes are genes similar to two other non-similar genes. Working on this application we determined that clique minimal separators were appropriate to modelize fused genes. We implemented a method, based on the detection of clique minimal separators with a variation of the extension of Blair and Peyton algorithm presented in Chapter 3, into a C++ program *MosaicFinder*, to find fused genes and to group them into families and to group genes they derived from that we call *family components*. The program, detailed in Chapter 6, can be downloaded at [30] and an article explaining it is submitted to Bioinformatics and currently in revision for them.

MosaicFinder: Identification of fused genes families in sequence
similarity networks.
P.A. Jachiet, R. Pogorelcnik, E. Bapteste and P. Lopez.
In revision for Bioinformatics.

We worked on level expression network data using our new representation, the atom tree, to extract information on experimental data. The implementation of the decomposition allowed us to detect atoms composed by genes that have a variation of their expression along time. The idea is to represent expression level measures of a large amount of genes by a graph and using clique minimal separators to decompose the graph. This process allows biologists to focus on a small subset of genes. This work was and is in collaboration with differents teams of INRA, and specially with the private partner BIOGEMMA. Two papers are under way, but due to the confidentility of the data, this work will just be explained on some partial examples using open data.

To summarize, the thesis is organized as follows:

- Chapter 1 gives some general graph, lattice and biological notions.
- Chapter 2 provides a precise definition of clique minimal separators, decomposition, properties and algorithms to find the clique minimal separators of a graph efficiently using known methods.
- In Chapter 3, we detail our work to find clique minimal separators and maximal cliques using LEX-BFS and MCS in a chordal graph.
- In Chapter 4, we recall the clique tree definitions and properties, as well as the algorithm to produce one. The rest of this chapter is devoted to the new notion of Atom Tree. We describe this tool and the different ways to compute it.
- Chapter 5 explains how this decomposition can be used on Galois lattices to break up some hard problems and optimize the graphic visualization.
- Chapter 6 emphasizes the detection of fused gene in a similarity network.
- Chapter 7 shows how the atom tree can be useful for the extraction of genes of interest in an expression level network.

Chapter 1

Preliminaries

Contents

1.1 Graph notions

In this thesis we assume readers are familiar with the elementary graph notions. Let $G = (V, E)$ be an undirected graph with V a set of vertices, $|V| = n$, and E a set of edges, $|E| = m$, an edge $\{x, y\}$ is denoted xy. The *complement* $\overline{G} = (V, \overline{E})$ of graph $G = (V, E)$ is the graph where, for any pair of vertices $\{x, y\}$ such that $x \neq y$, $xy \in \overline{E}$ if and only if $xy \notin E$. If $X \subset V$ is a set of vertices of $G = (V, E)$, $G(X)$ denotes the *subgraph induced* by X (which is a graph whose vertices are the elements of X and such that xy is an edge of $G(X)$ if $xy \in X$ and $xy \in E$). Two vertices x and y are said to be *adjacent* if $(x, y) \in E$. A *path* is a sequence $(x_1, x_2, ..., x_k)$ of different vertices such that for $i < k$, x_i is adjacent x_{i+1}. A *chord* in a path is an edge $x_i x_j$ where $j > i$ and $j \neq i + 1$. A path is *chordless* if it has no chord. A *cycle* is a sequence $x_1, x_2, ..., x_k, x_1$ of vertices, with $x_1, x_2, ..., x_k$ all different, such that for $i < k$, x_i is adjacent to x_{i+1} and x_k is adjacent to x_1. A graph is *connected* if there is a path between any pair of vertices. When a graph is disconnected, a maximal connected subgraph is a *connected component.* A *2-pair* is a pair $\{x, y\}$ of non-adjacent vertices such that every chordless path from x to y is of length 2.

Neighbourhood:

The neighbourhood of a vertex x in G is $N_G(x) = \{y \neq x \mid xy \in E\}$; we omit subscript G when it is clear from the context which graph we work on.

- A vertex is *isolated* if it has no neighbour.
- The *closed neighbourhood* of a vertex x is $N_G[x] = N_G(x) \cup \{x\}$.
- The neighbourhood of a set of vertices X is $N_G(X) = \bigcup_{x \in X} N_G(x) \setminus X$, and its closed neighbourhood is $N_G[X] = N_G(X) \cup X$.
- Let $X \subseteq V$ be a set of vertices, we will call *common neighbourhood*, denoted $CN(X)$, the intersection of the neighbourhoods of all the vertices of X.
$$CN(X) = \bigcap_{x \in X} N(x)$$
- A *module* is a set of vertices X such that for all $x, y \in X$ we have $N(x) \cap N(X) = N(y) \cap N(X)$

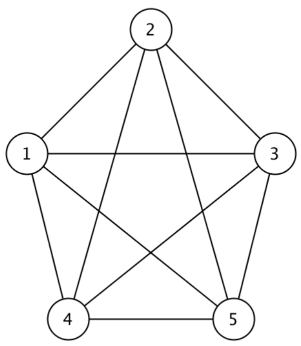

Figure 1: A clique of size 5, all vertices see the others

Clique:

A clique is a set of vertices which are pairwise adjacent. A vertex x is said to be *simplicial* if the subgraph induced by $N(x)$ is a clique.

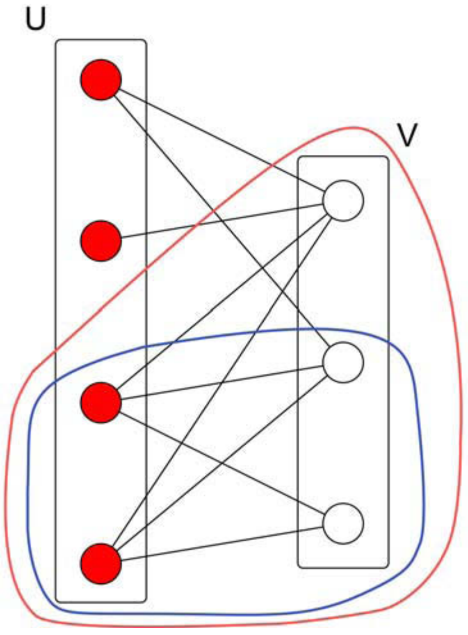

Figure 2: A bipartite graph with two vertex sets U and V. The vertex set circled in blue is a biclique included in a maximal biclique circled in red.

Bipartite graph:

- A *bipartite graph* G = (U +V, E), is built on two vertex sets, U and V, with no edge between vertices of U and no edge between vertices of V.
- A *biclique* of a bipartite graph $G = (U + V, E)$ is an induced subgraph $G' = (U' + V')$ with all the edges between vertices from U' and vertices from V'.

Chordal graph:

A graph is *chordal* or *triangulated* if it has no chordless cycle of length more than 3.

Perfect elimination ordering (peo):
A graph is chordal if and only if one can repeatedly find a simplicial vertex, and remove it, until no vertex left [33]. Since at each step a vertex is removed, this defines a series of transitory graphs, the simplicial vertex is found in this transitory graph. This process is called a *simplicial elimination scheme*, and defines an ordering on the vertices called *perfect elimination ordering (peo).*

In this work we will use the algorithm introduced by Rose, Tarjan and Lueker, so called *Lexicographic Breadth First Search* Lex-BFS [33] Algorithm Lex-BFS and a simplification of this algorithm *Maximum Cardinality Search* MCS [40] Algorithm MCS.

Algorithm 1: Lexicographic breadth first search - Lex-BFS [33]

```
Input: A graph $G = (V, E)$.
Output: A perfect elimination ordering $\alpha$ of $G$.
forall the vertices $v \in V$ do $L(v) = \emptyset$;
for $i = n$ downto 1 do
    Choose an unnumbered vertex $v$ with largest label;
    $\alpha(v) \leftarrow i$;
    foreach unnumbered neighbour $x$ of $v$ do
        Add $i$ to $label(x)$;
```

Algorithm 2: Maximum Cardinality Search - MCS [40]

```
Input: A graph $G = (V, E)$.
Output: A perfect elimination ordering $\alpha$ of $G$.
forall the vertices $v \in V$ do $w(v) = 0$;
for $i = n$ downto 1 do
    Choose an unnumbered vertex $v$ of maximum weight $w$;
    $\alpha(v) \leftarrow i$;
    foreach unnumbered neighbour $x$ of $v$ do
        $w(x) \leftarrow w(x) + 1$;
```

Minimal triangulation:
A *triangulation* is a chordal completion of a graph.

Let $G = (V, E)$ be a non chordal graph. A chordal graph $H = (V, E + F)$ is called a *triangulation* of G, and F is called the *fill*. The triangulation is said to be:

- *minimal* if for no proper subset F' of F such as $H' = (V, E + F')$ is chordal.
- *minimum* if no other minimal triangulation has less fill edges. Computing a minimum triangulation is NP-complete [44].

Minimal elimination ordering (meo):
One way to compute a triangulation is to force the graph into having a perfect elimination ordering by repeatedly picking a vertex and adding any edges missing in its neighbourhood and removing it, until no vertex is left. This will yield a triangulation whose fill is the set of all edges added in this process. Efficient algorithms have been designed for doing this, Lex M [33] and MCS-M [3] . Both these algorithms yield a minimal triangulation H of the input graph G and an ordering α that is both a minimal elimination ordering of G and a perfect elimination ordering of H.

***Madj*(*x*)** :
Given a chordal graph G and an ordering α of G which is a bijection $\alpha : V \rightarrow \{1, 2, ..., 3\}$. We use α to index the vertex set, such that, $\alpha(v_i) = i$ for $1 \leq i \leq n$ where i will be referred to as the *number* of v_i. Let $v_1, v_2, ..., v_n$ be an ordering of V, for any vertex v_i, $Madj_G(v_i)$ is the *monotone adjacency set* of v_i:

$$Madj_G(v_i) = \{v_j \in N_G(v_i) \mid j > i\}$$

Algorithm 3: Lex M [33]

```
Input: An undirected graph $G = (V, E)$.
Output: A minimal elimination ordering $\alpha$ on $V$ and a minimal
        triangulation $H = (V, E + F)$ of $G$.
$F \leftarrow \emptyset$;
forall the vertices $v \in V$ do $L(v) \leftarrow \emptyset$;
for $i = n$ downto 1 do
    Choose an unnumbered vertex $x$ of $G$ of maximal label ;
    $\alpha(x) \leftarrow i$;
    $Y \leftarrow N_G(x)$;
    foreach unnumbered vertex $y$ of $G$ not belonging to $N_G[x]$ do
        if there is a path from $x$ to $y$ in $G$ such that every internal
        vertex on the path has a label strictly smaller than $L(y)$ and is
        unnumbered then
            $F \leftarrow F \cup \{xy\}$; $Y \leftarrow Y \cup \{y\}$;
        foreach $y \in Y$ do Add $i$ to $L(y)$;
    $H \leftarrow (V, E + F)$;
```

Algorithm 4: MCS-M [3]

```
Input: An undirected graph $G = (V, E)$.
Output: A minimal elimination ordering $\alpha$ on $V$ and a minimal
        triangulation $H = (V, E + F)$ of $G$.
$F \leftarrow \emptyset$;
forall the vertices $v \in G$ do $w(v) \leftarrow 0$;
for $i = n$ downto 1 do
    Choose an unnumbered vertex $x$ of $G$ of maximal weight $w$;
    $Y \leftarrow N_G(x)$; $\alpha(x) \leftarrow i$;
    foreach unnumbered vertex $y$ of $G$ not belonging to $N_G[x]$ do
        if there is a path from $x$ to $y$ in $G$ such that every internal
        vertex on the path has a weight strictly smaller than $w(y)$ and
        is unnumbered then
            $F \leftarrow F \cup \{xy\}$; $Y \leftarrow Y \cup \{y\}$;
    foreach $y$ in $Y$ do $w(y) \leftarrow w(y) + 1$;
$H \leftarrow (V, E + F)$;
```

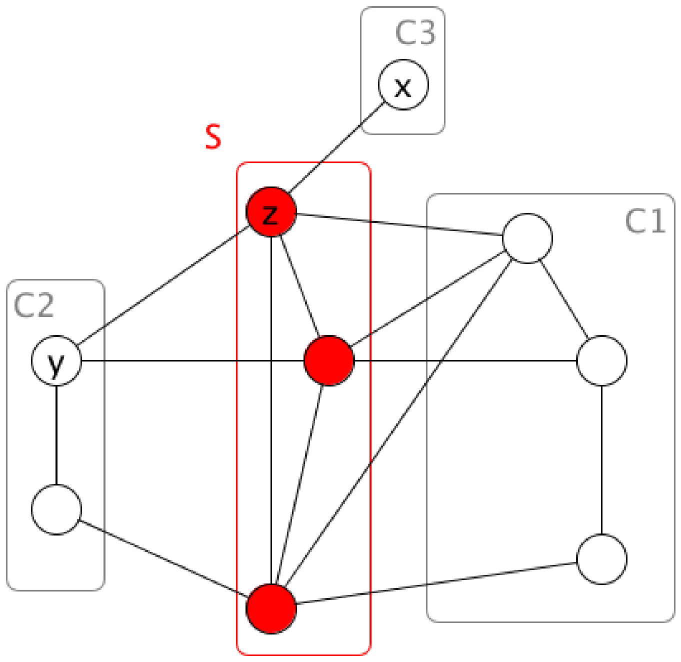

Figure 3: The set of red vertices form a clique minimal separator. Indeed its removal lead to two connected components $C1$ and $C2$ and $N(C1) = N(C2) = S$ and a connected component $C3$. The red vertices is an xy-separator non minimal as it includes z which is a minimal xy-separator

Separators:
A set S of vertices of a connected graph G is a *separator* if $G(V \setminus S)$ is not connected.

- A separator S is an *xy-separator* if x and y lie in two different connected components of $G(V \setminus S)$.
- S is a *minimal xy-separator* if S is an xy-separator and no proper subset S' of S is also an xy-separator.
- Equivalently, in a connected graph, S is a minimal separator if and only if $G(V \setminus S)$ has at least two connected components $C1$ and $C2$ such that $N_G(C1) = N_G(C2) = S$, called *full components*. A *clique minimal separator* is a minimal separator which is a clique.

- A graph is chordal if and only if all its minimal separators are cliques.
- In a bipartite graph, the clique minimal separators are of size one or two. They are called *articulation points* when they are of size one, and we will call them *articulation edges* when they are of size two.
- We will refer to *minimal separator generators* in a chordal graph H: given a perfect elimination ordering α of H, x is called a *minimal separator generator* if $Madj(x)$ is a minimal separator.

Clique separator decomposition:
This process introduced in [39] repeatedly finds a clique separator S and copies it into the connected components of $G(V \setminus S)$. When only minimal separators are used, the decomposition is unique and the subgraphs obtained at the end, called *atoms*, are exactly the maximal subgraphs containing no clique separator [25]. This decomposition is detailed in Chapter 2.

1.2 Galois lattice notions

Relation:
Let $\mathcal{O}$ and $\mathcal{A}$ be finite sets, a relation is a subset of the Cartesian product $\mathcal{O} \times \mathcal{A}$. The relation can be represented by a matrix as in Figure 4.
A relation $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ is associated with a bipartite graph G, which is denoted by $Bip(\mathcal{R})$; for $x \in \mathcal{O}$ and $y \in \mathcal{A}$, (x, y) is in $\mathcal{R}$ if and only if xy is an edge of G.

Context:
A context is a triplet $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ corresponding to a binary relation $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ on a finite set $\mathcal{O}$ of *objects* and a finite set $\mathcal{A}$ of *attributes*.

	a	b	c	d	e
1	×	×	×	×	
2		×	×		×
3				×	
4	×		×		×
5			×		

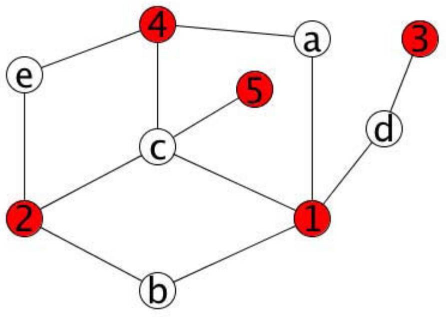

Figure 4: A relation $\mathcal{R}$ and the corresponding bipartite graph $Bip(\mathcal{R})$.

Concept:
Let $C = (\mathcal{O}, \mathcal{A}, \mathcal{R})$ be a context, $O_1 \subseteq \mathcal{O}$ and $A_1 \subseteq \mathcal{A}$. The pair (O_1, A_1) is a *concept* or *maximal rectangle* of C if:

$$O_1 = \{o \in \mathcal{O} : (o,a) \in \mathcal{R} \text{ for all } a \in A_1\},$$
$$A_1 = \{a \in \mathcal{A} : (o,a) \in \mathcal{R} \text{ for all } o \in O_1\}$$

and

$$\text{for all } x \in \mathcal{O} - O_1, \exists y \in A_1 \text{ such that } (x,y) \notin \mathcal{R},$$

$$\text{for all } y \in \mathcal{A} - A_1, \exists x \in O_1 \text{ such that } (x,y) \notin \mathcal{R}.$$

O_1 is called the *extent* and A_1 the *intent* of the concept (O_1, A_1).
The maximal bicliques of $bip(\mathcal{R})$ correspond to the concepts.

Lattice:
A *lattice* is a partially ordered set in which every pair of elements has both a lowest upper bound and greatest lower bound.

Galois lattice:
Let $C = (\mathcal{O}, \mathcal{A}, \mathcal{R})$ be a context. Let (O_1, A_1) and (O_2, A_2) be two concepts of C. We denote $(O_1, A_1) \preceq (O_2, A_2)$ if and only if $O_1 \subseteq O_2$ (or, equivalently $A_1 \supseteq A_2$). The concepts of C, ordered with $\preceq$, define the *Galois* lattice of C (also called *concept lattice*).

Hasse diagram:
The Hasse diagram is a directed graph representing a partial order omitting transitivity and reflexivity edges. The Hasse diagram of a Galois lattice can be drawn with two strategies:

- top-down: from the maximal extent to the minimal one.
- bottom-up: from the minimal extent to the maximal one.

Example 1. *Figure 5 shows the Hasse diagram of the Galois lattice of the relation in Figure 4.*

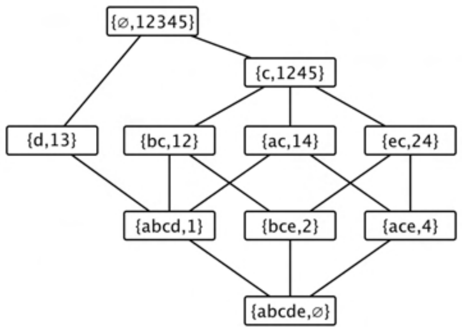

Figure 5: The Galois lattice of the relation $\mathcal{R}$ of Figure 4 .

Atom (resp. *co-atom*):
An *atom* (resp. *co-atom*) of $\mathcal{L}(\mathcal{R})$ is a concept covering the minimum element (resp. covered by the maximum element).

Introducer:
The *object-concept* associated with a given object x is defined as the concept which intent is $\mathcal{R}(x)$. The *attribute-concept* associated with a given attribute y is defined as the concept which extent is $\mathcal{R}^{-1}(y)$. The object-concept of x is the first (from bottom to top) concept having x in its extent. The attribute-concept of y is the last (from bottom to top) concept having y in its intent. Object-concepts and attribute-concepts are also called *introducers.*

1.3 Biological notions

Fused genes

Fused genes, as illustrated in Figure 6, result from the fusion of previously physically unrelated genes, or parts of their sequences. We called the process that leads to a fused gene a *fusion event*. Gene fusion can provide to new functions [26], a greater catalytic activity or a more efficient co-regulation between proteins whose genes have been fused. When such fused genes increase the fitness of their carrier, they are maintained in genomes and give rise to new gene families. Numerous cases of gene fusions have been reported, both in Prokaryotes [29] and in Eukaryotes [46].

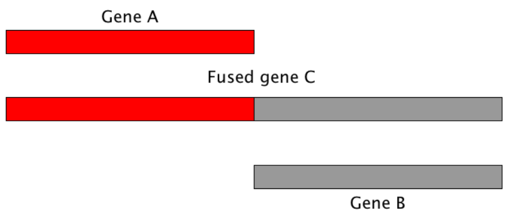

Figure 6: A fused gene C and the two sequences it stems from. A and B are similar to disjoint parts of C. A and B are dissimilar.

Microarray

A microarray is a solid surface where are bound synthesized probes. This allows to monitor the expression level of thousands of genes simultaneously in a cell or tissue in different conditions or at different times. The expression profiling allows to extract from a large number of genes some whose expression responds to a stimulus for further investigation.

Biological network

Applications of the thesis were on biological networks. These networks are graphs where vertices represent gene or protein sequences and where the edges represent a relation between them. This relation can be of sequence similarity, interaction, expression level similarity. Here we worked on:

- **expression level network:** Using technologies such as microarray, we can measure the expression level of genes in a specific condition. Using output data, we can compute a distance between all gene expression levels with the widely used distance measures Euclidean distance or Pearson correlation distance [20].

  – Euclidean Distance Measure

  $$D(x_i, x_j) = \sqrt{\sum_{l=1}^{d} (x_{il} - x_{il})^2}$$

  – Pearson Correlation-Based Distance Measure

  $$Cor(x_i, x_j) = \frac{\sum_{l=1}^{d} (x_{il} - \mu_{x_i})(x_{jl} - \mu_{x_j})}{\sqrt{\sum_{l=1}^{d} (x_{il} - \mu_{il})^2} \sqrt{\sum_{l=1}^{d} (x_{il} - \mu_{il})^2}}$$

  μ_{xi} and μ_{yi} represent the sample means. The distance between x_i and x_j is computed as:

  $$D(x_i, x_j) = 1 - Cor(x_i, x_j)$$

- **Sequence similarity network:** To represent relationships between proteins, a sequence similarity network can be constructed. The proteins are represented as vertices in a network (graph), and the significant similarities as edges connecting the proteins. One way to measure the similarity of proteins is to evaluate the similarity of their sequences by a pairwise sequence alignment. The region of sequence similarity detected may describe a relationship (evolutionary, functional...) between the proteins.

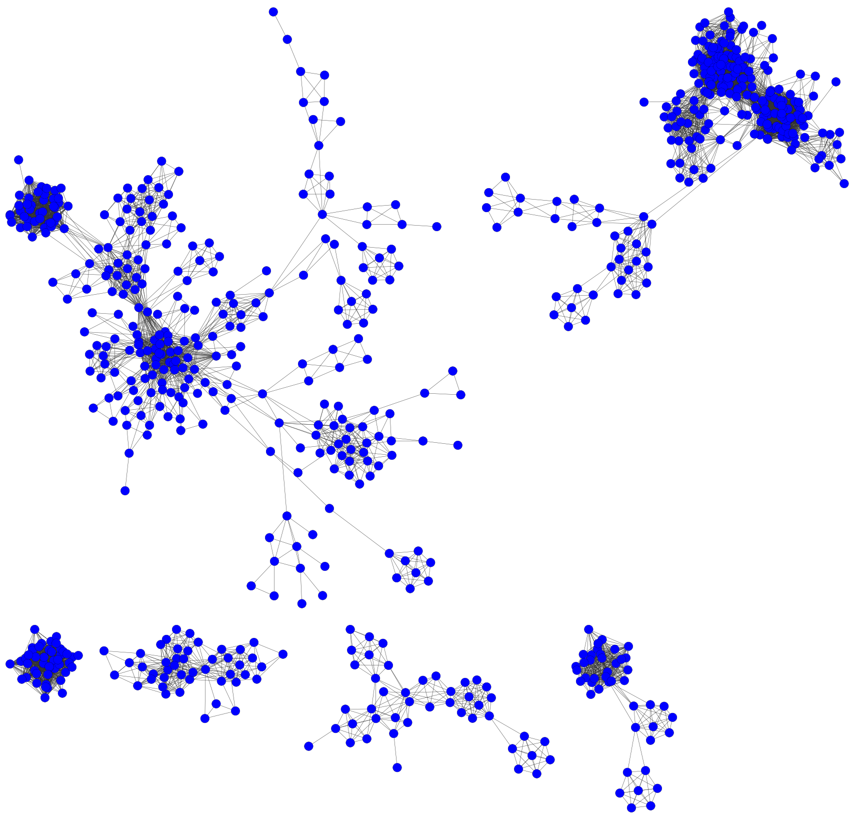

Figure 7: An example of a similarity network.

Chapter 2

Clique minimal separator decomposition: State of the art

Contents

Corresponding article:
An introduction to clique minimal separator decomposition.
A. Berry, R. Pogorelcnik, and G. Simonet.
Algorithms, 3:197–215, 2010.

In the genesis of the thesis, we studied the clique minimal separator decomposition in detail to improve the known algorithms to apply them on biological data. This led to the creation of article [9]. This chapter recalls this article to explain the history, definitions and to give algorithms to compute the decomposition as it was defined by Tarjan [39] and refined by Leimer [25].

2.1 History

Clique minimal separator decomposition was used to solve hard problems with a divide and conquer approach. The input graph is decomposed into smaller subgraphs called atoms, then the problem is solved on each subgraph. Finally results are merged until obtaining the input graph solution.

In 1976, Gavril [19] described the class of clique separable graphs in the context of solving in a polynomial time the hard problems of minimum colouring and maximum clique. In 1980, the problem of finding a clique separator in an arbitrary graph was addressed by Whitesides [42], who presented an $O(n^3)$ algorithm to find one clique separator. In 1983, Tarjan [39] addressed the problem of finding a decomposition of an arbitrary graph using clique separators. He noted that using Whitesides' algorithm would require $O(nm^3)$ time. He proposed an $O(nm)$ time algorithm to do this, by showing that no fill edge of a minimal triangulation can join two connected components defined by a clique separator. He proved that using a minimal elimination ordering, a clique separator decomposition can be computed in $O(nm)$ time. Tarjan left open the question of defining a unique clique separator decomposition. In 1990, Dahlhaus, Karpinsky and Novick [14] proposed a parallel algorithm for clique separator decomposition. In 1990, concurrently, Leimer [25], described how to obtain an optimal and unique decomposition using the clique minimal separators of the graph. Leimer also described how to use Lex M to find the clique minimal separator decomposition in $O(nm)$ time.

2.2 Definitions and examples

A decomposition step was defined by Tarjan [39] as follows:

Definition 1 (Decomposition Step). *Let $G = (E, V)$ be a undirected graph. Let A, B, C be a partition of V with A and B non empty and C is a clique. Then (A, B, C) is a decomposition of G into subgraph $G_1 = G(A \cup C)$ and $G_2 = G(B \cup C)$, separated by C.*

Repeating this decomposition step on a subgraph which has a clique separator, until none of the subgraphs has a clique separator, results in clique separator decomposition.

Definition 2 (Clique separator decomposition)**.** *The recursive decomposition of G using the Decomposition Step of Definition 1, until no further decomposition is possible, decomposes G into a collection of atoms, each a subgraph of G containing no clique separator.*

Tarjan showed that this decomposition is not unique, by the following counter example:

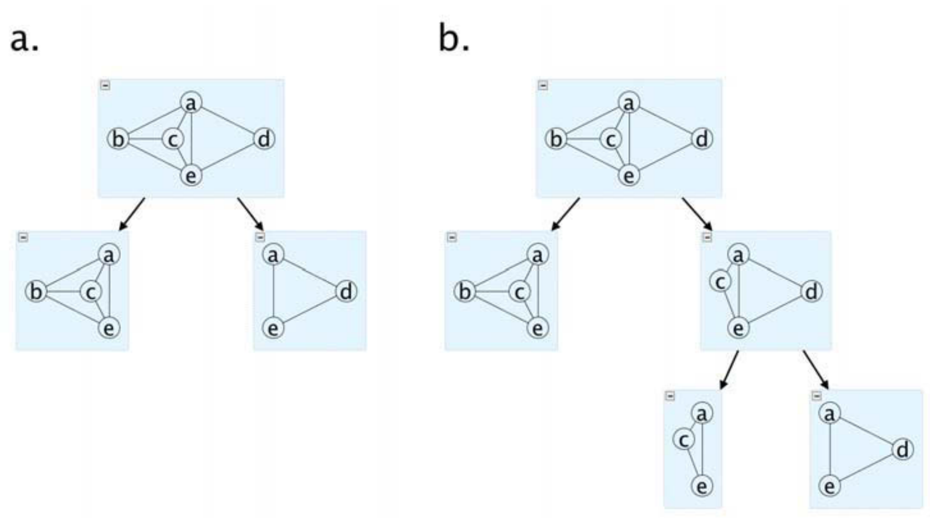

Figure 8: A graph with different decompositions. (a) A decomposition using the clique minimal separator $\{a, e\}$ (b) Another decomposition using the non-minimal clique separator $\{a, c, e\}$.

Tarjan left open the question of defining a unique clique separator decomposition. Leimer proved in [25] that by using only clique separators, which are minimal separators, at each decomposition step, the decomposition is unique.

Example 2. *In Figure 8 the example of Tarjan shows that his decomposition is not unique. The separator $\{a, c, e\}$ used in Figure 8 (b) is not minimal since $\{a, e\}$ is included in this separator and is also a bd-separator. Figure 8 (a) is the unique decomposition using Leimer's extension.*

Leimer gave the following characterization of the atoms obtained when using only clique minimal separators (he calls these atom *mp-subgraphs*). In the rest of this thesis, we will call these *atoms*.

Characterization 1. *An atom of a graph is a maximal induced subgraph containing no clique minimal separator.*

Note that the resulting atoms do not depend of the order in which the decomposition steps are executed. This is because of the following property:

Property 1. *[25] Let $G = (V, E)$ be a connected graph, let S be a clique minimal separator of G and let C be a full component of S in G; then the set of atoms of G is the disjoint union of the set of atoms of $G(C \cup S)$ and of $G(V \setminus C)$.*

2.3 Computing the decomposition

To decompose a graph G efficiently, a minimal triangulation H is required. Once H is computed it is sufficient to have a perfect elimination α of H: according to [32], for each clique minimal separator S of H, there is a vertex x such that $Madj_H(x) = S$ (x is then called a *minimal separator generator*). Once we find the separators of the triangulated graph H, we look for the minimal separators of H that are cliques in G due to the following property:

Property 2. *[25, 28, 5] Let $G = (V, E)$ be a connected graph, let $H = (V, E \cup F)$ be a minimal triangulation of G. Then the clique minimal separators of G are the minimal separators of H that are cliques in G; any minimal separator S of H is a minimal separator of G, and $G(V \setminus S)$ has the same connected components as $H(V \setminus S)$ with the same neighbourhoods.*

Based on these properties, we describe an algorithm to compute the unique decomposition using only clique minimal separators of H. The algorithm consists in three steps:

1. Compute a minimal triangulation H of the input graph G.

2. Compute a perfect elimination ordering α of H and the minimal separator generators of H with respect to α.

3. Process the vertices from 1 to n using α. For each vertex x, if x is a minimal separator generator of H, let X be a vertex set of the connected component of $V \backslash Madj_H(x)$ containing x, and $Y = V \backslash (Madj_H(x) \cup X)$. If $Madj_H(x)$ is a clique in G and $Y \neq \emptyset$, then $Madj_H(x)$ is a clique minimal separator of G: decompose G into $G_1 = G(X \cup Madj_H(x))$ and $G_2 = G(Y \cup Madj_H(x))$, store G_1 as an atom, and replace G with G_2.

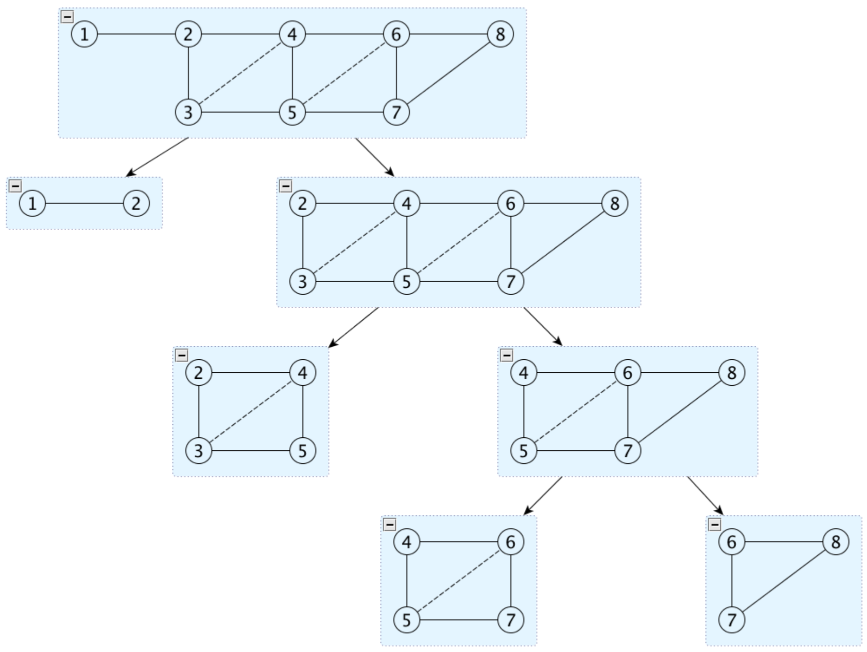

Figure 9: Decomposition using clique minimal separators. We start by a triangulation using a minimal elimination ordering.

Example 3. *In Figure 9 the vertices are numbered by a MCS. There are five clique minimal separators in the triangulated graph:* $\{2\}, \{4,3\}, \{4,5\}, \{5,6\}, \{6,7\}$. *The clique minimal generators are* $1,2,3,4,5$. $Madj(1) = \{2\}$ *is a clique of* G. G *is decomposed into the graphs* $G(\{1,2\})$ *and* $G(\{2,3,4,5,6,7,8\})$. $Madj(2) = \{3,4\}$ *is not a clique of* $G(\{2,3,4,5,6,7,8\})$. *The graph* *is not decomposed.* $Madj(3) = \{4,5\}$ *is a clique of* $G(\{2,3,4,5,6,7,8\})$*. This graph is decomposed into the graphs* $G(\{2,3,4,5\})$ *and* $G(\{4,5,6,7,8\})$*.* $Madj(4) = \{5,6\}$ *is not a clique of* $G(\{4,5,6,7,8\})$*. This graph is not decomposed.* $Madj(5) = \{6,7\}$ *is a clique of* $G(\{4,5,6,7,8\})$*. This graph is decomposed into the graphs* $G(\{4,5,6,7\})$ *and* $G(\{6,7,8\})$*.*

LEX M and MCS-M both yield a minimal triangulation H of the input graph G and an ordering α that is both a minimal elimination ordering of G and a perfect elimination ordering of H [6]. Thus, we can implement Steps 1 and 2 in a single pass using an ordering provided by algorithm MCS-M. Once the ordering and the triangulation are computed, we use a second algorithm which generates the atoms.

We give below a detailed algorithm, MCS-M+, to compute the triangulation, the perfect elimination ordering and the set of minimal separator generators which is MCS-M with an optimal implementation of "if there is a path from x to y in G such that every internal vertex on the path has a label strictly smaller than $label(y)$ and unnumbered".

We do this in a similar way as in the implementation of LEX M given in [33] by a single search in G. For each label value j, set $reach(j)$ contains the reached vertices having label j, as well as the vertices having a label strictly smaller than j reached from a vertex having label j.

Moreover in MCSM+, we compute the set X of vertices that generate the minimal separators of the triangulation H, as described in Chapter 3. The idea behind this, which we detail in Chapter 3, is that as long as the labels of the chosen vertices keep getting larger, we are inside a clique of H; when suddenly the label of the chosen vertex x stops getting larger, x is a minimal separator generator of H. Each such vertex x is added to set X.

Once we have obtained the ordering α, the corresponding minimal triangulation H of G and the set X of minimal separators generators of H, we run through the vertices from 1 to n as explained above, using the transitory subgraphs G' and H' of G and H, initialized as G and H, respectively. At each step i processing vertex $x = \alpha(i)$, we check whether x is in the set X.

Algorithm 5: MCS-M+

```
Input: An undirected graph $G = (V, E)$.
Output: A minimal triangulation $H = (V, E + F)$ of $G$, a minimal
        elimination ordering $\alpha$ of $H$ and the set $X$ of vertices which
        generate a clique minimal separator of $H$.
init: $F \leftarrow \emptyset$; $G' \leftarrow G$;
Initialize the labels of all vertices as 0; $s \leftarrow -1$ ; $X \leftarrow \emptyset$ ;
for $i = n$ downto 1 do
    Choose a vertex $x$ of $G'$ of maximal label ;
    $Y \leftarrow \emptyset$ ; $Z \leftarrow N_{G'}(x)$;
    if $label(x) \leq s$ then $X \leftarrow X + \{x\}$;
    $s \leftarrow label(x)$ ;
    Mark $x$ reached and mark all other vertices of $G'$ unreachable;
    for $j = 0$ to $n - 1$ do $reach(j) \leftarrow \emptyset$;
    foreach $y$ in $N_{G'}(x)$ do
        Mark $y$ reached;
        Add $y$ to $reach(label(y))$;
    for $j = 0$ to $n - 1$ do
        while $reach(j) \neq \emptyset$ do
            Remove a vertex $y$ from $reach(j)$;
            foreach $z$ in $N_{G'}(y)$ do
                if $z$ is unreached then
                    Mark $z$ reached;
                    if $label(z) > j$ then
                        $Y \leftarrow Y + \{z\}$;
                        $Z \leftarrow Z + \{z\}$;
                        Add $z$ to $reach(label(z))$;
                    else
                        Add $z$ to $reach(j)$;
    foreach $y$ in $Y$ do $F \leftarrow F + \{xy\}$;
    foreach $z$ in $Z$ do
        $label(z) \leftarrow label(z) + 1$;
    $\alpha(x) \leftarrow i$ ;
    Remove $x$ from $G'$;
$H \leftarrow (V, E + F)$.
```

If it is, the neighbourhood $N_{H'}(x)$ of x in H' is a minimal separator of H; we check whether it is a clique in G. If it is, then S is a clique minimal separator of G. In that case we compute the connected component C of $G(V \setminus S)$ which contains x; $G'(S \cup C)$ is an atom [39], and is stored as such; C is then removed from G'. In any case, we then remove x from H', as described by the following algorithm.

Algorithm 6: Atoms

```
Input: A graph $G = (V, E)$, a minimal triangulation $H = (V, E + F)$
       of $G$, the set $X$ of vertices that are minimal separator
       generators of $H$, a peo $\alpha$ of $H$.
Output: The set $\mathscr{A}$ of atoms of $G$, the set $\mathscr{S}_c$ of clique minimal
        separators of $G$.
$G' \leftarrow G$ ; $H' \leftarrow H$ ; $\mathscr{A} \leftarrow \emptyset$ ; $\mathscr{S}_H \leftarrow \emptyset$ ; $\mathscr{S}_c \leftarrow \emptyset$ ;
for $i = 1$ to $n$ do
    $x \leftarrow \alpha(i)$ ;
    if $x \in X$ then
        $S \leftarrow N_{H'}(x)$;
        if $S$ is a clique in $G$ then
            $\mathscr{S}_c \leftarrow \mathscr{S}_c \cup \{S\}$;
            $C \leftarrow$ the connected component of $G' \setminus S$ containing $x$ ;
            $\mathscr{A} \leftarrow \mathscr{A} + \{G'(S \cup C)\}$; $G' \leftarrow G' \setminus C$ ;
    Remove $x$ from $H'$;
$\mathscr{A} \leftarrow \mathscr{A} + \{G'\}$.
```

Chapter 3

Generating the clique minimal separators and the maximal cliques of a chordal graph

Contents

Corresponding paper:
A simple algorithm to generate the minimal separators and the maximal cliques of a chordal graph.
A. Berry and R. Pogorelcnik.
Information Processing Letters, 111(11):508 – 511, 2011.

At the beginning of this thesis we studied a way of finding clique minimal separators and maximal clique simply and quickly to allow us to implement the decomposition steps. We wrote an algorithm using Lex-BFS or MCS to obtain in a single pass the perfect elimination ordering, maximal cliques and minimal separators of a chordal graph, as well as the minimal separator generators. This algorithm is based on a theorem formulated and proved in this chapter.

3.1 Main theorem

In this section, we use graph search algorithms Lex-BFS and MCS, both generate a perfect elimination ordering. Both algorithms number the vertices from n to 1 (n is the number of vertices of the graph) and each vertex x carries a label, which is the list of numbers of the neighbours of x with a higher number for Lex-BFS or the cardinality of this list for MCS.

Definition 3. *Given a chordal graph G and a perfect elimination ordering α of G defined by either Lex-BFS or MCS, we call:*

- minimal separator generator *any vertex x_i with number i by α, such that $Madj(x_i)$ is a minimal separator of G and $label(x_i) \leq label(x_{i+1})$, where $x_{i+1} = \alpha(i+1)$.*
- maximal clique generator *a vertex y such that $Madj(y) \cup \{y\}$ is a maximal clique of G.*

Theorem 3.1.1. *Let α be a perfect elimination ordering defined by either Lex-BFS or MCS, let x_i be the vertex with number i, let x_{i+1} be the vertex with number $i+1$.*

a) x_i is a minimal separator generator if and only if $label(x_i) \leq label(x_{i+1})$.

b) x_{i+1} is a maximal clique generator if and only if $label(x_i) \leq label(x_{i+1})$ or $i+1=1$.

We now discuss the proof of theorem 3.1.1. We first present moplex elimination, which is a process that explains how both MCS and Lex-BFS scan the minimal separators and the maximal cliques of a chordal graph, then we prove Theorem 3.1.1 for Lex-BFS.

Definition 4. *[4] A* moplex *is a clique X which is a module and such that $N(X)$ is a minimal separator. We extend this definition to a clique whose neighbourhood is empty. A* simplicial moplex *is a moplex whose vertices are all simplicial.*

Property 3. *[4] Any chordal graph which is not a clique has at least two non-adjacent simplicial moplexes.*

Characterization 2. *[4] A graph is chordal if and only if one can repeatedly delete a simplicial moplex until the graph is a clique (which we will call the* terminal moplex*). We call this process* moplex elimination*.*

Note that moplex elimination on a chordal graph is a special case of simplicial elimination, since at each step a set of simplicial vertices is eliminated. Note also that for a connected graph G, the transitory elimination graph obtained at the end of each step remains connected.

Moplex elimination defines an ordered partition $(X_1, X_2, ..., X_k)$ of the vertices of the graph into the successive moplexes which are defined in the successive transitory elimination graphs. We will call this partition a *moplex ordering.*

Theorem 3.1.2. *[4] Let G be a chordal graph, let $(X_1, X_2, ..., X_k)$ be a moplex ordering of G. At each step $i < k$ of the elimination process finding moplex X_i in transitory graph G_i,*

- *$N_{G_i}(X_i)$ is a minimal separator of G,*
- *$X_i \cup N_{G_i}(X_i)$ is a maximal clique of G.*

The terminal moplex X_k is a maximal clique.
There are no other minimal separators or maximal cliques in G.

To prove this, we will first recall a result from Rose [32]:

Property 4. *[32] Let G be a chordal graph, let α be a perfect elimination ordering of G, let S be a minimal separator of G. Then there is some vertex x such that $Madj(x) = S$, and in every full component C of S, there is some vertex y such that $S \subseteq N(y)$ (such a vertex is called a* confluence point *of C).*

Proof of Theorem 3.1.2. $N_{G_i}(X_i)$ is a minimal separator of G: in transitory graph G_i, $S_i = N_{G_i}(X_i)$ is by definition a minimal separator of G_i, with at least two full components C_1 and C_2, each containing a confluence point, which we will call x_1 and x_2. Suppose S_i is not a minimal separator of G. Then there must be a chordless path from x_1 to x_2 in G which contains no vertex of S_i. Let y be the first vertex of this path to be eliminated, at step $j < i$; the vertex y must be simplicial in G_j, but this is impossible, since y has two non-adjacent neighbours on the path. Thus every $N_{G_i}(X_i)$ with $i < k$, is a minimal separator of G, and by property 4, all minimal separators of G have thus been encountered.

$X_i \cup N_{G_i}(X_i) = K_i$ is a maximal clique of G: K_i is a maximal clique of G_i and thus a clique of G. Suppose it is not a maximal clique of G. Let x_j be a vertex belonging to the moplex X_j with $j < i$, of largest number which can be added to K_i to make a larger clique in G.

In G_j, the set $S_j = N_{G_j}(x_j)$ is a minimal separator, which contains all the vertices of K_i. But in G_j, S_j must have a full component C, disjoint from X_j; C must contain a confluence point z, which is adjacent in G_j all the vertices of S_j, and thus all the vertices of K_i. The confluence point z must belong to a moplex of number larger than j, which contradicts the assumption that X_j is of maximum number. Thus every $X_i \cup N_{G_i}(X_i)$ $(i < k)$ is a maximal clique of G.

The same argument can be given to prove that the terminal moplex is a maximal clique of G. $\square$

Note that for a given chordal graph G, there may be many different moplex orderings, but there is always the same number of moplexes, since this is the number of maximal cliques of the graph.

With any moplex ordering $(X_1, X_2, ..., X_k)$, we can associate a peo α by processing the moplexes from 1 to n and giving consecutive numbers to the vertices of a given moplex. Using α, we can define the minimal separators and maximal cliques as vertex neighbourhoods:

Property 5. *Let G be a chordal graph, let $(X_1, X_2, ..., X_k)$ be a moplex ordering of G, let α be a peo associated with this moplex ordering. Then in the course of a moplex elimination, at each step i that processes moplex X_i.*

- *The vertex x of moplex X_i whose number is the smallest by α defines a maximal clique $\{x\} \cup Madj(x)$ of G.*
- *The vertex y of moplex X_i whose number is the largest by α defines a minimal separator $Madj(y)$ of G.*

3.2 Generators of Lex-BFS

Lex-BFS defines a moplex ordering and an associated peo: [4] showed that Lex-BFS always numbers as 1 a vertex belonging to a moplex (which we call X_1). They also proved that the vertices of X_1 receive consecutive numbers by Lex-BFS. These properties are true at each step of Lex-BFS in the transitory elimination graph. Therefore, Lex-BFS defines a moplex elimination $(X_1, ..., X_k)$, by numbering consecutively the vertices of X_1, then numbering consecutively the vertices of X_2, and so forth:

Theorem 3.2.1. *[4] In a chordal graph, Lex-BFS defines a moplex ordering.*

Note that it is easy to deduce from [12] that MCS also defines a moplex ordering. Since the vertices of any transitory moplex X_i are numbered consecutively, when running Lex-BFS (numbering the vertices from n to 1), as long as the labels increase, we are defining a moplex, X_i. When the labels stop increasing (when numbering vertex y), then we have started a new moplex X_{i-1} which contains y. Using Property 5, we can deduce that Lex-BFS and MCS generate the minimal separators and the maximal cliques.

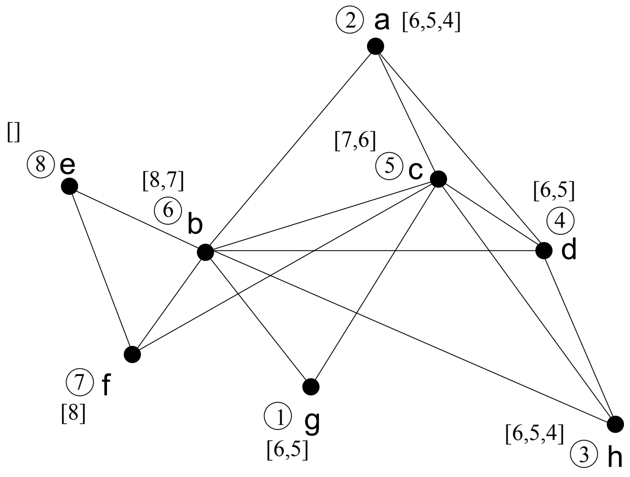

Figure 10: A graph with a Lex-BFS ordering. The circled numbers are labels of the vertices.

Example 4. *Figure 10 shows the numbers and labels of a Lex-BFS execution on a chordal graph.*

3.3 Algorithm

We can now derive from Theorem 3.1.1 a generalized algorithm to generate the minimal separators and the maximal cliques of a chordal graph.

In the Algorithm Minseps-Maxcliques, it is considered that either LexBFS or MCS is used. "Increment the label of y" is translated as $label(y) \leftarrow label(y) \cup \{i\}$ for LexBFS and as $label(y) \leftarrow label(y) + 1$ for MCS. The labels are all considered initialized at the beginning, as $\emptyset$ for LexBFS and as 0 for MCS.

$G_{NUM} \leftarrow G_{NUM} + \{x_i\}$ is shorthand for "$V_{NUM} \leftarrow V_{NUM} + \{x_i\}$; $G_{NUM} \leftarrow G(V_{NUM})$". In the same fashion, $G_{ELIM} \leftarrow G_{ELIM} - \{x_i\}$ is shorthand for $V_{ELIM} \leftarrow V_{ELIM} - \{x_i\}$; $G_{ELIM} \leftarrow G(V_{ELIM})$;

```
Algorithm 7: Minseps-Maxcliques
  Input: A chordal graph $G = (V, E)$.
  Output: Set $\mathscr{S}$ of minimal separators of $G$;
  Set $S$ of minimal separator generators of $G$;
  Set $\mathscr{K}$ of maximal cliques of $G$;
  Set $K$ of maximal clique generators of $G$;
  init: $G_{NUM} \leftarrow G(\emptyset)$; $G_{ELIM} \leftarrow G$;
  $\mathscr{S} \leftarrow \emptyset$; $S \leftarrow \emptyset$ ; $\mathscr{K} \leftarrow \emptyset$ ; $K \leftarrow \emptyset$;
  for $i = n$ downto 1 do
      Choose a vertex $x_i$ of $G_{ELIM}$ of maximum label;
      $G_{NUM} \leftarrow G_{NUM} + \{x_i\}$;
      if $i \neq n$ and $label(x_i) \leq \lambda$ then
          $//x_i$ is a min. sep. generator and $x_{i+1}$ is a max. clique
          generator $S \leftarrow S + \{x_i\}$; $\mathscr{S} \leftarrow \mathscr{S} \cup \{N_{G_{NUM}}(x_i)\}$;
          $K \leftarrow K + \{x_{i+1}\}$;
          $\mathscr{K} \leftarrow \mathscr{K} + \{(N_{G_{NUM}}(x_{i+1}) \cup \{x_{i+1}\}\}$;
      $\lambda \leftarrow label(x_i)$;
      foreach $y \in N_{G_{ELIM}}(x_i)$ do
          Increment the label of $y$;
      $G_{ELIM} \leftarrow G_{ELIM} - \{x_i\}$;
  $K \leftarrow K + \{x_1\}$ ; $\mathscr{K} \leftarrow \mathscr{K} + \{N_G(x_1) \cup x_1\}$;
```

Example 5. *In Figure 10, $\mathscr{S} = \{\{6,7\},\{5,6\},\{4,5,6\},\{5,6\}\}$; $S = \{5,4,2,1\}$; $\mathscr{K} = \{\{6,7,8\},\{5,6,7\},\{3,4,5,6\},\{2,4,5,6\},\{1,5,6\}\}$ $K = \{6,5,3,2,1\}$.*

The complexity of the above algorithm is the same as for Lex-BFS or MCS, which is in $O(n + m)$ time.

A minimal separator may be generated several times, depending on the number of full components it defines:

Property 6. *Let α be a peo defined by Lex-BFS or MCS, let S be a minimal separator, let k be the number of full components of S. Then S has $k - 1$ generators by α.*

Chapter 4

The Atom tree

Contents

Corresponding paper:
Efficiently computing clique separator decomposition using an atom tree.
A. Berry , R. Pogorelcnik, G. Simonet.
Submitted to Discrete Applied Mathematics.

In order to represent the atoms obtained by clique minimal separator decomposition, we introduced a new notion named atom tree. This extension of the notion of clique tree of a chordal graph yields not only its atoms, but also a compact representation of their organization.

An atom tree of a graph G is a tree whose vertices are the atoms of G, and whose edges are the clique minimal separators of G.

We examined how to compute an atom tree from a clique tree of a minimal triangulation, thereby providing a completely different approach from the classical one from [39] and [25], which requires $O(n)$ graph searches. Let $G = (V, E)$ be a graph, $H = (V, E \cup F)$ a minimal triangulation of G, with $|F| = f$. We can compute an atom tree in $O(min(m + nf, nm))$ time given a minimal triangulation. In the case we already have the triangulation and the clique minimal separators we can compute the atom tree in linear $O(m + f)$ time.

We also present a new technique to compute an atom tree of a graph in $O(n(n + t))$ time with t the number of 2-pair which is a pair $\{x, y\}$ of non-adjacent vertices such that every chordless path from x to y is of length 2 with $t \leq m$.

In the last part, we present an extension of MCS-M to produce a triangulation and a clique tree of a graph as well as an extension that produced its atom tree in a single pass.

4.1 The clique tree: state of the art

Let $G = (E, V)$ be a graph and $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ the set of maximal cliques with k the number of cliques. For any chordal graph there exist clique trees of $\mathcal{C}$ [40]. This allows to represent chordal graphs in a compact way.

Example 6. *Figure 11 shows a chordal graph composed of five cliques (3 of size 3, 1 of size 2 and 1 of size 4) and one of its clique trees is shown in Figure 12.*

Definition 5. *Let $H = (V, E)$ be a connected chordal graph. A* clique tree *is a tree $T = (\mathcal{C}, \mathcal{E})$ with $\mathcal{C}$ the set of nodes representing the maximal cliques and $\mathcal{E}$ the set of edges, such that for any vertex x of H, the subgraph T_x of T induced by the set of nodes of T containing x is a subtree of T.*

Property 7. *[12] Let H be a connected chordal graph, let T be a clique tree of H, and let S be a set of vertices of H; then S is a minimal separator of H if and only if there is an edge C_1C_2 of T such that $S = C_1 \cap C_2$.*

To prove that for each edge C_1C_2 of T, the intersection $C_1 \cap C_2$ is a minimal separator of H, [12] show the following result.

Property 8. *[12] Let H be a connected chordal graph, let T be a clique tree of H, let C_1C_2 be an edge of T, let $S = C_1 \cap C_2$, and for each $i \in \{1, 2\}$ let T_i be the connected component of $T - \{C_1C_2\}$ containing C_i and let V_i be the union of the nodes of T_i; then $(V_1 \setminus S, S, V_2 \setminus S)$ is an elementary decomposition by clique minimal separator of H.*

Property 9. *[18] A chordal graph has at most n maximal cliques, and the sum of their sizes is at most $n + m$.*

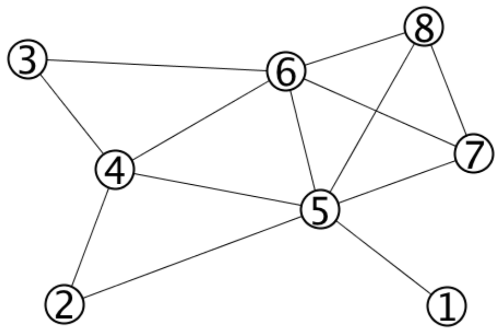

Figure 11: A chordal graph with five maximal cliques: {2,4,5}, {4,5,6}, {3,4,6},{5,6,7,8} and {1,5}.

Property 10. *[12] A vertex is simplicial if it belongs to only one clique.*

Property 11 (Clique intersection property [12])**.** *If two cliques C_i and C_j contain a vertex x, then all the cliques in the path between C_i and C_j contain x.*

Example 7. *In Figure 12, a clique tree from Figure 11 which respects the clique intersection property. For example $C_1 \cap C_4 = \{6\}$ is included in C_2.*

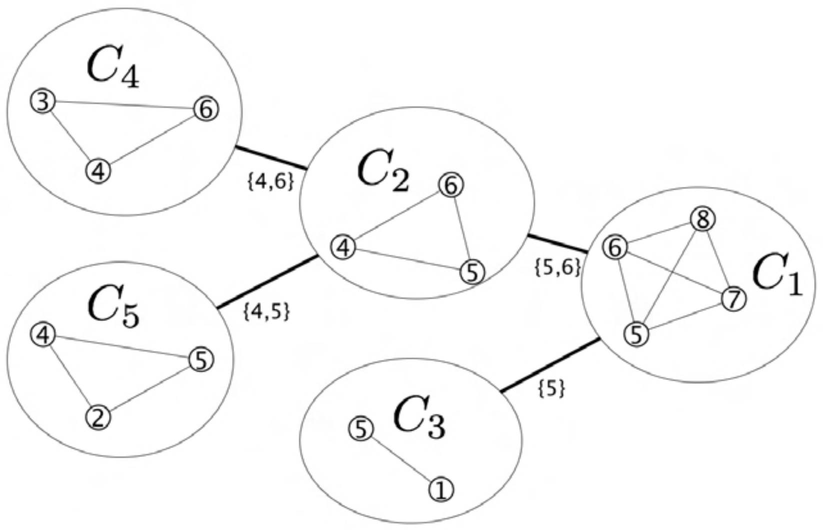

Figure 12: A clique tree of the chordal graph from Figure 11.

Property 12. *[12] A graph is chordal if and only if there exists a clique tree that respects the clique intersection property.*

A clique tree can be computed in linear time with the nodes labelled by the maximal cliques and the edges labelled by the minimal separators. The algorithm of Blair and Peyton [12] use an extension of MCS to create a tree of a chordal graph that respect the clique intersection property. We need the following terminology in this algorithm. An ordering α of G is a bijection $\alpha : V \rightarrow \{1, 2, ..., n\}$. We use α to index the vertex set, such that, $\alpha(v_i) = i$ for $1 \leq i \leq n$ where i will be referred to as the *number* of v_i. Let $v_1, v_2, ..., v_n$ be an ordering of V. We define $\mathcal{N}_i$ to be the set of vertices with number greater than $i - 1$. Let $v_{i_1}, v_{i_2}, ..., v_{i_m}$ be the representative vertices of the cliques $C_1, C_2, ..., C_m$. Blair and Peyton define a function $cli : V \rightarrow \{1, 2, ..., n\}$ by $cli(v_j) = r$ where $v_j \in \{v_{i_r}, v_{i_r+1}, ..., v_{i_{r-1}-1}\}$ with $i_r \leq j \leq i_{r-1}$. In other words, $cli(v)$ is the lowest index of a clique that contains v: $cli(v) = min\{r | v \in C_r\}$.

This algorithm, called Expanded-MCS, computes a perfect elimination ordering and each time a new clique begins it is linked to the one corresponding to the clique C_p for which $p = cli(v_j)$ where j is the vertex with the smallest number in the separator.

Algorithm 8: Expanded-MCS [12]

```
Input: A connected chordal graph $H = (V, E)$.
Output: A perfect elimination ordering $\alpha$ and a clique tree
        $T = (\mathcal{C}, \mathcal{E})$ of $H$.

$prev\text{-}card \leftarrow 0$;
$s \leftarrow 0$; $\mathcal{C} \leftarrow \emptyset$; $\mathcal{E} \leftarrow \emptyset$; $\mathcal{N}_{n+1} = \emptyset$;
foreach $i$ from $n$ downto 1 do
    Choose an unnumbered vertex $v$ in $V$ such that $|N(v) \cap \mathcal{N}_{i+1}|$ is
    maximum;
    $\alpha(v) \leftarrow i$; // $v$ becomes $v_i$
    $new\text{-}card \leftarrow |N(v_i) \cap \mathcal{N}_{i+1}|$;
    if $new\text{-}card \leq prev\text{-}card$ then              // a new clique begin
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_s\}$;
        $s \leftarrow s + 1$;
        $C_s \leftarrow N(v_i) \cap \mathcal{N}_{i+1}$;
        if $new\text{-}card \neq 0$ then                      // get edge to parent
            $k \leftarrow min\{j \mid v_j \in C_s\}$;
            $p \leftarrow cli(v_k)$;
            $\mathcal{E} \leftarrow \mathcal{E} \cup \{C_p, C_s\}$;
    $cli(v_i) \leftarrow s$;
    $C_s \leftarrow C_s \cup \{v_i\}$;
    $\mathcal{N}_i \leftarrow \mathcal{N}_{i+1} \cup \{v_i\}$;
    $prev\text{-}card \leftarrow new\text{-}card$;
$\mathcal{C} \leftarrow \mathcal{C} \cup \{C_s\}$;
```

4.2 The atom tree

We extend the definition of clique tree:

Definition 6. *Let $G = (V, E)$ be a connected graph. An* atom tree *of G is a tree $T = (\mathcal{A}, \mathcal{E})$ such that $\mathcal{A}$ is the set of atoms of G and for any vertex x of G, the subgraph T_x of T induced by the set of nodes of T containing x is a subtree of T.*

The intersection of two adjacent nodes is a clique minimal separator in the atom tree as in clique tree. Moreover each clique minimal separator is represented in the atom tree.

Property 13. *Let G be a graph, let T be an atom tree of G, and let S be a set of vertices of G. Then S is a clique minimal separator of G if and only if there is an edge A_1A_2 of T such that $S = A_1 \cap A_2$.*

Proof. To prove this, we will use the graph G^* introduced by [25]: G^* is the graph obtained from G by adding edges to the atoms to make them cliques. It is chordal and its maximal cliques are the atoms of G [25]. Hence T is a clique tree of G^*. S is a clique minimal separator of G if and only if it is a minimal separator of G^* [25], *i.e.* if and only if there is an edge A_1A_2 of T such that $S = A_1 \cap A_2$ by Property 7. $\square$

4.3 Computing an atom tree from a clique tree

In order to find clique minimal separators of a graph, we need to triangulate it. The resulting chordal graph can be represented by a clique tree. To compute the atom tree from the clique tree, we used a merging technique which contracts all nodes of the clique tree that are not separated by an edge representing a clique of the input graph.

Theorem 4.3.1. *Let $G = (V, E)$ be a connected graph, let $H = (V, E + F)$ be a minimal triangulation of G, let $T = (\mathcal{C}, \mathcal{E})$ be a clique tree of H, and let T' be the forest obtained from T by removing all edges C_1C_2 such that $C_1 \cap C_2$ is a clique in G, let T'' be the tree obtained from T by merging the nodes of each tree of T' into one node; then T'' is an atom tree of G.*

Proof. As the merging process preserves the connexity of the subgraph T_x for each vertex x of G, it is sufficient to show that the nodes of T'' are the atoms of G. We prove this by induction on the number of edges C_1C_2 of T such that $C_1 \cap C_2$ is a a clique in G. If there is no such edge then, by Properties 7 and 2, G has no clique minimal separator and therefore V is the unique atom, and T'' is reduced to node V, so the property holds. We assume that it holds if T has at most k edges C_1C_2 such that $C_1 \cap C_2$ is a a clique in G. Let us show that it holds if T has $k+1$ such edges. Let C_1C_2 be such an edge of T, let $S = C_1 \cap C_2$, for $i \in \{1,2\}$ let T_i be the connected component of $T - \{C_1C_2\}$ containing C_i, let T_i'' be the tree obtained from T_i by the merging process, let V_i be the union of the sets of vertices of the nodes of T_i, let $G_i = G(V_i)$ and let $H_i = H(V_i)$.

$(V_1 \setminus S, S, V_2 \setminus S)$ is a partition of V, no vertex of $V_1 \setminus S$ is adjacent to a vertex of $V_2 \setminus S$ in H and there exist a vertex x_1 of $V_1 \setminus S$ and a vertex x_2 of $V_2 \setminus S$ such that S is a minimal x_1x_2-separator in H [12]. By Property 7, this is also the case in G. It follows that the set of atoms of H (resp. G) is the disjoint union of the sets of atoms of H_1 and H_2 (resp. G_1 and G_2).

Next we show that G_1 and H_1 are connected, that H_1 is a minimal triangulation of G_1 and that T_1 is a clique tree of H_1. Clearly, H_1 is connected and is a triangulation of G_1. We claim that this triangulation is minimal: if it is not, let H' be a triangulation of G_1 that is a strict subgraph of H_1; the union of H' and H_2 is a triangulation of G that is a strict subset of H, which is impossible. As H_1 is a minimal triangulation of G_1 and H_1 is connected, G_1 is also connected. Let us show that T_1 is a clique tree of H_1. First we show that the nodes of T_1 are the maximal cliques of H_1. As the atoms of a chordal graph are its maximal cliques, the set of maximal cliques of H is the disjoint union of the sets of maximal cliques of H_1 and of H_2. As moreover no maximal clique of H_1 can be a subset of V_2 and no maximal clique of H_2 can be a subset of V_1 (otherwise it would be a subset of $V_1 \cap V_2 = S$, and therefore a strict subset of C_1 and C_2) the maximal cliques of H_1 are exactly the nodes of T_1. For each vertex x of H_1, T_x is a subtree of T, so its restriction to T_1 is a subtree of T_1. By induction hypothesis, the nodes of T_1'' are the atoms of G_1. Similarly, the nodes of T_2'' are the atoms of G_2, so the set of nodes of T'', which is the disjoint union of the sets of nodes of T_1'' and of T_2'', is the disjoint union of the sets of atoms of G_1 and of G_2, which is the set of atoms of G. □

From Theorem 4.3.1, we derive an algorithm to compute the atom tree from a clique tree of a minimal triangulation. Our algorithm defines the connected components of the forest T' by contracting each pair of adjacent

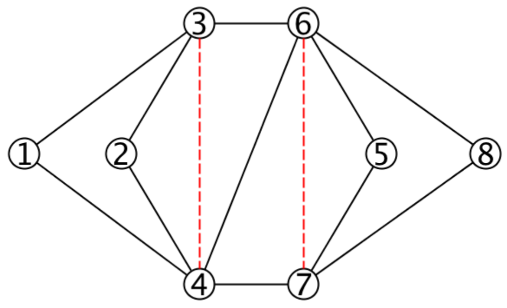

Figure 13: A minimal triangulation of graph. The red dashed lines represent the fill edges.

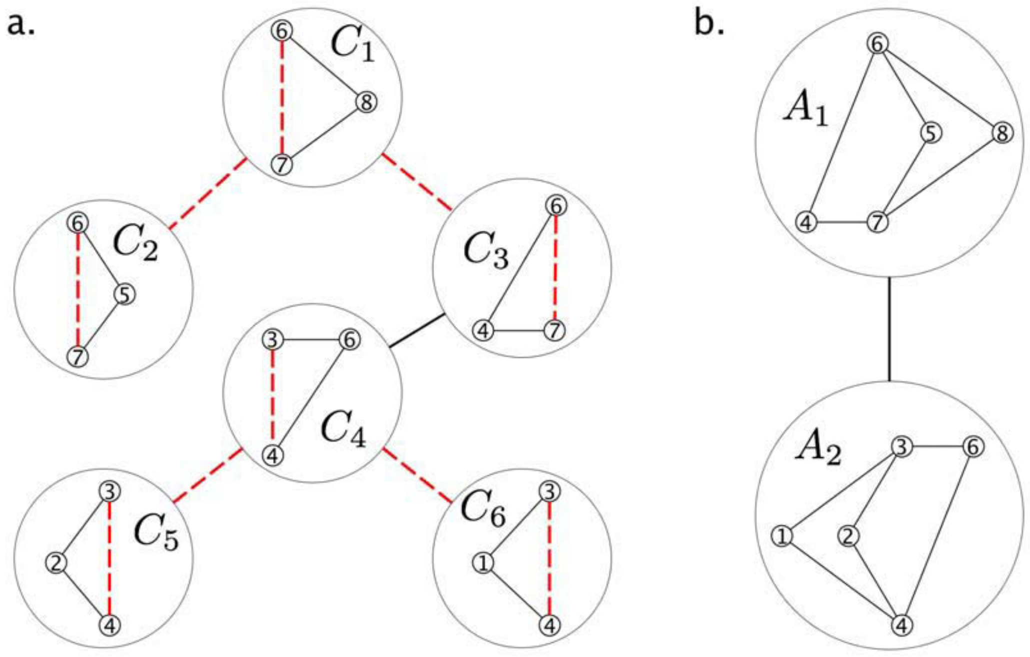

Figure 14: A clique tree (a) and its corresponding atom tree (b). The dashed edges represents separators that are not cliques in the graph G from Figure 13.

nodes in T separated by an edge which is not a clique minimal separator. The contraction of two nodes C_1 and C_2 into a single node classically consists in replacing C_1 and C_2 with node $C_1 \cup C_2$; the set of its neighbours in the new tree is the union of the sets of neighbours of C_1 and of C_2.

Algorithm 9: Atom-Tree

```
Input: A graph $G = (V, E)$, a clique tree $T = (\mathcal{C}, \mathcal{E})$ of a minimal
       triangulation of $G$.
Output: $T$ is merged into the corresponding atom tree of $G$.
foreach edge $C_1C_2$ of $T$ do
    if $C_1 \cap C_2$ is not a clique in $G$ then
        Contract $C$ with $C'$ ;
```

Example 8. *In Figure 14 (a) a clique tree of the triangulated graph of Figure 13. The separator C_3C_4 is the only complete one. To obtain the atom tree we merge the cliques of the two trees of the forest resulting from the deletion of edge C_3C_4. This leads to the atom tree shown in Figure 14 (b).*

The complexity of the algorithm is stated and proved in the following theorem.

Theorem 4.3.2. *Let $G = (V, E)$ be a graph, $H = (V, E \cup F)$ a minimal triangulation of G, with $|F| = f$; Algorithm Atom-Tree computes an atom tree of G from a clique tree of H in time:*

- *$O(m + f)$ if the clique minimal separators of G are provided.*
- *$O(min(nm, m + nf))$ otherwise.*

Proof. As T is a tree, it can be searched from one of its nodes in such a way that for each processed edge C_1C_2, C_2 is still unreached, and therefore is a node of the initial tree T with the same degree as in T. Computing $C_1 \cap C_2$ and contracting C_1 with C_2 can be done in $O(max(|C_1|, |C_2|) + |N_T(C_2)|)$ time, hence in $O(m+f)$ time globally since the sum of the sizes of the nodes of T is at most $n + m + f$ and T has less than n edges.

If the set clique minimal separators of the input graph $G = (V, E)$ are provided, determining whether $C_1 \cap C_2$ is one of them costs $O(m + f)$ time globally: we build the tripartite graph $I = (V, E_T, \mathscr{S}^*, E_I)$, where E_T is the set of edges of the clique tree, $\mathscr{S}^*$ is the set of clique minimal separators of G, and E_I is the set of edges of the tripartite graph, defined as follows: $vy \in E_I$ if $v \in V$ and y is either a minimal separator of $\mathscr{S}^*$ containing v or an edge of the clique tree representing a minimal separator containing v. The graph I has less than $3n$ vertices and less than $2(n+m+f)$ edges, since there are at most n clique minimal separators in a graph and since the sum of the sizes of the clique minimal separators is at most m. In I, we search for pairs of vertices $\{x, y\}$, $x \in \mathscr{S}^*, y \in E_T$, with identical neighbourhoods. Finding sets of vertices with identical neighbourhoods can be done in linear time using partition refinement on the neighbourhoods [23].

If the clique minimal separators of G are not provided, determining whether $C_1 \cap C_2$ is a clique in G costs $O(m)$ time, and therefore $O(nm)$ time globally. Another way to determine whether $C_1 \cap C_2$ is a clique in G is to check for each fill edge whether it belongs to $C_1 \cap C_2$, which can be done in $O(f)$ time per edge of T, hence in $O(nf)$ time globally, which makes the total complexity in $O(m + nf)$ time. □

The time complexity in $O(m+nf)$ may be better than $O(nm)$ when f is of small size. This is interesting in the context of triangulations computed with the Minimum Degree Heuristic, which approximates minimum triangulation and can yield a small fill in practice [11].

4.4 Computing an atom tree from a clique tree in $O(n(n + t))$ time

To determine which minimal separators of H are cliques in G, we use a new technique. We consider a directed version of the clique tree (by choosing an arbitrary root). We use the property that a minimal separator S is a clique in G if and only if it contains no edge of F. We traverse the clique tree from the root down, and maintain information on the subset X of V represented by the node or edge of the tree we are on. We maintain for each vertex y of V the number $dF_X(y)$ of vertices x in the current subgraph X such that $xy \in F$

(*i.e.* the number of neighbors of y in the graph (V, F) that are in X). Thus X is a clique in G if and only if for every vertex x of X, $dF_X(x) = 0$; dF_X is updated by vertex additions and removals when going downwards from node u to node v through edge (u, v).

The clique tree is represented by a rooted tree $T = (V_T, Child, Atom)$, where $Child$ maps each node of T to the set of its successors in T, and $Atom$ maps each node of T to a subset of V, which is a maximal clique of H at the beginning, and an atom of G at the end.

Algorithm 10: Algorithm DF-Atom-Tree

```
input : A connected graph $G = (V, E)$, the set $F$ of fill edges of a
        minimal triangulation $H$ of $G$, a rooted clique tree
        $T = (V_T, Child, Atom)$ of $H$, the root $r$ of $T$.
output: $T$ is merged into the corresponding atom tree of $G$.
init: $dF_r(x)$ is 0 for each vertex $x$ of $r$;
foreach $x \in Atom(r)$ do
    foreach $y \in V$ such that $xy \in F$ do $dF_r(y) \leftarrow dF_r(y) + 1$;
REC-AT-DF$(r, dF_r)$;
foreach edge $(u, v)$ of $T$ do
    if not $Complete(u, v)$ then contract $u$ with $v$ ;
```

Theorem 4.4.1. *Let $G = (V, E)$ be a connected graph. Given a minimal triangulation $H = (V, E + F)$ of G and a clique tree T of H, Algorithm DF-Atom-Tree computes an atom tree of G in $O(n(n + t))$ time, where t is the number of 2-pairs of H.*

Proof. The algorithm process in a way of a Depth-first search all the separators of H to check if they are clique in G. The cliques separated by non-clique separators are contracted, it follows from Theorem 4.3.1 that the tree produced is an Atom tree. Let (u, v) be an edge of T. Determining $Complete(u, v)$ costs $O(n)$ time, and therefore $O(n^2)$ time for all edges of T. A vertex x of V is added exactly once to the current subset X represented by dF (when processing the predecessor of the root of T_x if it ex-

Algorithm 11: REC-AT-DF

```
input : A node $u$ of $T$, the mapping $dF_u$ associated with $Atom(u)$
output: The mapping $Complete$ from the set of edges of the subtree of
        $T$ rooted at $u$ to $\{true, false\}$ defined by: $Complete(v, w) =$
        true if and only if $Atom(v) \cap Atom(w)$ contains no edge of $F$.
foreach $v \in Child(u)$ do
    $dF_v \leftarrow dF_u$;
    foreach $x \in Atom(u) - Atom(v)$ do
        foreach $y \in V \mid xy \in F$ do $dF_v(y) \leftarrow dF_v(y) - 1$;
    $Complete(u, v) \leftarrow true$ ; // $Atom(u) \cap Atom(v)$ is set as a clique
    in $G$;
    foreach $x \in Atom(u) \cap Atom(v)$ do
        if $dF_v(x) \neq 0$ then $Complete(u, v) \leftarrow false$;
    if $Child(v) \neq \emptyset$ then
        foreach $x \in Atom(v) - Atom(u)$ do
            foreach $y \in V \mid xy \in F$ do $dF_v(y) \leftarrow dF_v(y) + 1$;
        REC-AT-DF$(v, dF_v)$;
```

ists and when computing dF_r otherwise). Thus vertex additions to X cost $O(n^2)$ time globally. Let us show that the number of removals from X, *i.e.* $\Sigma_{(u,v)\ edge\ of\ T}|Atom(u) \setminus Atom(v)|$, is bounded by t. For this it is sufficient to show that there is an injective mapping f from the set of triples (x, u, v) such that (u, v) is an edge of T and $x \in Atom(u) \setminus Atom(v)$ to the set of 2-pairs of H. Let $f(x, u, v) = \{x, y\}$, where $y \in Atom(v) \setminus Atom(u)$. We show that $\{x, y\}$, is a 2-pair in H, or, equivalently, that $N_H(x) \cap N_H(y)$ is a minimal xy-separator in H [2]. As $Atom(u)$ and $Atom(v)$ are cliques of H containing x and y respectively, $Atom(u) \cap Atom(v) \subseteq N_H(x) \cap N_H(y)$. As $Atom(u) \cap Atom(v)$ is an xy-separator in H [12], $N_H(x) \cap N_H(y)$ is an xy-separator in H too, and it is a minimal one since it is a subset of each xy-separator. Thus vertex removals from X cost $O(nt)$ time globally, which puts the complexity of Algorithm DF-Atom-Tree in $O(n(n + t))$ time. □

Example 9. *In Figure 14a) $dF_{(C_1)}$ is initialized as follows* $(0\ 0\ 0\ 0\ 0\ 1\ 1\ 0)^t$ *$dF_{(C_1)}(6)$ and $dF_{(C_1)}(7)$ get 1 from each other. $dF_{(C_2)}$* $= (0\ 0\ 0\ 0\ 1\ 1\ 0\ 0)^t$, *since $dF_{(C_2)}(6) = 1$ the separator $6, 7$ is not complete so C_1 and C_2 will be merged.$dF_{(C_3)} = (0\ 0\ 0\ 0\ 0\ 1\ 1\ 0)^t$ and C_1 and C_3 are merged. $dF_{(C_3)}(4)$ is increased. When the completeness of C_3C_4 is tested $dF_{(C_4)} = (0\ 0\ 0\ 0\ 0\ 1\ 1\ 0)^t$ so it is complete. After the completeness test $dF_{(C_4)}(4)$ is increased. Since $dF_{(C_4)}(4) = 1$ C_4C_5 and C_4C_6 are merged.*

The complexity bottleneck of Algorithm DF-Atom-Tree resides in the number of vertex removals from dF_X. There are some instances where this number is small: this is the case when the number of 2-pairs is in $O(n)$, when the number of leaves of T is bounded, and when the minimal triangulation is an interval graph or a path graph. Note that in the general case, the number of vertex removals is often much smaller than the number of 2-pairs.

4.5 Computing the Atom Tree in one step

We derive from Algorithm Expanded-MCS from [12] an algorithm computing an atom tree directly from the input graph. The intermediate step was the creation of Algorithm Expanded-MCS-M which computes a minimal elimination ordering (peo of the corresponding triangulated graph) and a clique tree from a random graph. This work logically results in the creation of the MCS-Atom-Tree algorithm that generates a triangulation, a minimal elimination ordering and an atom tree of a graph.

4.5.1 Expanded-MCS-M

Algorithm Expanded-MCS-M is derived from Algorithm MCS from computing a perfect elimination ordering of a chordal graph in linear time. Expanded-MCS computes both a perfect elimination ordering and a clique tree of a chordal graph in linear time. We define Algorithm Expanded-MCS-M from algorithm MCS-M in the same way as Algorithm Expanded-MCS is defined from Algorithm MCS.

Algorithm MCS-M computes a minimal elimination ordering of the input graph and the associated minimal triangulation in $O(nm)$ time. Algorithm

Algorithm 12: Expanded-MCS-M

```
**Input**: A connected graph $G = (V, E)$.
**Output**: a meo $\alpha$ of $G$, the associated minimal
            triangulation $H = (V, E_H)$ of $G$ and a clique tree
            $T = (\mathcal{C}, \mathcal{E}, MinSep)$ of $H$.
$prev\text{-}card \leftarrow 0$; $\mathcal{N}_{n+1} = \emptyset$; $E_H \leftarrow E$;
$s \leftarrow 0$; $\mathcal{C} \leftarrow \emptyset$; $\mathcal{E} \leftarrow \emptyset$; $H \leftarrow (V, E_T)$;
foreach $i$ from $n$ downto 1 do
    Choose an unnumbered vertex $v$ in $V$ such that $|N(v) \cap \mathcal{N}_{i+1}|$ is maximum;
    $\alpha(v) \leftarrow i$; // $v$ become $v_i$
    $new\text{-}card \leftarrow |N_H(v_i) \cap \mathcal{N}_{i+1}|$;
    if $new\text{-}card \leq prev\text{-}card$ then
        $\mathcal{C} \leftarrow \mathcal{C} \cup C_s$;
        $s \leftarrow s + 1$;
        $C_s \leftarrow N_H(v_i) \cap \mathcal{N}_{i+1}$;
        if $new\text{-}card \neq 0$ then
            $k \leftarrow min\{j \mid v_j \in C_s\}$;
            $p \leftarrow cli(v_k)$;
            $\mathcal{E} \leftarrow \mathcal{E} \cup \{C_p, C_s\}$;
            $MinSep(ps) \leftarrow |N_H(v_i) \cap \mathcal{N}_{i+1}|$;
    $cli(v_i) \leftarrow s$;
    $C_s \leftarrow C_s \cup \{v_i\}$;
    $F \leftarrow \emptyset$;
    foreach $w \in V \setminus (\mathcal{N}_{i+1} \cap N_G(v_i))$ do
        if there is an $v_i w$-path $\mu$ in $G(V \setminus \mathcal{N}_{i+1})$ such that for each
        internal node $x$ of $\mu$ $|N_H(x) \cap \mathcal{N}_{i+1}| < |N_H(w) \cap \mathcal{N}_{i+1}|$ and $w$ is
        unnumbered then
            $F \leftarrow F + \{v_i w\}$;
    $E_H \leftarrow E_H + F$;
    $\mathcal{N}_i \leftarrow \mathcal{N}_{i+1} \cup \{v_i\}$;
    $prev\text{-}card \leftarrow new\text{-}card$;
$\mathcal{C} \leftarrow \mathcal{C} \cup C_s$;
```

Expanded-MCS-M computes in $O(nm)$ a minimal elimination ordering of G, the chordal graph H and a clique tree of H in $O(nm)$ time. We remind the following terminology used in Algorithm Expanded-MCS-M. An ordering α of G is a bijection $\alpha : V \to \{1, 2, ..., n\}$. We use α to index the vertex set, such that, $\alpha(v_i) = i$ for $1 \leq i \leq n$ where i referred to as the *number* of v_i. Let $v_1, v_2, ..., v_n$ be an ordering of V. We define $\mathcal{N}_i$ to be the set of vertices with number greater than $i - 1$. Let $v_{i_1}, v_{i_2}, ..., v_{i_m}$ be the representative vertices of the cliques $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_m$. The function $cli : V \to \{1, 2, ..., n\}$ by $cli(v_j) = r$ and $v_j \in \{v_{i_r}, v_{i_{r+1}}, ..., v_{i_{r-1}-1}\}$. In other words, $cli(v)$ is the lowest index of a clique that contains v: $cli(v) = min\{r|v \in \mathcal{C}_r\}$.

In Algorithm Expanded-MCS-M, a clique tree of a chordal graph H is represented by a labelled tree $T = (\mathcal{C}, \mathcal{E}, MinSep)$, where $\mathcal{C}$ represents the set of maximal cliques of H, $\mathcal{E}$ represents the set of edges and $MinSep$ maps each edge of $\mathcal{E}$ to a minimal separator of H, with $MinSep(ps) = C_p \cap C_s$.

The algorithm computes a minimal elimination ordering of G, the triangulation H of G. Each time a new clique begins it is linked to the one corresponding to the clique C_p for which $p = cli(v_j)$, where j is the vertex with the smallest number in the separator.

4.5.2 MCS-Atom-Tree

To explain the Algorithm MCS-Atom-Tree, we need the following notation. Let $v_{i_1}, v_{i_2}, ..., v_{i_m}$ be the representative vertices of the atoms $A_1, A_2, ..., A_m$. In the same fashion as in Algorithm MCS-Expanded we define a function $ato : V \to \{1, 2, ..., n\}$ by $ato(v_j) = r$ where $v_j \in \{v_{i_r}, v_{i_r+1}, ..., v_{i_{r-1}-1}\}$ with $i_r \leq j \leq i_{r-1}$. In other words, $ato(v)$ is the lowest index of an atom that contains v: $ato(v) = min\{r|v \in A_r\}$.

In the MCS-Atom-Tree algorithm an atom tree of a graph G is represented by a labelled tree $T = (\mathcal{A}, \mathcal{E}, CliqueMinSep)$, where $\mathcal{A}$ represents the set of atoms of G and $CliqueMinSep$ maps each edge of $\mathcal{E}$ to a clique minimal separator of G, with $CliqueMinSep(ps) = A_p \cap A_s$.

In Expanded-MCS-M each edge $\{A_p, A_s\}$ of the clique tree of the minimal triangulation H computed is a minimal separator. When adding edge $\{A_p, A_s\}$ to $\mathcal{E}$, $N_H(v_i) \cap \mathcal{N}_{i+1}$ corresponds to the separator $A_p \cap A_s$ of H.

Thus, in order to compute an atom tree of a graph G, it is sufficient to modify Algorithm Expanded-MCS-M, as follows. In case *new-card* $\leq$ *prev-card*, we determine whether $N_H(v_i) \cap \mathcal{N}_{i+1}$ is a clique in G. If it is a clique in G and *prev-card* $\neq 0$ then we increment s, and add edge $\{A_p, A_s\}$ to $\mathcal{E}$, where $p = ato(v_j)$ where j is the vertex with the smallest number in the separator. If it is not, then we merge A_p and A_s by setting s to p.

Theorem 4.5.1. *Let $G = (V, E)$ be a connected graph. Algorithm MCS-Atom-Tree computes an atom tree of G in $O(nm)$ time.*

Proof. Correctness follows from correctness of Algorithm Expanded-MCS-M and Theorem 4.3.1. $O(nm)$ time complexity follows from $O(nm)$ time complexity of Algorithm Expanded-MCS-M and from the fact that determining whether $N_H(x)$ is a clique in G costs $O(m)$ time, and therefore $O(nm)$ time globally. $\square$

4.6 Conclusion

In conclusion, our work on clique minimal separator decomposition and on generating the minimal separators and the maximal cliques of a chordal graph enabled us to define a new approach to generate the atoms of a graph, by introducing the notion of atom tree.

The atom tree not only enables to find the atoms in a single pass (which may be interesting for very large graphs) but also yields structural information and helps to visualize how the atoms interact. This is useful for the biological applications described in Chapter 7.

Algorithm 13: MCS-Atom-Tree

```
Input: A connected graph $G = (V, E)$.
Output: an atom tree $T = (\mathcal{A}, \mathcal{E}, CliqueMinSep)$ of $G$.
$prev\text{-}card \leftarrow 0$; $\mathcal{N}_{n+1} = \emptyset$ ;
$s \leftarrow 0$; $\mathcal{A} \leftarrow \emptyset$; $\mathcal{E} \leftarrow \emptyset$;
$E_H \leftarrow E$; $H \leftarrow (V, E_H)$;
foreach $i$ from $n$ downto $1$ do
    Choose an unnumbered vertex $v$ in $V$ such that $|N(v) \cap \mathcal{N}_{i+1}|$ is maximum;
    $\alpha(v) \leftarrow i$; // $v$ become $v_i$
    $new\text{-}card \leftarrow |N_H(v_i) \cap \mathcal{N}_{i+1}|$;
    if $new\text{-}card \leq prev\text{-}card$ then
        if $new\text{-}card \neq 0$ then
            $k \leftarrow min\{j \mid v_j \in A_s\}$;
            $p \leftarrow ato(v_k)$;
        if $N_H(v_i) \cap \mathcal{N}_{i+1}$ is a clique in $G$ or an empty set then
            $\mathcal{A} \leftarrow A_s$;
            $s \leftarrow s + 1$;
            $A_s \leftarrow N_H(v_i) \cap \mathcal{N}_{i+1}$;
            if $new\text{-}card \neq 0$ then
                $\mathcal{E} \leftarrow \mathcal{E} + \{A_p, A_s\}$;
                $CliqueMinSep(ps) \leftarrow N_H(v_i) \cap \mathcal{N}_{i+1}$;
        else
            $s \leftarrow p$;
    $ato(v_i) \leftarrow s$;
    $A_s \leftarrow A_s + \{v_i\}$;
    $F \leftarrow \emptyset$;
    foreach $w \in V \setminus (\mathcal{N}_{i+1} \cap N_H(v_i))$ do
        if there is an $v_i w$-path $\mu$ in $G(V \setminus \mathcal{N}_{i+1})$ such that for each
        internal node $x$ of $\mu$ $|N_H(x) \cap \mathcal{N}_{i+1}| < |N_H(w) \cap \mathcal{N}_{i+1}|$ and $w$ is
        unnumbered then
            $F \leftarrow F \cup \{v_i w\}$;
    $E_H \leftarrow E_H \cup F$;
    $\mathcal{N}_i \leftarrow \mathcal{N}_{i+1} \cup \{v_i\}$;
    $prev\text{-}card \leftarrow new\text{-}card$;
$\mathcal{A} \leftarrow A_s$;
```

Chapter 5

Decomposing bipartite graphs

Contents

Corresponding paper:
Vertical decomposition of a lattice using clique separators.
A. Berry, R. Pogorelcnik, and A. Sigayret.
The 8th International Conference on Concept Lattices and their Applications (CLA11), 2011.
Submitted as a journal paper in the special issue of AMAI.

While examining possible biological representations using lattices, we realized that clique minimal separator decomposition could be applied efficiently to a bipartite graph, and the resulting decomposition had interesting effects on the Galois lattice, which is an ordering of the maximal bicliques of the bipartite graph.

We show how the lattice is decomposed into smaller lattices, from which the original lattice can be deduced. This is an interesting application of clique minimal separator decomposition, providing a Divide and Conquer approach to generate a Galois lattice.

In addition, a bipartite graph has clique minimal separator of size one or two but not more; algorithms exist to compute the clique minimal separator decomposition in linear time in this case.

5.1 Decomposing the bipartite graph and the relation

We use the bipartite graph $Bip(\mathcal{R})$ defined by a relation $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$. Figure 15 shows an example of a relation with the corresponding bipartite graph.

	a	b	c	d	e	f	g	h	i
1	×		×	×	×				
2		×	×			×		×	×
3	×	×				×			
4				×	×				
5							×	×	
6		×					×		

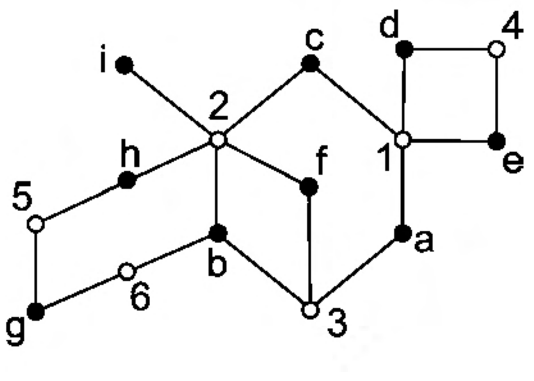

Figure 15: A relation and the corresponding bipartite graph

We describe the decomposition of the bipartite graph and how it decomposes the associated lattice. The decomposition is done in two steps, the first step consists in decomposing the lattice using minimal separators of size one. The articulation points (clique minimal separators of size one) can be found by a simple depth-first search [21], as well as the corresponding decomposition of the graph (called decomposition into biconnected components).

The removal of an articulation point $\{x\}$ in a graph G results into components C_1, ..., C_k, which correspond to a partition $V = C_1 + ... + C_k + \{x\}$. The vertex x is preserved, with its local neighbourhood, in each component, so that the decomposition step finally replaces G by k subgraphs $G(C_1 \cup \{x\})$, ..., $G(C_k \cup \{x\})$.

The second step is the search for clique separators of size two, and corresponds to a more complicated algorithm, described in [22]: all separators of size 2 are obtained, whether they are cliques or not. Once this list of separators is obtained, it is easy to check which are joined by an edge. The decomposition can then be obtained easily, and corresponds to the following decomposition step: the removal of a clique minimal separator $\{x, y\}$ in a connected bipartite graph G results into components C_1,..., C_k, corresponding to a partition $V = C_1 + ... + C_k + \{x, y\}$. The decomposition step replaces G with $G(C_1 \cup \{x, y\})$, ..., $G(C_k \cup \{x, y\})$.

A clique minimal separator of size two may include an articulation point. Thus, it is important to complete the decomposition by the articulation points first, and then go on to decompose the obtained subgraphs using their clique separators of size two.

Example 10. *Figure 16 shows that the input graph of Figure 15 is decomposable into four bipartite subgraphs:* $G_1 = G(\{2, i\})$, $G_2 = G(\{2, 5, 6, b, g, h\})$, $G_3 = G(\{1, 2, 3, a, b, c, f\})$ *and* $G_4 = G(\{1, 4, d, e\})$.

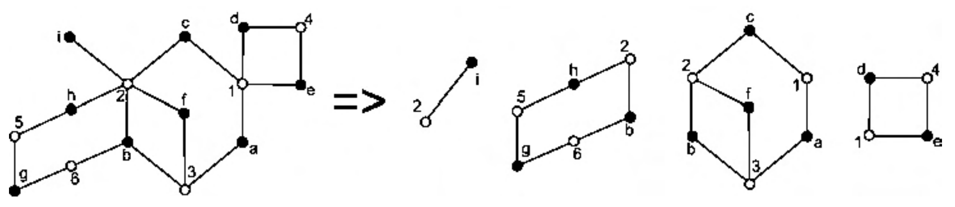

Figure 16: Complete decomposition of a bipartite graph

5.2 Decomposing the lattice

Let $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ be a relation with $\mathcal{O}$ a set of objects and $\mathcal{A}$ a set of attributes, associated with a bipartite graph $G = (\mathcal{O} + \mathcal{A}, E)$ denote $Bip(\mathcal{R})$. The maximal rectangles of the relation correspond exactly to the maximal bicliques of $Bip(\mathcal{R})$ and to the concepts of the Galois lattice $\mathcal{L}(\mathcal{R})$. We now examine how the set of concepts is modified and partitioned into the subgraphs obtained. As clique minimal separators are copied in all the components induced, most of the concepts will be preserved by the decomposition. Furthermore, only concepts including a vertex of a clique minimal separator may be affected by the decomposition.

Definition 7. *We say that a maximal biclique is a* star maximal biclique *if it contains either exactly one object or exactly one attribute. This single object or attribute is called the center of the star.*

Lemma 5.2.1. *Let $R \subseteq O \times A$ be a relation, a star maximal biclique $\{x\} \cup N(x)$ of $Bip(\mathcal{R})$ is an atomic concept of $\mathcal{L}(\mathcal{R})$ (atom or co-atom), unless x is universal in $Bip(\mathcal{R})$. More precisely, $\{x\} \times N(x)$ is an atom of $\mathcal{L}(\mathcal{R})$ if $x \in \mathcal{O}$ and $N(x) \neq \mathcal{A}$, or $N(x) \times \{x\}$ is a co-atom if $x \in \mathcal{A}$ and $N(x) \neq \mathcal{O}$.*

Proof. Let $\{x\} \cup N(x)$ be a star maximal biclique of $Bip(\mathcal{R})$. As a maximal biclique, it corresponds to a concept of $\mathcal{L}(\mathcal{R})$. Suppose the star has center on $x \in \mathcal{O}$. By definition, it contains no other element of $\mathcal{O}$; as a biclique, it includes all $N(x) \subseteq \mathcal{A}$, and no other element of $\mathcal{A}$ by maximality. The corresponding concept is $\{x\} \times N(x)$ which is obviously the first concept from bottom to top including x. As the biclique is maximal, and as x is not universal, this concept cannot be the bottom of $\mathcal{L}(\mathcal{R})$ but only an atom. A similar proof holds for $x \in \mathcal{A}$ and co-atomicity. □

We will now give the property which describes how the maximal bicliques are dispatched or modified by the decomposition. We will give a general theorem and its proof, from which these properties can be deduced.

Property 14. *Let $G = (X + Y, E)$ be a bipartite graph, let S be a clique minimal separator of G which decomposes G into subgraphs G_1, ..., G_k. Then:*

1. *For all $x \in S$, $\{x\} \cup N_G(x)$ is a star maximal biclique of G.*

2. *For all $x \in S$, $\{x\} \cup N_G(x)$ is not a maximal biclique of any G_i.*

3. *For all $x \in S$, $\{x\} \cup N_{G_i}(x)$ is a biclique of G_i, but it is maximal in G_i if and only if it is not strictly contained in any other biclique of G_i.*

4. *All the maximal bicliques which are not star bicliques with any $x \in S$ as a center are partitioned into the corresponding subgraphs.*

With the help of Lemma 5.2.1, this property may be translated in terms of lattices. Given a relation $\mathcal{R}$, its associated graph G, its lattice $\mathcal{L}(\mathcal{R})$, and a decomposition step of G into some G_is by articulation point $\{x\}$:

If $x \in \mathcal{O}$ (resp. $\in \mathcal{A}$) is an articulation point of G, $\{x\} \times N_G(x)$ (resp. $N_G(x) \times \{x\}$) is a concept of $\mathcal{L}(\mathcal{R})$. After the decomposition step, in each subgraph G_i of G, either this concept becomes $\{x\} \times N_{G_i}(x)$, or this concept disappears from G_i; this latter case occurs when there is in G_i some $x' \in \mathcal{O}$, the introducer of which appears after the introducer of x in $\mathcal{L}(\mathcal{R})$, from bottom to top (resp. from top to bottom if $x, x' \in \mathcal{A}$). Every other concept will appear unchanged in exactly one lattice associated with a subgraph G_i.

The same holds for each vertex of a size two clique minimal separator .

5.3 Reconstructing the lattice from the sublattices

We now explain how, given the subgraphs obtained by clique decomposition, as well as the corresponding subrelations and subsets of concepts, we can reconstruct the set of concepts of the global input bipartite graph.

Theorem 5.3.1. *Let $G = (X + Y, E)$ be a bipartite graph, let $\Sigma = \{s_1, ...s_h\}$ be the set of all the vertices which belong to any clique separator of G, let $G_1, ...G_k$ be the set of subgraphs obtained by the corresponding clique separator decomposition. Then:*

1. *For every $s \in \Sigma$, $\{s\} \cup N_G(s)$ is a star maximal biclique of G.*

2. *Any maximal biclique of a subgraph G_i which is not a star with a vertex of Σ as center is also a maximal biclique of G.*

3. *There are no other maximal bicliques in G: $\forall s \in \Sigma$, no other star maximal biclique of G_i with center s is a star maximal biclique of G, and these are the only maximal bicliques of some graph G_i which are not also maximal bicliques in G.*

Proof.

1. For every $s \in \Sigma$, $\{s\} \cup N_G(s)$ is a star maximal biclique of G:
   Case 1: s is an articulation point, let G_i, G_j be two graphs which s belongs to; s must be adjacent to some vertex y_i in G_i and to some vertex y_j in G_j. Suppose $\{s\} \cup N_G(s)$ is not a maximal biclique: there must be a vertex z in G which is adjacent to y_i and y_j, but then $\{s\}$ cannot separate y_i from y_j, a contradiction.
   Case 2: s is not an articulation point, let s' be a vertex of S such that $\{s, s'\}$ is a clique separator of G, separating G_i from G_j. s must as above see some vertex y_i in G_i and some vertex y_j in G_j. Suppose $\{s\} \cup N_G(s)$ is not maximal: there must be some vertex w in G which is adjacent to all of $N_G(s)$, but w must see y_i and y_j, so $\{s, s'\}$ cannot separate G_i from G_j.

2. Let B be a non-star maximal biclique of G_i, containing $o_1, o_2 \in \mathcal{O}$ and $a_1, a_2 \in \mathcal{A}$. Suppose B is not maximal in G: there must be a vertex y in $G - B$ which augments B. Let y be in G_j, WLOG $y \in \mathcal{A}$: y must see o_1 and o_2. Since G_i is a maximal subgraph with no clique separator, $G_i + \{y\}$ must have a clique separator. Therefore $N(y)$ must be a clique separator of this subgraph, but this is impossible, since y is adjacent to two non-adjacent vertices of G_i.

3. Any star maximal biclique B of G_i whose center is not in Σ is also a star maximal biclique of G: suppose we can augment B in G.

Case 1: v is adjacent to an extra vertex w; $G_i + \{w\}$ contains as above a clique separator, which is impossible since $N(w) = v$ and $v \notin S$.
Case 2: A vertex z of G_j is adjacent to all of $N(v)$: again, $G + \{z\}$ contains a clique separator, so $N(z)$ is a clique separator, but that is impossible since $N(z)$ contains at least two non-adjacent vertices. For $s \in \Sigma$, no star maximal biclique of G_i is a star maximal biclique of G: let B be a star maximal biclique of G_i, with $s \in \Sigma$ as center. $s \in \Sigma$, so s belongs to some clique separator which separates G_i from some graph G_j. Therefore, s must be adjacent to a vertex y_j in G_j, so $B + \{y_j\}$ is a larger star including B thus B cannot be maximal in G.

□

Example 11. *We illustrate Theorem 5.3.1 using graph G from Figure 16, whose decomposition yields subgraphs G_1, ..., G_4, with $G_1 = G(\{1, 2, i\})$, $G_2 = G(\{2, 5, 6, b, g, h\})$, $G_3 = G(\{1, 2, 3, a, b, c, f\})$ and $G_4 = G(\{1, 4, d, e\})$. The corresponding lattices are shown in Figure 17, and their concepts are presented in the table below. In this table, braces have been omitted; symbol $\Rightarrow$ represents a concept of the considered G_i which is identical to a concept of G; the other concepts of G_is will not be preserved in G while recomposing.*

$\mathcal{L}(G)$	$\mathcal{L}(G_1)$	$\mathcal{L}(G_2)$	$\mathcal{L}(G_3)$	$\mathcal{L}(G_4)$	star max biclique of G ?
$1 \times acde$			$1 \times ac$		yes
$2 \times bcfhi$	$2 \times i$	$2 \times bh$	$2 \times bcf$		yes
$3 \times abf$			$\Rightarrow$		
$14 \times de$				$\Rightarrow$	
$5 \times gh$		$\Rightarrow$			
$6 \times bg$		$\Rightarrow$			
$13 \times a$			$\Rightarrow$		
$236 \times b$		$26 \times b$			yes
$12 \times c$			$\Rightarrow$		
$23 \times bf$			$\Rightarrow$		
$56 \times g$		$\Rightarrow$			
$25 \times h$		$\Rightarrow$			

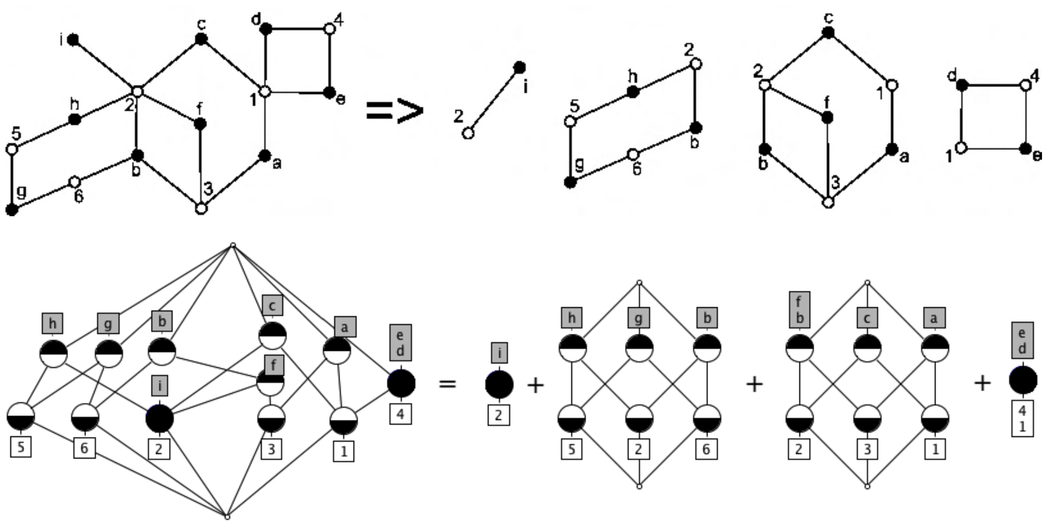

Figure 17: Recomposition of a lattice

According to Theorem 5.3.1, the steps to reconstruct the maximal concepts of the global lattice from the concepts of the smaller lattices are:

1. Compute Σ, the set of attributes and objects involved in a clique minimal separator. (In our example, $\Sigma = \{1, 2, b\}$.)
2. Compute the maximal star bicliques for all the elements of Σ. (In our example, we compute star maximal bicliques $1 \times acde$, $2 \times bcfhi$ and $26 \times b$.)
3. For each smaller lattice, remove from the set of concepts the atoms or co-atoms corresponding to elements of Σ; maintain all the other concepts as concepts of the global lattice. (In our example, for $\mathcal{L}(G_3)$, we remove $1 \times ac$ and $2 \times bcf$, and maintain $3 \times abf$, $13 \times a$, $12 \times c$ and $23 \times bf$ as concepts of $\mathcal{L}(G)$.)

5.4 Using the decomposition as a layout tool

When there is a size two clique separator in the bipartite graph which divides the graph into two components,the elements not involved in the separator can be displayed on the two sides of the separator, thus helping to minimize the number of line crossings in the Hasse diagram.

To illustrate this, we have used our running example with 'Concept Explorer' [45], which is a user-friendly tool for handling lattices. Notice however how clique separator $\{1, d\}$ is better displayed when put at the right extremity.

Figure 18 shows the lattice as proposed by Concept Explorer, and then redrawn with insight on the clique separators of the bipartite graph.

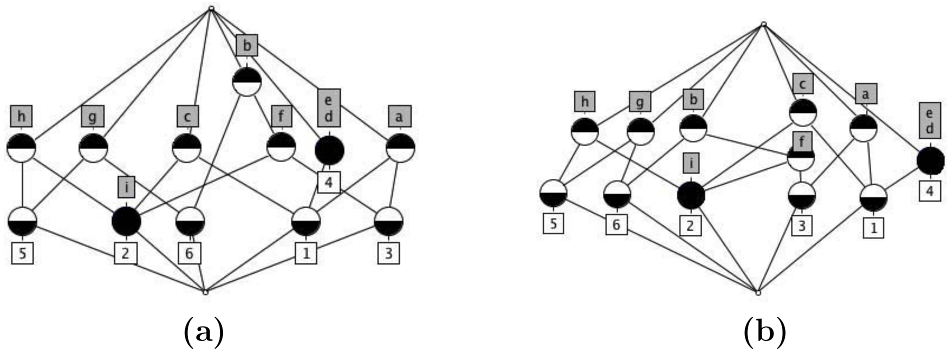

Figure 18: **(a)** Lattice constructed by Concept Explorer using the minimal intersection layout option (8 crossings). **(b)** Lattice re-drawn using the information on clique separators (5 crossings).

Chapter 6

Fused gene detection

Contents

Corresponding paper:
MosaicFinder: Identification of fused genes families in sequence similarity networks.
P.A. Jachiet, R. Pogorelcnik, E. Bapteste and P. Lopez.
In revision for Bioinformatics.

We propose to represent similarity between sequences as a network and a new characterization of families of fused genes as clique minimal separators. Our purpose was to find not only fused genes but to group them into families. The software we implemented, MosaicFinder, not only directly groups fused and component gene families, but also reduces the risk of outputting a large number of false positives.

6.1 Fused gene detection: state of the art

In this section we will need the following definitions:

Definition 8 (Gene fusion point)**.** *The fusion point is the point where two originally unrelated sequences are linked.*

Definition 9 (Fused genes family)**.** *A group of fused genes coming from the same fusion event will be called* Fused gene family*.*

Definition 10 (Component family)**.** *The two unrelated groups of sequences which give birth to a fused gene family will be called* Component family*.*

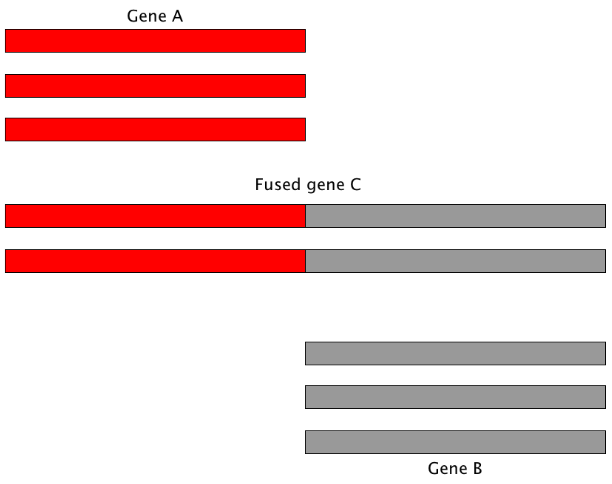

Figure 19: The red sequences (Gene A) and grey sequences (Gene B) sets are the component families of the grey and red sequences (Fused gene C) composing a fused gene family.

Example 12. *In Figure 19, we show a fused gene family formed by two non similars sequences. Red and grey sequences are part of the two component families and the bicolored sequences are part the fused gene family. The fusion point is at the confluence of the red and grey sequences in the fused genes.*

To the best of our knowledge, current *in silico* methods for finding fused genes are based on sequence similarities [27, 16, 37, 38, 15, 31, 34]. The idea is that a fused gene is similar to two genes, which are not pairwise similar and align on disjoint parts of the fused gene. We designate as a *fused triplet* a triplet of genes that exhibits this non transitive pattern of similarity. Many variations around this idea have been implemented to identify fused genes and their components since [27] and [16]. They encounter four types of issues:

- First, the number of fused triplets rapidly becomes enormous for big datasets, namely n^3, where n is the number of similarity links. Previous authors usually distinguished *a priori* between a query dataset (genome) within which fused genes were searched for, and a reference dataset (genomes, COGs) in which components could be found. This greatly reduced the number of candidate triplets, since only a subset are thus investigated.

- Second, some triplets may not result from a fusion but from sequencing or prediction errors [29], if a gene is artificially split into two separate genes or if two adjacent genes are artificially fused into a single one. Two types of tests are usually performed to get rid off those false positives. First, one cross-checks that component genes are not similar, either with the same algorithm at a more permissive threshold (most of the time a higher BLAST E-value), or with a more accurate algorithm such as Smith-Waterman [36]. Second, one tests that component genes align along non-overlapping regions of the candidate fused genes. These controls eliminate many false positives.

- A third and central issue is the grouping of identified components and fused genes *a posteriori* into gene families descending from a common fusion event. This grouping is necessary to count evolutionary events, to perform general functional analyses. First, if fused triplets descending from the same fusion event are grouped, it would summarize the information contained in this set of triplets of genes. Second, potentially artefactual fused or component genes would be easily identified, as they are the only representatives of their family.

> To group fusion events into families a method is to map genes on pre-existing family classifications [38, 43], usually COG/KOG [41]. This is only partially satisfactory, as by definition families of fused genes do not match a single COG and therefore are overlooked by that approach. Novel gene families (e.g. environmental) that have not been associated to a COG family are likewise difficult to detect. Alternatively, [17] grouped fused genes into families by simple linkage. This is straightforward as similarities between sequences are already computed to look for fused triplets. But simple linkage will aggregate unrelated fused genes if multiple fusion events have occurred in the history of some genes. Moreover, this method does not allow to reconstruct component gene families and their relation with fused gene families.

We implemented a program called FusedTriplets that unified the known methods in the fusion events research based on finding two genes not similar to each other but similar to a third one.

6.2 MosaicFinder implementation

When exploring a large dataset including several genomes, there may be several representatives of a given fusion event, which we want to group into fused families and component gene families.

The approach we choose to deal with this problem is to use a sequence similarity network. These networks are graphs with sequences (or proteins) as vertices, directly connected by edges when they show a similarity greater than a user-defined threshold. We use the tool BLAST [1] to examine this similarity. BLAST give scores (e-value, pident) to evaluate the sequence similarities. The user can choose as a threshold the e-value or/and the pident respectively below and above which sequences as considered similar. As shown in Figure 20, at stringent thresholds, the sequences are highly disconnected. By relaxing the threshold, edges corresponding to less significant similarities are added to the network until all the proteins are interconnected. We leave the threshold decision to the user in order to obtain a network which met our requirements: sequences that are aligned on most of their length must all be connected and the threshold must be permissive enough to add edges between sequences with a partial alignment. In this context, a family of fused genes has the particularity to connect non connected groups of vertices which prompts us to characterize fused genes families as clique minimal separators of the sequence similarity network. A fused gene family is a separator, since its removal would disconnect component gene families and by extension the graph. The minimal condition avoids creating different families for the same event. The additional condition that the separator is a clique describes the requirement that the family of fused genes is conserved (all members of the family are similar).

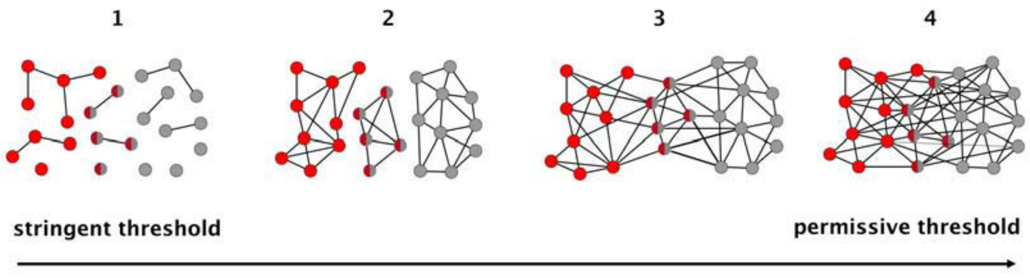

Figure 20: Effect of the threshold on a sequence similarity network. Red and grey vertices represent sequences of the component families and bicolored vertices sequences of the fused genes. If the alignment of two sequences is better than a threshold an edge is added between the two representative vertices. At stringent threshold (1), most of sequences are disconnected. As the threshold is relaxed sequences from the same families merge together until they form a complete network. In the intermediate threshold (3), the sequences of fused genes are connected to genes of the two component families.

Example 13. *In Figure 21, the fused genes (bicolored) form a clique which separates the two sets of vertices (grey and red) which represent the component families.*

MosaicFinder works in several consecutive steps. It first uses BLAST information to construct a sequence similarity network. A variation of the Algorithms Minseps-Maxcliques and Expanded-MCS-M is then applied to

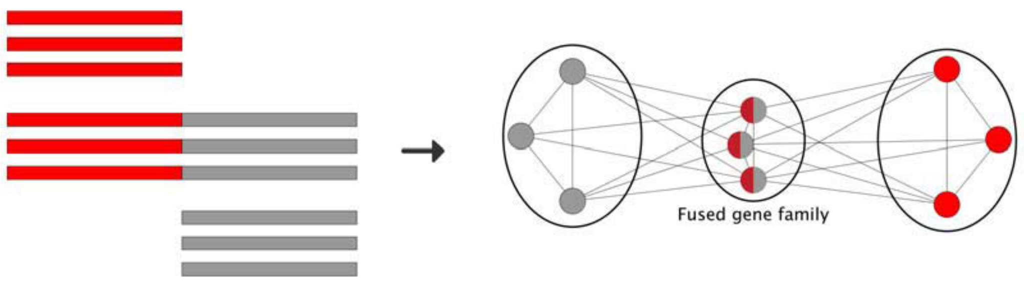

Figure 21: (a.) Multiple alignment of fused genes (bicolored) and component genes (grey and red). (b.) Similarity network of those genes. The bicolored vertices form a fused gene family. They are a clique minimal separator of the network. The red vertices and the grey vertices form two separate component families.

triangulate the graph and find clique minimal separators in this network in a single pass, to propose candidate families of fused genes. The component families are then defined. False positives are discarded by testing that component families align along non overlapping regions of the candidate fused genes. Finally, MosaicFinder proposes a fusion point family.

STEP 1: Construction of the similarity network

MosaicFinder takes as input the result of all-against-all BLAST comparisons between the sequences under study, in the form of a simple flat file, including information about the region which aligns pairs of sequences. To determine whether two sequences are similar, MosaicFinder considers the user threshold defined on the BLAST scores.

STEP 2: Identification of fused gene families

We applied Expanded-MCS-M algorithm but we only look for separators and ignore the atom computing part. These separators are tested for completeness.

STEP 3: Identification of component families

Once a clique minimal separator is found, we associate it with component families in the following fashion, as shown in Figure 22:

- We first compute the common neighbourhood of the clique minimal separator vertices.
- We then consider the subgraph formed by this common neighbourhood.
- We search the connected components of this subgraph which will be our component family.

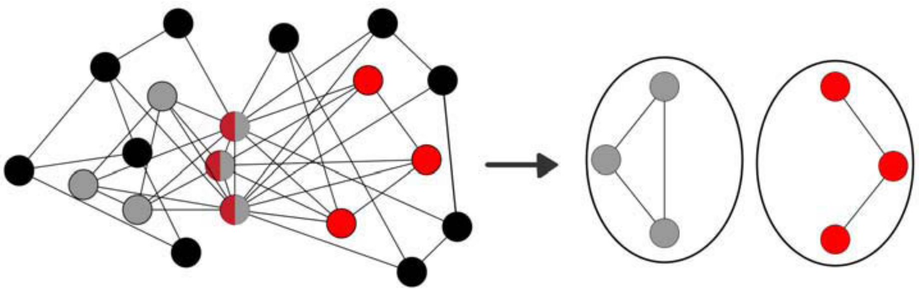
Figure 22: Bicolored nodes are in a clique minimal separator. Black, red and grey nodes represent common neighbourhood of the clique minimal separators. The subgraph induced by the common neighbourhood contains two connected components, which define its component families.

STEP 4: Cross-checking the component families (optional)

MosaicFinder can test if a separator is reliable. If the threshold was too stringent, similarities between families might not have been detected and similar genes not connected in the common neighbourhood of a clique minimal separator because some edges remain hidden at this stringent threshold. Component families defined at that threshold result from artefactual disconnection in this neighbourhood. These disconnections would not be observed at a less stringent threshold of similarity. Consequently, MosaicFinder uses a second less stringent threshold of similarity to evaluate the number of connected components in the the common neighbourhood. If at this threshold there are less than 2 component families the separator is not considered.

STEP 5: Use of alignments to eliminate false positives

MosaicFinder tests the presence of false positives using information about the regions of the sequences that align in BLAST. There is a false positive when the different component families align in the same region of a candidate fused gene, because such a significant overlap in an alignment suggests that similarity between sequences of different component families was undetected. As different genes from a component family may align to slightly different parts of a potential fused gene, we compute the median alignment. MosaicFinder rejects a candidate fused gene if the median alignment of different component families overlaps on more than 20 amino acids. This small overlap is allowed because BLAST extends alignments as far as possible and small regions align artefactually. Otherwise, the fused gene is accepted, and a fusion point is calculated as the middle point between the median alignments of component families.

STEP 6: Output

MosaicFinder outputs a table of genes involved in fusion events. This table indicates the fusion event genes are involved in, and their grouping into fused or component families. It additionally indicates a fusion point for fused genes.

6.3 Results

We implemented MosaicFinder to compare the detection of fused genes families with the existing methods for detecting fused genes (FusedTriplets implementation). As there exists no large manually curated database of fused genes for testing, we used simulated data to test the accuracy of MosaicFinder.

We also ran tests on real databases, but we have less information on the validity of our methods in this context. We focussed our attention on the number of fused genes detected.

6.3.1 Test of MosaicFinder on simulated fused genes families

Our dataset contains 92,229 sequences, from 2000 independent gene families and 1000 fused genes families. We compared all pairs of genes from this dataset with BLAST. We searched the resulting similarity network with MosaicFinder and FusedTriplets, either with a unique 1e-5 E-value threshold, or with a 1e-10 E-value threshold and a cross-check of the absence of similarity between component genes/families with a more permissive 1e-5 E-value threshold (STEP 4).

Table 1 reports our results: the total number of true and false positive is given for each of the two programs, either with one or two thresholds (optional cross-check). Without distant homology cross-check, FusedTriplets identified most of the fused genes (92%) but has a high false positive rate (35% of identified fused genes here). On the other hand, MosaicFinder has a lower positive rate (76%), but almost no false positives (1%). With the optional second threshold cross-check, fewer true fused genes are identified, but the false positive rates drops to 5% for FusedTriplets and 0% for MosaicFinder. These results stress the importance of the two-threshold test for FusedTriplets, and the comparative robustness of MosaicFinder. We believe that the robustness of MosaicFinder may be a great advantage on biological datasets. Real evolution of sequences is more complex than simulated data which may increase the number of false positives. Regarding the groupings into families, all the genes which were grouped by MosaicFinder belong to the same simulated family. MosaicFinder finds 92% of the conserved fused genes, and exhibits no error in grouping them. Regarding the fusion points, we find that MosaicFinder accordingly estimates the position of the fusion points. 94% of the computed fusion points are less than 5 amino acids away from the true fusion point, and 99% less than 16 amino acids away. This variation is due to the imprecision of BLAST alignments. Those numbers validate the 20 amino acids overlap allowed between component families on fused genes. Regarding the running time, we find that MosaicFinder runs much faster (30s) on this simulated dataset than FusedTriplets (1m39s).

	E-value : 1e-5		E-value : 1e-10 then 1e-5	
	MosaicFinder	Triplets	MosaicFinder	FusedTriplets
Simulated fused genes	4833	4833	4833	4833
True positive	3663	4444	3260	4130
False positive	45	2380	0	221
Simulated fused events	1000	1000	1000	1000
Identified fused events	892	-	813	-

Table 1: Results of MosaicFinder and FusedTriplets on simulated gene fusion events, either with or without two-threshold distant homology cross-check

6.3.2 Test on real data

We built datasets composed of various sizes, composed of 1 to 30 prokaryotic complete genomes, to search for fused gene families and component families.

Figure 23 presents the number of fusion events and fused triplets respectively output by MosaicFinder and FusedTriplets, compared to the number of similarity links (edges) between sequences. For the biggest dataset, FusedTriplets outputs an enormous amount of fused triplets (up to 11 millions) grouped into 5339 potential fused genes (note that a given fused gene can correspond to many different triplets), whereas MosaicFinder finds 2490 fused genes grouped into 1821 fusion events. According to our results on simulated data, FusedTriplet leads to more false positives than MosaicFinder, this leads us to presume that there is a great number of potentially misleading fusion events generate by fusedTriplets for this dataset, and shows the interest of MosaicFinder to identify them.

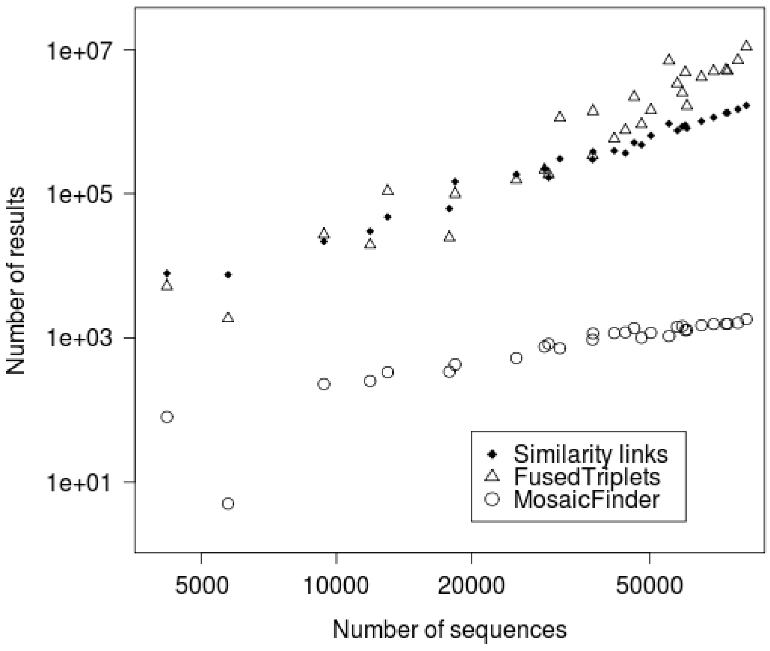

Figure 23: Comparison of the number of similarity links between sequences, the number of identified fused families by MosaicFinder, and the number of identified fused triplets by FusedTriplets (logarithmic scales).

6.4 Discussion

We proposed a new characterization of families of fused genes, as clique minimal separators in sequence similarity networks, and implemented this method into the C++ program MosaicFinder. We showed that on simulated data MosaicFinder identifies conserved fused genes families very well and partially unconserved fused gene families.MosaicFinder proves to have a very low false positive rate. Moreover, this information may be visualized as an annotated graph using tool as Cytoscape [35]. Figure 24 gives an example.

We show that MosaicFinder gives good results quickly, with the advantage that genes are grouped into families, thus avoiding the extra work of regrouping the fused genes after they have been found.

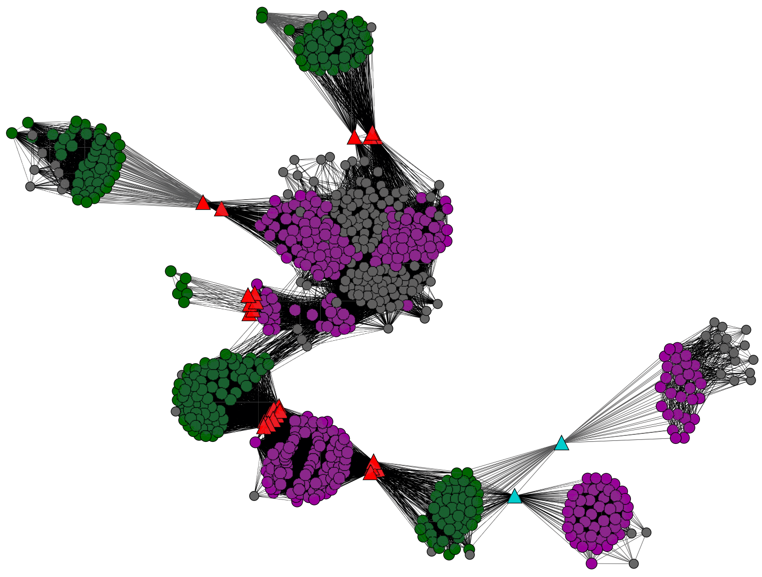

Figure 24: Results layout using Cytoscape. Fused families are represented as red triangle (blue for families of size one). Component families are represented in green and purple.

Chapter 7

Extraction of genes of interest

Contents

One of the applications of our work on clique minimal separators was the decomposition of networks constructed using gene expression data. This work was in collaboration with different laboratories but the work exposed here is the fruit of the collaboration with INRA.

To understand the behaviour of cells it is necessary to understand how genes interact together, transcriptomic tries to answer this question based on gene expression levels. We consider that genes having a close expression or correlated expression must be related. The aim of this work was to find sets of genes of interest using the atom tree to represent microarrays. The set of genes of interest correspond to genes involved in the cellularisation and accumulation stage in the wheat grains.

7.1 Dataset

The data we describe here are issued from wheat (*Tricutun aestivium*) grains collected at 11 stages after flowering. The two first were taken at the cellularisation stage and the nine others one were collected during the accumulation stage at 50 growing degree days (# of days * temperature) interval. The studied genes are 2295 transcription factors.

7.2 From expression level data to graph

	t1	t2	t3	t4	t5	t6
Gene1	2.5	3	5	2.5	3	5
Gene2	5	9	5	2.5	3	5
Gene3	6	8	4	2.5	3	5

Table 2: Expression level data.

As explained in the preliminary chapter, we start by creating an expression level network. The dataset is a flat file formatted as shows in Table 2, each line is a gene, the first row will be the gene names and the following row are the expression at different times. Computing the distance between gene expression levels, we obtain a distance matrix. Once we have this distance matrix we will apply a threshold under which genes are considered close enough to have an edge between them in the network. This process leads to the creation of an adjacency matrix. The threshold was chosen empirically.

7.3 Results

The first investigation was on expression level network computed with the Euclidean distance. Using our software we created an atom tree of the expression level network on a range of thresholds. We investigated the atom tree allowing to extract a humanly manageable amount of genes. The network shown in Figure 25 was created for a threshold of 2.2.

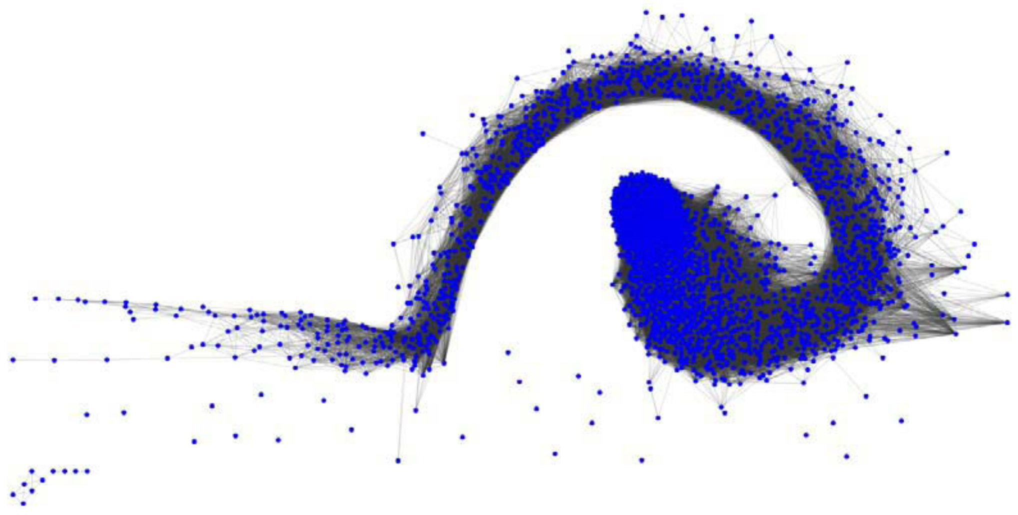

Figure 25: The expression level network for a threshold of 2.2. Vertices correspond to the genes. There is an edges between two vertices if the the Euclidean distance between their expression level for the 11 data points is lower than 2.2.

At this threshold we have 22 isolated vertices, two connected components of size two, a connected component of size 8 with one clique minimal separator, a connected component of size 8 with one separator of size two and the major connected component of size 2273 with 34 clique minimal separators. We will describe expression of the genes of these different connected components.

Connected components of size one or two

All these connected components are represented by one atom composed by one or two genes. The expression profiles of these genes exhibit a variation in their expression levels among the experiments, as can be seen in Figure 26. These genes extracted due to the conformation of the network can be retained for further investigation.

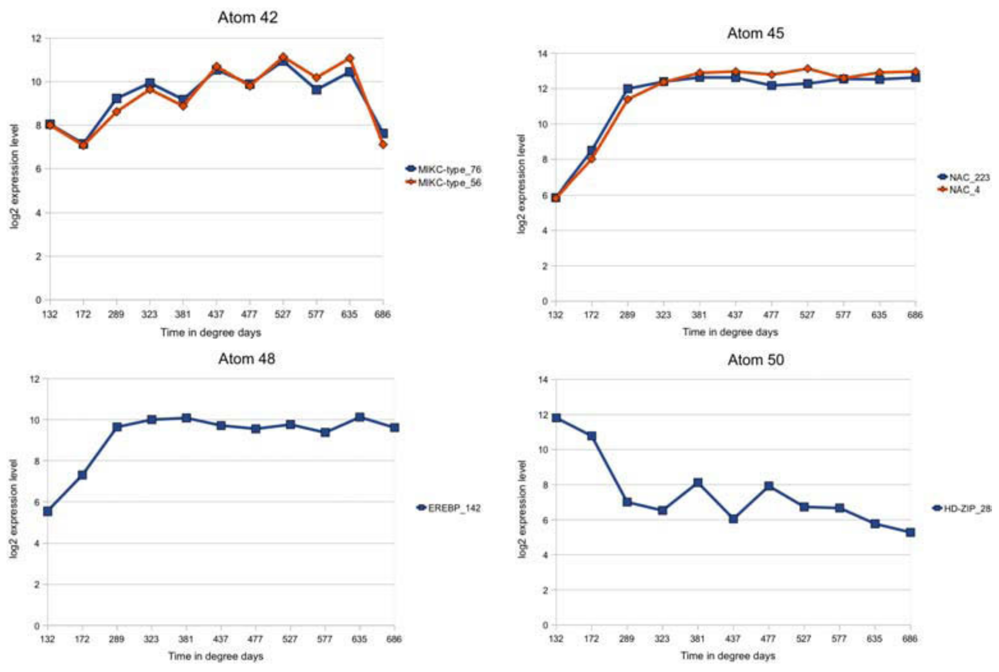

Figure 26: Expression profiles of four atoms, two of size one and two of size two. The expression profiles of the genes of Atom 42 increased through the first ten stages of the wheat grain development and decreased at the last stage. Gene expressions of Atom 45 and Atom 48 increased sharply during the third stage of development before stabilizing. The expression of the gene in Atom 50 decreased through the eleven stages of wheat grain development.

Connected component of size 8

This connected component with one clique minimal separator can be decomposed into two atoms. Genes composing these atoms are highly expressed in all samples (Figure 27), which explains the fact that they are grouped and isolated in the graph at this threshold. We can observe how coherently genes are grouped in these atoms.

Main connected component

For the principal connected component we observe a sun shaped atom tree with disjoint atoms (Figure 28). Atom 1 in the center of the sun is the atom regrouping most of the genes (2224/2295). This is due to the fact it

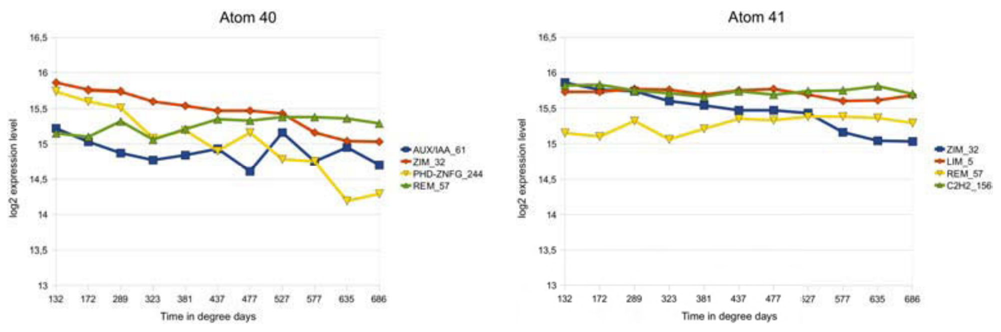

Figure 27: Expression profiles of two atoms of the connected component of size 8. The expression of the genes of Atom 40 and of Atom 41 is high and stable through the 11 development stages of the wheat grains.

groups genes with an expression stable through the eleven stages of the wheat grain development which dominate the dataset. On closer inspection of the expressions of each gene in Atom 1, we can observe that most of them have a flat expression profile in the range of expressions from 4.37 to 14.11. The average expression variance of these genes through the experiment is 0.28. No genes in Atom 1 have an expression variance greater than 0.69 unlike genes that are grouped in the other atoms which have an expression variance greater than 1.

Atoms linked to Atom 1 formed our groups of interests. Genes composing these atoms have different expression profiles exhibiting variation during time (Figure 29). The variations of the expressions of these genes show that they play a specific role during the wheat development and should be investigated. We can highlight that expression profiles of the atoms are coherent.

When we look at the branch starting from Atom 1 (Atom 17, Atom 26, Atom 28, Atom 36), we observe an expression profile evolution (Figure 30). Atom 17 at distance 1 is composed of genes with an expression increased during the 4 first collected samples and stabilizes at a high expression value in the seven last samples. As the distance from Atom 1 grows, genes composing atoms have the same pattern with a stronger incrementation of the expression in the fourth stage.

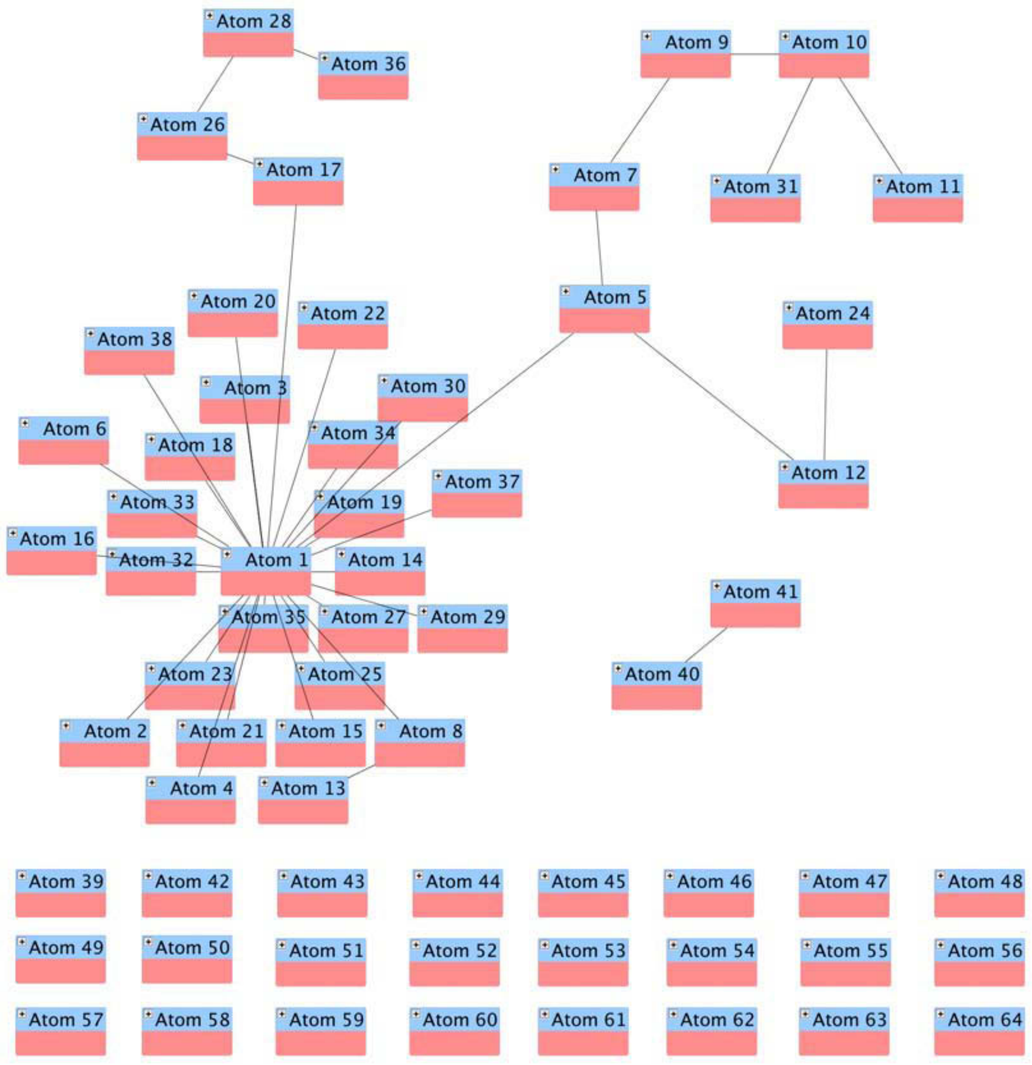

Figure 28: The atom tree of the network of Figure represented using yEd.

7.4 Discussion

Using our decomposition, we easily and quickly creates a subset of interesting genes. Decomposition of the network using our method often create a big central cluster at small distance thresholds. This aspect allows biologists to extract informative genes quickly from experimental data. It is important to extract modules of closed genes for further experimentation. Through the

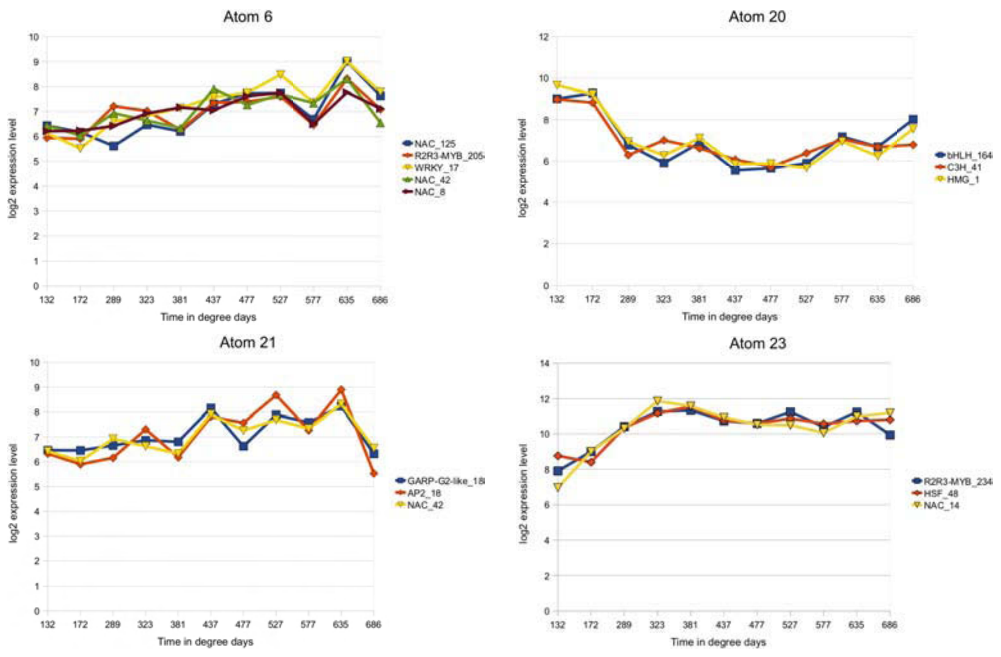

Figure 29: Expression profiles of atoms linked to the central atom. The expression of the genes from Atom 6 and Atom 21 increased through the first ten stages of the wheat grain development and decreased at the last stage. The expression of the genes of Atom 20 decreased sharply during the first and second stages of development before it stabilizes. The expression of genes from Atom 23 increased sharply during the first three stages of development before stabilizing.

atom tree we can see very coherent clusters of genes with special patterns that give good hopes for further biological investigation. It is necessary to highlight that an atom tree computed using the Pearson correlation also creates coherent atoms but with patterns not corresponding to the same genes.

The implementation of this method uses our work on the atom tree which allows a good visualization. We can quickly detect disjoint atoms and groups of atoms evolving in the same way but with different amplitudes following chains. Our implementation allows to output the atom tree in GraphML [13] format as well as dot format. The GraphML output can be vizualized tools as yEd (in our example). This allows to see the atom tree and to open the

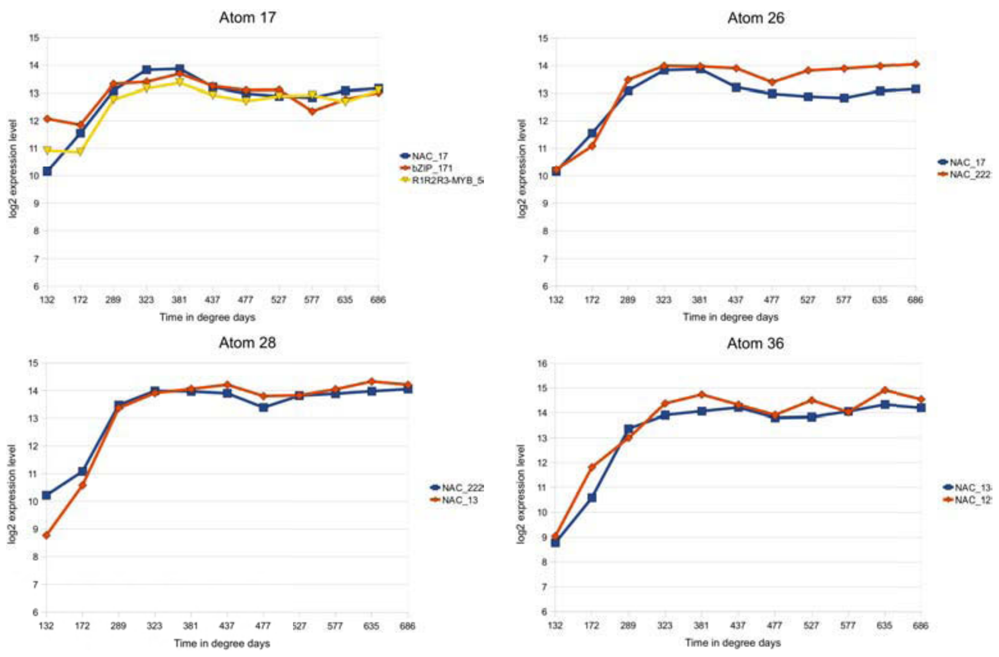

Figure 30: One branch of size 4 of the atom tree composed of Atom 17, Atom 26, Atom 28 and Atom 36. The expression of all the genes from these atoms increased within the first four development stages before it stabilized. The farther an atom is from Atom 1 the greater the increasement of the expression level is.

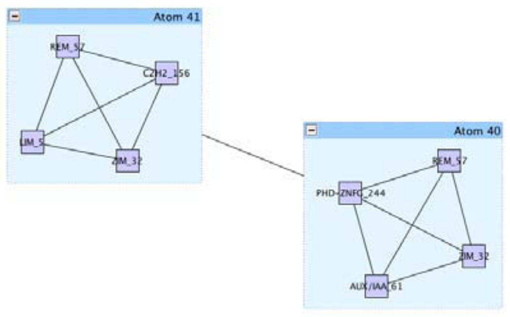

Figure 31: Atom 40 and Atom 41 are opened to see the interactions of genes in this atoms.

atom we want to inspect and see the gene interaction within the atom Figure 31. The dot output is less friendly for the inspection of atoms but GraphML for big dataset produce heavy files because of the verbosity of this language.

This approach will give rise to two papers currently in preparation, one presenting the implementation of our decomposition, the second one elaborating on the atom results in collaboration BIOGEMMA on public data.

Conclusion

In this thesis we worked on graph decomposition using clique minimal separators. Our motivation was to improve existing algorithms and to apply this decomposition to different fields such as biology.

Since the first step of clique separator decomposition is to make the graph chordal, our first contribution was the creation of an algorithm to generate minimal separators and maximal cliques of a chordal graph using either Lex-BFS or MCS.

This first result was followed by a work on the clique tree. Based on this notion, we defined the atom tree which allows to represent atoms and clique minimal separators of a graph. Using this structure we developed two algorithms that in certain contexts allowed a faster decomposition.

We applied our decomposition to bipartite graphs to provide a Divide and Conquer approach to generate the corresponding Galois lattice and may help visualizing the lattice by minimizing edges crossing.

We implemented a software, based on our algorithms called MosaicFinder, to detect fused genes families and their component families. We show that Mosaic Finder gives good results quickly, with the advantage that genes are grouped into families, thus avoiding the extra work of grouping the fused genes after they are output.

We also implemented our method to decompose a gene expression level network in order to extract genes of interest using microarray data. Several experiments on expression level data using the atom tree were successful and the collaboration with BIOGEMMA continues in the spirit of intense cooperation.

This thesis yields some interesting perspectives.

We ran some experiments to obtain a minimal triangulation by an exact algorithm or an heuristic adding edges not crossing the clique minimal separators. We also have algorithms avoiding the triangulation step that will be investigated.

For the detection of fused genes our current work consists in breaking up very long cycles with a local approach since long cycles may mask fusion families. We are also running some experiment to find minimal separators which are quasi-cliques to detect unconserved fused gene families.

We also plan to investigate the use of this decomposition to tackle problems in other biological contexts where a network can be used.

Bibliography

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, Oct. 1990.

[2] S. R. Arikati and C. P. Rangan. An efficient algorithm for finding a two-pair, and its applications. *Discrete Applied Mathematics*, 31(1):71 – 74, 1991.

[3] A. Berry, J. R. S. Blair, P. Heggernes, and B. W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39:287–298, 2004. 10.1007/s00453-004-1084-3.

[4] A. Berry and J.-P. Bordat. Separability generalizes dirac's theorem. *Discrete Applied Mathematics*, 84(1-3):43–53, May 1998.

[5] A. Berry, J.-P. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. *Journal of Algorithms*, 58(1):33 – 66, 2006.

[6] A. Berry, R. Krueger, and G. Simonet. Maximal label search algorithms to compute perfect and minimal elimination orderings. *SIAM Journal on Discrete Mathematics*, 23(1):428–446, 2009.

[7] A. Berry and R. Pogorelcnik. A simple algorithm to generate the minimal separators and the maximal cliques of a chordal graph. *Information Processing Letters*, 111(11):508 – 511, 2011.

[8] A. Berry, R. Pogorelcnik, and A. Sigayret. Vertical decomposition of a lattice using clique separators. In *The 8th International Conference on Concept Lattices and their Applications (CLA11)*, 2011.

[9] A. Berry, R. Pogorelcnik, and G. Simonet. An introduction to clique minimal separator decomposition. *Algorithms*, 3:197–215, May 2010.

[10] M. D. Biha, B. Kaba, M.-J. Meurs, and E. SanJuan. Graph decomposition approaches for terminology graphs. In *Proceedings of the artificial intelligence 6th Mexican international conference on Advances in artificial intelligence*, MICAI'07, pages 883–893, Berlin, Heidelberg, 2007. Springer-Verlag.

[11] J. R. S. Blair, P. Heggernes, and J. A. Telle. A practical algorithm for making filled graphs minimal. *Theoretical Computer Science*, 250(1-2):125 – 141, 2001.

[12] J. R. S. Blair and B. W. Peyton. An introduction to chordal graphs and clique trees. *Contract*, 56:1–1, 1991.

[13] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall. Graphml progress report - structural layer proposal, 2002.

[14] E. Dahlhaus, M. Karpinski, and M. B. Novick. Fast parallel algorithms for the clique separator decomposition. Technical report, 1990.

[15] P. Durrens, M. Nikolski, and D. Sherman. Fusion and fission of genes define a metric between fungal genomes. *PLoS Comput Biol*, 4(10):e1000200, Oct. 2008.

[16] A. J. Enright, I. Iliopoulos, N. C. Kyrpides, and C. A. Ouzounis. Protein interaction maps for complete genomes based on gene fusion events. *Nature*, 402(6757):86–90, Nov. 1999.

[17] A. J. Enright and C. A. Ouzounis. GeneRAGE: a robust algorithm for sequence clustering and domain detection. *Bioinformatics*, 16(5):451–457, Jan. 2000.

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

[19] F. Gavril. Algorithms on clique separable graphs. *Discrete Mathematics*, 19(2):159 – 165, 1977.

[20] J. Han. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[21] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[22] J. Hopcroft and R. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.

[23] W.-L. Hsu and T.-H. Ma.

[24] B. Kaba, N. Pinet, G. Lelandais, A. Sigayret, and A. Berry. Clustering gene expression data using graph separators. *In Silico Biology*, 7(4):433–452, 01 2007.

[25] H.-G. Leimer. Optimal decomposition by clique separators. *Discrete Mathematics*, 113(1-3):99–123, Apr. 1993.

[26] M. Long. A new function evolved from gene fusion. *Genome Research*, 10(11):1655–1657, Jan. 2000.

[27] E. M. Marcotte, M. Pellegrini, H. Ng, D. W. Rice, T. O. Yeates, and D. Eisenberg. Detecting protein function and Protein-Protein interactions from genome sequences. *Science*, 285(5428):751 –753, July 1999.

[28] A. Parra and P. Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Applied Mathematics*, 79(1 - 3):171 – 188, 1997.

[29] S. Pasek, J. Risler, and P. Brézellec. Gene fusion/fission is a major contributor to evolution of multi-domain bacterial proteins. *Bioinformatics*, 22(12):1418 –1423, June 2006.

[30] R. Pogorelcnik. MosaicFinder. `http://www.isima.fr/~pogorelc/MosaicFinder.zip`.

[31] R. L. Rogers, T. Bedford, and D. L. Hartl. Formation and longevity of chimeric and duplicate genes in drosophila melanogaster. *Genetics*, 181(1):313–322, Jan. 2009. PMID: 19015547 PMCID: 2621179.

[32] D. J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(3):597 – 609, 1970.

[33] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

[34] H. M. W. Salim, A. M. Koire, N. A. Stover, and A. R. O. Cavalcanti. deFuser / detection of fused genes in eukaryotic genomes using gene deFuser: analysis of the tetrahymena thermophila genome. *BMC Bioinformatics*, 12:279, 2011. PMID: 21745395.

[35] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, Jan. 2003.

[36] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, Mar. 1981.

[37] B. B. Snel, P. P. Bork, M. M. Huynen, et al. Genome evolution-gene fusion versus gene fission. *Trends in genetics*, 16:9–11, 2000.

[38] K. Suhre. FusionDB: a database for in-depth analysis of prokaryotic gene fusion events. *Nucleic Acids Research*, 32(90001):273D–276, Jan. 2004.

[39] R. E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221 – 232, 1985.

[40] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, July 1984.

[41] R. L. Tatusov, N. D. Fedorova, J. D. Jackson, A. R. Jacobs, B. Kiryutin, E. V. Koonin, D. M. Krylov, R. Mazumder, S. L. Mekhedov, A. N. Nikolskaya, B. S. Rao, S. Smirnov, A. V. Sverdlov, S. Vasudevan, Y. I. Wolf, J. J. Yin, and D. A. Natale. The COG database: an updated version includes eukaryotes. *BMC Bioinformatics*, 4(1):41, Sept. 2003.

[42] S. Whitesides. An algorithm for finding clique cut-sets. *Information Processing Letters*, 12(1):31 – 32, 1981.

[43] I. Yanai, A. Derti, and C. DeLisi. Genes linked by fusion events are generally of the same functional category: A systematic analysis of 30 microbial genomes. *Proceedings of the National Academy of Sciences*, 98(14):7940–7945, Mar. 2001.

[44] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, 1981.

[45] S. A. Yevtushenko. System of data analysis ‘concept explorer’. *Proc. of the 7th National Conference on Artificial Intelligence KII-2000*, pages 127–134, 2000. Russian.

[46] Q. Zhou, G.-j. Zhang, Y. Zhang, S.-y. Xu, R.-p. Zhao, Z. Zhan, X. Li, Y. Ding, S. Yang, and W. Wang. On the origin of new genes in drosophila. *Genome Research*, Jan. 2008.

List of Figures

List of Tables

List of Algorithms