Université Montpellier II Sciences et Techniques du Languedoc

Thesis Dissertation

for the grade of Docteur de l'Université Montpellier II

Doctoral School École Doctorale Information, Structures, et Systèmes
Doctoral Program Informatique

with a public presentation and defence on September 9th 2008

Tree-Representation of Set Families in Graph Decompositions and Efficient Algorithms

by

Binh-Minh Bui-Xuan

Thesis committee

Thomas Erlebach	(rapporteur)	Professor at University of Leicester
Michel Habib	(supervisor)	Professor at Université Paris VII
Pavol Hell	(rapporteur)	Professor at Simon Fraser University
Christophe Paul		Chargé de Recherche at Cnrs, Lirmm
Stéphan Thomassé		Professor at Université Montpellier II
Ioan Todinca	(president)	Professor at Université d'Orléans

The dissertation was prepared at the research group "Visualisation and Algorithms for Graphs" of the "Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier"

Foreword and Acknowledgements

This thesis is conducted within the "École Doctorale Information, Structures, et Systèmes" at the University Montpellier II, France. It is written in English, and an extended abstract of its contents is given in French at the end.

Also, and most importantly, this thesis would not have been possible without the priceless help of many people. As well, for most of them not only I am grateful as being the author of the thesis, but also as being the lucky person who had the privilege to meet – and for a few cases to live with – them here below. (I can only speak for "most of them" because there are some I have never met in real life.) These are important encounters to me and have many consequences to, among other things, the very current dissertation as I strongly believe my perspectives would have never become what they are now if ever these meetings have differed, for better or for worse that is.

Of course it is impossible to make an exhaustive enumeration of such acknowledging declarations, and it is by no means the purpose of the following lines since I fail in seeing how thanks could be expressed by mere words. However, this kind of acknowledgement sections is usually said to be the only place in thesis dissertations where Ph. D. students could feel free to put anything inside (isn't it!). Then, according to the common practice, I would like to take the opportunity and sketch some personal feelings here.

Firstly, I am grateful to Michel Habib for, well, an unbounded amount of topics (and also of beer). Among other things, *Castelnau-le-Lez* is a pleasant town, *Palavas-les-Flots* a good haven, *Café de la Mer* a strange place, *Men Meur III* a suntanning sailboat (at least it is for lazy me), and the baby-bed still in use. Besides these important checkpoints and matters, I am also grateful for his supervision calibre despite the geographical distance between Montpellier and Paris. On the one hand, it was particularly nice during some everlasting months a year ago or so to still trust, work together, and give encouragements when no other sensible person I know seemed to be willing to continue to believe any further in the project I was working on at that time. I acknowledge this as a kind of seldom magic which I personally would take into account in those now-trendy discussions around the strange concept of a "chercheur vs. trouveur", should I ever have to consider

funny such discussions. On the other hand, I was also particularly impressed that day, some years ago now, by how well the piece of advice was given against some research ventures I was to commit, leaving no place there to be taken as an administering action. This probably vindicates true instances of leadership, also I would acknowledge it as is. Finally, though I will ever – and ever again – be surprised by the eventually quite low frequency in which we meet for *working* purposes, I am willing to acknowledge this fact as a unique opportunity for feeding my personal topics of interest – as well as some beliefs, from a certain point of view.

I am grateful to the other members of the committee: Thomas Erlebach, Pavol Hell, Christophe Paul, Stéphan Thomassé, and Ioan Todinca, and especially to the *rapporteurs*. This is not only for their interest in my work and the subsequent refereeing, but also for numerous comments which help improving greatly the thesis at many planes.

I would like to send a special thank to all those who did the big favour of reading some early versions of the thesis and giving feedback on them. They are, in lexicographic order: Attila Bernáth, Paul Bonsma, Bùi Xuân Hải, David Coudert, Serge Gaspers, Erik Jan van Leeuwen, Daniel Meister, Adèle Mennerat, Fabien de Montgolfier, Michaël Rao, and Jan Arne Telle. I cannot even think about what the thesis would look like without their great help.

I am also specially grateful to those who helped me in finding, probably not the best, but at least the most acceptable way for presenting the thesis in 45 minutes. For this I thank, in addition to the previously mentioned persons: Vincent Berry, Olivier Cogis, Jean Daligault, Sylvain Durand, Philippe Gambette, Daniel Gonçalves, Vincent Limouzy, Jérôme Palaysi, Anthony Perez, Alexandre Pinlou, Chedy Raïssi, Guillaume Verger, and Marie-Catherine Vilarem. A second thank to Vincent Limouzy for more suntanning sessions on sweet and swift *Maderka II*.

I had the opportunity to learn a lot of things from working sessions and occasional discussions with very good fellow researchers. Sometimes the discussions consisted of unfamiliar concepts, most of the time they were just plainly strange. For these unexpected adventures, I am also grateful to: Sandeep Bhadra, Pierre Charbit, Christophe Crespelle, Benoit Darties, Afonso Ferreira, Aubin Jarry, Marc Plantevit, Martin Vatshelle, and Yngve Villanger.

Teaching has been a dear exercise to me, and I have learned much from such experience. Thus, I am grateful to those who provided me with teaching duties for broadening my perspectives. Here, the additional names will be: Stéphane Bessy, Norbert Kern, Frédéric Koriche, Gaël Isoird, Pierre Pompidor, and Richard Terrat. A second and specially-big thank to Pierre Pompidor for some quasi-ritual sessions. I would also like to thank my

past students for unfailingly forcing me to find better explanations (again, and still again), no matter what the topic is about. Somehow, this thesis would have never been the same without their efforts.

I am also grateful to the other colleagues of the LIRMM for some fun moments during the thesis period. An additional but not exhaustive list is: Maxime Collomb, Emeric Gioan, Flavio Guinez, Mountaz Hascoët, Philippe Janssen, Nancy Rodriguez, and Gilles Simonin.

I would like to send esteems to my past teachers, and especially to Thầy Trần Văn Hạo and Richard Antetomaso. I am grateful to Thầy Hạo for the monthly assignments he gave. They probably settled the most solid basis for my current mathematical standpoint. I am grateful to Richard "Toto" for excessive training and for letting me know one can at the same time "do maths", be a fan of *Olympique de Marseille*, and drink soda at the blackboard. I am also grateful to Thầy Hạo for making me realize that outranking several contests does not necessarily imply being good at the exercise, and to Richard "Toto" for making me realize that some contest just has to be won.

I thank Yann Palu and François Simenhaus for some badminton contests a decade ago, and more recently for some refreshing phone calls on linear algebra. For purely scientific reasons (perhaps), I would like to thank my long-lasting friend Minh Ngọc, who, I believe, was the first person to make me realize that some generic algorithms just won't work properly, and the automatic reflexes I had at that time with occidental chess would not let me beat her at oriental chess. Another person who has been doing the same job as winning a lot on me is Quang Hiển. I thank him for that. I am grateful to Paven Byzov for saving my life and making me realize that some research ventures are simply not meant to be committed, especially when it comes to a half-frozen lake.

Thank Gram'p for, among other things, a sample of Ginette Mathiot's *La cuisine pour tous*, the only programming language I am willing to bring back home. Thank Bà Ngoại and Tat' for demonstrating their art in that discipline. Thank Ông Nội for his verses, even if they include some strange grammar sometimes. Thank Mẹ for bringing me up without a word about computers, despite her expertise on the matter. Somehow I think she showed me the way... Thank Ba for never being the first of us to talk about mathematics, for always answering to my questions with delight and enthusiasm, and for making me take mathematics as a cheeriness, not a burden, as a trick, not plainly magick, finally – and most importantly – as a job, not a life. I thank him also for providing me with the *Moka of Buôn Mê Thuột district*, without the use of which this dissertation would have never started. I am afraid to say, but thank Bé Hà and Ngay for serving the *rice spirits** *of Phát*

*This is not to be mistaken with the quite fashionable *sake*, which comes originally from Japan.

Diệm district at their wedding, as ever since I have been hoarding its alike, and without the use of which this dissertation would have never ended.

Not that I expect to meet them someday (nor that I am willing to, perhaps), nor that I am positive on whether this is the proper place for such a purpose but I have to thank Christophe Arleston, René Goscinny, Jean-Louis Mourier, Jacques Rouxel, and Bill Watterson for without some of their compilations, it would have been very difficult for me to produce mine.

Thank Adèle for a great amount of things. Part of them simply cannot be listed here, and for the remaining I'd keep them private anyway.

To be frank, Juliette was of no help at all. Not to mention the contrary. However, as a matter of fact, more than half my scientific activities so far has been conducted and/or concluded after her arrival here around. I am grateful for the motivations she brought.

It is said that times, places, and events could split people, sometimes forever. Should it be needless to say, I would like to let all the people who have – or had – been accompanying my interests and perspectives to attain this current state know that there was, is, and will be a humble, but favoured place to host them in my mind. And sometimes also in my heart.

Bùi Xuân Bình Minh

to my grand mother Bà Nội who left when I was to be born
to my unborn child

Contents

Introduction to the Thesis

Modelling is fundamental to mathematics as it allows the very basic act of *defining* a problem. In computer science, modelling in most cases means modelling some discrete structure. For this, at the first level of investigation one can consider collections of objects. This allows to define not only basic operations such as the addition of a member or the inversion to the complementary, but also simple properties such as being a member, being non-empty, disjoint, included, overlapping or crossing. Then, for deeper details, collections of objects that are themselves collections of objects will obviously allow the modelling of more information. For instance, the collection of a certain number of collections of *exactly two* objects defines a graph. This, since the publication of L. Euler's paper in 1736 (known as *the Seven Bridges of Königsberg* paper), has been acting as the seminal notion for a very well-studied and fruitful theory called graph theory (e.g., Figure 1). The first objective of this thesis is the study of some structural properties of the former and more general notion of a collection of arbitrary collections of objects, a so-called set family.

Individual	Dated	From	To
Alice	Bob	January 1st	December 31st
Célia	Dicky	January 17th	June 28th
C. J.	Mitch	July 1st	ongoing
Craig	Stephanie	May 5th	August 8th
C. J.	Matt	August 5th	August 6th
C. J.	Jimmy	August 5th	August 6th
Matt	Caroline	August 5th	August 6th
Matt	Shauni	August 5th	August 6th
Jimmy	Caroline	August 5th	August 6th
Jimmy	Shauni	August 5th	August 6th
Stephanie	Matt	August 5th	August 6th
Stephanie	Jimmy	August 5th	August 6th
Craig	C. J.	August 10th	August 16st
Craig	Caroline	August 17th	ongoing
Mitch	Stephanie	August 9th	ongoing
Mitch	Shauni	August 7th	ongoing
Nutella	Honey	July 7th	August 8th
Polymerase	Promoter	July 2nd (8:30)	July 2nd (9:15)
Male Lark	Fem. Lark	March 15th	July 15th
Romeo	Juliet	long ago	ongoing

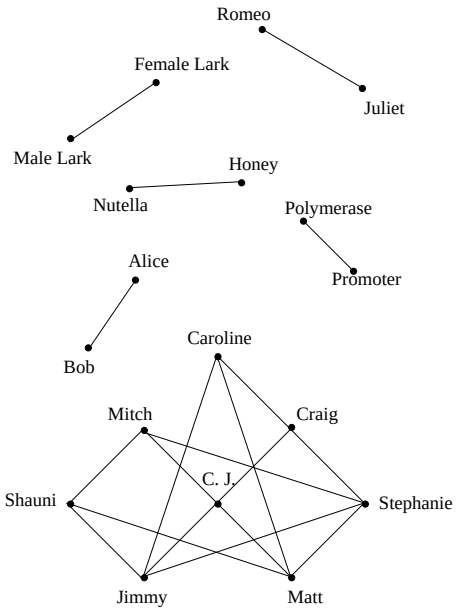


Figure 1: Modelling summer data (from July 1st to August 31st) of some polls.

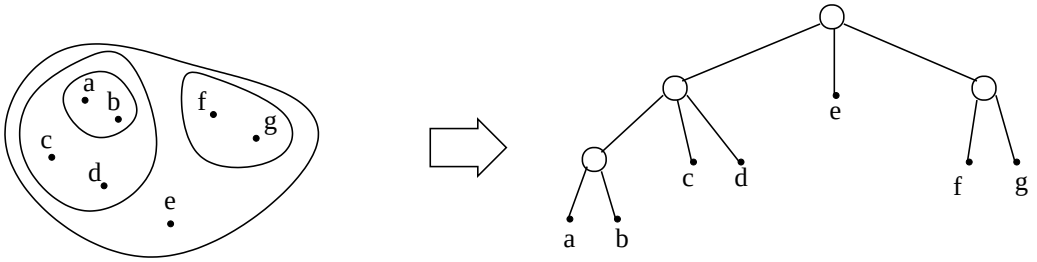


Figure 2: Modelling a laminar family.

The focus is to find efficient representations for a given set family. A mathematical motivation could be seen as follows. Starting from a ground set X of n elements, one clearly has 2^{2^n} choices of a family of subsets of X. However, if the family satisfies some simple axioms, the situation could be completely different. For instance, let us say that two subsets A and B overlap if they have a non-empty intersection $A \cap B \neq \emptyset$, as well as two non-empty differences $A \setminus B \neq \emptyset$ and $B \setminus A \neq \emptyset$. Then, a family is said to be laminar (or overlap-free) if no two of its members overlap. By elimination, two members of such a family can only be either disjoint, or included one in the other. Therefore, ordering them by inclusion will result into a (partial) subgraph of a tree, with a root corresponding to X, with leaves corresponding to the singletons $\{x\}$ (for all $x \in X$), and with internal nodes having at least two children each (e.g., Figure 2). This clearly states that a laminar family over X cannot have more than $2n$ members (and hence there are no more than 2^{2n} choices of a laminar family over X). In classical complexity theory for computer science, such exponential jumps are important for computational purposes. Accordingly, we would like to study some such situations, where a set family has a representation using not an exponential, but a polynomial space encoding.

Our biggest applicative motivation for such a study comes from a graph theory point of view. This could seem quite strange given the aforementioned fact that graphs are particular instances of set families: how about just studying graphs instead? However, many graph decompositions make intensive use of particular families of vertex subsets. This is, for instance, the case for modular and split decompositions: the corresponding notions of a module and a split are basically some subsets of the vertex set. Then, it is now a quite well-known fact that the family of modules of a graph meets the axioms of a so-called partitive family while the family of splits of a connected graph meets the axioms of a so-called symmetric crossing family. Under this standpoint, both modular and split decompositions can be seen as corollaries of some well-known results for representing the corresponding families. To give another example, the use of set families intervenes also in tree decomposition and clique decomposition via their respective alternatives: branch

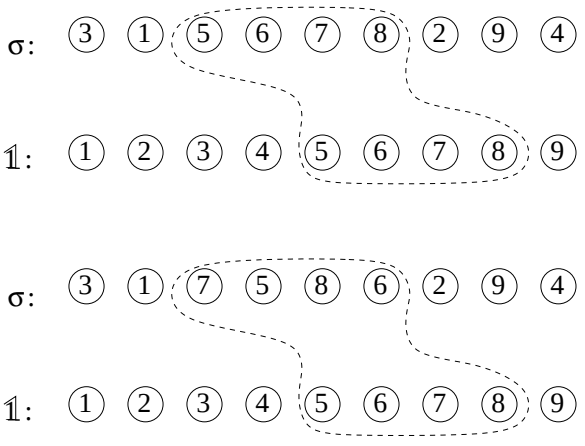


Figure 3: In both examples, $\{5, 6, 8, 7\}$ is a common interval.

decomposition and rank decomposition. Here, the more general framework is the so-called branch decomposition of a symmetric and submodular function. The focus therein is the family of subsets of the ground set where the function has a value lower than some given threshold. Then, the decomposition can be seen as a tree-like representation of a sub-family of the previous family.

Beside graph theoretic matters, combinatorial optimization is another well-studied area in combinatorics. A fundamental issue therein is to compute the minimizers of a given submodular function. This helps with, among other things, computing a maximum flow in some networking concerns (the corresponding equivalence is well-known under the name of min-max duality). Here, the family of non-empty minimizers of the function meets the axioms of a so-called intersecting family. Then, efficiently representing such a family has been acting as a crux for a class of solutions to the submodular function minimization problem. This is for instance the case for the so-called tree-of-posets decomposition.

Set families find also applications in computational biology. For instance, they help with handling particular instances of some computational problems around the concept of a gene cluster. Indeed, let us restrict the modelling of a genomic sequence to the case of a permutation over the set X of all genes it contains. Therein, an interval is a set of genes which come successively in the sequence (the permutation can be seen as a total order). Then, a set C of genes is called a common interval of a given group of several genomic sequences if C is an interval for every sequence in the group (e.g., Figure 3). This is among the first attempts to formalize the notion of a gene cluster. Now, the family of common intervals of a set of permutations over X meets the axioms of a so-called weakly partitive family. As a consequence, one can use a well-studied representation result on weakly partitive families to give a decomposition scheme of the genomic sequences into their

gene clusters. Under this framework, one can handle not only the efficient computation of those gene clusters but also their behaviour through successive modifications on the genomic sequences. While the former fact gives motivations for the computation of the very gene clusters, the latter fact helps with computing the so-called reversal distance between two species in evolutionary sorting.

Other decompositions of discrete structures where the representation of particular families of sets plays a central role include: canonical decomposition of symmetric and submodular functions [56], decomposition of submodular functions and decomposition of matroids [40], clan decomposition of 2-structures [48], block decomposition of inheritance graphs [76] (see also [23, 66] for some English versions), bimodular decomposition of bipartite graphs [55], bijoin decomposition of graphs [93], and decomposition of tournaments into unordered-modules [18].

At the same time, if efficiently representing set families finds its applications in the decomposition of several discrete structures, then it is also a related motivation to look into some applications of the decomposition paradigm itself. For instance, the decomposition philosophy intervenes in algorithm design at a quite basic level: the conception. Here, given a problem that one wishes to solve by an autonomous system, say a computer, wouldn't the first thing to do be to understand the problem, analyse how it behaves, figure out all the configurations it can lead to, etc? In other words, one would like to find an underlying structure. Then, a way to achieve that aim consists of decomposing the given problem into simple configurations. Now, if the decomposition itself is simple enough, one gives birth to simple, and efficient algorithms. The simplicity here refers to the use of regular and conventional algorithmic schemes: all technical difficulties have to be done by theoretical work during the structural analysis and should not figure in the proper use of the algorithm. In practice, simple algorithms are desirable to avoid implementation errors, while efficient algorithms are required when one wishes to manipulate huge amounts of data and cannot allow the extra cost that a naive approach might bring.

The second part of the thesis follows this philosophy. By means of exemplifications, we will try to defend how the use of decomposition approaches may help to design not only efficiently competitive algorithms, but also algorithms answering to some criterion of simplicity. The latter claim is motivated by the fact that we intensively follow the divide-and-conquer approach, widely recognized as one of the most fundamental and conventional algorithmic schemes. Note however that, beside the common point of using decomposition approaches to solve discrete problems, each chapter of the second part is rather case specific. For each of them, we study how to decompose and simplify the input, then show how to solve each thus decomposed situation, as well as how to subsequently

solve the global problem.

We now shape more specifically the structure of the thesis. As previously mentioned, we divide the composition into two main parts. Though highly related, the two subjects we discuss are not absolutely dependent on each other, and sometimes can be quite severed.

For convenience, we start each part of the thesis with a specific overview of its contents. Beside introductory matters, a more detailed summary of each chapter can be found in those two overview sections. Roughly, the first part of the thesis focuses on some combinatoric issues on set families in general. It addresses also some applications of those issues into a branch of graph theory called graph decomposition. The second part of the thesis is oriented towards algorithmic graph theory. The leading idea is to use the connection between divide-and-conquer algorithms on graphs and some structural decomposition on them.

Our activities so far have brought to fruition references [14, 15, 16, 17, 18, 19, 20, 21, 22] (details are below). In this composition, we develop [15, 19, 20, 21, 22]. We also mention [16, 17, 18] but do not go into their details. We do not address [14] here.

Last but not least, we highlight that *all* discrete structures in this thesis are *finite*.

---

[14] B. Bui Xuan, A. Ferreira, and A. Jarry. Computing Shortest, Fastest, and Foremost Journeys in Dynamic Networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.

[15] B.-M. Bui-Xuan and M. Habib. A Representation Theorem for Union-Difference Families and Application. In *8th Latin American Theoretical Informatics (LATIN'08)*, volume 4957 of *LNCS*, pages 492–503, 2008.

[16] B.-M. Bui-Xuan, M. Habib, V. Limouzy, and F. de Montgolfier. Algorithmic Aspects of a General Modular Decomposition Theory. *Discrete Applied Mathematics: special issue of the 3rd conference on Optimal Discrete Structures and Algorithms (ODSA'06), to appear.*

[17] B.-M. Bui Xuan, M. Habib, V. Limouzy, and F. de Montgolfier. Homogeneity vs. Adjacency: generalising some graph decomposition algorithms. In *32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'06)*, volume 4271 of *LNCS*, pages 278–288, 2006.

[18] B.-M. Bui-Xuan, M. Habib, V. Limouzy, and F. de Montgolfier. Unifying two Graph Decompositions with Modular Decomposition. In *18th Annual International Symposium on Algorithms and Computation (ISAAC'07)*, volume 4835 of *LNCS*, pages 52–64, 2007.

[19] B.-M. Bui Xuan, M. Habib, and C. Paul. Revisiting T. Uno and M. Yagiura's Algorithm. In *16th Annual International Symposium on Algorithms and Computation (ISAAC'05)*, volume 3827 of *LNCS*, pages 146–155, 2005.

[20] B.-M. Bui-Xuan, M. Habib, and C. Paul. Competitive Graph Searches. *Theoretical Computer Science*, 393(1-3):72–80, 2008.

[21] B.-M. Bui-Xuan, M. Habib, and M. Rao. Representation Theorems for two Set Families and Applications to Combinatorial Decompositions. *Extended abstract in Proceedings of the International Conference on Relations, Orders and Graphs: Interaction with Computer Science (ROGICS'08), Nouha editions,* pages 532–546, 2008.

[22] B.-M. Bui-Xuan and J. A. Telle. H-join and dynamic programming on graphs of bounded rankwidth. *Abstract presented in the Workshop on Graph Decomposition: Theoretical, Algorithmic and Logical Aspects,* 2008.

Part I

Representation of Set Families and Decomposition

"A natural beauty is a beautiful thing;
artificial beauty is a beautiful representation of a thing."
Immanuel Kant, 1892.
(*Kant's Critique of Judgement,* translated with Introduction and Notes
by J. H. Bernard, 2nd ed. revised, London: Macmillan, 1914.)

Overview of Part I

Representing a given finite set, and more specifically finding a bijective mapping for it, has always been one of the fundamental issues in combinatorics. It is related to many other areas of mathematics, such as algebra, probability theory, ergodic theory and geometry, as well as to applied subjects in computer science and statistical physics. The issue has a long history throughout the five continents, with the probably first written mention appearing no later than some three hundred years Before Christ in an Indian text. In theoretical computer science, it is mostly used to obtain estimates on the number of elements of certain sets, with "estimates" being the key word of the sentence.

For instance, the Kolmogorov-Chaitin complexity is a theoretical formalization of the computational resources needed to specify a given object. It is roughly the minimum number $K(s)$ of bits required to encode a program which outputs the object s. However, the exact value of such a definition is hard to find. Moreover, one usually does not need to know exactly such a measure, but an upper bound for the value. The naive upper bound for the complexity of a family over a given set X is *a priori* exponential on the number of elements in X since the family might count that many members. Now, studying the problem of representing some set families can help lowering the bound. Indeed, whenever a representation theorem is known for some class of set families, the complexity of every family of that class is lowered to that of the representation, to a multiplicative factor. This phenomenon is the direct application of the folklore fact where every loss-less data compression method leads to an upper bound for the corresponding Kolmogorov-Chaitin complexity of the compressible data. Accordingly, representing efficiently a given set family can also be seen as an approach for reaching its complexity.

We in fact will have little discussion on the actual Kolmogorov-Chaitin complexity, and define, with some abuse, the complexity of a set family as the space complexity to represent the family. This is abusive in the sense that space complexity is usually used with some abuse itself: one does not necessarily count at the bit-level. To give an example, the space complexity to represent the linked list containing all integers of the set $X = \{1, 2, \ldots, n\}$ is commonly accepted as in $O(n)$. However, to represent each

integer bound to each element of the linked list, one may need $\log n$ bits, namely the description of an integer smaller than n in binary numeral system. Then, the actual space to be used for storing the set X should be in $\Theta(n \log n)$ bits. The abusive counting of the space complexity of X as *in* $O(n)$ *space complexity* is widely known as *the* $\log n$ *neglect*. Likewise, a graph on n labelled vertices and m edges is commonly regarded as of $O(n+m)$ space complexity while the actual space to store such a structure by adjacency list is in $\Theta(m + n \log n)$. Even more strange, if the graph is a tree, then, from the Cayley formula, there are exactly $n^{n-2} = 2^{(n-2)\log n}$ trees on n labelled vertices. Therefore, it is asymptotically impossible to represent such a tree with a number of bits proportional to n. Even so, the space complexity of this case is still commonly regarded as in $O(n)$. For this reason, one says that the popular counting of space complexity in theoretical computer science is subject to the $\log n$ neglect.

An attempt to defend the $\log n$ neglect could come from the fact that, in practice, the base unit of a computer is more likely an *octet* (a.k.a. byte), and can already represent up to $2^8 = 256$ integers (octet: "eight bits"). Moreover, if one considers that accessing to *"some hundreds"* of bytes (read/write) is a unit operation, then there are $O(n)$ unit operations to create the above linked list, corresponding to X, with already a relatively large n. Also, there will be that many unit operations to read the contents of the list. This abusive $\log n$ neglect is not that far from reality given that the actual computers do not follow exactly the Turing Machine model, and follow more likely the Random Access Machine one. For a side note, the situation is different when counting the time complexity of running a loop $\log n$ times. Here, one will have to repeat the same set of operations $\log n$ times, which cannot be neglected.

In this part of the thesis, a major concern is to apprehend the distance of a family to its ground set. Recall the remark in the introduction where a laminar family behaves roughly the same manner as its ground set X (space-wise), namely it has no more than $2 \times |X|$ members. Then, an equivalent purpose could be to settle laminar families as standards and apprehend the distance of a family to them. Now, we will see in the upcoming chapter that, from the Edmonds-Giles theorem, laminar families can be bijectively mapped to trees. Then, another equivalent purpose could be to apprehend the distance of a set family to a tree structure. Besides, we have briefly seen in the introduction that the representation problem and the decomposition problem are closely related. Accordingly, we will also discuss in detail some decomposition issues in an "on-the-fly" manner: the discussion about a decomposition scheme will be given at about the same place where a representation theorem responsible for that scheme is presented.

For convenience, we will take into account the abusive counting of space complexity

when evaluating a set family. We also make the convention that the complexity of a ground set X is in $O(|X|)$ space. Thus, the $\log n$ neglect will intervene from the very root of our question, namely in the standard defined by the ground set of the input family. Then, it will not change much to count the neglect in the space complexity of the set family itself. Finally, if one's concern is only to know if some family can be polynomially representable or not, then the $\log n$ neglect does not change any result.

We now specify more the contents of the first part.

The opening Chapter 1 builds the basis for representing set families in general. To this purpose we present new tools and techniques to find a tree-like representation for an arbitrarily given set family. They capture and extend many well-known concepts in different branches of combinatorics, ranging from Edmonds-Giles's tree-representation of cross-free families to Ehrenfeucht-Harju-Rozenberg's framework for graph decompositions. The technique we develop in Chapter 1 will be crucial for almost all other results on set families that will be presented in the thesis. Beside this, we also give in Chapter 1 a brief discussion on some preliminary results of the representation problem. Finally, we end the chapter with a synopsis in Figures 1.4 and 1.5 of all representation results involved in the whole composition.

In Chapter 2 we recall some known results in the topic of representing set families, as well as some applications. In particular, the chapter includes the review of results on the so-called intersecting and crossing families. They are important for submodular function minimization purposes. The chapter includes also a detailed review of well-known results on the so-called partitive families and symmetric crossing families. While partitive families are seminal for the modular decomposition of graphs, symmetric crossing families are important for symmetric submodular function minimization studies. At the same time, we revisit a well-known result for representing partitive families using a linear encoding space. More precisely, we give an alternative approach for obtaining the same result. The motivation is that, unlike the classical approach, our alternative follows the general framework developed in Chapter 1. Then, the approach of Chapter 1 can be seen among other things as a unification under a same framework of various schemes for representing different classes of set families. Chapter 2 ends with a series of applications of the above mentioned results in an important branch of graph theory called graph decomposition. Here, the application list includes no fewer than seven decomposition schemes of various discrete structures (a list can be found in the table of contents). While most cases come from previous works in the topic, two of them derive from our activities during the thesis period. However, we will not go into their details in this composition.

The third and last chapter of the current part is devoted to two recent results. For their

introduction, recall from the definitions given in the introduction that two sets overlap if they have a non-empty intersection, as well as two non-empty differences. Here, we say moreover that two subsets of a given ground set cross if they overlap and so do their respective complements in the ground set. Then, a set family is called weakly partitive crossing if it is closed under the union, the intersection, and the difference of its crossing members. Besides, a set family is called a union-difference family if it is closed under the union and the difference of its overlapping members. In Chapter 3, we give linear and quadratic space representation for weakly partitive crossing families and union-difference families, respectively. These results rely on the framework given in Chapter 1. We close Chapter 3 with the presentation of two new combinatorial decompositions. Both of them come under the common name of sesquimodular decomposition. One of the two is a strict generalization of the modular decomposition of digraphs, while the other is a strict generalization of the clan decomposition of so-called 2-structures. Both decompositions are polynomially computable.

Most of the ideas presented in Chapter 1, the alternative approach for partitive families in Chapter 2, two of the decomposition schemes at the end of Chapter 2, as well as all results presented in Chapter 3, are based on [15, 16, 17, 18, 21]. However, we will not go into the details of [16, 17, 18].

Chapter 1

A General Representation Method

This is a very short chapter which, nonetheless, will settle the main basis for the current part of the thesis. We present here some tools that will be fundamental to represent almost all set families related to our composition. The ideas are mainly based on [15, pages 493–495].

We address three abstraction levels: elements, sets, and families. They are concisely defined as follows. Let X be a finite set. Those that belong to X are called *elements*, sets included in X are *subsets*, and sets of subsets of X are *families*. For more clarity, elements of a family are called *members*. The *ground set* of a family is the minimum set of which members of the family are subsets. When the ground set is not mentioned, families are also called *set families*. Notice that a finite family always admits a finite ground set. The converse is a triviality.

Among the subsets of X, the universal set X and singletons $\{x\}$ (for all $x \in X$) are called *the trivial subsets* of X. By abusive notations, we may sometimes denote the singleton $\{x\}$ by x. Without loss of generality (w.l.o.g.) in the representation problem, a family is always supposed to exclude the empty set and include all trivial subsets of the ground set, which we denote by a *proper and connected* family. Indeed, starting from an arbitrary family, we can obtain a proper and connected family just by removing and adding the corresponding subsets. This process will not increase the space complexity any higher than that of the ground set. Then, as far as the representation problem is concerned, supposing a family proper and connected does not change the asymptotic space complexity of the family. When dealing with such a family, we refer to the trivial subsets of the ground set as *the trivial members* of the family. Besides, the power set of X will be denoted by 2^X. For instance, $2^{\{a,b\}} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$. Then, a minor remark would claim here that representing a proper and connected family over a ground set X with $|X| < 3$ is trivial since it would mean the family is exactly $2^X \setminus \{\emptyset\}$. For convenience we suppose, unless otherwise stated, that the ground set of a family includes at least three

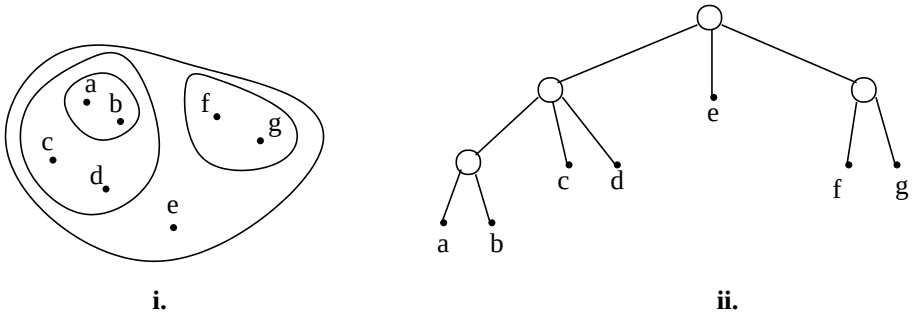


Figure 1.1: i. An overlap-free (laminar) family. ii. Its tree representation.

elements, that is

Remark 1.1 (Conventions) *Unless otherwise stated explicitly, not only all set families in this thesis are supposed to cover a ground set of at least three elements, but they are also supposed to be proper and connected, namely they exclude the empty set, and include all trivial subsets of the ground set. In particular, sentences like "Let $\mathcal{F}$ be an arbitrary set family..." refer in reality to "Let $\mathcal{F}$ be a set family which is proper and connected...".*

We refer to the complexity of a set family as the space complexity to represent the family. The counting of the space complexity is subject to *the* $\log n$ *neglect*, for which we refer the reader to the overview section for a short presentation and discussion. The aim of this part of the composition is to compare the complexity of a set family to that of a tree. We start with giving an explanation why trees are selected as unit of our measure.

1.1 The Baseline: Cross-Free Families

We start with the representation problem of a simple class of families, so-called *cross-free families* in combinatorial optimization related literature (e.g., [106]). Let us recall that two subsets A and B of a ground set X *overlap*, denoted by $A \owedge B$, if none among $A \cap B$, $A \setminus B$, and $B \setminus A$ is empty. They *cross*, if we have both $A \owedge B$ and $\overline{A} \owedge \overline{B}$, where $\overline{A} = X \setminus A$. Then, a set family is called *overlap-free* (resp. *cross-free*) if no two members of the family overlap (resp. cross). Note that overlap-free families are more widely known under the name of *laminar* families. Besides, as crossing clearly implies overlapping, an overlap-free family is also cross-free. Finally, that cross-free families are basic for tree-like representations of set families is due to J. Edmonds and R. Giles's characterization of them as ones in bijection with unrooted trees [46], which can be summarized as follows.

In a (proper and connected) overlap-free family, two members are either disjoint, or included one in the other. Then, their ordering by inclusion will result in a tree, with the

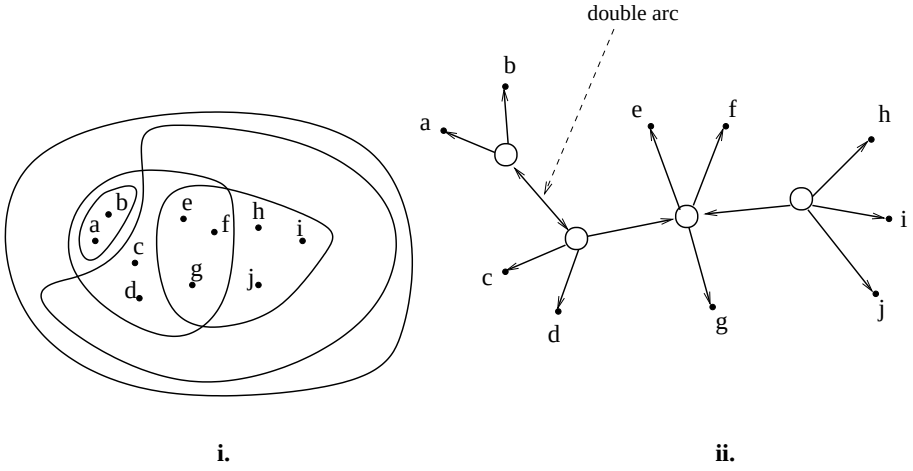


Figure 1.2: i. A cross-free family. ii. Its tree representation.

root corresponding to the ground set, and with the leaves corresponding to the singletons (e.g., Figure 1.1). Now let $\mathcal{F} \subseteq 2^X$ be a cross-free family. Let $x \in X$, we consider $\mathcal{G} = \{A \mid A \in \mathcal{F} \wedge x \notin A\} \cup \{\overline{A} \mid A \in \mathcal{F} \wedge A \neq X \wedge x \in A\}$, which is a proper and connected, overlap-free family over the ground set $X \setminus \{x\}$. Then, applying the previous tree representation on $\mathcal{G}$ results in a tree whose root corresponds to $X \setminus \{x\}$. Let us add x to the children of the root and unroot the tree.

Remark 1.2 *Such a tree has no degree* 2 *nodes as long as* $|X| \geq 3$.

The set of leaves is now in bijection with X: by some abusive notations we confound the two sets. In this tree, deleting any edge gives rise to two connected components. If each component is regarded as the set of its leaves, then at least one of them is a member of $\mathcal{F}$. Thus, edge orientation can denote which ones belong to $\mathcal{F}$ (see Figure 1.2). On the other hand, it is straightforward to prove that each member of $\mathcal{F}$ corresponds to one edge of the tree. It follows that

Theorem 1.1 (Edmonds-Giles [46]) *A set family is overlap-free (resp. cross-free) if and only if it has a rooted (resp. unrooted) tree representation.*

Remark 1.3 *If a set family is overlap-free (hence cross-free), then either its* cross-free *tree representation (as described in the above) has one and only one source, or there is in that tree a unique double-arc and setting a source in between the subdivision of the double-arc results into a tree with one and only one source. In both cases, inverting the orientations will result into a rooted tree. Furthermore, this rooted tree turns out to be exactly the* overlap-free *tree representation of the initial family.*

Accordingly, we would say that the distance from overlap-free and cross-free families to a tree structure is almost null. Notice for the sake of rigour that the above transformation is not necessarily a bijection between the set $\mathcal{C}_X$ of all cross-free families over X and the set $\mathcal{T}_X$ of all trees whose leaves are in bijection with X. Regardless, it is still a bijection between the set $\mathcal{C}_X$ and a subset of $\mathcal{T}_X$, namely the image of the transformation over the domain $\mathcal{C}_X$. This clearly gives an upper bound for the size of $\mathcal{C}_X$. Now, from the fact that an overlap-free family is also cross-free, we can unify their complexity statements as

Corollary 1.1 *The complexity of a cross-free family on a ground set X is in $O(|X|)$.*

Roughly, handling overlap-free families is a more convenient task than that with cross-free families. However, cross-free families are clearly a strict generalization of overlap-free families. This situation will be all the more emphasized in the upcoming discussion around the definition of a decomposition tree.

1.2 How to Represent an Arbitrary Family

To represent a set family in a unique manner, we would like to find an injective function which maps every set family to some object. Then, the injection can be seen as a bijection between the set of all set families and the image of the function. We will build this injection step by step, starting with a function which is not necessarily injective.

Decomposition Tree: Extraction Approach

In this thesis, we use *decomposition* as a general name for associating a first category of objects with a second category of objects. The only requirement is that the second category object should have some cutting ability with respect to (w.r.t.) the first category object. In the following, we will define such an association for arbitrary set families. The cutting ability will appear from the subsequent notion of a quotient.

At the same time, a set family might be complex. Then, we would also like to perform some simplifications upon it. For instance, the terminology of *uncrossing* usually refers to methods which transform a family into a cross-free family by the repeated application of some easy steps. It is widely used in combinatorial optimization as a pre-processing step. Here also, we will see how, for decomposition purposes, one can use a very naive method to uncross the input family. Indeed, one can obtain the cross-free property simply by some greedy *extraction* (without transformations) of qualified members as follows.

A member $A \in \mathcal{F}$ is an *overlap-free member of* $\mathcal{F} \subseteq 2^X$ if A does not overlap any $B \in \mathcal{F}$. Likewise, $A \in \mathcal{F}$ is a *cross-free member of* $\mathcal{F}$ if it does not cross any $B \in \mathcal{F}$. Let

$\mathcal{S} \subseteq \mathcal{F}$ be the family of cross-free members of $\mathcal{F}$. For the sake of simplicity, X is *excluded* from $\mathcal{S}$ although it is clearly cross-free. Clearly, $\mathcal{S}$ is a cross-free family.

Definition 1.1 (Decomposition Tree and Cross-free Decomposition Tree) The *cross-free decomposition tree* of a family $\mathcal{F} \subseteq 2^X$ is defined as the Edmonds-Giles's representation [46] of its cross-free members, where X is excluded. Throughout the first part of the thesis, unless otherwise stated, a *decomposition tree* always refers to a cross-free decomposition tree.

Remark 1.4 *Associating a set family with its cross-free decomposition tree results in a well-defined function.*

Before continuing, note that for uncrossing purposes, one can of course consider the overlap-free members of $\mathcal{F}$ as well: they form clearly an overlap-free (hence cross-free) family and, more importantly, have also a tree-representation. Indeed, a more detailed discussion on the overlap-free alternative will be given around Definition 2.5 in the next chapter. However, since an overlap-free member is also a cross-free member, it is clear that addressing cross-free members allows to define a finer decomposition tree. This being said, some cases of set families, such as those bound to modular decomposition, prefer using the overlap-free members in order to define the decomposition tree. A bit more precisely, the modular decomposition of a graph is essentially an efficient representation of the family of modules of the graph. For this purpose, one defines the (modular) decomposition tree of the graph using the Edmonds-Giles's representation of the sub-family of overlap-free members of the previous family, namely the sub-family containing all modules of the graph which do not overlap another module of the graph. At least this is one of the most common points of view to approach modular decomposition (some further details are given in Section 2.3.1). This could be intriguing given what has just been said. However, as long as modules are concerned, there is a very simple reason for using overlap-free members instead of cross-free members: their corresponding trees coincide, except for the root and orientations (cf. Lemma 2.2 in the next chapter). Then, for the sake of terseness, overlap-free terminologies are used at the expense of generality. This is by no means a general situation.

Decomposition & Cutting Property: the Notion of a Quotient

In the cross-free decomposition tree, the deletion of an internal node u gives rise to $k = d(u)$ connected components, which can also be seen as a $k-$partition of X. Let $\{X_1, X_2, \ldots, X_k\}$ denote this partition (e.g., Figure 1.3). Notice that if $|X| \geq 3$ then $k \geq 3$. Let us consider $Y = \{X_1, X_2, \ldots, X_k\}$ as a set.

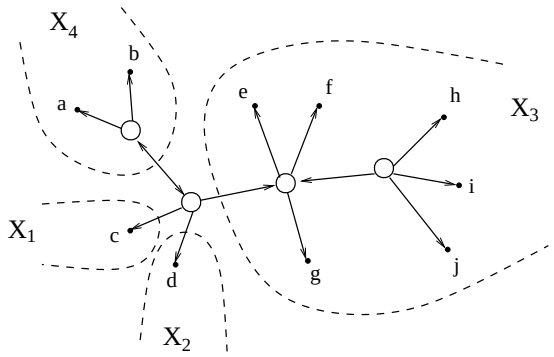


Figure 1.3: The ground set $\{X_1, X_2, X_3, X_4\}$ of a quotient, with $X_1 = \{c\}$, $X_2 = \{d\}$, $X_3 = \{e, f, g, h, i, j\}$, and $X_4 = \{a, b\}$.

Definition 1.2 (Quotient and Cross-free Quotient) Keeping the same notation of $Y = \{X_1, X_2, \ldots, X_k\}$ as above, we define the *cross-free quotient of $\mathcal{F}$ with respect to node u* as the family $\mathcal{Q}(u) \subseteq 2^Y$ such that

$$\begin{cases} \{X_i\} \text{ belongs to } \mathcal{Q}(u) \text{ for all } 1 \leq i \leq k, \\ Q = \{X_i \mid i \in I\} \text{ with } |Q| \neq 1 \text{ belongs to } \mathcal{Q}(u) \;\Leftrightarrow\; \bigcup_{i \in I} X_i \text{ belongs to } \mathcal{F}. \end{cases}$$

Throughout the first part of the thesis, unless otherwise stated, a *quotient* always refers to a cross-free quotient.

Notice that $\mathcal{Q}(u)$ is a proper and connected family over the ground set Y, and Y has at least three elements as long as X has at least three elements. The main point of defining the decomposition tree and the various quotients bound to the internal nodes of the tree could be informally seen as follows. In an arbitrary set family, every cross-free member in some sense "splits" the other members of the family by its very definition: no other member crosses the cross-free member. Then, the definition of a quotient allows to somehow classify the members of the initial family with respect to the various cross-free members. We come to the most important claim of this chapter:

Essential fact: *As it will be showed next (in Remark 1.7) and also exemplified in the next two chapters (e.g., partitive families, symmetric crossing families, union-difference families, and partitive crossing families), characterizing the various quotients of $\mathcal{F}$ will be the only left-over work if one wishes to represent a set family $\mathcal{F}$.*

This is what we referred to as the cutting ability of the decomposition tree. Of course, this is by no means a concise point of view. The informal cutting ability, nonetheless, will be crucial for various situations in the next part of the thesis, when we would like to properly "divide-and-conquer" a problem. For the current objective, namely obtaining a bijection over arbitrary set families, let us highlight that

Remark 1.5 *Let f be the function mapping a set family to its decomposition tree. Then, for every set family $\mathcal{F}$, the function $g_{\mathcal{F}}$ over the domain $\{u,\ u$ is an internal node of $f(\mathcal{F})\}$, which maps every node u to the quotient $g_{\mathcal{F}}(u)$ of $\mathcal{F}$ with respect to node u, is well-defined. One consequence is that the function h, mapping a set family $\mathcal{F}$ to the pair $h(\mathcal{F}) = (f(\mathcal{F}), g_{\mathcal{F}})$, is well-defined.*

In the following, the objective is to prove that the function h, defined in the above remark, is injective. Turning our attention back to the quotient $\mathcal{Q}(u)$, notice that the membership of X_i in $\mathcal{F}$ (resp. exclusion of X_i from $\mathcal{F}$) can already be stored by the edge orientation of the decomposition tree. Roughly, each member Q of the quotient $\mathcal{Q}(u)$ corresponds to one and only one member of $\mathcal{F}$, except for the singletons $\{X_i\}$. Moreover, it is not so obvious but quite popular that the converse holds as well.

Proposition 1.1 *Let A be a member of a family $\mathcal{F} \subseteq 2^X$. If A is not a cross-free member of $\mathcal{F}$, then, there exists one and only one node u in the decomposition tree of $\mathcal{F}$ such that A corresponds to a member of the quotient $\mathcal{Q}(u)$ of $\mathcal{F}$ with respect to node u.*

Proof: There are many ways to prove this fact. A graphical one, extending the ideas of [94, proof of Lemma 1], is as follows. With respect to the member $A \in \mathcal{F}$, we will define a special orientation on the arcs of the decomposition tree $\mathcal{T}$ of $\mathcal{F}$. Let us consider an arc a of $\mathcal{T}$ which links a (not necessarily internal) node u to node v of $\mathcal{T}$. Let S_u and S_v be the 2-partition of X induced by the leaves of the two connected components of $\mathcal{T} - a$, the forest obtained by removing arc a from $\mathcal{T}$. Notice that if A crosses one among S_u and S_v, then A crosses both. Since at least one among S_u and S_v is a cross-free member of $\mathcal{F}$, A does not cross any of them. Hence, there are only two cases, which are self-exclusive:

- either S_u or S_v is strictly included in A: w.l.o.g. suppose it was S_u, then the special orientation is defined to be from u to v.
- A is strictly included in either S_u or S_v: w.l.o.g. suppose it was S_u, then the special orientation is defined to be from v to u.

We claim that the special orientation has one and only one sink (i.e. it defines a rooted tree). Indeed, let uv be an arbitrary arc in $\mathcal{T}$ with the special orientation from u to v. For the claim, it suffices to prove that, for every arc st belonging to the connected component which contains u when we remove arc uv from $\mathcal{T}$, the special orientation is from the further node s (w.r.t. u) to the nearer node t (w.r.t. u). This can be proved for example with a straightforward case analysis. Let now u be the unique sink defined by the special orientation. Let $\{X_1, X_2, \ldots, X_k\}$ be the ground set of the quotient $\mathcal{Q}(u)$ of $\mathcal{F}$

w.r.t. node u. Here, A cannot be included in some X_i, otherwise the special orientation from X_i to the sink u would be reversed. By elimination, the only case left possible is when every X_i is either included in A or included in the complement of A. We can then conclude by applying Definition 1.2 on $\mathcal{Q}(u)$. □

Corollary 1.2 *The function h defined in Remark 1.5 is injective.*

Proof: Let $\mathcal{F}$ and $\mathcal{G}$ be two set families such that $h(\mathcal{F}) = h(\mathcal{G})$. By symmetry it suffices to prove that $\mathcal{F} \subseteq \mathcal{G}$. Let $A \in \mathcal{F}$ be a member of $\mathcal{F}$. If A is a cross-free member of $\mathcal{F}$ then we can conclude using $f(\mathcal{F}) = f(\mathcal{G})$ and Theorem 1.1. Otherwise, from Proposition 1.1, there exists an internal node u of $f(\mathcal{F}) = f(\mathcal{G})$ such that, there exists a member $Q \in g_{\mathcal{F}}(u)$ of the quotient of $\mathcal{F}$ with $A = \bigcup_{S \in Q} S$ and $|Q| \neq 1$. Now, $h(\mathcal{F}) = h(\mathcal{G})$ also implies $g_{\mathcal{F}} = g_{\mathcal{G}}$. We then have: u is an internal node of $f(\mathcal{G})$, $Q \in g_{\mathcal{G}}(u)$, $A = \bigcup_{S \in Q} S$, and $|Q| \neq 1$. This, by Definition 1.2, implies $A \in \mathcal{G}$. □

A stronger claim than that of Corollary 1.2 is as follows. Its proof is straightforward.

Remark 1.6 *In order to represent a family $\mathcal{F}$, it is sufficient to represent the quotient family $\mathcal{Q}(u)$, for every node u of the decomposition tree of $\mathcal{F}$.*

How to Represent a Class of Set Families

According to Remark 1.6, in order to represent every family of a class $\mathcal{K}$ of set families, it suffices to represent the class $\mathfrak{L}$ of every family $\mathcal{Q}$ which is the quotient of some node appearing in the decomposition tree of some family $\mathcal{F} \in \mathcal{K}$. However, it is not so obvious how to describe the class $\mathfrak{L}$. Instead, we will focus on a super-class of $\mathfrak{L}$. Let us say that a member A of a family $\mathcal{F} \subseteq 2^X$ is *quasi-trivial* if $|A| = |X| - 1$. Clearly, trivial and quasi-trivial members of $\mathcal{F}$ are also cross-free members. Then, one can notice a second non obvious but folklore fact that

Proposition 1.2 *Let $\mathcal{F} \subseteq 2^X$ be a family (such that $|X| \geq 3$). Then, for every node u of the decomposition tree of $\mathcal{F}$, all cross-free members of the quotient $\mathcal{Q}(u)$ with respect to u are either trivial or quasi-trivial with respect to the ground set of $\mathcal{Q}(u)$. Besides, the ground set of $\mathcal{Q}(u)$ contains at least three elements.*

Proof: Recall the notation where $\mathcal{S}$ denotes the set of all cross-free members of $\mathcal{F}$, but the ground set X. Here, any cross-free member of $\mathcal{Q}(u)$, except for its ground set, corresponds to a member of $\mathcal{S}$. Then, a cross-free member of $\mathcal{Q}(u)$ that is neither trivial nor quasi-trivial w.r.t. $\mathcal{Q}(u)$ would lead to a member of $\mathcal{S}$ which is not present in the tree representation of $\mathcal{S}$. This contradicts Theorem 1.1. Finally, the fact that $\mathcal{Q}(u)$ has a ground set of at least three elements is straightforward. □

Definition 1.3 (Quotient property) We say that a set family satisfies the *(cross-free) quotient property* if all its cross-free members are either trivial or quasi-trivial.

Straight from Proposition 1.2, every family $\mathcal{Q} \in \mathfrak{L}$ of the above mentioned class $\mathfrak{L}$ satisfies the quotient property. Obviously, the converse does not necessarily hold. However, we will still study all the families which satisfy the quotient property since, in all situations described in this thesis, the relaxation will not increase the difficulty in representing the family. This leads us to the definition of a quotient-hereditary class of set families.

The Case for Quotient-Hereditary Classes of Set Families

Basically, all the classes addressed in this thesis turn out to have an intrinsic property that makes the description of their quotients even more convenient (cf. Proposition 1.3 below).

Definition 1.4 (Quotient-Hereditarity) A class $\mathcal{K}$ of set families is *quotient-hereditary* if, for every family $\mathcal{F} \in \mathcal{K}$ in this class, the quotient family $\mathcal{Q}(u)$ with respect to every node u of the decomposition tree of $\mathcal{F}$ belongs also to the class: $\mathcal{Q}(u) \in \mathcal{K}$.

If a class of set families is quotient-hereditary, all what has been said leads to the following *essential* fact.

Remark 1.7 (Main Tool) *In order to represent a class $\mathcal{K}$ of set families which is quotient-hereditary, it suffices to represent the sub-class $\mathcal{K}' \subseteq \mathcal{K}$ containing every family $\mathcal{F} \in \mathcal{K}$ which satisfies also the quotient property.*

Note that, in Remark 1.7, $\mathcal{K}'$ is a super-class of $\mathfrak{L}$, the class of every quotient of some node appearing in the decomposition tree of some family $\mathcal{F} \in \mathcal{K}$. Let us attempt to informally explain why the remark is important. Basically, another aspect of the notion of a quotient is to eliminate all non-trivial cross-free members from a set family (according to Proposition 1.2). This can roughly be seen as the exact opposite of an uncrossing process. However, in order to do so, the heavy weaponry of a quotient may not make much sense at first glance: how about just simply removing the non-trivial cross-free members whenever they are not desired? Actually, the drawback of simply removing undesirable members when studying a set family belonging to some special class is that one might end up with a family that does not belong to the special class anymore. On the other hand, when a class of set families is quotient-hereditary, the notion of a quotient allows to, at the same time, eliminate the non-trivial cross-free members *and* stay inside the class. In the

next two chapters, our study ranges over the classes of families which are closed under a number of set operations on their overlapping, or crossing, members. All of them are quotient-hereditary, and we will intensively exploit the notion of a quotient, as well as the quotient-hereditary property.

Definition 1.5 (Overlap-$\mathcal{X}$, Cross-$\mathcal{X}$, and $\mathcal{X}$ Closure) Let $\mathcal{X}$ be a set of operations over sets. A set family is called an $\mathcal{X}$ *closed family* if it is closed under all operations belonging to $\mathcal{X}$. Given a set $\mathcal{X}$ of binary operations over sets, a set family is called an *overlap-$\mathcal{X}$ closed family* (resp. a *cross-$\mathcal{X}$ closed family*) if it is closed under the action of any operation of $\mathcal{X}$ on a pair of its overlapping (resp. crossing) members. For more convenience, we refer to overlap-$\mathcal{X}$ closed and cross-$\mathcal{X}$ closed families simply as overlap-$\mathcal{X}$ and cross-$\mathcal{X}$ families.

Proposition 1.3 *For every set $\mathcal{X}$ of arbitrary operations over sets, the class of $\mathcal{X}$ closed families satisfies the quotient-hereditary property. This is also the case for overlap-$\mathcal{X}$ and cross-$\mathcal{X}$ families, when $\mathcal{X}$ contains only binary operations. Finally, this also holds for an arbitrary number of intersections of the above classes.*

We will focus more particularly on $\mathcal{X}$ being a subset of the set of basic operations: complementation, intersection, union, difference, and symmetric difference. Note that, for the sake of generality, we introduce the above (and heavy!) notations of overlap-$\mathcal{X}$ and cross-$\mathcal{X}$ families. However, most cases have specific interests and own a different (and sensible) designation. An overview of some known results, as well as some new results, is given at the end of this chapter. Figure 1.4 will capture the instances of the representation problem with some known applications. It also gives the corresponding designation of the involved families in each case. The subsequent Figure 1.5 presents in detail the possible combinations of overlap-$\mathcal{X}$, cross-$\mathcal{X}$, and $\mathcal{X}$ closed families with $\mathcal{X}$ being a subset of $\mathcal{Z} = \{ \cap, \cup, \setminus, \Delta \}$.

Remark 1.8 *For inclusion reasons, if a $\mathcal{Z}$ closed family is not representable in polynomial space complexity, neither will be the overlap-$\mathcal{X}$, cross-$\mathcal{X}$, and $\mathcal{X}$ closed families, for every $\mathcal{X} \subseteq \mathcal{Z}$. On the other hand, a representation of a cross-$\mathcal{Y}$ family of size τ gives a representation of the same size τ for overlap-$\mathcal{X}$, cross-$\mathcal{X}$, and $\mathcal{X}$ closed families, for every $\mathcal{X} \supseteq \mathcal{Y}$.*

Representing an Arbitrary Family

We now have the necessary ingredients to finally picture the general representation of an arbitrary family $\mathcal{F} \subseteq 2^X$. Assume that we already know how to represent every member

of a set FOO containing only families satisfying the quotient property. More precisely we know how to map bijectively every member of FOO to a small object. For instance, let us consider a family which is both bipartitive and quotient, where bipartitivity denotes the property of being both cross-$\{ \cap, \cup, \setminus, \Delta \}$ closed and closed under the complementation. Then we know from a lemma in [39] that one can represent them in $O(1)$ space. (More precisely, we know that there are only two such families: 2^X and $\{X\} \cup \{\{x\},\ x \in X\}$.) Thus, the bipartitive quotient families figure in the list FOO. We can represent the family $\mathcal{F}$ as follows.

- Compute the decomposition tree $\mathcal{T}$ of $\mathcal{F}$.
- For every internal node of $\mathcal{T}$, do
  - Ask whether the corresponding quotient ranges over the known ones from the list FOO. This requires a recognition algorithm.
  - If the answer is yes, represent it accordingly. The representation is supposed to be of small space.
  - Otherwise, represent the quotient by itself. This increases the used space, due to the amortised storage of $\mathcal{T}$.
- Return the annotated decomposition tree.

Roughly, the more representable quotient families we know, the more efficient this compression method is. Clearly, this is a lossless compression method. A consequence is that representing a random set family will more likely result in compressing ratio null (and even negative, due to the storage of the decomposition tree). Notwithstanding, set families in the applications occur mostly in specific situations, coming along with specific structural properties. Then, it could be worth checking the representation paradigm. We will meet such situations throughout the thesis in various graph decompositions (part I) and also in the algorithmic solution of several graph problems (part II).

1.3 Brief Notes on Two Simple Cases

This section is suggested by S. Thomassé.

A Non-Polynomial Case

Sometimes it is simply impossible to represent set families in a reasonable way. Presuming that "reasonable" in computer science refers to some polynomial size assumption, it is

actually more accurate to claim the previous statement *most of the time*. Indeed, let us consider a ground set of n elementary objects. We will count how large may be the universal class containing every family over this ground set.

Obviously, there are 2^n choices for each member of a family in this class. Therefore, there are $C_{2^n}^k$ choices for each family of size k. This implies $2^{2^n} = C_{2^n}^0 + C_{2^n}^1 + \cdots + C_{2^n}^{2^n}$ choices for each family of the universal class. However, there are only $2^{P(n)}$ distinct ways to encode something from $P(n)$ bits. Then, if $P(n)$ is polynomial on n, it is obviously not sufficient to cover 2^{2^n} families with $P(n)$ bits when n gets an arbitrarily high value. We deduce that an arbitrary family over a ground set of size n cannot be encoded in a number of bits that is asymptotically polynomial on n.

There are actually many ways to consider the universal class. Basically, its cardinal is exactly that of the power set of the boolean lattice of dimension n. In other words, and for a more graphical intuition, one can also address the number of vertex subsets of the hypercube of dimension n. We will use the point of view of a lattice to give a proof for the following Proposition 1.4.

Let us define a strict subclass of the universal class, denoted by SQL (as in *"still quite large"*). We begin with a special family, that we call $Trunk$. For instance, $Trunk$ could be the family of all subsets of the ground set of cardinality strictly less than $\lfloor \frac{n}{2} \rfloor$. We then consider another family, that we call (potential) $Branches$. The requirement is that no member of $Branches$ belongs to $Trunk$. For the same example, let us say that $Branches$ is the collection of all subsets of the ground set of cardinality exactly equal to $\lfloor \frac{n}{2} \rfloor$. Obviously, adding some members of $Branches$ to the family $Trunk$ results in a family over the same ground set. We define SQL as the class of all families that can be obtained by adding some members of $Branches$ to the $Trunk$. Then, it follows directly from definition that there are as many families in SQL as there are subsets of $Branches$.

In the above example, the cardinality of $Branches$ is $C_n^{\lfloor \frac{n}{2} \rfloor} \geq 2^{\lfloor \frac{n}{2} \rfloor}$ (it is asymptotically equivalent to the width – the maximum size of an anti-chain – of the boolean lattice of dimension n; equivalently, it is also roughly the size of a diagonal of the hypercube of dimension n). Hence, there are at least $2^{2^{\lfloor \frac{n}{2} \rfloor}}$ pairwise distinct families in our special SQL class. Then, the previous size argument implies that such a family cannot be encoded with a number of bits asymptotically polynomial on n. Notice also that every family belonging to SQL is closed under the intersection and the difference. In other words, the $\{ \cap, \setminus \}$ closed families form a superclass of SQL, hence cannot be represented with a number of bits asymptotically polynomial on n. Now, replacing the $Trunk$ by the family of all subsets of the ground set of cardinality strictly greater than $\lfloor \frac{n}{2} \rfloor$ while keeping the same $Branches$ yields the same property for $\{ \cup \}$ closed families. We sum up with

Proposition 1.4 *It is not possible to represent an arbitrary* $\{ \cap, \setminus \}$ *closed family in an asymptotically polynomial space on the size of the ground set. Neither that is possible for an arbitrary* $\{ \cup \}$ *closed family.*

A Polynomial Case

From a similar point of view of what has just been said, the following polynomial result can be obtained, which is somewhat strange. We already saw that the universal class is in bijection with the family of vertex subsets of a hypercube of dimension n. Alternatively, it can also be seen as the family of subsets of vectors of a space of dimension n over the field $GF(2)$ of integers modulo 2 (informally: fix an origin, then every points of the hypercube can be seen as a vector). More concisely, we can just address every singleton as a vector of the basis, define them to be linearly independent, and consider there are no other vectors in the basis. It is then straightforward that every subset of the ground set is a $GF(2)$-linear combination of vectors of the basis: just choose scalar 1 for members of the subset and scalar 0 for non-members. This is usually called the "bit vector representation" of a subset, and is also the reason why the power set of X is usually denoted by $\{0,1\}^X$, or simply by 2^X.

In this vector space, the symmetric difference operation (over sets) is exactly the addition operation (over vectors) of the space. Then, it follows directly from definition that any $\{ \Delta \}$ closed family is a subspace of the vector space. Since any subspace can be represented with a basis, namely a set of at most n vectors (of exactly n bits each), the existence of a representation with at most n^2 bits follows. It is noteworthy to highlight that, as long as the ground set is already encoded, those n^2 bits are exact: there is no "$\log n$ neglect"! Now, on the other hand, each subspace of the previous vector space gives rise to a distinct $\{ \Delta \}$ closed family. Since their number* is asymptotically greater than a constant times $2^{\frac{n^2}{4}}$, it is not possible to have a representation requiring less than quadratic space, that is

Proposition 1.5 *Let* $\mathcal{F}$ *be an arbitrary* $\{ \Delta \}$ *closed family which is not necessarily proper and connected. Then, it is possible to represent* $\mathcal{F}$ *with at most* n^2 *bits, where* n *is the size of the ground set of* $\mathcal{F}$*. It is not possible to represent* $\mathcal{F}$ *using an encoding space that is asymptotically sub-quadratic on* n*.*

Firstly, compared to the exponential cases of the difference operation (cf. $\{ \setminus \}$ closed families), this result is surprising by its compactness: at most n^2 bits *"cash"*, and none

*This number is (in-)famous in the area of enumerative combinatorics under the different names of q-binomial coefficients, Gaussian coefficients, and Gaussian binomials.

of an $O(n^2)$ asymptotic analysis. Roughly, any "cash-result" is somehow similar to the case of overlap-free families, where the linear space complexity is obtained from a direct counting of their cardinality (at most $2n$). Secondly, it is clear that the slight relaxation to overlap-$\{ \Delta \}$ families ruins totally the result. For a side note, it is from my personal feelings that the structural behaviour of an overlap-$\{ \Delta \}$ family is rather counter-intuitive. However, as the compactness of the above result means there are quite few families which are $\{ \Delta \}$ closed, we still conjecture that

Conjecture: *Cross-$\{ \Delta \}$ families are polynomially representable.*

Proving this conjecture will imply a positive answer to most of the open questions in Figure 1.5 at the end of this chapter: the only case left open would be cross-$\{ \cup, \setminus \}$.

A general note: The consequential work reviewed in [47] by A. Ehrenfeutch, T. Harju and G. Rozenberg includes a framework for graph decomposition. This relies on a solution for the representation problem of a particular class of set families, the so-called class of *weakly partitive families* or *siba*'s, a shortcut for *semi-independent boolean algebras*. The latter result was established by several authors in the early 1980s: cf. M. Chein, M. Habib and M.-C. Maurer [27], and also R. Möhring and F. Radermacher [91, 92]. Actually, one can slightly modify the framework described in [47] in order to obtain another general method for representing an arbitrary set family, namely to obtain an alternative to this chapter. More precisely, the result on weakly partitive families investigates the overlap-free members of the input family. It can lead to what can be qualified as a framework for representing set families via an "overlap-free decomposition tree" (see Definition 2.5 in the upcoming chapter), as opposed to the "cross-free decomposition tree" of previously seen Definition 1.1. We borrowed much on the ideas presented in [47]. For instance, they are seminal for the very approach presented in this chapter itself! Actually, we will also revise the overlap-free framework in Section 2.2.2. We show how, for weakly partitive families, the overlap-free and cross-free approaches coincide. However, we will see further how the overlap-free approach fails on some cases, including the symmetric crossing families (see Corollary 2.4), while the cross-free approach results in polynomial representations in these cases. Accordingly, the cross-free approach is a strict generalization of the overlap-free approach. Then, the framework we have presented in this chapter can also be seen as a strict generalization of the framework described in [47].

Note on Edmonds-Giles Theorem 1.1: The so-called *Edmonds-Giles theorem* [46] in the area of combinatorial optimization is a much stronger result, which was among the seminal works of an important theory in modern combinatorial optimization, so-called *min-max duality*. However, a proper introduction to duality is beyond the purposes of our composition (see for example [106] for more details). This note only gives some information in order to avoid possible misunderstandings. In [46], Theorem 1.1 acted as a preliminary lemma for more complex results. Also, it can be enhanced in order to result in trees of smaller size (roughly, some arcs in the corresponding trees can be contracted). This is the case for the versions explained in [46], in [57, 105], and also in [78, Chapter 2, Section 2.1.1]. For this kind of "enhanced" trees, we recommend the latter reference, which is a good (and also recent) introduction. The version we gave for Theorem 1.1 follows from [106, page 215]. It does not necessarily result in the most space-efficient tree. However, the asymptotic space complexity is the same, and this (detailed) version is much more suited for the purposes addressed in our composition, in particular with the notion of a quotient.

Note on the terminologies of a cross-free/overlap-free member: In topics around modular graph decomposition, a – and probably *the* – seminal result, due to T. Gallai, denotes an overlap-free member by the terminology of a strong member [62, cited in [85]]. One of the most consequential references on those topics [47] also follows this terminology, as well as various recent studies [38, 81, 94, 101, 103]. For cross-free members, the first reference to our knowledge denotes them by *good* members [39, cited in [41]]. Some recent studies [38, 94, 103] denote abusively overlap-free and cross-free members by strong members. However, such an abuse hides important insights for the purposes of representation. Among other things, both views provided by the dual result of Edmonds-Giles theorem are important for our discussion, this will especially be the case for Sections 2.2.1 and 2.2.2. Accordingly, we opt for overlap-free/cross-free terminologies for more concision. Last but not least, doing this provides us with a broader view over binary set relations.

description	known as	representable in	applicable for
cross-$\{ \cap, \cup \}$	crossing family	$O(n^2)$ [58]	minimum s,t-cuts of a network, umodules of a digraph
overlap-$\{ \cap, \cup \}$	intersecting family	$O(n^2)$ [58]	minimum cuts of a network
closed under the complementation and cross-$\{ \cap \}$ closed	symmetric crossing family	$O(n)$ [39]	symmetric submodular function minimization
closed under the complementation and cross-$\{ \cap, \Delta \}$ closed	bipartitive family	$O(n)$ [39]	splits of a graph, bijoins of a graph, canonical bimodules of a bipartite graph
overlap-$\{ \cap, \cup, \setminus \}$	weakly partitive family	$O(n)$ [27]	modules of a digraph, clans of a 2-structure, common intervals of permutations
overlap-$\{ \cap, \cup, \setminus, \Delta \}$	partitive family	$O(n)$ [27]	modules of a graph, modules of a symmetric 2-structure
cross-$\{ \cap, \cup, \setminus \}$	weakly partitive crossing family	$O(n)$ [21] (cf. Section 3.1)	sesquimodules of a digraph
overlap-$\{ \cup, \setminus \}$	union-difference family	$O(n^2)$ [15] (cf. Section 3.2)	sesquimodules of a 2-structure

Figure 1.4: Applications of the representation problem of set families. The size of the ground set is denoted by n.

$\mathcal{X}$	$\mathcal{X}$ closed	overlap-$\mathcal{X}$	cross-$\mathcal{X}$
$\{ \cup \}$	**N.R.P.** from Proposition 1.4	$\longleftarrow$	$\longleftarrow$
$\{ \cap \}$	$\downarrow$	$\downarrow$	$\downarrow$
$\{ \cap, \setminus \}$	**N.R.P.** from Proposition 1.4	$\longleftarrow$	$\longleftarrow$
$\{ \setminus \}$	$\uparrow$	$\uparrow$	$\uparrow$
$\{ \cup, \Delta \}$	$\downarrow$	– **OPEN** –	– **OPEN** –
$\{ \Delta \}$	$\leq n^2$ bits from Proposition 1.5	– **OPEN** –	– **OPEN** –
$\{ \cap, \setminus, \Delta \}$	$\uparrow$	– **OPEN** –	– **OPEN** –
$\{ \setminus, \Delta \}$	equivalent to $\{ \cap, \setminus, \Delta \}$	equivalent to $\{ \cap, \setminus, \Delta \}$	equivalent to $\{ \cap, \setminus, \Delta \}$
$\{ \cap, \Delta \}$	equivalent to $\{ \cap, \setminus, \Delta \}$	equivalent to $\{ \cap, \setminus, \Delta \}$	equivalent to $\{ \cap, \setminus, \Delta \}$
$\{ \cap, \cup \}$	$\longrightarrow$	$O(n^2)$ [58]	$O(n^2)$ [58]
$\{ \cup, \setminus \}$	$\longrightarrow$	$O(n^2)$ [15] (cf. Section 3.2)	– **OPEN** –
$\{ \cap, \cup, \setminus \}$	$\longrightarrow$	$O(n)$ [27]	$O(n)$ [21] (cf. Section 3.1)
$\{ \cap, \cup, \setminus, \Delta \}$	$\longrightarrow$	$O(n)$ [27]	$O(n)$ [21] (cf. Section 3.1)
$\{ \cup, \setminus, \Delta \}$	equiv. to $\{ \cap, \cup, \setminus, \Delta \}$	equiv. to $\{ \cap, \cup, \setminus, \Delta \}$	equiv. to $\{ \cap, \cup, \setminus, \Delta \}$
$\{ \cap, \cup, \Delta \}$	equiv. to $\{ \cap, \cup, \setminus, \Delta \}$	equiv. to $\{ \cap, \cup, \setminus, \Delta \}$	equiv. to $\{ \cap, \cup, \setminus, \Delta \}$

Figure 1.5: Possible combinations of overlap-$\mathcal{X}$, cross-$\mathcal{X}$, and $\mathcal{X}$ closed families with $\mathcal{X} \subseteq \{ \cap, \cup, \setminus, \Delta \}$. The size of the ground set is n. **N.R.P.** is short for *not representable in polynomial size on* n, while – **OPEN** – refers to the open question whether the class is polynomially representable or not. Arrows from cell A to cell B denote the fact the result of cell A follows from that of cell B (cf. Remark 1.8).

Chapter 2

Fundamental Representations

Some aspects presented in this chapter are based on [16, 17, 18, 21]. This includes in particular Sections 2.2.2 and 2.3.2. However, we will not go into the details of [16, 17, 18]. Further information on the decomposition schemes given in Section 2.3.2 are developed in [83]. Most results presented in Section 2.2.2 have not been published so far.

The aim of this chapter is to survey a number of classes of set families for which a polynomial space representation is known. For such a class, we recall the basic notions related to that class, and show at least one representation theorem. Indeed, as set families involve in many research areas, the same class of set families might have been studied independently, leading to different representation theorems. When such a situation occurs, we will select, among the most efficient representations, those that are most simple (in our opinion!), as well as those that fit best into the scope of our discussion. Then, we would rather suggest readers with further interests in the other representations to follow the references given for each case.

For instance, intersecting families have two quadratic size representations: so-called *tree of posets* [58] and a more recent representation given in [6]. As much as we will recall the latter representation, which is simpler in our opinion, we only give a brief comment for the former representation by trees of posets. To give another example, weakly bipartitive families are also known as symmetric crossing families. The first terminology leads to a representation that usually refers to the so-called *split decomposition*. This decomposition was introduced and studied in the early 1970s [39] and figured also in some recent works [94, 103]. The other terminology of symmetric crossing leads to the so-called *cactus (hyper-)tree* representation, and was studied by several authors. The very definition of a cactus hypertree seems to be first introduced in Russian in the mid 1970s. For a quite recent introduction in English refer to, e.g., [54] (therein, the reference to the mid 1970s Russian paper is cited as [45]). When facing this class of families, we will favour bipartitive terminology, as it fits better in our general method described in Chapter 1.

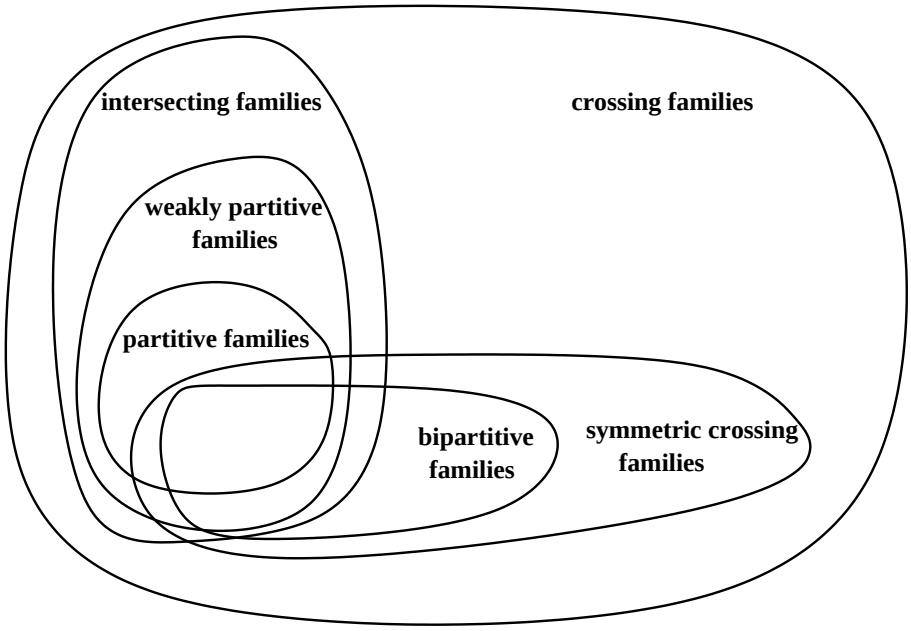


Figure 2.1: Some fundamental classes of set families. When comparable, the inclusion of a class is strict.

Another noteworthy remark on all the classes addressed in this chapter would be that they are seminal. Firstly, each of them has been rather well-studied for half a century (at least*!). Moreover, they will be fundamental for this thesis since the next chapter will deepen two of their generalizations. As well, this chapter is concluded with selected applications of the representation paradigm based on those set families. We will focus on applications in graph theory, and more specifically in the problem of decomposing graphs and their common generalization to 2-structures (roughly, a 2-structure is a complete directed graph given along with a colouring on its arcs). Before going into the details, let us recall that, unless otherwise stated explicitly, not only all families in the thesis are supposed to cover a ground set of at least 3 elements, but they are also supposed to be *proper and connected*, namely

Definition 2.1 (Proper and Connected Family) A family $\mathcal{F}$ over a ground set X is *proper and connected* if it excludes the empty set, and if it contains all the trivial subsets of X, namely the sets $\{x\}$ (for all $x \in X$) and the set X.

This assumption will not change the results on space complexity since any family can be made proper and connected by adding/removing the corresponding subsets. The "cleaning" process will not take a space complexity higher than that of the ground set.

*For instance, according to M. Habib [64], the first notions related to contemporaneous modular decomposition and its closed associates the partitive families were already mentioned in 1949 in the works of A. A. Zykov.

2.1 Intersecting and Crossing Families

We first focus on two seminal cases: intersecting and crossing families. Both of them are quite well worked since they intervene very naturally among submodular function minimization issues. Indeed, the family of non-empty minimizers of a submodular function is an intersecting family, while the family of minimizers that are neither the empty set nor the universal set is a crossing family. Let us begin with the intersecting families.

Definition 2.2 (Intersecting Family) A proper and connected family is an *intersecting* family if it is closed under the union and the intersection of its overlapping members. (They are exactly the overlap-$\{\cap, \cup\}$ families.)

H. Gabow gave a representation for intersecting families using quadratic space on the size of the ground set [58]. It is so-called the *tree-of-posets* representation and is probably the first quadratic-size result for those families. This was used to speed-up a number of optimization problems, presented in the same paper. Let us call a lattice family a set family which is closed under the intersection and the union of arbitrary members. Informally, the approach of [58] can be seen as the extraction of a lattice family from the input family $\mathcal{F} \subseteq 2^X$ when looking into members of $\mathcal{F}$ of high cardinality (strictly greater than $\frac{|X|}{2}$). After this, members belonging to the lattice family are represented by a well-known poset representation, while the remaining members are represented using some recursive process (hence the terminology of a tree-of-posets). To obtain the quadratic size result, the approach also requires some non-trivial counting argument. The main drawback of this representation lays on both its number of intermediary notions and the fact that the global approach is rather heavy.

On the other hand, A. Bernáth proved the following direct counting result [6]. For every element $x \in X$, a subset $A \subseteq X$ is called *an x-free subset* if x does not belong to the subset: $x \notin A$. For every subset $A \subseteq X$, a subset $B \subseteq X$ is called *an A-free subset* if A and B are disjoint: $A \cap B = \emptyset$.

Lemma 2.1 (Bernáth [6]) *If a family $\mathcal{F} \subseteq 2^X$ is closed under the union of its crossing members, then the following family, which is not necessarily proper and connected, has at most $2|X| - 2$ members.*

$$\mathcal{H} = \{A \subseteq X \ , \ \exists x \in X \text{ s.t. } A \text{ is a maximal } x\text{-free member of } \mathcal{F}\}.$$

Proof: The proof given in [6] is as follows. For every $x \in X$, let $\mathcal{H}_x$ denote the family of maximal x-free members of $\mathcal{F}$. Under this notation, $\mathcal{H} = \bigcup_{x \in X} \mathcal{H}_x$. To prove the lemma, we proceed by induction on $n = |X|$.

Clearly, the cases of $n = 1$ and $n = 2$ are trivial. Let us consider the case where $n \geq 3$. Firstly, $X \notin \mathcal{H}$ by definition. As well, if all members of $\mathcal{H}$ are trivial subsets of X, then their number cannot exceed n, and we are done. Now suppose there are non-trivial subsets of X among the members of $\mathcal{H}$. We take $A \in \mathcal{H}$, A non-trivial, and minimum by size. Note by minimality of A that every member $B \in \mathcal{H}$ with $B \subsetneq A$ is a singleton. We partition $\mathcal{H}$ into two parts:

- $\mathcal{K} = \{\ B \in \mathcal{H},\ A \cap B = \emptyset \text{ or } A \subseteq B\ \}$, and
- $\mathcal{L} = \{\ B \in \mathcal{H},\ A \circledcirc B \text{ or } B \subseteq A\ \}$.

We first evaluate $\mathcal{K}$ by proving the following claim:

$$\mathcal{K} \subseteq \bigcup_{x \in X \setminus A} \mathcal{H}_x \ \cup\ \{\ B \subseteq X\ ,\ \ B \text{ is a maximal } A\text{-free member of } \mathcal{F}\ \}.$$

Indeed, let B be a member of $\mathcal{K}$. Basically, B is a member of $\mathcal{H}$, so let $b \notin B$ be such that $B \in \mathcal{H}_b$. If $b \in X \setminus A$ then we are done. Otherwise, A and B have to be disjoint, from definition of $\mathcal{K}$ and the fact $A \setminus B$ is non-empty (for the latter fact b is a witness). In other words, B is an A-free member of $\mathcal{F}$. Then, the maximality of B from being a member of $\mathcal{H}_b$ allows to conclude.

We now build another family $\mathcal{F}'$ over another ground set X'. First, let us shrink A into one new element α: $X' = (X \setminus A) \cup \{\alpha\}$. Then, we define $\mathcal{F}'$ to contain the subsets of X' corresponding to members of $\mathcal{F}$ that are either disjoint from A or a super-set of A: $\mathcal{F}' = \{B \in \mathcal{F},\ A \cap B = \emptyset\} \cup \{(B \setminus A) \cup \{\alpha\},\ B \in \mathcal{F} \text{ and } A \subseteq B\}$. From the non-triviality of A, we have $|X'| < |X|$, which enables the use of an inductive argument. Finally, one can check that $\mathcal{F}'$ is closed under the union of its crossing members, and deduce from induction that $|\mathcal{K}| \leq 2|X'| - 2$, that is $|\mathcal{K}| \leq 2n - 2|A|$.

We now conclude by proving that $|\mathcal{L}| \leq 2|A| - 2$. Indeed, it is firstly straightforward to notice that $\mathcal{L} \subseteq \bigcup_{x \in A} \mathcal{H}_x$. Moreover, let $a \notin A$ be such that $A \in \mathcal{H}_a$, then, for every $x \in A$, for every member B of $\mathcal{H}_x$, one and only one of the following holds:

- $A \cap B = \emptyset$, and B is not a member of $\mathcal{L}$.
- $B \subseteq A$, and B is a singleton by minimality of A.
- $A \circledcirc B$. But then we have two further facts. First, $a \in B$ otherwise the maximality of A as a member of $\mathcal{H}_a$ would fail. Second, B is unique w.r.t. x otherwise the maximality of B as a member of $\mathcal{H}_x$ would fail. Finally, whenever this case occurs, we denote B by Z_x, since B is unique w.r.t. x.

Note that, for every $x \in A$, Z_x exists if and only if the third case in the above list occurs. The remainder of the proof of the lemma is achieved by the following case analysis. Let $Z = \{x \in A,\ Z_x \text{ exists}\}$. We distinguish three cases

- $|Z| \leq |A|-2$. Then, we can conclude as $\mathcal{L}$ contains at most $|Z|$ non-trivial members.
- $Z = A \setminus \{x\}$. Then $\{x\} \notin \mathcal{L}$: otherwise there would be $y \in A$ such that $\{x\} \in \mathcal{H}_y$; this would imply $x \notin Z_y$; but then Z_x would exist. After this observation, we can conclude as $\mathcal{L}$ contains at most $|Z|$ non-trivial members and at most $|A| - 1$ singletons.
- $Z = A$. We first prove that $\{Z_x,\ x \in A\}$ has at least two minimal members (by inclusion): otherwise let Z_{x_0} be the unique minimal member; but then, for every $y \in A \cap Z_{x_0}$ (y exists since $A \not\subset Z_{x_0}$), we would have $\{y\} \subseteq Z_{x_0} \subseteq Z_y$, which is impossible. Finally, for every minimal member Z_x of the previous set, we have two mutually exclusive cases
  - either $\{x\} \notin \mathcal{L}$,
  - or $\{x\} \in \mathcal{L}$. But then, for some $y \in A$, $Z_x = Z_y$. Indeed, since $\{x\} \in \mathcal{L}$, there is $y \in A$ such that $\{x\} \in \mathcal{H}_y$. As $Z_y \in \mathcal{H}_y$ also holds, we have $x \notin Z_y$. Now, assuming $Z_x \subsetneq Z_y$ would contradict the maximality of Z_x in $\mathcal{H}_x$; assuming $Z_x \not\subset Z_y$ would also lead to the same contradiction; and assuming $Z_y \subsetneq Z_x$ would contradict the minimality of Z_x in $\{Z_z,\ z \in A\}$. The only possibility is that $Z_x = Z_y$.

  In both cases we can conclude easily using the fact there are at least two minimal members in $\{Z_x,\ x \in A\}$.

We have seen that in all three cases we can conclude the proof of the lemma. □

We can now represent intersecting families by combining Bernáth lemma and the following folklore fact:

Proposition 2.1 *(see, e.g., [6]) Let $\mathcal{F} \subseteq 2^X$ be an arbitrary intersecting family. Let $\mathcal{H}$ denote the family $\mathcal{H} = \{A \subseteq X\ ,\ \ \exists x \in X$ s.t. A is a maximal x-free member of $\mathcal{F}\}$, which is not necessarily proper and connected. Then,*

$$\mathcal{F} \setminus \{X\} = \left\{ \bigcap_{i \in \mathcal{I}} H_i\ ,\ \ \forall i \in \mathcal{I},\ H_i \in \mathcal{H} \right\} \setminus \{\emptyset\}.$$

Proof: The inclusion of the right hand-side to the left hand-side follows directly from definition. Let us prove the inclusion of the other way around. Let us pick any member A of $\mathcal{F}$ which is different from X. If $A \in \mathcal{H}$ then we are done. Otherwise, let $X \setminus A = \{x_1, x_2, \ldots, x_k\}$. Let $H_1, H_2, \ldots, H_k \in \mathcal{H}$ be the members of $\mathcal{H}$ such that $A \subseteq H_i$ and H_i is x_i-free, for all $1 \leq i \leq k$. We define $B = \bigcap_{i=1}^{k} H_i$. Straight from definition $A \subseteq B$. Now, for all $1 \leq i \leq k$, $x_i \notin H$ clearly implies $x_i \notin B$. Hence $B \subseteq A$ and we deduce that A is the intersection of the H_i's. □

Corollary 2.1 *An intersecting family $\mathcal{F} \subseteq 2^X$ can be represented in $O(|X|^2)$ space.*

Proof: From Proposition 2.1, the family $\mathcal{H}$ defined therein is a representation of $\mathcal{F}$. From Bernáth Lemma 2.1, the needed space to store $\mathcal{H}$ is in $O(|X|^2)$. □

Corollary 2.2 *The complexity of an intersecting family on a ground set X is in $O(|X|^2)$.*

Basically, Bernáth Lemma 2.1 is very nice by the simplicity of its statement. Now, conversely, there is a second folklore fact stating it is not possible to have a representation that is asymptotically better than the one given by Corollary 2.1, namely

Proposition 2.2 *The quadratic complexity of an arbitrary intersecting family is tight.*

Proof: We proceed with a counting argument as what has been done at the end of Chapter 1 for Remark 1.4. Roughly, we need to display sufficiently "many" intersecting families over the same ground set X. Concisely, let $X = \{x_1, x_2, \ldots, x_n\}$, $p = \lfloor \frac{n}{2} \rfloor$, and $q = \lceil \frac{n}{2} \rceil$. We will define 2^{pq} intersecting families over X. This would imply that we cannot represent an arbitrary intersecting family with fewer than $\lfloor \frac{n}{2} \rfloor^2$ bits, and the proposition follows. Let G be a directed graph over the vertex set X such that

- G is a bipartite graph between $\{x_1, x_2, \ldots, x_p\}$ and $\{x_{p+1}, x_{p+2}, \ldots, x_n\}$,
- and there are no arcs in G of the form (x_j, x_i) with $i < j$.

The number of such graphs is clearly 2^{pq} since there are only two choices per pair (i, j), ranging as $1 \leq i \leq p < j \leq n$, whether the arc (x_i, x_j) belongs to a graph or not. Let $\mathcal{F}_G = f(G)$ be the family of all non-empty vertex subsets of X of in-degree zero, where the in-degree of a vertex subset counts the number of arcs of G with an out-going extremity inside that subset and the other extremity outside the subset. Then, it is straightforward to prove that $\mathcal{F}_G$ is an intersecting family. Therefore, it suffices to prove that the function f is injective to obtain the desired amount of different intersecting families over X. Indeed, let G and G' be two different graphs which both satisfy the above requirement. Let us

prove that $\mathcal{F}_G \neq \mathcal{F}_{G'}$. Firstly, since the two graphs are different we can suppose without loss of generality that there are i and j with $1 \leq i \leq p < j \leq n$ such that arc (x_i, x_j) belongs to G' and does not belong to G (just permute G and G' if G' is a partial subgraph of G). Now let A be the minimal member of $\mathcal{F}_G$ which contains x_j. The minimality of A implies that, for all $k > p$ and $k \neq j$, A does not contain x_k. Now, if x_i does not belong to A then we are done because this would mean A cannot be a member of $\mathcal{F}_{G'}$. Otherwise, $B = A \setminus \{x_i\}$, which cannot belong to $\mathcal{F}_G$ by minimality of A, should have a strictly positive in-degree in G. However, as A is a member of $\mathcal{F}_G$, the previous fact can only occur when there is an arc in G linking x_i to some vertex in B, say x_{k_0}. Clearly $k_0 > p$ and $k_0 \neq j$. This contradicts the minimality of A. □

Turning our attention to the second case of this section, the class of crossing families is the broadest class presented in this chapter. The situation is not the same for the remaining of the thesis since the next chapter will present a class that is incomparable to this class (inclusionwise).

Definition 2.3 (Crossing Family) A proper and connected family is a *crossing* family if it is closed under the union and the intersection of its crossing members.

Since the crossing of two subsets implies their overlapping, an intersecting family is clearly a crossing family. Conversely, we can represent a crossing family $\mathcal{F}$ with two intersecting families as follows. Let x be an element of the ground set of $\mathcal{F}$. Let $\mathcal{F}'$ be the family of all complements of members of $\mathcal{F}$ containing x, and $\mathcal{F}''$ the family of all members of $\mathcal{F}$ excluding x. Then, one can check that both $\mathcal{F}'$ and $\mathcal{F}''$ are intersecting families. Finally, representing $\mathcal{F}$ by the two latter families follows trivially from $\mathcal{F} = \overline{\mathcal{F}'} \uplus \mathcal{F}''$, where $\overline{\mathcal{F}'}$ is the family of all complements of members of $\mathcal{F}'$. Now, intersecting families own polynomial space representations from what has been previously said. It follows that

Remark 2.1 *The complexity of a crossing family is asymptotically proportional to that of an intersecting family on the same ground set.*

2.2 More Specific Families

Due to the quadratic complexity of the result on intersecting and crossing families, one can suppose there is space for more efficient representations, for example with a better theorem or when restricting to some subclass. However, the previous Proposition 2.2 states that improving the representation theorem is impossible for arbitrary intersecting families. Then, by inclusion, this is also impossible for arbitrary crossing families. As a

natural consequence, it is interesting (by elimination perhaps?) to look for subclasses of crossing families which have a more efficient representation. This will be the topic of the section. The fact that the next section gives applications of all these classes in some quite important branches of graph theory is all the more instructive.

2.2.1 Partitive Families – the Classical Approach

Partitive families are particular cases of intersecting families. Their structure is fundamental for a very well-studied field in graph theory, so-called modular decomposition (see Section 2.3.1). The terminology of partitivity was introduced and studied in [27]. In one of the most complete references of topics around modular decomposition, namely in [47], a weakly partitive family is also called a *siba* – shortcut for *semi-independent boolean algebra*. Therein, the notion is fundamental for the description of clan decomposition.

Definition 2.4 (Weakly Partitive Family and Partitive Family) A proper and connected family is *weakly partitive* if it is closed under the union, the intersection, and the difference of its overlapping members. Moreover, if the family is also closed under the symmetric difference of its overlapping members, it is called *partitive*.

The terminology of weakly partitivity as opposed to (bare) partitivity is historical. It could be interesting to replace them by partitivity and strongly partitivity, respectively. Indeed, we will see in Theorem 2.1 that a weakly partitive axiom is sufficient to obtain a linear space representation. Also, the additional closure provided by the partitive axiom is quite marginal in a large part of the related literature in the topic. More precisely, the common approach consists of getting a representation using the weakly partitive axiom. Only then, the partitive axiom is used to eliminate some cases, and simplify the representation. This being said, we will follow the historical terminologies.

We have mentioned in several places in the last chapter that modular decomposition makes use of a version of decomposition tree that is not based on the cross-free members. The actual situation is as follows. Modular decomposition relies on the representation of partitive families. A partitive family, in turn, is usually studied under a different framework, namely via the Edmonds-Giles rooted tree representation of the (overlap-free) subfamily of overlap-free members of the input family, that is

Definition 2.5 (Overlap-free Decomposition Tree) The *overlap-free decomposition tree* of a family $\mathcal{F} \subseteq 2^X$ is defined as the Edmonds-Giles's representation [46] of its overlap-free members.

The following lemma shows how the two views of overlap-free decomposition tree and cross-free decomposition tree are almost equivalent for weakly partitive families. Actually,

we show the result for a larger class of set families. However, note that this does not necessarily hold for families such as the union-difference families that will be introduced in the next chapter. Before continuing, recall that an overlap-free member of any set family is also a cross-free member of the family. On the other hand,

Lemma 2.2 *If a family is closed under the intersection and the difference of its overlapping members, then every cross-free member of the family is either an overlap-free member or the complement of an overlap-free member. A consequence is that, except for the root and the edge orientations, the cross-free decomposition tree of such a family coincides with its overlap-free decomposition tree.*

Proof: Assume that a cross-free member A is not an overlap-free member. Then, A must be the complement of an overlap-free member. Indeed, there exists another member B of the family such that A and B overlap, but do not cross. Therefore the union of A and B is equal to the ground set of the family. A consequence is that $C = B \setminus A$ is exactly the complement of A. Now, by difference closure, C is a member of the family. If $|C| = 1$ then C is clearly an overlap-free member and we can conclude. Otherwise, suppose that C is not an overlap-free member: there exists a member D of the family such that C and D overlap. Then, the union of C and D is equal to the ground set of the family, otherwise D would cross (the cross-free member) A. But this also would mean that $D \cap B$, which is a member of the family by the intersection closure, crosses A. We conclude that C is actually an overlap-free member, and A the complement of an overlap-free member. □

Chapter 1 was meant for the general case of an arbitrary family. There, we prefer addressing cross-free decomposition trees in order to classify the members of the input family into as many quotients as possible. However, for weakly partitive families, this will not allow to classify any more than the approach via overlap-free decomposition trees (cf. Lemma 2.2). Then, for the sake of simplicity, considering the overlap-free decomposition trees allows some shortcuts in obtaining a representation result. In our study, we nonetheless deepen the cross-free decomposition tree approach, even though it requires more writing space. A (double) motivation is that the more general cross-free approach gives more details, and at the same time, it can be used in a more general context. For instance, a situation where the overlap-free approach fails while the cross-free approach results in a polynomial representation will be exemplified in Section 2.2.3.

In order to recall the overlap-free approach, we will need to introduce the following shortcut of the notion of a (cross-free) quotient. In an overlap-free decomposition tree, every internal node u has $k = d(u) - 1$ children, except for the root which has $k = d(u)$ children. Here, the leaf sets of the subtrees rooted at the children of u can also be seen as

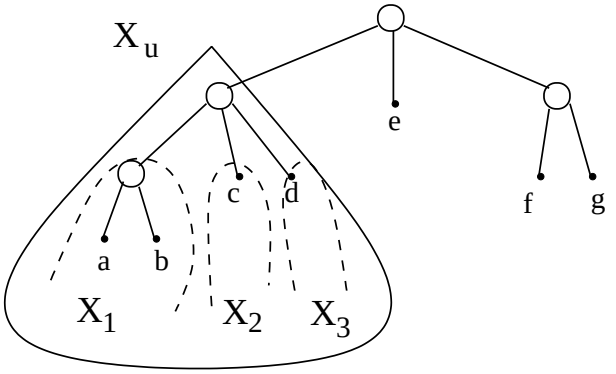


Figure 2.2: The ground set $\{X_1, X_2, X_3\}$ of an overlap-free quotient, with $X_1 = \{a, b\}$, $X_2 = \{c\}$, and $X_3 = \{d\}$. It is a partition of $X_u = \{a, b, c, d\}$.

a $k-$partition of X_u, the leaf set of the subtree rooted at u. Let $\{X_1, X_2, \ldots, X_k\}$ denote this partition (e.g., Figure 2.2). Note that $k \geq 2$. Let us consider $Y = \{X_1, X_2, \ldots, X_k\}$ as a set.

Definition 2.6 (Overlap-free Quotient) Keeping the same notation of the partition $Y = \{X_1, X_2, \ldots, X_k\}$ as above, we define the *overlap-free quotient of $\mathcal{F}$ with respect to node u* as the family $\mathcal{Q}(u) \subseteq 2^Y$ such that $Q = \{X_i \mid i \in I\}$ belongs to $\mathcal{Q}(u)$ if and only if $\bigcup_{i \in I} X_i$ belongs to $\mathcal{F}$.

Notice that an overlap-free quotient is always proper and connected. Also, its ground set may have less than three elements, but it always has at least two elements. Now, one can check that the mapping of a set family to an object made of both the overlap-free decomposition tree of the family and the overlap-free quotients of all nodes of the tree is an injective function. One can also check that the initial family can be built back from the knowledge of its overlap-free decomposition tree and the overlap-free quotients of all nodes of the tree. We here would like to highlight the following straightforward property.

Proposition 2.3 *Let $\mathcal{F} \subseteq 2^X$ be a family. If $|X| < 3$ then every member of $\mathcal{F}$ is at the same time an overlap-free member and a trivial member. Otherwise, trivial members of $\mathcal{F}$ are clearly overlap-free members. On the other hand, for every node u of the overlap-free decomposition tree of $\mathcal{F}$, all overlap-free members of the quotient $\mathcal{Q}(u)$ with respect to u are trivial with respect to the ground set of $\mathcal{Q}(u)$.*

Definition 2.7 (Overlap-free Quotient property) We say that a set family satisfies the *overlap-free quotient property* if all its overlap-free members are trivial.

One can also check that the class of partitive (resp. weakly partitive) families is *overlap-free quotient-hereditary*, namely, for every partitive (resp. weakly partitive) family $\mathcal{F}$, the

overlap-free quotient family $\mathcal{Q}(u)$ with respect to every node u of the decomposition tree of $\mathcal{F}$ is also a partitive (resp. weakly partitive) family. Accordingly, in order to represent a partitive (resp. weakly partitive) family, it suffices to represent a family that is at the same time partitive (resp. weakly partitive) and overlap-free quotient.

In the following, Theorem 2.1 will recall a result given by M. Chein, M. Habib, and M.-C. Maurer* [27]. This can be used as an intermediary step to establish a linear space representation for weakly partitive families. It is seminal for several aspects throughout the current part of the thesis. However, we will not discuss in detail how to prove this theorem. Instead, we will recall how it can be used to obtain the linear space representation. Then, we revisit the theorem and give an alternative approach to obtain exactly the same representation. Our approach follows the more general framework of cross-free decomposition developed in Chapter 1. Recall that not only all set families in this part of the thesis are supposed to have a ground set of at least 3 elements, but also they are considered to be proper and connected, that is they exclude the empty set and include all trivial subsets of the ground set.

Theorem 2.1 (Chein-Habib-Maurer [27]) *If a family $\mathcal{F} \subseteq 2^X$ is at the same time weakly partitive and overlap-free quotient, then one and only one of the following holds:*

- *$\mathcal{F}$ has no other member than the trivial members (we say that $\mathcal{F}$ is* prime*),*
- *$\mathcal{F} = 2^X \setminus \{\emptyset\}$ (we say that $\mathcal{F}$ is* complete*),*
- *there is an ordering $(x_1, x_2, \ldots, x_n)$ of the elements of X such that $\mathcal{F}$ is the family of all intervals $\{x_i, x_{i+1}, \ldots, x_j\}$ (for all $1 \leq i \leq j \leq n$) of this ordering (we say that $\mathcal{F}$ is* linear*).*

Moreover, if $\mathcal{F}$ is partitive, the third case cannot occur.

Though we will not formally prove this theorem, notice that a proof can be deduced from the proofs of Theorems 2.2 and 2.3 (see Section 2.2.2). Here, let us discuss on how Chein-Habib-Maurer theorem can be used to obtain a linear space representation of (weakly) partitive families. Let $\mathcal{F} \subseteq 2^X$ be a weakly partitive family. Let $\mathcal{T}$ be its overlap-free decomposition tree. Let $Y = \{X_1, X_2, \ldots, X_k\}$ denote the partition associated to an internal node u in $\mathcal{T}$ as in the definition of an overlap-free quotient (Definition 2.6, also exemplified in Figure 2.2). If $k = 2$ then representing the overlap-free quotient is straightforward (we can for example denote them by a special "*trivially representable*" label). Let us suppose $k \geq 3$. Then, if the overlap-free quotient is prime, its representation

*M.-C. Maurer is now M.-C. Vilarem.

requires only 1 bit stating it has no other members than the trivial subsets of Y. In order to represent an overlap-free quotient that is complete, again we need only 1 bit stating that it is made of all unions of some X_i's (informal view in the rooted tree $\mathcal{T}$: all unions of some children of the node u). For an overlap-free quotient that is linear, we also need to code an ordering over the children of the node, namely k pointers. Then, 1 bit can state it is the family of all unions of some consecutive X_i's (informal view: all intervals of children w.r.t. the aforementioned ordering). Accordingly, the global labelling for the various overlap-free quotients of the overlap-free decomposition tree requires a space that is a constant times the number of edges of the tree. The latter number is bounded by $2 \times |X| - 3$. Notice that the previous constant is low: we need to code 3 types of nodes, and at most 1 pointer per incident edge of the node (when the overlap-free quotient w.r.t. the node is linear). We conclude from Theorem 2.1 and from what has been said on overlap-free decomposition trees that weakly partitive families are representable in a space of linear complexity on the size of the ground set, that is

Corollary 2.3 *The space complexity of a weakly partitive family over X is in $O(|X|)$.*

2.2.2 Partitive Families – a New Viewpoint

In order to give the cross-free alternative to obtain the Chein-Habib-Maurer result, we need the formalism of simply-linkedness and its dual notions of a guard and a quotiental parent. These notions were introduced in [15] for the study of a strict generalization of partitive families.

Let us emphasize that the current section abandons the overlap-free terminology as long as decomposition trees and quotients are concerned. Instead, we address cross-free decomposition trees and cross-free quotients, as in Chapter 1. Recall also that not only all set families in this part of the thesis are supposed to have a ground set of at least 3 elements, but also they are considered to be proper and connected, that is they exclude the empty set and include all trivial subsets of the ground set. Clearly, a quasi-trivial member of a family is also a cross-free member. Moreover, we say that

Definition 2.8 (Simply-linked Property) A family is *simply-linked* if none of its quasi-trivial members is an overlap-free member of the family.

Definition 2.9 (Guard) If a family $\mathcal{F} \subseteq 2^X$ is not simply-linked, then directly from definition there exists one and only one element $x \in X$ such that $\mathcal{G} = \mathcal{F} \setminus \{X, \{x\}\}$ is a (proper and connected) family over $Y = X \setminus \{x\}$. In this case, the element x is called a *guard* of the family. In the remaining of the thesis, the fact the ground set of $\mathcal{G}$ is Y, and not X, is crucial.

Definition 2.10 (Quotiental Parent) Let $\mathcal{F} \subseteq 2^X$ be an arbitrary family over X. Let u be a node of the (cross-free) decomposition tree $\mathcal{T}$ of $\mathcal{F}$ such that the (cross-free) quotient $\mathcal{Q}(u)$ w.r.t. u is

- not simply-linked. Then, the guard of $\mathcal{Q}(u)$ corresponds to a unique connected component C of the forest $\mathcal{T} - u$, obtained by removing u from $\mathcal{T}$. Let v be the unique neighbour of u in $\mathcal{T}$ that belongs to C. We say that v is the *quotiental parent* of u, and define the quotiental parent orientation from u as an arc from u to v.
- simply-linked. Then, we say that u is the *quotiental parent* of itself, and define the quotiental parent orientation from u as a loop from u to u.

Definition 2.11 (Quotiental Parent Arborescence) Let $\mathcal{T}$ be the decomposition tree of a set family $\mathcal{F}$. We define the *quotiental parent arborescence* $\overrightarrow{\mathcal{T}}$ *of* $\mathcal{F}$ as the orientation given by the quotiental parent orientation on the underlying graph of $\mathcal{T}$. This might disconnect the underlying graph, namely when two neighbours u and v in $\mathcal{T}$ are such that neither u is the quotiental parent of v nor v the quotiental parent of u.

Notice that a quotiental parent arborescence might have loops. Roughly, loops therein display simply-linked quotients. Now, Chein-Habib-Maurer Theorem 2.1 can be divided into two pairwise exclusive cases as follows.

Theorem 2.2 *If a family $\mathcal{F} \subseteq 2^X$ satisfies: the weakly partitive property, the quotient property, and the simply-linked property, then one and only one of the following holds:*

- *$\mathcal{F}$ is the family of trivial subsets of X (we say that $\mathcal{F}$ is* prime*),*
- *$\mathcal{F} = 2^X \setminus \{\emptyset\}$ (we say that $\mathcal{F}$ is* complete*),*
- *there is an ordering $(x_1, x_2, \ldots, x_n)$ of the elements of X such that $\mathcal{F}$ is the family of all intervals $\{x_i, x_{i+1}, \ldots, x_j\}$ (for all $1 \leq i \leq j \leq n$) of this ordering (we say that $\mathcal{F}$ is* linear*).*

Moreover, if $\mathcal{F}$ is partitive, the last case cannot occur.

Proof: By successive applications of Lemma 2.4, Lemma 2.5, and Remark 2.2 (all three below). □

Theorem 2.3 *If a family $\mathcal{F} \subseteq 2^X$ satisfies both the weakly partitive and the quotient properties, but fails the simply-linked property, then, for $x \in X$ being the guard of $\mathcal{F}$, $\mathcal{G} = \mathcal{F} \setminus \{X, \{x\}\}$, and $Y = X \setminus \{x\}$, one and only one of the following holds:*

- $|Y| = 2$ *(we say that* $\mathcal{F}$ *is* trivially representable*: like in the case when* $|X| = 2$*, here* $\mathcal{G} = 2^Y \setminus \{\emptyset\}$*, but the difference is that* $\mathcal{F}$ *now has a guard),*
- $|Y| \geq 3$ *and* $\mathcal{G}$ *is the family of trivial subsets of* Y *(by abuse of terminology, we still say that* $\mathcal{F}$ *is* prime*, the difference is that* $\mathcal{F}$ *now has a guard),*
- $|Y| \geq 3$ *and* $\mathcal{G} = 2^Y \setminus \{\emptyset\}$ *(we still say that* $\mathcal{F}$ *is* complete*, the difference is that* $\mathcal{F}$ *now has a guard),*
- $|Y| \geq 3$ *and there is a linear ordering of the elements of* Y *such that* $\mathcal{G}$ *is the family of all intervals of this ordering (we still say that* $\mathcal{F}$ *is* linear*, the difference is that* $\mathcal{F}$ *now has a guard).*

Moreover, if $\mathcal{F}$ *is partitive, the last case cannot occur.*

Proof: (same as previous theorem). By successive applications of Lemma 2.4, Lemma 2.5, and Remark 2.2 (all three below). □

Before heading to the proof of the two theorems, let us discuss on how they can be used to retrieve the linear representation of (weakly) partitive families via the overlap-free decomposition tree. For this we establish a rooted point of view of the unrooted (cross-free) decomposition tree of a weakly partitive family. Let us proceed in two steps.

Lemma 2.3 *In the decomposition tree* $\mathcal{T}$ *of a weakly partitive family* $\mathcal{F}$*, there is at most one internal node* u *such that the quotient w.r.t. node* u *is simply-linked.*

Proof: (by contradiction). Suppose there are two internal nodes u, v in $\mathcal{T}$ such that the quotient w.r.t. each node is simply-linked. For convenience, we make use of abuse of terminology, and also denote the underlying graph of $\mathcal{T}$ by $\mathcal{T}$. Let $(u = u_1, u_2, \ldots, u_p = v)$ be the path linking u to v in $\mathcal{T}$. Let A be the leaf set of the connected component containing u we get when removing the edge between u_{p-1} and v in $\mathcal{T}$. Let B be the leaf set of the connected component containing v we get when removing the edge between u and u_2 in $\mathcal{T}$.

We first prove that both A and B are members of $\mathcal{F}$ in two steps.

- If u and v are neighbours in $\mathcal{T}$, then A and B are complementary. From definition of a cross-free decomposition tree, either A or its complement B is a member of $\mathcal{F}$. W.l.o.g. suppose that A is a member of $\mathcal{F}$. Let us examine the quotient $\mathcal{Q}(u)$ w.r.t. u. There, B corresponds to an element of the ground set of $\mathcal{Q}(u)$, and the complement of the singleton $\{B\}$ belongs to $\mathcal{Q}(u)$ since $A = X \setminus B$ belongs to $\mathcal{F}$. Since $\mathcal{Q}(u)$

is simply-linked, $\mathcal{Q}(u)$ can only be complete or linear according to Theorem 2.2. In both cases, there exists an element C of the ground set of $\mathcal{Q}(u)$ for which it holds that $C \subsetneq X \setminus B$ and $\{B, C\}$ belongs to $\mathcal{Q}(u)$ (to see that $C \neq X \setminus B$ recall that the ground set of $\mathcal{Q}(u)$ has at least 3 elements according to Proposition 1.2). One consequence is that $B \cup C$ is a member of $\mathcal{F}$. This member overlaps A, and the difference closure of $\mathcal{F}$ implies $B = (B \cup C) \setminus A$ is a member of $\mathcal{F}$.

- If u and v are not neighbours in $\mathcal{T}$, then A and B overlap (noted $A \otimes B$). From definition of a cross-free decomposition tree, either B or its complement $X \setminus B$ is a member of $\mathcal{F}$. Let us prove that B is a member of $\mathcal{F}$. For this, it suffices to prove that if $X \setminus B$ is a member of $\mathcal{F}$, then so is B. We proceed exactly as before: B corresponds to an element of the ground set of $\mathcal{Q}(u)$, and the complement of the singleton $\{B\}$ belongs to $\mathcal{Q}(u)$ since $X \setminus B$ belongs to $\mathcal{F}$. Since $\mathcal{Q}(u)$ is simply-linked, $\mathcal{Q}(u)$ can only be complete or linear according to Theorem 2.2. In both cases, there exists an element C of the ground set of $\mathcal{Q}(u)$ for which it holds that $C \subsetneq X \setminus B$ and $\{B, C\}$ belongs to $\mathcal{Q}(u)$ (to see that $C \neq X \setminus B$ recall that the ground set of $\mathcal{Q}(u)$ has at least 3 elements according to Proposition 1.2). One consequence is that $B \cup C$ is a member of $\mathcal{F}$. Since $B \cup C$ and $X \setminus B$ overlap, their difference $(B \cup C) \setminus (X \setminus B) = B$ is a member of $\mathcal{F}$ by the difference closure of $\mathcal{F}$. By a similar argument, we can also prove that A is a member of $\mathcal{F}$.

Hence, both A and B are members of $\mathcal{F}$. But then, we can also obtain that their respective complements are members of $\mathcal{F}$: if u and v are neighbours, then it is trivial; otherwise, just notice that $A \otimes B$ and $X \setminus B = A \setminus B$, then use the difference closure. Here again, since $X \setminus B$ is a member of $\mathcal{F}$, all arguments in the last paragraph applies: there exists $C \subsetneq X \setminus B$ such that $B \cup C$ is a member of $\mathcal{F}$. By a similar argument on A and the quotient w.r.t. v, we can also obtain that there exists $D \subsetneq X \setminus A$ such that $A \cup D$ is a member of $\mathcal{F}$. But then $(B \cup C) \cap (A \cup D)$ is a member of $\mathcal{F}$ which crosses both A and $X \setminus A$ (it also crosses both B and its complement). This would mean that the edge between u_{p-1} and v cannot be an edge of the cross-free decomposition tree $\mathcal{T}$. Contradiction. □

Theorem 2.4 *Let $\overrightarrow{\mathcal{T}}$ be the quotiental parent arborescence of a weakly partitive family $\mathcal{F}$. Then, one and only one of the following holds:*

- *$\overrightarrow{\mathcal{T}}$ has one and only one sink: it is then a rooted tree, noted $\widehat{\mathcal{T}}$.*
- *$\overrightarrow{\mathcal{T}}$ has one and only one double-arc, says between nodes u and v. Moreover, when subdividing that double-arc by adding a new node r such that both u and v have an*

outgoing arc to r, one results in an arborescence having r as unique sink. In other words, this is a rooted tree, noted $\widehat{\mathcal{T}}$.

Finally, $\widehat{\mathcal{T}}$ in both cases turns out to be exactly the overlap-free decomposition tree of $\mathcal{F}$.

Proof: The fact that, in both cases, $\widehat{\mathcal{T}}$ is exactly the overlap-free decomposition tree of $\mathcal{F}$ is straightforward from the definition of the various quotiental parents in $\mathcal{T}$. Accordingly, we only show how to obtain the two items in the theorem. From Lemma 2.3, we have two pairwise exclusive configurations. Before continuing we highlight that one of the important facts that is implicit in the following is that the ground set of a quotient always has at least 3 elements (cf. Proposition 1.2).

In one configuration of Lemma 2.3, $\mathcal{T}$ has exactly one internal node u s.t. the quotient $\mathcal{Q}(u)$ w.r.t. u is simply-linked. Let us prove that u is the unique sink of $\overrightarrow{\mathcal{T}}$. Here it suffices to prove that, for every pair of neighbours s, t in $\mathcal{T}$, the nearer node t (w.r.t. u) is the quotiental parent of the further node s (w.r.t. u). By contradiction suppose that there exists a neighbour w of s such that $w \neq t$ and w is the quotiental parent of s. Notice that t might coincide with u but $s \neq u$ and $w \neq u$. Let $(s = u_1, t = u_2, u_3, \ldots, u_p = u)$ be the path linking s to u in $\mathcal{T}$. Before continuing we prove a useful fact. Let Z be the leaf set of the connected component containing u_{p-1} we get when removing the edge between u_{p-1} and u in $\mathcal{T}$. We claim that Z is a member of $\mathcal{F}$. Indeed, by definition of the cross-tree decomposition tree either Z or $\overline{Z}$ is a member of $\mathcal{F}$. Moreover, if $\overline{Z}$ is a member of $\mathcal{F}$ then we can proceed by a similar argument as in the proof of Lemma 2.3 as follows. Z corresponds to an element of the ground set of $\mathcal{Q}(u)$. Since $\overline{Z}$ is a member of $\mathcal{F}$, by Theorem 2.2 $\mathcal{Q}(u)$ is not prime. Besides, all cases lead to the existence of another element W of the ground set of $\mathcal{Q}(u)$ s.t. $W \subsetneq \overline{Z}$ and $\{W, Z\}$ belongs to $\mathcal{Q}(u)$. But then $W \cup Z$ is a member of $\mathcal{F}$ which overlaps the member $\overline{Z}$ of $\mathcal{F}$. By difference closure, $Z = (W \cup Z) \setminus \overline{Z}$ is a member of $\mathcal{F}$. We now achieve the proof of this case by exhibiting a contradiction. Let A be the leaf set of the connected component containing w we get when removing node s from $\mathcal{T}$. By definition of a quotiental parent, A corresponds to the guard of the quotient $\mathcal{Q}(s)$ w.r.t. node s. Let B be the leaf set of the connected component containing s we get when removing the edge between s and t in $\mathcal{T}$. Note that $\overline{B}$ corresponds to an element of the ground set of $\mathcal{Q}(s)$. Clearly B and $\overline{A}$ overlap. We claim that B is a member of $\mathcal{F}$. Indeed, if $t = u$ then $B = Z$ and we have just proved that Z is a member of $\mathcal{F}$. Otherwise, by definition of the cross-tree decomposition tree either B or $\overline{B}$ is a member of $\mathcal{F}$. Moreover, if $\overline{B}$ is a member of $\mathcal{F}$ then it overlaps the member Z of $\mathcal{F}$, and $B = Z \setminus \overline{B}$ will also be a member of $\mathcal{F}$. But then in $\mathcal{Q}(s)$, the membership of B in $\mathcal{F}$ would lead to the existence of a (quasi-trivial) member of $\mathcal{Q}(s)$

which overlaps the complement of the guard A. Contradiction by definition of a guard. We conclude that this case leads to the first item in the theorem.

In the other configuration of Lemma 2.3, every internal node u of $\mathcal{T}$ is such that the quotient w.r.t. u is not simply-linked. The crucial point is the following. Let u and v be two nodes of $\mathcal{T}$ such that v is the quotiental parent of u. Let st be an edge of the connected component containing u when removing the edge uv from $\mathcal{T}$, with t being the nearer node (w.r.t. u) and s being the further node (w.r.t. u). Then, we claim that t is the quotiental parent of s. Indeed, suppose by contradiction that there exists a neighbour w of s such that $w \neq t$ and w is the quotiental parent of s. Notice that t might coincide with u but $s \neq u$ and $w \neq u$. Let A be the leaf set of the connected component containing w we get when removing the edge between s and w in $\mathcal{T}$. By definition of a quotiental parent, A corresponds to the guard of the quotient $\mathcal{Q}(s)$ w.r.t. node s. Let Z be the leaf set of the connected component containing s we get when removing the edge between s and t in $\mathcal{T}$. Note that $\overline{Z}$ corresponds to an element of the ground set of $\mathcal{Q}(s)$. Clearly Z and A overlap. Then Z cannot be a member of $\mathcal{F}$ since this would contradict the fact A is the guard of $\mathcal{Q}(s)$. However, either Z or its complement is a member of $\mathcal{F}$. Besides, let B be the leaf set of the connected component containing u we get when removing the edge between u and v in $\mathcal{T}$. Clearly, B is a member of $\mathcal{F}$ for v is the quotiental parent of u. Now, if $\overline{Z}$ is a member of $\mathcal{F}$ then it would overlap B and $Z = B \setminus \overline{Z}$ would also be a member of $\mathcal{F}$. Contradiction. We conclude that either $\overrightarrow{\mathcal{T}}$ has one sink then the sink is unique and this case leads to the first item of the theorem, or there exist in $\overrightarrow{\mathcal{T}}$ one and only one double-arc between u and v and it is straightforward to check that this case leads to the second item of the theorem. □

Theorem 2.4 is the crucial point. Roughly, it states that if one decomposes a weakly partitive family w.r.t. the cross-free decomposition framework, and labels the quotients according to Theorems 2.2 and 2.3 (with a special "*trivially representable*" label for the first case in the items of Theorem 2.3), then the orientation of arcs given by the quotiental parents leads to exactly the same representation of the family as what has been done in the previous section via the overlap-free approach (with the same special "*trivially representable*" label for overlap-free quotients which have a ground set of 2 elements; see the end of Section 2.2.1 for more concise details).

Turning our attention back to the proof of both Theorems 2.2 and 2.3, let us introduce some minor terminologies.

Definition 2.12 (2-Graph of a Set Family) A family $\mathcal{F} \subseteq 2^X$ can also be seen as an undirected hypergraph with vertex set X. Let us define the 2-*graph of* $\mathcal{F}$ as its restriction to size 2 hyperedges: $G_{\mathcal{F}} = (X, E)$ with $E = \{A \in \mathcal{F} \text{ and } |A| = 2\}$.

We assume that the reader is familiar with the basics of graph theory. However, since the following terminologies could be different in some literature, we give here the definition we use in this composition: a stable is $G = (X, \emptyset)$, a clique is the complement of a stable, a cycle is a 2-regular connected graph, a path is a cycle minus one and only one edge. The size, and also the length, of a cycle is its number of vertices. So is the *size* of a path. The length of a path is its number of edges. The P_n path denotes the path of *size* n. For instance, P_4 denotes the four vertex path and P_3 has length 2 (two).

The following two Lemmas 2.4 and 2.5 borrow heavily on a property given in [47]. For the following lemma, the case of simply-linked quotients figured already in that reference. Our context of cross-free decomposition trees introduces a second case to the statement of the property. However, for the proof, the extended case follows trivially from the ideas of the proof of the first case.

Lemma 2.4 *([47]) Let $\mathcal{F} \subseteq 2^X$ be both weakly partitive and quotient. If moreover $\mathcal{F}$ is simply-linked then either its 2-graph is connected, or $\mathcal{F}$ is the family of trivial subsets of X. If $\mathcal{F}$ is not simply-linked then, for $x \in X$ being its guard, $\mathcal{G} = \mathcal{F} \setminus \{X, \{x\}\}$, and $Y = X \setminus \{x\}$, either the 2-graph of $\mathcal{G}$ is connected, or $\mathcal{G}$ is the family of trivial subsets of Y.*

Proof: We extend the ideas given in [47]. Suppose that $\mathcal{F}$ is simply-linked and does not contain only the trivial subsets of X. The crucial point is, for every non-trivial member of $\mathcal{F}$, there exists another member of $\mathcal{F}$ such that the two *overlap* (if the first member was quasi-trivial then the simply-linked property provide such a second member, if the first member is not quasi-trivial then the quotient property can be used). We first prove that $G_{\mathcal{F}}$ has an edge. Let A be a minimal member of $\mathcal{F}$ among the non-trivial members of $\mathcal{F}$. As explained in the above, there exists another member B of $\mathcal{F}$ such that A and B overlap. If $|A| = 2$, we are done. Otherwise by the corresponding closure, either $A \setminus B$ or $A \cap B$ is a non-trivial member of $\mathcal{F}$ included in A. This contradicts the minimality of A. Hence, $G_{\mathcal{F}}$ has an edge. To continue, let Z be the connected component of $G_{\mathcal{F}}$ which contains that edge: $|Z| \geq 2$. We now want to prove that $Z = X$. Suppose this is not the case. Z, which is a member of $\mathcal{F}$ by the union closure, is then a non-trivial member of $\mathcal{F}$. Therefore, there are members in $\mathcal{F}$ which overlap Z. Let C be a minimal member of $\mathcal{F}$ among those of $\mathcal{F}$ which overlap Z. Firstly, $|C \cap Z| = 1$ otherwise by the difference closure between C and some well selected edge in Z there would be a contradiction to the minimality of C. Secondly, suppose that $D = C \setminus Z$ is such that $|D| > 1$. Notice that $D \neq X$ since Z is not empty. Then, D is a non-trivial member of $\mathcal{F}$ (membership by the difference closure). Therefore, there exists a member E of $\mathcal{F}$ such that D and E overlap.

This clearly implies overlapping between C and E. But then either $C \setminus E$ or $C \cap E$ will contradict the minimality of C. Hence $|D| = 1$, or in other words $|C| = 2$. But then, Z is no more a connected component in $G_{\mathcal{F}}$. We conclude that $Z = X$, namely $G_{\mathcal{F}}$ is connected.

Suppose that $\mathcal{F}$ is not simply-linked and $\mathcal{G} = \mathcal{F} \setminus \{X, \{x\}\}$ does not contain only the trivial subsets of $Y = X \setminus \{x\}$, where $x \in X$ is the guard of $\mathcal{F}$. The crucial point is, for every non-trivial member of $\mathcal{G}$ (w.r.t. its ground set Y), there exists another member of $\mathcal{G}$ such that the two *overlap* (it is provided by the quotient property of $\mathcal{F}$). The remaining proceeding is like before. □

Before continuing we would like to highlight that, though the following property was discovered for partitive families, its proof mainly requires the union and difference closures. The lemma can be found in [47].

Lemma 2.5 *([47]) Let $\mathcal{F} \subseteq 2^X$ be a weakly partitive family. If its 2-graph $G_{\mathcal{F}}$ is connected then $G_{\mathcal{F}}$ is either a clique or a path of size at least 3. Moreover, if $\mathcal{F}$ is partitive, then $G_{\mathcal{F}}$ is not a path of size ≥ 3.*

Proof: The proof given in [47] is as follows. If $|X| = 3$ then the lemma is trivial. Otherwise, suppose that $G_{\mathcal{F}}$ has a vertex x with degree at least 3, and let y, z, t be three distinct neighbours of x. In other words, $\{x, y\}$ and $\{x, z\}$ are members of $\mathcal{F}$, and so is $\{x, y, z\}$ by the union closure. But $\{x, t\}$ is also a member of $\mathcal{F}$. By the difference closure we deduce that $\{y, z\}$ is an edge of $G_{\mathcal{F}}$. Likewise, we can deduce that x, y, z, and t form a clique in $G_{\mathcal{F}}$. Now, let v be a vertex that is connected to the previous clique at some point, say t. Then, by a similar argument on the fact that t is of degree at least 3, we can show that v is connected to all other vertices of the clique. Thus the previous clique plus vertex v form a bigger clique, and so on. The connectivity of $G_{\mathcal{F}}$ then can be used to conclude that the whole graph $G_{\mathcal{F}}$ is a clique. Also, the only connected graphs of degree at most 2 are paths and cycles.

We now have to use the intersection closure to forbid cycles. Indeed, if $G_{\mathcal{F}}$ is a cycle $(x_1, x_2, \ldots, x_n)$ of size at least $n \geq 4$, then $\{x_1, x_2, x_3\}$ and $\{x_3, x_4, \ldots, x_n, x_1\}$ are overlapping members of $\mathcal{F}$. Therefore, the closure under intersection implies that $\{x_1, x_3\}$ is a member of $\mathcal{F}$, which in other words enforces an edge in $G_{\mathcal{F}}$ between x_1 and x_3. But then $G_{\mathcal{F}}$ is no more a cycle.

Finally, suppose that $\mathcal{F}$ is partitive and $G_{\mathcal{F}}$ a path $(x_1, x_2, \ldots, x_n)$ of size $n \geq 3$. Here, $\{x_1, x_2\}$ and $\{x_2, x_3\}$ are overlapping members of $\mathcal{F}$. Then, by the symmetric difference closure, we obtain an edge in $G_{\mathcal{F}}$ between x_1 and x_3. Contradiction. □

Remark 2.2 *Let $\mathcal{F}$ be a weakly partitive family, and $G_{\mathcal{F}}$ its 2-graph. Then*

- *$G_{\mathcal{F}}$ is a clique if and only if $\mathcal{F}$ is complete.*
- *$G_{\mathcal{F}}$ is a path of size at least 3 if and only if $\mathcal{F}$ is linear.*

The remark is folklore. The only potentially tricky part is to prove the left-to-right implication in the second item. Actually, if $G_{\mathcal{F}}$ is the path $(x_1, x_2, \ldots, x_n)$ of size $n \geq 3$, then it is clear that any intervals of the ordering $(x_1, x_2, \ldots, x_n)$ is a member of $\mathcal{F}$ by the union closure. Suppose that $\mathcal{F}$ had a member A that is not an interval of this ordering. Then, there exist x_i and x_j with $|j - i| \geq 2$ such that $x_i, x_j \in A$ and $x_k \notin A$ for all $i < k < j$. Thus, A and $\{x_i, x_{i+1}, \ldots, x_j\}$ are overlapping members of $\mathcal{F}$. Their intersection, which is $\{x_i, x_j\}$, has to belong to $\mathcal{F}$. But then $G_{\mathcal{F}}$ is no more a path.

2.2.3 Symmetric Crossing and Bipartitive Families

Bipartitive families are particular instances of crossing families. Their relaxation, so-called weakly bipartitive families, is still a particular case of crossing families, and also appears under the name of symmetric crossing families. The notion was probably first introduced in [39] under the name of "decomposition frame with the intersection and transitivity properties". Later, the work of [39] are well formalized in [94, 103] under the name of bipartitivity. The same notion appears also in [75] under the name of "unrooted set families".

From the references of [54], symmetric crossing families were also studied in [45] (in Russian), which leads to the notion of a so-called *cactus (hyper-)tree*. Similarly as what will be presented in this section, a cactus hypertree is also a linear size representation of the input symmetric crossing family [45, cited in [54]]. However, note that we do not address cactus hypertrees in this thesis at all.

Definition 2.13 (Symmetric Crossing Family and Bipartitive Family) A proper and connected family is *symmetric crossing* if it is a crossing family and if it is closed under the complementation. Moreover, the family is *bipartitive* if it is also closed under the symmetric difference of its crossing members. A symmetric crossing family is also called a *weakly bipartitive* family. By abuse of terminology, we still keep the convention of excluding the empty set from all those families, though the complementation closure on the ground set should clearly imply the membership of the empty set.

Given the result on partitive families via the overlap-free decomposition tree, it could be interesting to investigate symmetric crossing families by the same approach. However, the following observation, though simple, formally states that such a practice would fail.

Proposition 2.4 *A (proper and connected) set family which is closed under complementation is also overlap-free quotient.*

Proof: Let A be a non trivial member of a symmetric crossing family $\mathcal{F} \subseteq 2^X$. Let $x \in A$. Since $\{x\}$ is a member of $\mathcal{F}$, so is its complement $Y = X \setminus \{x\}$. Then, A and Y overlap. □

Corollary 2.4 *The overlap-free decomposition tree of a symmetric crossing family is in bijection with the trivial members of the family.*

Basically, the overlap-free decomposition tree of a symmetric crossing family is a star: one root and all leaves, with no internal nodes other than the root. Moreover, the only overlap-free quotient of the tree (w.r.t. the unique internal node) is exactly equal to the initial family itself! Accordingly, the overlap-free decomposition approach which was explained in Section 2.2.1 does not provide any help in solving the representation problem of symmetric crossing families. This being said, symmetric crossing families are closely related to weakly partitive families by the now quite folklore fact that

Proposition 2.5 *Let $\mathcal{F} \subseteq 2^X$ be a symmetric crossing (resp. bipartitive) family such that $|X| \geq 4$. Let $x \in X$ be any element of the ground set. Then, the family $\mathcal{F}'$ of all members of $\mathcal{F}$ excluding x is a weakly partitive (resp. partitive) family.*

Accordingly, those families are simply structured by the following theorem on their cross-free quotients.

Theorem 2.5 (Cunningham [39]) *If a symmetric crossing family $\mathcal{F} \subseteq 2^X$ satisfies the (cross-free) quotient property, then one and only one of the following holds:*

- *$\mathcal{F}$ consists of only the trivial subsets of X and possibly some of their complements (we still say that $\mathcal{F}$ is* prime*, by abuse in the terminology),*
- *$\mathcal{F} = 2^X \setminus \{\emptyset\}$ ($\mathcal{F}$ is* complete*),*
- *$|X| \geq 4$ and there is a circular ordering of the elements of X such that $\mathcal{F}$ is the family of all circular intervals of this ordering (we say that $\mathcal{F}$ is* circular*).*

Moreover, if $\mathcal{F}$ is bipartitive, the last case cannot occur.

Proof: There are many ways to prove this claim. The laziest one is probably as follows. If the ground set of $\mathcal{F}$ has three elements, then the family is complete. Otherwise, take any element x therein. Define $\mathcal{F}'$ as the family of all members of $\mathcal{F}$ excluding x. Check that

- Proposition 2.5 applies on $\mathcal{F}'$, among other facts, $\mathcal{F}'$ is proper and connected, and the ground set of $\mathcal{F}'$ has at least three elements.
- Moreover, if $\mathcal{F}$ is quotient, then so is $\mathcal{F}'$.

Apply Chein-Habib-Maurer Theorem 2.1 on $\mathcal{F}'$. Finally, we can conclude each case of the claim using the fact that every member of $\mathcal{F}$: either is exactly the ground set; or belongs to $\mathcal{F}'$; or is the complement of some member of $\mathcal{F}'$. □

The new quotient type consists of the circular nodes, which are slightly different from linear ones. However, as a circular ordering can also be encoded as a linear ordering, we code circular nodes according to the encoding of linear ones (see previous Section 2.2.1), and conclude that

Corollary 2.5 *The space complexity of a symmetric crossing family over X is in $O(|X|)$.*

2.3 Applications in Graph Theory

This section exemplifies the usefulness of the representation paradigm in the area of graph decomposition.

2.3.1 Modular Decomposition and Clan Decomposition

We start with probably the best known application of the representation of set families: modular decomposition. Indeed, this is now a well-studied notion in graph theory [47, 62, 92]. As having been rediscovered in other fields, the notion appears also under the various names of autonomous sets, homogeneous sets, externally related sets, clans, partitive sets, and also intervals.

Though it has a quite storied origin, the last terminology of an *interval* is rather inappropriate for its purpose nowadays. An attempt to explain the situation could be as follows. Roughly, the biggest motivation for such a terminology comes from the fact the closure axioms in the definition of the notion refer to those of an interval of the real line: if A and B are intervals, then so are their intersection $A \cap B$ and differences $A \setminus B$ and $B \setminus A$; moreover, if the intersection of A and B is not empty, then $A \cup B$ is also an interval. Then, calling a graph module an interval would have the merit to generalize intervals of $\mathbb{R}$ to those of a (directed) graph. However, when applied to an empty digraph, or more generally to a symmetric digraph (that is, an undirected graph), if both A and B are graph modules, and if moreover they have a non-empty intersection, then their symmetric difference $A \Delta B$ will also be a graph module. Then, the fact that the

symmetric difference of two intersecting intervals of the real line is by no means an interval could be an explanation why the use of this terminology is more and more marginal among graph theorists.

Turning our attention back to decomposition purposes, let us highlight the following aspects around the notion of a quotient. Firstly, when translated to graphs, the notion of an overlap-free quotient (cf. Definition 2.6) consists roughly in "shrinking" a vertex subset of the graph into one single vertex. The "shrinking" operation is so-called the *quotient operation* in the literature related to the topic. Then, its reverse action can be seen as the injection of a second graph to some vertex of the first graph. This is denoted by the *substitution operation*. After this, the so-called *modular decomposition tree* can be seen as a construction of the input graph from the one vertex graph through successive substitution operations. Conversely, it is also a division scheme of the input graph into terminal cases (indecomposable graphs) through quotient operations. For a side note, modular decomposition has also been known under the name of *decomposition by substitution* for a quite long time.

It turns out that many graph problems *"go through quotient"*, namely there is a way to combine the solutions of some sub-instances to solve an instance that is obtained through a *substitution* operation over the latter sub-instances. If so, to solve the problem on some graph, it suffices to solve the same problem on the *quotients* of the modular decomposition tree of that graph. The list of such problems includes: transitive orientation, maximum clique, maximum independent set, colouring, minimum dominating set, longest induced path, feedback vertex set, and number of minimal separators (cf. [92, 95] for short versions, however, [103] gives a better introduction and discussion). In other words, in order to solve those problems, modular decomposition fits into the academic divide-and-conquer algorithmic philosophy. It is important to notice that, when translated to the graph, there are only three kinds of quotients: the cliques, the stables, and the indecomposable graphs. Also, it is an important fact that a modular decomposition tree can always be computed in linear time on the size of the input graph [19, 25, 37, 43, 65, 87, 89, 108].

Accordingly, in order to solve some graph problem, it does not "cost" anything to modular decomposing the input graph before the effective search for a solution. Thus, modular decomposition becomes a very well studied, very useful decomposition of graphs, and is used among other things as the first algorithmic step for many problems including recognition, decision, and optimization. Its direct applications include tractable constraint satisfaction problems [28], computational biology [59], graph clustering for, e.g., network analysis, and graph drawing. This rich research field relies heavily on the nice combinatorial properties of modules of a graph. Among most important ones, that modules form

a partitive family allows representing them compactly with a tree, as will be formalized afterwards in Corollary 2.3. Let us first reformulate the definition of a graph from the scope of its generalization to 2-structures. Also, in order to keep a homogeneity in the notations occurring in this part of the thesis, the vertex set of a graph will be denoted by X and not by the commonly used V.

Definition 2.14 (2-Structure) A 2-*structure* $G = (X, C)$ is a vertex set X along with a colour function C mapping every pair (x, y) (with $x, y \in X$ and $x \neq y$) to some value in $\mathbb{N}$. The 2-structure is *symmetric* if $C(x, y) = C(y, x)$ for every pair of vertices x and y in X. Moreover, if the colour function C has only 2 values or fewer, then the 2-structure is called a *directed graph* (the two values denote the existence and the non-existence of an arc). Finally, a symmetric directed graph is called an *undirected graph*, or simply a *graph*.

Definition 2.15 (Module) A *module* M of a symmetric 2-structure $G = (X, C)$ is a non-empty vertex subset (i.e. $M \subseteq X$) such that for all $x, y \in M$ and $s \notin M$, the colour function satisfies $C(s, x) = C(s, y)$.

The probably first application of modules was a solution to the transitive orientation problem (given an undirected graph, if exists, find an edge orientation which follows the transitive law: the existence of arcs (x, y) and (y, z) implies that of arc (x, z)). Later on, the notion has been shown to be fundamental for an increasing number of combinatorial problems. For instance, twins and anti-twins in a graph are exactly the modules of size 2 of the graph. Then, for many optimization problems, contracting twins and anti-twins as pre-processing is a common approach to simplify the problem. This practice is convenient for computational purposes since twins and anti-twins can be found easily. To give another example, all common intervals of two permutations are also modules of the permutation graph associated to the two permutations. On the other hand, all overlap-free modules of this graph are overlap-free common intervals of the two permutations. (Section 5.1 in the next part of the thesis will deepen those matters.) Also, in graph drawing, a module can be contracted and drawn as one single vertex without any information loss on the incident edges, since they all are same. The following folklore fact is probably the most important and seminal property of modules.

Proposition 2.6 *The family of modules of a symmetric* 2*-structure is partitive. In particular, the property holds when the* 2*-structure is an undirected graph.*

Proof: Let $G = (X, C)$ be a symmetric 2-structure. By definition, the empty set is not a module while every trivial subset of X is a module of G. It is also straightforward that

the modules of G form an intersecting family. Let A and B be two modules of G which overlap. The only remaining thing to prove is that both $Y = A \setminus B$ and $Z = A \Delta B$ are modules of G.

First suppose that Y is not a module and let $x, y \in Y$ and $s \notin Y$ be such that $C(s, x) \neq C(s, y)$. That A is a module implies $s \in A \cap B$. That A and B overlap provides us with a vertex t belonging to $B \setminus A$. That B is a module implies $C(x, s) = C(x, t)$ and $C(y, s) = C(y, t)$. Now, G is a symmetric 2-structure, leading to $C(s, x) = C(t, x)$ and $C(s, y) = C(t, y)$. However, A is a module: $C(t, x) = C(t, y)$. Then, $C(s, x) = C(s, y)$. Contradiction.

Let us now suppose that Z is not a module. Let $x, y \in Z$ and $s \notin Z$ be such that $C(s, x) \neq C(s, y)$. From the last paragraph, we can suppose without loss of generality that $x \in A \setminus B$, $y \in B \setminus A$, and $s \in A \cap B$. Both A and B are modules: $C(x, s) = C(x, y)$ and $C(y, x) = C(y, s)$. G is symmetric: $C(x, y) = C(y, x)$. Hence $C(x, s) = C(y, s)$. But then, $C(s, x) = C(s, y)$ (still) by symmetry of G. Contradiction. □

Consequently, one can map every symmetric 2-structure G to a unique tree, namely the decomposition tree of the family of modules of G. In fact, this tree defines the modular decomposition tree of G, that is

Definition 2.16 (Modular Decomposition Tree) The *modular decomposition tree* of a symmetric 2-structure is the decomposition tree of the family of its modules, which is partitive. We mostly consider this tree as the overlap-free decomposition tree (cf. Sections 2.2.1 and 2.2.2 for further details).

Moreover, the symmetry of a 2-structure implies also the following property, which gives further information on its modular decomposition tree.

Proposition 2.7 *The family of modules of a symmetric 2-structure $G = (X, C)$ is complete if and only if the colour function C is constant: $C(x, y) = C(z, t)$ for all $x, y, z, t \in X$.*

Proof: It is clear that the condition is sufficient. Let us prove its necessity. Indeed, the completeness provides us with, among other things, two modules: $\{x, y, t\}$ and $\{y, z, t\}$. This includes two facts: $C(z, x) = C(z, y) = C(z, t)$ and $C(x, y) = C(x, z) = C(x, t)$. Now, G is symmetric, and $C(z, x) = C(x, z)$. Hence, $C(x, y) = C(z, t)$. □

In the following remark, the modular decomposition theorem was first stated by T. Gallai under the framework of undirected graphs. Its generalization to symmetric 2-structure is absolutely straightforward.

Remark 2.3 (On Gallai Modular Decomposition Theorem [62])
There is a unique labelled tree associated to a symmetric 2-structure, so-called modular decomposition tree, such that all modules of the 2-structure can be enumerated by the tree without the knowledge of the 2-structure. Moreover, the size of this tree is proportional to the number of vertices of the 2-structure, while its labels are of only 2 types, so-called complete and prime. Finally, if k is the number of values of the colour function of the 2-structure, then the modular decomposition tree can have a further labelling, which,

- *for a complete node, consists of one among k (arbitrarily given) constants,*
- *for a prime node, consists of a symmetric 2-structure, and,*

from which labelled tree the initial 2-structure can be rebuilt.

Proof: Mostly follows from Propositions 2.6 and 2.7. Let $G = (X, C)$ denote the initial 2-structure. For a prime node n with the associated quotient partition $\{X_1, X_2, \ldots, X_k\}$ of the vertex set X, the 2-structure used in the further labelling is the sub 2-structure $G[Q] = (Q, C_{|Q})$ of G that is induced by the vertex subset $Q = \{x_1, x_2, \ldots, x_k\}$, where each x_i is a representative belonging to the vertex subset X_i, and where $C_{|Q}$ refers to the restriction of the function C on the domain Q. □

There is a classical generalization of modules of a symmetric 2-structure to those of a not necessarily symmetric 2-structure, under the name of a clan. It is somewhat an equivalent of modules in the directed case, *from a structural point of view.* Indeed, from a more intrinsic point of view, we would prefer the subsequent notion of a genuine-module.

Definition 2.17 (Clan) A *clan* of a 2-structure $G = (X, C)$ is a non-empty vertex subset $M \subseteq X$ such that for all $x, y \in M$ and $s \notin M$, the colour function satisfies the *two-way-condition* $C(s, x) = C(s, y)$ and $C(x, s) = C(y, s)$. Moreover, when the 2-structure is a directed graph (meaning when the colour function has 2 values or fewer), the clans are also called *modules**.

According to the standard notation of the neighbourhood in graph theory, let us define the out-going neighbourhood $N_c^+(x)$, for every vertex $x \in X$ and every colour $c \in \mathbb{N}$, as the set of all vertices y such that $C(x, y) = c$. Likewise, we define the in-coming neighbourhood $N_c^-(x)$ as the set of all vertices y such that $C(y, x) = c$. Moreover, if

*For the sake of history, we keep this terminology of "module of a directed graph", although it is quite little apropos. A better candidate would be the genuine-module which will be presented further in Section 2.3.2.

the 2-structure is symmetric, we define the neighbourhood as $N_c(x) = N_c^+(x) = N_c^-(x)$. Then, an equivalent definition of a module of a symmetric 2-structure is

$$M \text{ is a module if for all } x, y \in M \text{ and for all } c \in \mathbb{N},\ N_c(x) \setminus M = N_c(y) \setminus M.$$

From this point of view, the definition of a clan can be seen as a *double* module-like condition:

$$M \text{ is a clan if for all } x, y \in M \text{ and for all } c \in \mathbb{N}, \begin{cases} N_c^+(x) \setminus M = N_c^+(y) \setminus M \\ N_c^-(x) \setminus M = N_c^-(y) \setminus M \end{cases}.$$

Actually, the definition of a clan has the important merit to organize the clans of a 2-structure into a weakly partitive family, and hence, to result in a decomposition theorem.

Proposition 2.8 *The family of clans of a 2-structure is weakly partitive.*

Proof: The proof is very similar to that of Proposition 2.6. □

After this, almost all – and actually all – the previous discourse about modular decomposition of symmetric 2-structures holds for arbitrary 2-structures after some slight modifications.

Remark 2.4 (On Ehrenfeutch-Rozenberg Clan Decomposition Theorem [48]) *There is a clan decomposition for 2-structures. It is also called modular decomposition when the 2-structure is a digraph.*

We will not give the proof for Ehrenfeutch-Rozenberg theorem, nor discuss more about clan decomposition. The purpose here is not to deepen every topic related to graph decomposition. Clan decomposition is very similar to modular decomposition of symmetric 2-structures. We would suggest the reader with further interests in clan decomposition to refer to [47] for more details.

The discussion right below can be seen as a side note on modular decomposition, from a partitive families's point of view. The class of cographs can probably be seen as the most studied class of graphs in the framework of modular decomposition. This class owns many intrinsic properties. For instance, it is the class of graphs excluding the four vertex path P_4 as induced subgraph. As well, it is the smallest class of graph which contains the one vertex graph and which is closed under the complementation and the disjoint union. Almost all the familiar NP-complete problems are polynomially – and often linearly –

solvable when restricting to this class. Simple examples could be: maximum independent set or clique, chromatic number, and Hamiltonian path. Also, the graph isomorphism problem is linearly solvable for cographs.

Here, an important notion bound to cographs which implies many of the nice computational linearity is probably the cotree. Let us revise this notion, from a 2-structures' point of view. We exhibit a straightforward generalization of the representation theorem for cographs.

Definition 2.18 (Co-structure, Cograph, and Cotree) A symmetric 2-structure is a *co-structure* if there are no prime nodes in its modular decomposition tree. Moreover, if the 2-structure is an undirected graph, then it is called *cograph*. Finally, a *cotree* is the modular decomposition tree of a co-structure, labelled with respect to the labelling given in Corollary 2.3.

Corollary 2.6 (Cotree Theorem) *The cotree gives a representation of a co-structure, using linear space on the size of its vertex set. In particular, this holds when the co-structure is a cograph.*

Proof: The linearity follows as there are no prime nodes, and the labels used for complete nodes are of constant size. □

Basically, the cotree is a representation of the co-structure using a sub-linear encoding, since the number of edges of a co-structure can be quadratic on the number of its vertices. Moreover, this still holds for the particular case when the co-structure is a cograph. In algorithmic graph theory, those facts are important for, among other things, designing fast algorithms on cographs.

Let us come back to the discussion on modular decomposition in general. As previously said, modular decomposition can be used to solve graph problems via the divide-and-conquer paradigm. However, the main drawback of this approach comes from the terminal cases, namely the indecomposable graphs (or prime nodes of the decomposition tree). Indeed, when using modular decomposition to solve a problem, one needs to be able to solve the problem on terminal cases of the decomposition. Accordingly, it is sensible to search for relaxations of modular decomposition in order to further decompose.

Besides, in the research field of social networks, several vertex partitioning methods have been introduced in order to catch the idea of putting together all vertices having a similar behaviour, in other words finding regularities [113]. Modular decomposition provides such a partitioning, yet seemingly too restrictive for real life applications. The concept of a role [50] on the other hand seems promising, however its computation is

unfortunately NP-hard [52]. Here, there is need for the search of not only relaxed, but also tractable, variations of the modular decomposition scheme. In the following, as well as in the last section of the next chapter, we present a range of such relaxations. The corresponding decomposition schemes are more powerful than the classical modular decomposition scheme in the sense that the corresponding indecomposable graphs form a strict subclass of the classical indecomposable graphs w.r.t. modular decomposition. In other words, one will be able to decompose a graph into smaller pieces, making the terminal cases more restricted, hence potentially easier to handle.

2.3.2 Genuine-Modules and Unordered-Modules

There is a broader generalization of modules of a symmetric 2-structure. We will present this notion under the name of a *genuine-module*, given that the terminology of "module of a directed graph" has already been reserved for (actually) its clans. Before this composition, we have presented the same notion under the name of a *homogeneous module*, following the study then on the so-called *homogeneous relations* [16].

Definition 2.19 (Genuine-Module) A *genuine-module* M of a 2-structure $G = (X, C)$ is a non-empty vertex subset such that for all $x, y \in M$ and $s \notin M$, the colour function satisfies the *one-way-condition* $C(s, x) = C(s, y)$. An equivalent condition could be: for all vertices $x, y \in M$ and for every colour $c \in \mathbb{N}$, we have $N_c^-(x) \setminus M = N_c^-(y) \setminus M$.

Clearly, a clan is always a genuine-module, but the converse does not necessarily hold. Besides, it follows directly from the definition that the genuine-modules of an undirected graph or a symmetric 2-structure are exactly their modules, and also their clans. As for their representation, the following property (which is also straightforward from definition) shows that the genuine-modules of a 2-structure form an intersecting family. Then, we can of course use the representation result of intersecting families to show a quadratic space representation for the genuine-modules of a given 2-structure. Notice that this is not much a space improvement in representing the genuine-modules since the 2-structure itself is already an encoding using equivalent asymptotic space. Notwithstanding, the representation obtained via intersecting families could be instructive, as it focuses on the structure of the genuine-modules themselves.

Proposition 2.9 *The genuine-modules of a 2-structure form an intersecting family.*

We now describe an application of crossing families. We will use a newly introduced notion [18], so-called *unordered-module*, or *umodule* to be short. It has high connections with the notion of a bi-join (see Section 2.3.3). Also, we will restrict the discussion to the

case of digraphs, which here refer to loopless simple directed graphs where 2-cycles are allowed. We address them using both standard graph theory's and 2-structure theory's terminologies.

Definition 2.20 (Umodule) A *umodule* M of a digraph G over a vertex set X is a non-empty vertex subset such that the partition of $X \setminus M$ into $\{N^+(x) \setminus M,\ \overline{N^+(x)} \setminus M\}$ is the same for every vertex $x \in M$, where $\overline{N^+(x)}$ is the complement of the out-going neighbourhood of x. An equivalent condition could be: for all vertices $x, y \in M$ and for every colour $c \in \mathbb{N}$, there exists a colour $d \in \mathbb{N}$ such that we have $N_c^+(x) \setminus M = N_d^+(y) \setminus M$.

For digraphs and non-symmetric 2-structures, we can also define the genuine-module using a similar approach, namely with the partition of the exterior w.r.t. the colour of the corresponding arc. The only difference between the two notions is that the order from which the parts of the partition come is irrelevant in the definition of a umodule. On the other hand, this order is important in the definition of a genuine-module. This is the reason for the terminology of unordered-modules.

Proposition 2.10 *The umodules of a digraph form a crossing family.*

Proof: The closure under the intersection is the less obvious part. Let G be a digraph over a vertex set X. Let A and B be two umodules of G which cross. Let us prove that $C = A \cap B$ is a umodule of G. For this, if C is reduced to a singleton then we are done by vacuity. Otherwise let x and y be two vertices of C. Since A and B cross there exists an exterior vertex $z \notin A \cup B$. A proof can be obtained by exhaustive checking on the arcs (x, z) and (y, z) as follows. If both of them exist in G, then, by umodule definition of A and B, we have both $N^+(x) \setminus A = N^+(y) \setminus A$ and $N^+(x) \setminus B = N^+(y) \setminus B$. This implies $N^+(x) \setminus C = N^+(y) \setminus C$. If neither of them exist in G, then the situation is similar. If one of them exists in G and the other does not, then we have both $N^+(x) \setminus A = \overline{N^+(y)} \setminus A$ and $N^+(x) \setminus B = \overline{N^+(y)} \setminus B$. This implies $N^+(x) \setminus C = \overline{N^+(y)} \setminus C$. □

2.3.3 Split Decomposition, Bijoin Decomposition, and Decomposition of Symmetric Submodular Functions

We close the chapter with the quite fruitful field of bipartitivity. To begin with, the split decomposition defined by W. Cunningham and J. Edmonds [39, 41] can be seen among other things as an application of bipartitivity. Later, W. Cunningham gave a decomposition scheme for symmetric submodular functions [40], which can be seen as an application of symmetric crossing families. The recent development of bipartitivity is mainly for the study of bijoins of a graph [93, 94, 103]. We here recall all these notions,

starting with the simplest case in the list (proof-wise!). Before continuing, let us recall the definition of a symmetric submodular function.

Definition 2.21 (Submodular and Symmetric Submodular Function) A set function $f : 2^X \to \mathbb{R}$ is *submodular* if for all subsets $A \subseteq X$ and $B \subseteq X$, f holds the submodular inequality:

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B).$$

The function is *symmetric* if for every subset $A \subseteq X$, f has the same value on A as on its complement $X \setminus A$. In some literature, a symmetric and submodular function is also called a *function of connectivity*.

Proposition 2.11 *Both the empty set and the universal set are trivial minimizers of a symmetric submodular function. Moreover, the family of non-trivial minimizers of the function is weakly bipartitive.*

Proof: It is quite straightforward from definition. The submodular inequality on a subset and its complement implies the two trivial minimizers. As for the family of non-trivial minimizers of the function, the symmetry of the function gives the closure under the complementation. Then, the submodular inequality gives the closure under the intersection and union of crossing members. □

Let us continue with the bijoins. In the following, we revert to standard graph theory terminologies and abandon 2-structure theory terminologies.

Definition 2.22 (Bijoin) A vertex subset $M \subseteq X$ of an undirected graph $G = (X, E)$ is a *bijoin* if there are $A \subseteq M$ and $B \subseteq \overline{M}$ such that the edges between M and $\overline{M}$ are exactly those forming the complete bipartite graph between A and B plus those forming the complete bipartite graph between $M \setminus A$ and $\overline{M} \setminus B$.

Proposition 2.12 *The family of bijoins of a graph is bipartitive.*

Proof: The closure under the complementation follows directly from definition. The closure under the union of crossing members follows from the following characterization. M is a bijoin if and only if, for all $x, y \in M$ and $s, t \notin M$, the adjacencies of x, s and x, t differ if and only if the adjacencies of y, s and y, t differ. Then, using the two closures, we deduce the closure under intersection and difference of crossing members. Finally, we need to prove the closure under the symmetric difference of crossing members. Let $G = (X, E)$ be a graph. Let A and B be two bijoins of G which cross. We would like to

prove that $C = A\Delta B$ is a bijoin of G. For this, we still use the above characterization. Let $x, y \in C$ and $s, t \notin C$. If both x and y belong to A then we can conclude by the fact $A \setminus B$ is a bijoin. By symmetry, the only case left is w.l.o.g. $x \in A$ and $y \in B$. Here, if neither of s and t belong to $A \cup B$, then it is straightforward to conclude for $A \cup B$ is a bijoin. If both s and t belong to $A \cap B$, then we conclude likewise using that $\overline{A \cap B}$ is a bijoin. By symmetry, we can suppose w.l.o.g. $s \in A \cap B$ and $t \notin A \cup B$. Then, a simple case analysis allows to conclude. □

Finally, we end the list with the probably first application of bipartitivity.

Definition 2.23 (Split) A vertex subset $M \subseteq X$ of an undirected graph $G = (X, E)$ is a *split* if there are $A \subseteq M$ and $B \subseteq \overline{M}$ such that the edges between M and $\overline{M}$ are exactly those forming the complete bipartite graph between A and B.

Proposition 2.13 *The family of splits of a connected graph is bipartitive.*

Proof: The closure under the complementation is clear. However, unlike bijoins, splits do not own a nice characterization. The only proof to our knowledge for the closure under the intersection of crossing members makes use of a quite long case analysis. It uses among other things the connectivity of the graph for some forbidden configurations. For the sake of smoothness in the reading, the case analysis is postponed to the end of this chapter, in Lemma 2.6. The same case analysis can be done for the closure under the symmetric difference of crossing members. □

We now conclude the section with a unified statement, which follows as corollary of Propositions 2.11, 2.12, and 2.13.

Remark 2.5 *There are*

- *a decomposition of symmetric submodular functions,*
- *a bijoin decomposition of graphs, and*
- *a split decomposition of connected graphs.*

Here again, the purpose is not to discuss in detail about decomposition schemes. We refer readers with further interests to [40] for the first decomposition scheme, [93, 94, 103] for the second, and [39, 41] for the last one. Before opening the next chapter, we give the promised lemma for the bipartitivity of splits.

Lemma 2.6 *The family of splits of a connected graph is closed under the intersection of its crossing members.*

Proof: Let A and Z be two crossing splits of a given connected graph $G = (X, E)$. By definition, there is a partition of A into $\{B, C\}$ and its complement into $\{D, E\}$ such that the matrix adjacency between (B, C) and (D, E) is made of one 1-block and three 0-blocks. W.l.o.g. the 1-block is between B and D. The same holds for Z: there is a partition of Z into $\{U, V\}$ and its complement into $\{S, T\}$ such that the only 1's form an 1-block, which is between U and S. In the following, we denote any intersection $A \cap Z$ by AZ. We would like to investigate the adjacency matrix between AZ and its complement. For this purpose, we will use the following pairwise exclusive configurations.

- If BU is not empty, then both DT and ES are empty.
- If one among DT and ES is not empty, then BU is empty.
- Beside this, there are three similar couples of statements, by circular replacement of BU by respectively BS, DS, and DU.

Moreover, we will also use a second kind of pairwise exclusive configurations.

- If BV is not empty, then DT is empty.
- If DT is not empty, then BV is empty.
- Beside this, there are three similar couples of statements, by circular replacement of BV by respectively CU, CS, and BT.

Finally, we will also use the connectivity of G in the following forbidden configurations.

- One among CU, BU, and BV is not empty (otherwise either CV is empty, which is forbidden since A and Z cross, or CV is disconnected from the other vertices in G).
- Beside this, there are three similar statements, by the same circular replacement as before.

Now, one can check the adjacency matrix between AZ and its complement under a case analysis and conclude that AZ is a split. □

Chapter 3

Representable Generalizations

This chapter is based on [15, 21].

All the classes of set families presented in the previous chapter range over two main categories: those having a tight linear space complexity, and those having a tight quadratic space complexity. Moreover, every class of the first category is subclass of some class of the second category, namely weakly partitive families are particular cases of intersecting families while symmetric crossing families are particular cases of crossing families. As well, note that all the involved families are particular cases of crossing families (see Figure 3.1).

We address in this chapter two new classes of families, the so-called *(weakly) partitive crossing* and *union-difference* families. The motivation is manifold. Some theoretical motivations could be as follows. At first, both classes own a polynomial space complexity. Secondly, weakly partitive crossing families are in fact proper generalizations of, not only weakly partitive families, but also symmetric crossing families. Besides, they all are particular cases of crossing families. Furthermore, we will show that each of them has a representation of linear size on the size of the ground set. This fact broadens the known subclass of crossing families of linear space complexity. As for union-difference families, while being a quite simple generalization of partitive families, they form a class which is incomparable (inclusionwise) with the class of crossing families (a synopsis is given in Figure 3.2). Also, we will give a representation of quadratic size on the size of the ground set for such a family. This is a widening of the superclass of partitive families containing some families known to have at most a quadratic space complexity. However, the question whether the quadratic space complexity of union-difference families is tight, or not, will be left open (see Figure 3.3).

At the same time, we will exemplify the applicative potentials of the above mentioned families by introducing a new notion of graph components, a so-called *sesquimodule*. Then, interests in studying those set families can also be seen as a motivation arising from the two leading facts that: the sesquimodules of a given digraph form a weakly partitive crossing

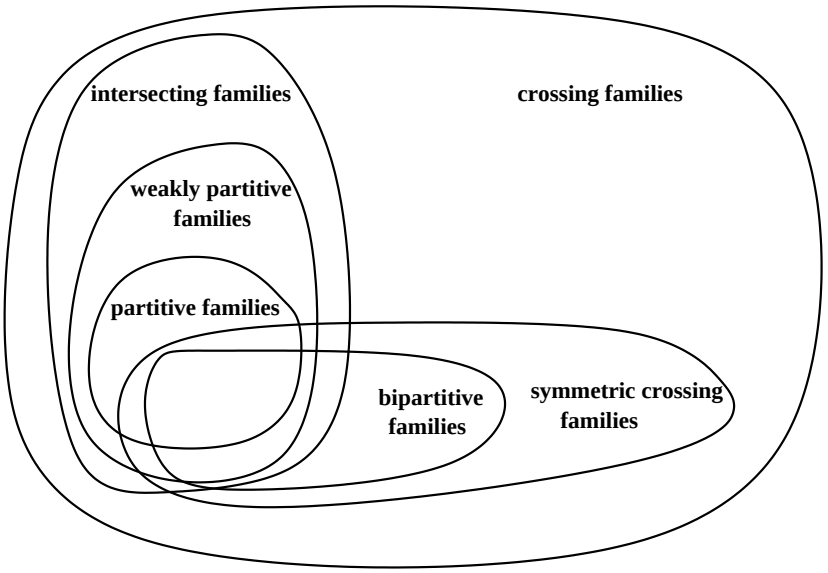


Figure 3.1: Inclusion relationship of some classes of set families. When comparable, the inclusion of a class is strict.

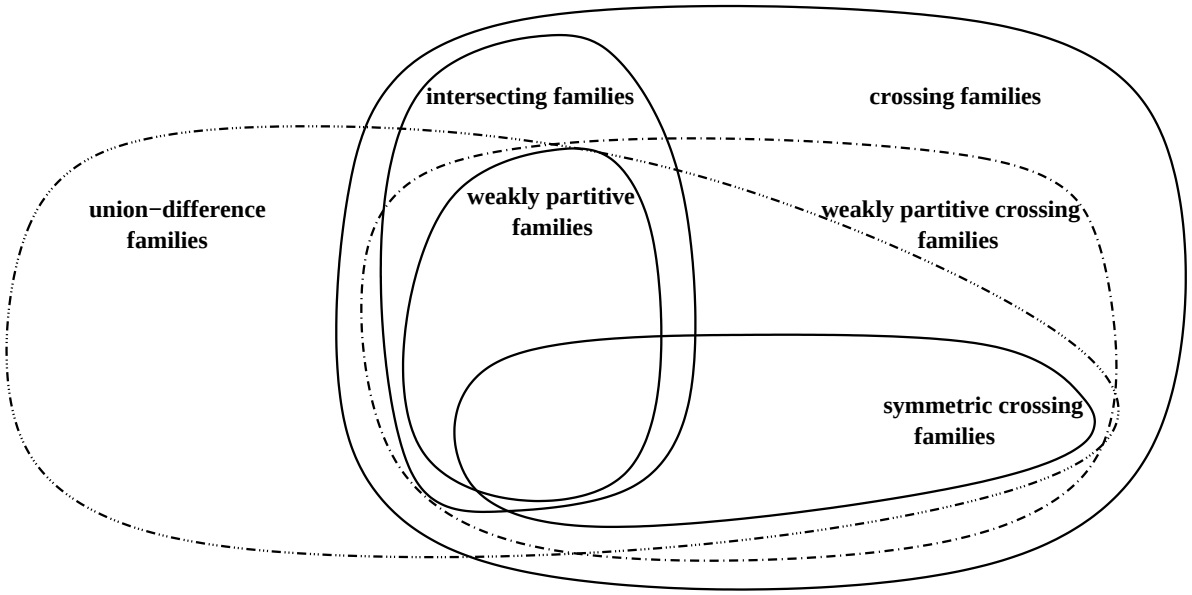


Figure 3.2: Two new classes of set families. When comparable, the inclusion of a class is strict.

class	space complexity	tight
weakly partitive families	$O(n)$ [27]	yes
symmetric crossing families	$O(n)$ [39]	yes
weakly partitive crossing families	$O(n)$ [21]	yes
union-difference families	$O(n^2)$ [15]	– **OPEN** –
intersecting families	$O(n^2)$ [58]	yes
crossing families	$O(n^2)$ [58]	yes

Figure 3.3: Some classes of set families by increasing space complexity. The size of the ground set is n.

family; moreover, their generalization to the sesquimodules of a given 2-structure form a union-difference family which is not necessarily weakly partitive crossing. Furthermore, sesquimodules of digraphs turn out to be proper generalizations of their modules*, while sesquimodules of 2-structures are proper generalizations of their clans.

We then depict two new combinatorial decompositions. By some abuse of terminology, we denote them simultaneously by *sesquimodular decomposition*. One of them can be used for digraphs as a proper generalization of modular decomposition. It is based on weakly partitive crossing families. The other can be used for arbitrary 2-structures as a proper generalization of clan decomposition. It is based on union-difference families. However, despite their common name, the two decomposition schemes are by no means equivalent. For instance, given an n-vertex digraph G, the sesquimodular decomposition tree of G via weakly partitive crossing families is of $O(n)$ size. It is different from the sesquimodular decomposition tree of G via union-difference families, which is of $\Theta(n^2)$ size. In both cases, all sesquimodules of G can be enumerated in a straightforward manner from the only knowledge of the tree (without the knowledge of, e.g., G).

3.1 Partitive Crossing Families

The last chapter has displayed a quite tidy gap between the tight results of, on the one hand, the space complexity of crossing families, and on the other hand, the space complexity of families which are either partitive or bipartitive. This naturally moves us

*Here, modules are in the sense of clans of the digraph (cf. Definition 2.17 in the last chapter).

to look into the gap and define

Definition 3.1 (Weakly Partitive Crossing and Partitive Crossing Families) A proper and connected family is *weakly partitive crossing* if it is a crossing family which is closed under the difference of its crossing members. Moreover, if the family is also closed under the symmetric difference of its crossing members, it is called *partitive crossing*.

Clearly, the notion of a partitive crossing (resp. weakly partitive crossing) family can be seen simultaneously as: a simple generalization of a partitive (resp. weakly partitive) family; a simple generalization of a bipartitive (resp. symmetric crossing) family; and a simple restriction of a crossing family. Besides, it is also a straightforward exercise to check that those generalizations and restriction are strict. The main focus of this section is to prove that every weakly partitive crossing family has also a representation using a linear space on the size of the ground set.

At the same time, our study on partitive crossing families is moved by the fact that Section 3.3 gives an application of those families in a strict generalization of the well-known and well-studied modular decomposition of digraphs.

3.1.1 A Linear-Size Representation Theorem

We proceed as before with partitive families: we first give the claim of the main theorems, and only give the proofs subsequently. Here, partitive crossing families are structured by the two following theorems. Their respective hypothesis are pairwise exclusive.

Theorem 3.1 *If a family $\mathcal{F} \subseteq 2^X$ satisfies: the weakly partitive crossing property, the quotient property, and the simply-linked property, then one and only one of the following holds:*

- $|X| \leq 4$ *(we say that $\mathcal{F}$ is* basic*),*
- $|X| \geq 5$ *and $\mathcal{F}$ consists of only the trivial subsets of X and possibly some of their complements (we still say that $\mathcal{F}$ is* prime*, by abuse in the terminology),*
- $|X| \geq 5$ *and $\mathcal{F} = 2^X \setminus \{\emptyset\}$ ($\mathcal{F}$ is* complete*),*
- $|X| \geq 5$ *and there is a linear ordering of the elements of X such that $\mathcal{F}$ is the family of all intervals of this ordering ($\mathcal{F}$ is* linear*).*
- $|X| \geq 5$ *and there is a circular ordering of the elements of X such that $\mathcal{F}$ is the family of all circular intervals of this ordering ($\mathcal{F}$ is* circular*).*

Moreover, if $\mathcal{F}$ is partitive crossing, the two last cases cannot occur.

Proof: By successive applications of Lemma 3.1, Lemma 3.2, and Remark 3.1 (all three below). □

Theorem 3.2 *If a family $\mathcal{F} \subseteq 2^X$ satisfies both the weakly partitive crossing property and the quotient property, but fails the simply-linked property, then, for $x \in X$ being the guard of $\mathcal{F}$, $\mathcal{G} = \mathcal{F} \setminus \{X, \{x\}\}$, and $Y = X \setminus \{x\}$, one and only one of the following holds:*

- $|Y| \leq 4$ *(we still say that $\mathcal{F}$ is* basic*, the difference is that $\mathcal{F}$ now has a guard),*
- $|Y| \geq 5$ *and $\mathcal{G}$ consists of only the trivial subsets of Y and possibly some of their complements (we still say that $\mathcal{F}$ is* prime*, the difference is that $\mathcal{F}$ now has a guard),*
- $|Y| \geq 5$ *and $\mathcal{G} = 2^Y \setminus \{\emptyset\}$ (we still say that $\mathcal{F}$ is* complete*, the difference is that $\mathcal{F}$ now has a guard),*
- $|Y| \geq 5$ *and there is a linear ordering of the elements of Y such that $\mathcal{G}$ is the family of all intervals of this ordering (we still say that $\mathcal{F}$ is* linear*, the difference is that $\mathcal{F}$ now has a guard).*

Moreover, if $\mathcal{F}$ is partitive crossing, the last case cannot occur.

Proof: (same as previous theorem). By successive applications of Lemma 3.1, Lemma 3.2, and Remark 3.1 (all three below). □

Linear and circular quotients are represented like before, with (weakly) partitive and bipartitive families. The representation of the new prime quotients is already done by the arc orientations of the decomposition tree: all members of a prime quotient are cross-free members of the initial family. As for basic quotients, they can be represented in constant space using an exhaustive encoding. In Section 3.3, Figure 3.9(b) exemplifies the representation on a restricted framework (of sesquimodular digraph decomposition). We conclude from the two above theorems that

Corollary 3.1 *The space complexity of a weakly partitive crossing family over X is in $O(|X|)$.*

We now prove Theorems 3.1 and 3.2 in two steps.

Lemma 3.1 *Let $\mathcal{F}$ be a weakly partitive crossing and quotient family. Let S be a non trivial and non-quasi trivial member in $\mathcal{F}$. For every element $x \in S$, x has a neighbour in the 2-graph $G_{\mathcal{F}}$ of $\mathcal{F}$.*

Proof: Let $A \in \mathcal{F}$ such that $x \in A$, A is non trivial, non quasi-trivial and minimum by size. If $|A| = 2$, then by definition x has a neighbour in $G_{\mathcal{F}}$. Otherwise, we will exhibit a contradiction. Firstly, there is a member of $\mathcal{F}$ which crosses A since $\mathcal{F}$ is quotient. Let B be a member of minimum size among the members of $\mathcal{F}$ which crosses A.

Suppose that $x \in B$. Here, B is *not necessarily* of minimum size among the members of $\mathcal{F}$ which contains x, but A is one such. A consequence is that $|B| \geq |A|$. Also, $A \cap B$ is a member of $\mathcal{F}$ (intersection closure) which contains x. By minimality of A, $A \cap B = \{x\}$. Besides, $B \setminus A$ is a member of $\mathcal{F}$ (difference closure) and is not trivial nor quasi-trivial since $|B| \geq |A| > 2$. Then, there is a member C of $\mathcal{F}$ which crosses $B \setminus A$ (quotient property). If $A \cap C = \emptyset$, C crosses B, and $B \setminus C$ would contradict the minimality of B. Therefore, $A \cap C \neq \emptyset$. If $x \notin C$, then $B \setminus C$ would contradict the minimality of B. Otherwise, $B \cap C$ would contradict the minimality of B. In any case there is a contradiction.

Suppose that $x \notin B$. Then, $A \setminus B = \{x\}$ (closure and minimality of A). Moreover, $A \cap B$ is a non trivial, non quasi-trivial member of $\mathcal{F}$ (closure and size argument). Then, there is a member C of $\mathcal{F}$ which crosses $A \cap B$ (quotient). If $C \subseteq A$, then $C \neq A$, $x \in C$, and C contradicts the minimality of A. Otherwise we distinguish two cases. If $A \cup C \neq V$, then A and C cross, and $A \setminus C$ would contradict the minimality of A. If $A \cup C = V$, then B and C cross, and $B \cap C$ would contradict the minimality of B. □

Lemma 3.2 *Let $\mathcal{F}$ be a weakly partitive crossing and quotient family over a ground set X with $|X| \geq 5$. Then, either $\mathcal{F}$ only consists of the trivial subsets of X and possibly some of their complements, or one and only one of the following holds*

- *$G_{\mathcal{F}}$ is a clique,*
- *$G_{\mathcal{F}}$ is the disjoint union of a clique and an isolated vertex,*
- *$G_{\mathcal{F}}$ is a cycle,*
- *$G_{\mathcal{F}}$ is a path,*
- *$G_{\mathcal{F}}$ is the disjoint union of a path and an isolated vertex.*

Moreover, if $\mathcal{F}$ is also a partitive crossing family, then the three last cases cannot occur.

Proof: By Lemma 3.1, $G_{\mathcal{F}}$ is not empty. Let W be a connected component of $G_{\mathcal{F}}$ of size at least 2. We first prove that $|W| \geq |X| - 1$ by contradiction.

Assume this is not the case. Clearly we have $W \in \mathcal{F}$ (union closure). Let A be a member in $\mathcal{F}$ which crosses W, minimum by size. Then, $|A| > 2$ (maximality of W). Besides, $|A \cap W| = 1$, otherwise $A \setminus \{v\}$ would contradict the minimality of A, where uv

is an edge of $G_{\mathcal{F}}[W]$ which crosses A. Then, there is a member C of $\mathcal{F}$ which crosses $A \setminus W$. Here, either $A \cap C$ or $A \setminus C$ would contradict the minimality of A.

We have now two cases. Either $G_{\mathcal{F}}$ is connected, or $G_{\mathcal{F}}$ has one connected component of size $|X|-1$ and one isolated vertex. In both cases, let W be the connected component of size $|X|$ or $|X|-1$.

We now conclude using the same technique described in [47] and recalled in Lemma 2.5. Suppose that there is in $G_{\mathcal{F}}$ a vertex $v \in W$ adjacent to at least 3 vertices a, b and c. Clearly $\{a, v, b\}$ and $\{v, c\}$ are members of $\mathcal{F}$. Since $|X| > 4$, there is a vertex $u \in X \setminus \{v, a, b, c\}$, and the latter members $\{a, v, b\}$ and $\{v, c\}$ cross. Therefore, $\{a, b\}$ is a member of $\mathcal{F}$, and by symmetry we can deduce that $\{v, a, b, c\}$ induces a clique in $G_{\mathcal{F}}$. Let C be a maximal clique in $G[W]$ and suppose that $C \neq W$. Since W is a connected component there is a vertex $w \in W \setminus C$ adjacent to a vertex $v \in C$. By similar argument as before, we can prove that w is linked to every vertex of C. But this contradicts the maximality of C, so we must have $C = W$.

Hence either W induces a complete graph, or every vertex of W has degree at most two, i.e. $G_{\mathcal{F}}[W]$ is either a path or a cycle. Now suppose that $|W| = |X| - 1$ and W induces a cycle of length at least 4. Let v be the isolated vertex in $G_{\mathcal{F}}$ and let a, b, c be pairwise distinct vertices of $X \setminus \{v\}$ such that a is adjacent to b and c in $G_{\mathcal{F}}$ (they exist mostly for $|X| > 4$). But from the existence of v, $\{a, b\}$ and $\{a, c\}$ are *crossing* members of $\mathcal{F}$. Therefore $\{a, b, c\} \in \mathcal{F}$. But we also have that $X \setminus \{v, a\}$ is a member of $\mathcal{F}$ from the union closure on the other vertices of the cycle. From the existence of v, $\{a, b, c\}$ and $X \setminus \{v, a\}$ are crossing members of $\mathcal{F}$. Hence, bc is an edge in $G_{\mathcal{F}}$. Contradiction.

Finally, suppose that $\mathcal{F}$ is also a partitive crossing family. If there is any vertex a of $G_{\mathcal{F}}$ of degree at least 2, then the symmetric difference closure will forbid the cases of cycles and paths. □

Remark 3.1 *Let $\mathcal{F} \subseteq 2^X$ be a weakly partitive crossing family, and $G_{\mathcal{F}}$ its 2-graph. Then*

- *$G_{\mathcal{F}}$ has an isolated vertex if and only if $\mathcal{F}$ has a guard.*
- *$G_{\mathcal{F}}$ is a clique of size at least 5 if and only if $\mathcal{F} = 2^X \setminus \{\emptyset\}$.*
- *$G_{\mathcal{F}}$ is a path of size at least 5 if and only if there is a linear ordering of the elements of X such that $\mathcal{F}$ is the family of all intervals of this ordering.*
- *$G_{\mathcal{F}}$ is a cycle of size at least 5 if and only if there is a circular ordering of the elements of X such that $\mathcal{F}$ is the family of all circular intervals of this ordering.*

The size assumption over the ground set has one and only one aim: whenever A and B are overlapping members of the family with $|A| = 3$ and $|B| = 2$, the assumption enforces that A and B are also crossing members. This allows the use of closure axioms on crossing members, like in the proof of Lemma 3.2.

3.1.2 Between Crossing Families and Linear-Size Representable Families?

We have seen that weakly partitive crossing families are at the same time generalizations of weakly partitive families and symmetric crossing families. Also, they are crossing families satisfying the additional axiom of closure under the difference of their crossing members. Accordingly, they help in increasing the subclass of crossing families known to have a linear space complexity (see Figure 3.2). However, such a gap is large, and it is a natural question to look into the space between weakly partitive crossing families and crossing families.

Unfortunately, the approach consisting of adding some closure axioms to those of crossing families reveals unhelpful here. For instance, it is by definition that the addition of the difference closure to a crossing family leads already to a weakly partitive crossing family. Moreover, the following direction fails also. Both the class of crossing families and the class of $\{\ \Delta\ \}$ closed families have a tight quadratic space complexity (see Section 1.3). Then, it could at first sight be an interesting question to look into their intersection. However, we have here that a family which is at the same time crossing and closed under the symmetric difference will also be a partitive crossing family. This leads back to a known case. By elimination, the only closure related to our composition which remains is the closure under complementation. But we have also seen that a crossing family which is closed under complementation is called a symmetric crossing family. This also leads back to a known case.

Even the following relaxation is a failure. Let us define FOO as the class containing every crossing family which satisfies that, for every crossing members A and B of the family, at least one of the three following sets is also a member of the family: $A \setminus B$, $A\Delta B$, and $\overline{B}$. Here, one can check that $A \setminus B$ is always a member of the family. Indeed, if $C = A\Delta B$ is a member, then C crosses A and $A \setminus B = A \cap C$ will be a member of the family. As well, if $\overline{B}$ is a member, then it crosses A and $A \setminus B = A \cap \overline{B}$ will also be a member. Hence, FOO is exactly the class of weakly partitive crossing families. And we are back to a known case.

Despite all this, the duality quadratic vs. linear space complexity is still an interesting question. For instance, the minimizers of submodular functions are very naturally bound

to intersecting and crossing families. On the other hand, the minimizers of symmetric submodular functions are by definition bound to symmetric crossing families. Given the tidy gap between the space complexity of crossing versus symmetric crossing families, the above statements on submodular functions can also be seen as a theoretical motivation to find other directions to investigate what is in between crossing families and symmetric crossing families. The next section investigates one such direction.

3.2 Union-Difference Families

From what has just been said on the gap between families of linear and quadratic space complexity, a direction could also be on how to extend *partitive* families while keeping the linear space complexity. If we wish to do this by means of removing successively the closure axioms on overlapping members, then the first axiom which comes to mind would probably be the symmetric difference closure. (For an exhaustive checking on the possible removals, we refer the reader to Figure 1.5 at the end of Chapter 1.) Indeed, we then result in the class of weakly partitive families, which is also of linear space complexity. After this, we come to a junction. However, we have seen that two among the three remaining choices are bad: either we remove the difference axiom and result in intersecting families, which have a tight quadratic space complexity (see Section 2.1); or we remove the union axiom and run into the overlap-$\{ \cap, \setminus \}$ families, which cannot be represented in polynomial space (see Section 1.3). Accordingly, let us check the last choice, namely the case of

Definition 3.2 (Union-difference Family) A proper and connected family is a *union-difference* family if it is closed under the union and the difference of its overlapping members.

Remark 3.2 *If a union-difference family is also closed under the symmetric difference of its overlapping members, then it is a partitive family.*

Remark 3.3 *Clearly, a weakly partitive family is also a union-difference family. It is a bit less straightforward, but a symmetric crossing family is also a union-difference family.*

A potential hitch of the above defined families is that none of what has been said so far give a clue on their space complexity. Indeed, it is straightforward to check that the class of union-difference families is incomparable with, on the one hand, the class of crossing families, and on the other hand, the class of $\{ \Delta \}$ closed families. From this, we *a priori* cannot conclude with a polynomial assumption, from any of the results given in the previous chapters. On the other hand, union-difference families are strict restrictions

of $\{ \cup \}$ closed families and $\{ \cap, \setminus \}$ closed families, the two minimal classes we know to own an argument against a polynomial representation. Accordingly, we *a priori* cannot conclude with a non-polynomial assumption, neither. The aim of this section is to give a representation of quadratic size on the size of the ground set for any union-difference family. However, we leave open the question whether our representation is tight, or union-difference families have a sub-quadratic space complexity.

In any case, the complexity result of this section is also a broadening of the known classes of set families having a polynomial space complexity. Last but not least, the next section gives an application of union-difference families in a strict generalization of clan decomposition of 2-structures.

3.2.1 A Quadratic-Size Representation Theorem

The structuralization of a union-difference family via the cross-free decomposition tree, and the associated notion of a quotient, is strange. Indeed, while simply-linked quotients are simply organized by the following theorem, structural properties of other quotients are rather scarce.

Theorem 3.3 *If a family $\mathcal{F} \subseteq 2^X$ satisfies: the union-difference property, the quotient property, and the simply-linked property, then one and only one of the following holds:*

- *$\mathcal{F}$ consists of only the trivial subsets of X and possibly some of their complements ($\mathcal{F}$ is* prime*),*
- *$\mathcal{F} = 2^X \setminus \{\emptyset\}$ ($\mathcal{F}$ is* complete*),*
- *there is a linear ordering of the elements of X such that $\mathcal{F}$ is the family of all intervals of this ordering ($\mathcal{F}$ is* linear*).*
- *there is a circular ordering of the elements of X such that $\mathcal{F}$ is the family of all circular intervals of this ordering ($\mathcal{F}$ is* circular*).*

Proof: First we have to prove Lemmas 3.3 and 3.4 (below). Then, notice by Lemma 3.5 (below) that if $G_{\mathcal{F}}$ is connected, it is either a clique, a path, or a cycle. We use Remark 3.4 (below) in order to conclude. □

Roughly, the simply-linked quotients of a union-difference family are more organized than the simply-linked quotients of a partitive crossing family (there are fewer cases). However, we have failed so far in characterizing the non-simply linked quotients of a union-difference family. Instead, for the representation of those quotients we use the

very properties of a guard and a quotient-hereditary family. This is formally stated in Theorem 3.4 below. Its proof follows directly from definition.

Theorem 3.4 *If a family $\mathcal{F} \subseteq 2^X$ satisfies both the union-difference property and the quotient property, but fails the simply-linked property, then,*

- *there is a guard $x \in X$ such that $Y = X \setminus \{x\}$ and $\mathcal{G} = \mathcal{F} \setminus \{X, \{x\}\}$ is a union-difference family over the ground set Y (we say that $\mathcal{F}$ is* recursive*, it can be represented via the guard x and any representation of $\mathcal{G}$).*

In Section 3.3, Figure 3.9(c) gives an example under a restricted framework. Clearly, we have to perform the recursive process of Theorem 3.4 for every non-simply linked quotient of the decomposition tree $\mathcal{T}$ of a union-difference family $\mathcal{F}$. Moreover, we have to do the same process for all non-simply linked quotients of the decomposition tree $\mathcal{T}'$ of every union-difference family $\mathcal{G}$ associated to a non-simply linked quotient of $\mathcal{T}$. And so on. In other words, a decomposition tree of a union-difference family, along with all its quotients, may have recursive levels. Fortunately enough, its total size is still polynomial:

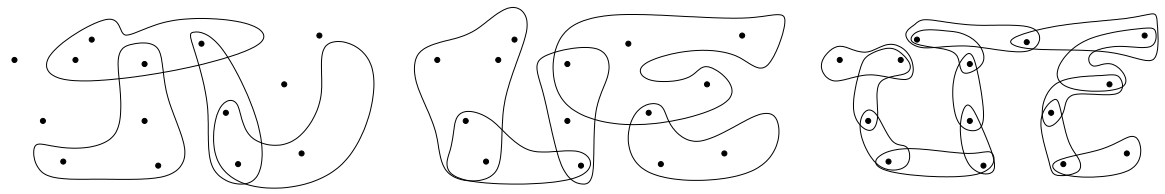
Theorem 3.5 *Let us label every internal node of the decomposition tree $\mathcal{T}$ of a given union-difference family $\mathcal{F} \subseteq 2^X$ w.r.t. the matching between its quotient and the cases described in Theorems 3.3 and 3.4. Then, the global size of the labelled tree is in $O(|X|^2)$.*

Proof: (by induction on $n = |X|$). Let $f(n)$ be the maximum size of all decomposition trees of n leaves. Obviously, $f(1)$ and $f(2)$ are non null constants. Let $f(k) \leq \alpha \times k^2$ hold for all $k < n$. We suppose without loss of generality that α is greater than any other constant in this proof. Let us consider a decomposition tree of n leaves and let N be the set of its internal nodes. For each $i \in N$, let n_i be its degree. The label of i is either of constant size (cf. prime and complete nodes), of linear size on n_i (cf. linear and circular nodes), or of size bounded by $f(n_i - 1) + \beta$ (cf. nodes that are not simply-linked). In all cases, it is bounded by $\alpha \times (n_i - 1)^2 + \alpha$ since $n_i \geq 3$ and $\alpha \geq \beta$. The total size of leaves, edges, and orientations is linear on n, hence bounded by $\alpha \times n$. We deduce that

$$f(n) \leq \alpha \times \left(\sum_{i \in N} ((n_i - 1)^2 + 1) + n \right) \leq \alpha \times \left(\sum_{i \in N} (n_i - 1)^2 + n' + n \right),$$

where $n' = |N|$. Notice that $\sum_{i \in N} n_i = n + 2 \times (n' - 1)$ (the n pendant edges are counted once while other edges are counted twice). In other words, $S = \sum_{i \in N} (n_i - 1) = n + n' - 2$. Then, the greatest value that $\sum_{i \in N} (n_i - 1)^2$ can reach happens when one among the n_i gets the greatest value possible. Since $n_i - 1 \geq 2$, we have

$$\sum_{i \in N} (n_i - 1)^2 \leq (n' - 1) \times 2^2 + (S - (n' - 1) \times 2)^2.$$



i. ii. iii.

Figure 3.4: i. A chain. ii. A covering chain. iii. An irreducible covering chain.

Then, $f(n) \leq \alpha \times (n^2 + n'^2 + 5n' + n(1-2n') - 4)$. Besides, that there are no degree 2 nodes in the tree provides us with $n \geq n' + 2$. Moreover, it is clear that $1 - 2n' \leq 0$. Hence, $n(1-2n') \leq (n'+2)(1-2n')$, which is also $n(1-2n') \leq -2n'^2 - 3n' + 2$. Therefore, $f(n) \leq \alpha \times (n^2 - n'^2 + 2n' - 2) \leq \alpha \times (n^2 - (n'-1)^2 - 1) \leq \alpha \times n^2$. □

Corollary 3.2 *The space complexity of a union-difference family over X is in $O(|X|^2)$.*

Turning our attention back to the proof of Theorem 3.3, let us introduce the following terminologies. A *chain* of length k of $\mathcal{F}$ is a sequence $(A_1, A_2, \ldots, A_k)$ of members of $\mathcal{F}$ such that $A_i \circledcirc A_{i+1}$ for all i, and $A_i \cap A_j = \emptyset$ for all $|i - j| > 1$. The chain is *covering* if $A_1 \cup A_2 \cup \cdots \cup A_k = X$, and *irreducible* if $|A_i| = 2$ for all $1 \leq i \leq k$. An irreducible and covering chain of $\mathcal{F}$ can also be seen as a Hamiltonian path in the 2-graph $G_{\mathcal{F}}$, which would imply its connectivity, and enable the use of Lemma 3.5 (see Figure 3.4).

Lemma 3.3 *If a union-difference family $\mathcal{F} \subseteq 2^X$ satisfies both quotient and simply-linked properties, then, either $\mathcal{F}$ only consists of the trivial subsets of X and possibly some of their complements, or $\mathcal{F}$ has a length three covering chain.*

Proof: Suppose that $\mathcal{F}$ does not contain only the trivial subsets of X and some of their complements. Let $A \in \mathcal{F}$ be neither trivial nor quasi-trivial. We take A maximal by inclusion. The quotient property provides us with $B \in \mathcal{F}$ such that A and B cross. The closure under the union implies $A \cup B \in \mathcal{F}$. Moreover, A is maximal. Hence $A \cup B$ is either trivial or quasi-trivial. However, $A \cup B$ cannot be trivial since A and B cross. Then, the simply-linked property implies that $A \cup B$ is not an overlap-free member. Hence it overlaps some member $C \in \mathcal{F}$. Here, all cases lead to either $D = C \cup B \setminus A$ or $E = C \cup A \setminus B$ is a member of $\mathcal{F}$. Then, either (A, B, D) or (B, A, E) is a covering chain of length three. □

Lemma 3.4 *If a union-difference family $\mathcal{F} \subseteq 2^X$ satisfies both quotient and simply-linked properties, and has a covering chain of length at least three, then $\mathcal{F}$ has an irreducible covering chain (then $G_{\mathcal{F}}$ is connected).*

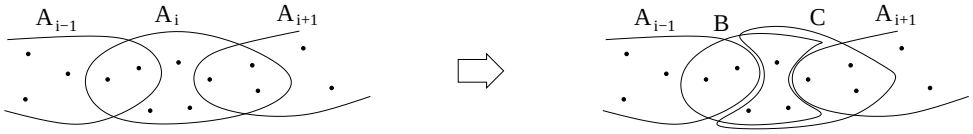


Figure 3.5: Illustration for the proof of Lemma 3.4.

Proof: By hypothesis $\mathcal{F}$ has a covering chain $\mathcal{A} = (A_1, A_2 \ldots, A_k)$ with $k \geq 3$. We take k maximum.

Assume for some $1 < i < k$ that $A_i \setminus (A_{i-1} \cup A_{i+1}) \neq \emptyset$. In this case $B = A_i \setminus A_{i+1}$ and $C = A_i \setminus A_{i-1}$ are overlapping members of $\mathcal{F}$ (see Figure 3.5). Then, replacing $\mathcal{A}$ with $(A_1, A_2 \ldots, A_{i-1}, B, C, A_{i+1}, \ldots, A_k)$ would improve k. Hence, $A_i \setminus (A_{i-1} \cup A_{i+1}) = \emptyset$ for all $1 < i < k$.

We now assume that $|A_i| > 2$ for some $1 < i < k$. Then at least one among $B = A_i \setminus A_{i+1}$ and $C = A_i \setminus A_{i-1}$ is neither trivial nor quasi-trivial (hence not cross-free by quotient property). By symmetry we suppose it was B. Let $D \in \mathcal{F}$ cross B. We show in all cases a contradiction as follows (see also Figure 3.6).

- *Case 1:* $D \subseteq A_i$. Among other, D and A_{i-1} overlap. Let $E = A_{i-1} \setminus D$, we can improve k by replacing $\mathcal{A}$ with $(A_1, A_2, \ldots, A_{i-2}, E, B, D, A_{i+1}, \ldots, A_k)$.

- *Case 2:* $D \setminus A_i \neq \emptyset$ and $C \setminus D \neq \emptyset$. Then, we are conducted to *Case 1* by replacing D with $D' = A_i \setminus D$.

- *Case 3:* $D \setminus A_i \neq \emptyset$ and $C \subseteq D$. We define the left and right as $L = A_1 \cup \cdots \cup A_{i-2}$ and $R = A_{i+1} \cup \cdots \cup A_k$. Notice that $L \cup R = \overline{B}$. Since D and B cross, there is some element in either L or R that does not belong to D. If it was L, replacing D with $A_i \setminus (A_{i-1} \setminus (D \setminus L))$ leads back to *Case 1*. If it was R, the same can be done with $A_i \setminus (D \setminus R)$.

Hence, $|A_i| = 2$ for all $1 < i < k$. Now, assume that $|A_1| > 2$, and let $D \in \mathcal{F}$ cross $B = A_1 \setminus A_2$. Let $Z = A_k \setminus A_{k-1}$. We will examine whether $Z \setminus D \neq \emptyset$ or $Z \subseteq D$ (see Figure 3.7). In the first case, let $E = A_3 \cup \cdots \cup A_k$ and $F = D \cup A_2 \cup \cdots \cup A_{k-1}$: they overlap. Then, $G = F \setminus E$ is a member of $\mathcal{F}$, and replacing $\mathcal{A}$ with $(B, G, A_2, \ldots, A_k)$ would improve k. In the second case, since D and B cross, there is some element in $A_2 \cup \cdots \cup A_{k-1}$ that does not belong to D. In other words, $A_2 \cup \cdots \cup A_{k-1}$ and D overlap.

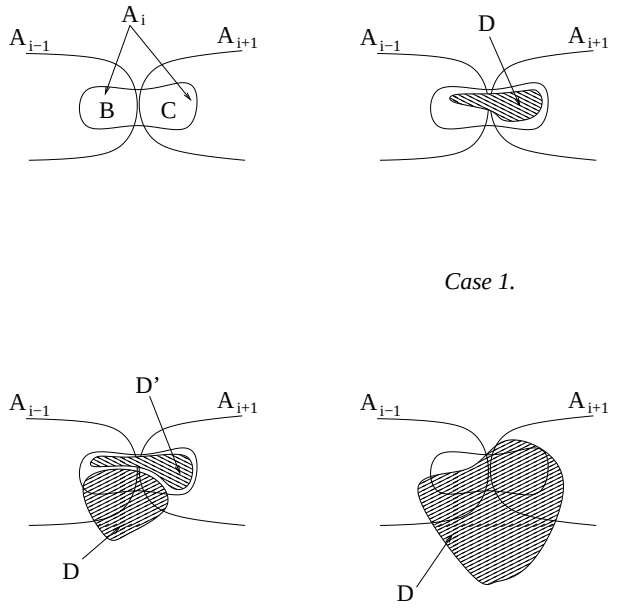


Figure 3.6: Illustration for the proof of Lemma 3.4.

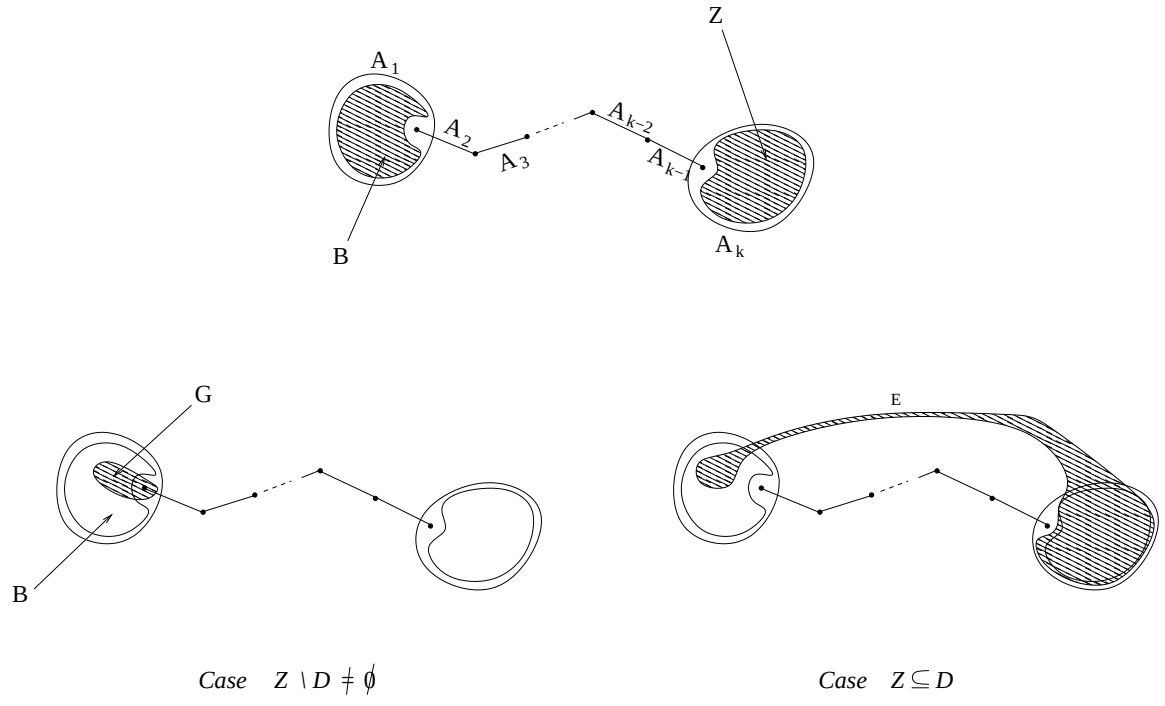


Figure 3.7: Illustration for the proof of Lemma 3.4.

Then, $E = D \setminus (A_2 \cup \cdots \cup A_{k-1})$ is a member of $\mathcal{F}$ which contains Z. That $E \in \mathcal{F}$ implies $(E, A_1, A_2, \ldots, A_{k-1})$ is a chain of $\mathcal{F}$. That E contains Z implies the chain is covering. Moreover, it is of length k, i.e. of maximum length. However, from the last paragraph, this chain cannot have A_1 with more than two elements. Therefore, $|A_1| = 2$. Then, by symmetry we obtain $|A_k| = 2$, and $\mathcal{A}$ is an irreducible covering chain. □

To conclude, we use a tool that was discovered from previous works on partitive families. It has been addressed in the last chapter in the statement of Lemma 2.5. The actual situation is: the proofs of Lemma 2.5 given in [47] mainly required the union and difference closures. Then, the following property can be seen as part of that lemma.

Lemma 3.5 *(cf. [47] with partitive families) Let $\mathcal{F}$ be a union-difference family. If its 2-graph $G_{\mathcal{F}}$ is connected then $G_{\mathcal{F}}$ is either a clique, a path, or a cycle.*

Proof: The proof given in [47] is as follows. Suppose that $G_{\mathcal{F}}$ has a vertex x with degree at least 3, and let y, z, t be three distinct neighbours of x. In other words, $\{x, y\}$ and $\{x, z\}$ are members of $\mathcal{F}$, and so is $\{x, y, z\}$ by union closure. But $\{x, t\}$ is also a member of $\mathcal{F}$. By difference closure we deduce that $\{y, z\}$ is an edge of $G_{\mathcal{F}}$. Likewise, we can deduce that x, y, z, and t form a clique in $G_{\mathcal{F}}$. Now, let v be a vertex that is connected to the previous clique at some point, say t. Then, by a similar argument on the fact that t is of degree at least 3, we can show that v is connected to all other vertices of the clique. Thus the previous clique plus vertex v form a bigger clique, and so on. The connectivity of $G_{\mathcal{F}}$ then can be used to conclude that the whole graph $G_{\mathcal{F}}$ is a clique. Finally, the only connected graphs of degree at most 2 are paths and cycles. □

Remark 3.4 *Let $\mathcal{F} \subseteq 2^X$ be a union-difference family, and $G_{\mathcal{F}}$ its 2-graph. Then*

- *$G_{\mathcal{F}}$ is a clique if and only if $\mathcal{F} = 2^X \setminus \{\emptyset\}$.*
- *$G_{\mathcal{F}}$ is a path of size at least 3 if and only if there is a linear ordering of the elements of X such that $\mathcal{F}$ is the family of all intervals of this ordering.*
- *$G_{\mathcal{F}}$ is a cycle of size at least 4 if and only if there is a circular ordering of the elements of X such that $\mathcal{F}$ is the family of all circular intervals of this ordering.*

Before closing the section, let us comment that the way we proceeded for non-simply linked quotients of union-difference families is somewhat brute-force (cf. Theorem 3.4), and conjecture that

Conjecture: *Union-difference families have sub-quadratic space complexity.*

3.3 Applications in Graph Theory

We have seen at the end of Section 2.3.1 a natural question for the search of not only relaxed, but also tractable, variations of the modular decomposition scheme. This section investigates the case of directed graphs, and their common generalization to arbitrary (and not necessarily symmetric) 2-structures. In order to further decompose, a weakened definition of module is proposed. Fortunately enough, we still obtain a well-structured variation, thanks to partitive crossing and union-difference families.

We recall some terminologies presented in the last chapter. Digraphs throughout the thesis refer to loopless simple directed graphs where 2-cycles are allowed. A 2-structure $G = (X, C)$ is a vertex set X along with a colour function C mapping every pair (x, y) (with $x, y \in X$ and $x \neq y$) to some value in $\mathbb{N}$. The out-going neighbourhood $N_c^+(x)$, for every vertex $x \in X$ and every colour $c \in \mathbb{N}$, is defined according to standard notions of neighbourhood in graph theory as the set of all vertices y such that $C(x, y) = c$. Likewise, the incoming neighbourhood $N_c^-(x)$ is the set of all vertices y such that $C(y, x) = c$.

The focus of this section is on the following notion.

Definition 3.3 (Sesquimodule) A *sesquimodule* M of a 2-structure $G = (X, C)$ is a non-empty vertex subset such that

- $\forall x, y \in M,\ \forall c \in C,\quad N_c^-(x) \setminus M = N_c^-(y) \setminus M$, and
- $\forall x, y \in M,\ \forall c \in C,\ \exists d \in C,\quad N_c^+(x) \setminus M = N_d^+(y) \setminus M$.

Let us recall also that, in an undirected graph, a module is a non-empty vertex subset M such that $\forall x, y \in M,\ N(x) \setminus M = N(y) \setminus M$. For a symmetric 2-structure, the same definition holds with the condition $\forall x, y \in M,\ \forall c \in C,\ N_c^-(x) \setminus M = N_c^-(y) \setminus M$. The generalization to arbitrary 2-structures presented in Section 2.3.2 under the name of a genuine-module appeals to the very same condition. On the other hand, the classical generalization of module to so-called digraph modules and clans of 2-structures requires *two full conditions* on M: $\forall x, y \in M,\ \forall c \in C$, both $N_c^-(x) \setminus M = N_c^-(y) \setminus M$ and $N_c^+(x) \setminus M = N_c^+(y) \setminus M$ must hold.

In the new notion, there is a full condition on in-neighbours, and a relaxed condition on out-neighbours. For the relaxed condition, the partition of the exterior w.r.t. the colour of the out-going arcs still has to be the same, however, the order of the parts in the partition is irrelevant. We qualify such a notion as a half-condition compared to the condition on in-neighbours. This is the reason for the terminology of a sesquimodule*.

**Sesqui-* is a Latin prefix for one-and-one-half.

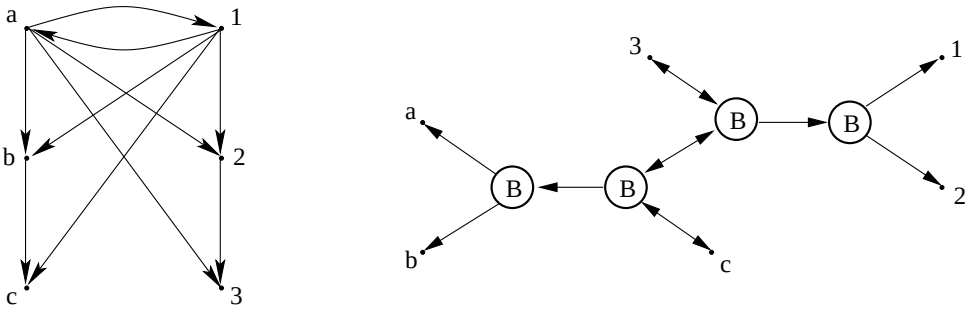
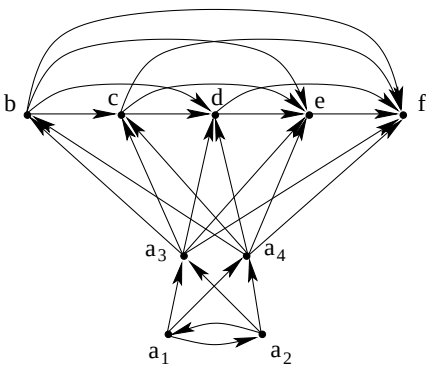
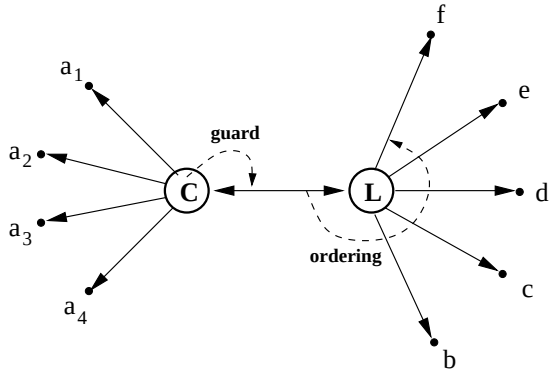


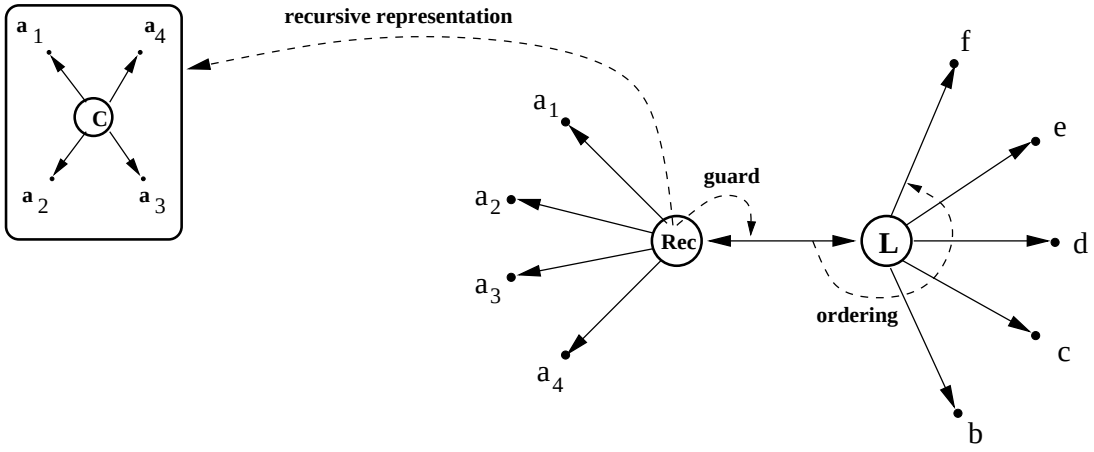
Figure 3.8: A modular prime digraph with its sesquimodular decomposition tree. The label "B" stands for *basic* quotient nodes.



(a) A directed graph.



(b) Its weakly partitive crossing sesquimodular decomposition tree.



(c) Its union-difference sesquimodular decomposition tree.

Figure 3.9: Sesquimodular decomposition. Some sesquimodules of the digraph are: all subsets of $A = \{a_1, a_2, a_3, a_4\}$, $A \cup \{b\}$, $A \cup \{b, c\}$, $A \cup \{b, c, d\}$, $\{b, c, d\}$, $\{b, c, d, e\}$, $\{b, c, d, e, f\}$, $\{c, d, e\}$, ...

In Figure 3.8, we show that the generalization of clans to sesquimodules is proper. An example of sesquimodular decomposition tree is given in Figure 3.9.

Besides, notice that the condition on the out-neighbours of a sesquimodule defines exactly an unordered-module (cf. Section 2.3.2). Then, a sesquimodule can also be seen as a vertex subset which is at the same time a genuine-module and an unordered-module. For this reason sesquimodules are also called *bi-dules* in French. Finally, notice that the sesquimodules of an undirected graph or a symmetric 2-structure are exactly its modules, and also its clans. We now distinguish two cases, following some structural properties of the family of sesquimodules, depending on whether the given structure is a digraph, or a 2-structure.

3.3.1 Sesquimodular Decompositions

We begin with the two leading facts of this section.

Proposition 3.1 *The sesquimodules of a digraph form a weakly partitive crossing family. Furthermore there are no circular nodes in its decomposition tree.*

Proof: Let $G = (X, A)$ be a digraph. Clearly, the trivial vertex subsets are sesquimodules of G. Let P and Q be two crossing sesquimodules of G. By Proposition 3.2 (right below), $P \cup Q$ and $P \setminus Q$ are sesquimodules of G. We only need to prove that $R = P \cap Q$ is a sesquimodule.

It follows directly from definition that $\forall x, y \in R,\ \forall c \in C,\quad N_c^-(x) \setminus R = N_c^-(y) \setminus R$.

From now on, the fact that a 2-structure which is also a digraph must have at most two arc colours will be important. W.l.o.g. suppose the two possible colours are 0 and 1. From definition of the sesquimodule P, there is a partition of $\overline{P}$ into *two* parts $\{S, T\}$ such that every vertex $x \in P$ holds $\{N_0^+(x) \setminus P, N_1^+(x) \setminus P\} = \{S, T\}$ (this is an equality between sets). Likewise, there is a 2-partition of $\overline{Q}$ into $\{S', T'\}$ such that, for all $x \in Q$, $\{N_0^+(x) \setminus Q, N_1^+(x) \setminus Q\} = \{S', T'\}$. Now take a vertex $x \in R$: it belongs to both P and Q. Since P and Q cross, there is a vertex $s \in \overline{P \cup Q}$.

From $\{N_0^+(x) \setminus P, N_1^+(x) \setminus P\} = \{S, T\}$, suppose w.l.o.g. that $s \in S$. Likewise, from $\{N_0^+(x) \setminus Q, N_1^+(x) \setminus Q\} = \{S', T'\}$, suppose w.l.o.g. that $s \in S'$. Moreover, vertex s has only two choices: either $s \in N_0^+(x)$ or $s \in N_1^+(x)$. But then, $N_0^+(x) \setminus R$ is either $S \cup S'$ or $T \cup T'$, accordingly. Hence, $\{N_0^+(x) \setminus R, N_1^+(x) \setminus R\} = \{S \cup S', T \cup T'\}$.

Finally, a circular sesquimodule quotient node would be a complete one. □

Proposition 3.2 *The sesquimodules of a 2-structure form a union-difference family. Furthermore there are no circular nodes in its decomposition tree.*

Proof: Let $G = (X, C)$ be a 2-structure. Clearly, the trivial vertex subsets are sesquimodules of G. Let P and Q be two overlapping sesquimodules of G. It follows directly from definition that $P \cup Q$ is a sesquimodule. We only need to prove that $R = P \setminus Q$ is also a sesquimodule.

First suppose that there exist an exterior vertex $s \notin R$ and two vertices $x, y \in R$ s.t. $C(s, x) \neq C(s, y)$. Since P is a sesquimodule s belongs to $P \cap Q$. Moreover, that P and Q overlap implies there is a vertex t belonging to $Q \setminus P$. Notice that $s, t \in Q$ and $x, y \notin Q$. Since Q is a sesquimodule, $C(t, x)$ would be different from $C(t, y)$. But then P is no more a sesquimodule as $t \notin P$ and $x, y \in P$. Contradiction.

To finish, let us introduce a notation. For every vertex triplet x, y, z we denote the fact $C(x, y) = C(x, z)$ by $x|yz$. Then, the second condition in the definition of a sesquimodule M, namely $\forall x, y \in M,\ \forall c \in C,\ \exists d \in C,\ \ N_c^+(x) \setminus M = N_d^+(y) \setminus M$, can be translated by: for all $x, y \in M$, for all $s, t \notin M$, $x|st \Leftrightarrow y|st$.

Now let $x, y \in R$ and $s, t \notin R$. We need to prove that $x|st \Leftrightarrow y|st$. If none of s and t belong to P, that P is a sesquimodule allows to conclude. If both s and t belong to Q, that Q is a sesquimodule allows to conclude. By symmetry, the only remaining case is when $s \in P \cap Q$ and $t \notin P \cup Q$. In this case, let $u \in Q \setminus P$. Since P is a sesquimodule, we already have $x|tu \Leftrightarrow y|tu$, but we would like the same property with vertex u replaced by vertex s. For this, notice that $x \notin Q$, but $s, u \in Q$, and Q is a sesquimodule. Therefore, $x|su$. Likewise, $y|su$. Then, we have $(x|tu \Leftrightarrow y|tu) \wedge (x|su) \wedge (y|su)$, which is equivalent to the desired property.

Finally, a circular sesquimodule quotient node would be a complete one. □

Basically, with a digraph (resp. a 2-structure), one can associate a labelled tree which represents exactly the family of sesquimodules of the digraph (resp. the 2-structure).

Definition 3.4 (Sesquimodular Decomposition Tree)
The *sesquimodular decomposition tree* of a digraph refers to the decomposition tree of the family of its sesquimodules, which is weakly partitive crossing. The *sesquimodular decomposition tree* of a 2-structure refers to the decomposition tree of the family of its sesquimodules, which is a union-difference family. They are cross-free decomposition trees (cf. Definition 1.1).

The discourse of the previous chapters applies on sesquimodular decomposition trees. In particular, we can proceed with them using a similar approach as that with modular graph decomposition (see Section 2.3.1). To begin with, it is clear that from the only knowledge of the sesquimodular decomposition tree, all sesquimodules of the digraph (resp. the 2-structure) can be retrieved, that is

Theorem 3.6 (Sesquimodular Decomposition Theorem) *There is a unique labelled tree associated to a 2-structure, a so-called sesquimodular decomposition tree, such that all sesquimodules of the 2-structure can be enumerated from the tree without the knowledge of the 2-structure. In particular, the claim holds if the 2-structure is a digraph.*

Besides, we have also seen that one of the major interests – and, from a certain algorithmic point of view, *the* major interest – of modular decomposition is that the computation of the modular decomposition tree of a given graph is polynomially solvable. Here also, we would like to highlight that there are polynomial algorithms that, given a digraph (resp. 2-structure), compute the corresponding sesquimodular decomposition tree. More precisely,

Theorem 3.7 *([15, 21]) The sesquimodular decomposition tree of a 2-structure over n vertices can be computed in $O(n^7)$ time. This can be sped up to be in $O(n^3)$ time if the 2-structure is a digraph.*

However, we will not discuss in detail this matter, which goes slightly beyond the purposes of our composition. Instead, we will keep our focus on representability issues. More specifically, we have seen in the previous chapter that one of the nice aspects of modular decomposition is that from the only knowledge of the decomposition tree, one can build back the graph itself! This is not the case for many other graph decompositions, including notorious decompositions such as tree decomposition, branch decomposition, and rank decomposition. Basically, the end of this section can be seen as an attempt to answer to the following question.

Question 1: *Can the sesquimodular decomposition tree of a digraph (resp. a 2-structure) be labelled with further information in such a way that the digraph (resp. 2-structure) can be built back from the only knowledge of the tree and its labels?*

At the same time, we would like to investigate a second question. The motivation comes from Corollary 2.6 at the end of Section 2.3.1, which states that the degenerate case of modular graph decomposition, on the so-called cographs, leads to a sub-linear encoding of the input cograph. This case occurs precisely when the modular decomposition tree of some given graph has no prime node. Then, the graph will be called a cograph, and the modular decomposition tree of the graph its cotree. It is then proved that one can label the cotree in such a way that the cograph can be built back from the only knowledge of the cotree and its labels. Moreover, the total size of the labelled cotree turns out to be proportional to the number of vertices of the cograph, while the size of the cograph can be quadratic on this number. Accordingly, one says that the cotree is a sub-linear encoding

of the corresponding cograph. This is an important fact for, e.g., computational purposes in graph theory. Here, we would like to investigate Question 2 below, which can be seen as an extended version of Question 1 on the particular instance of what we call a totally sesquimodular decomposable digraph.

Definition 3.5 (Totally Sesquimodular Decomposable 2-Structure & Digraph) A 2-structure is *totally sesquimodular decomposable* if there are no prime nodes in its sesquimodular decomposition tree. In particular, the definition holds when the 2-structure is a digraph.

Question 2: *Can a digraph which is totally sesquimodular decomposable be represented using linear space on the size of its vertex set?*

Before continuing, it is important to highlight that we have failed so far in giving an answer to either Question 1 or Question 2, no matter if the answer were positive or negative. Instead, the leading idea of the remaining of this section should be seen as an attempt to investigate these two questions, with a tendency towards positive answers. To this purpose, the following property will be essential. However, since this is a property with a long statement, let us first give the idea behind the claim. Roughly, our aim is to encode every quotient of a sesquimodular decomposition tree with additional information in such a way that we can partly build back the digraph (resp. the 2-structure) from the knowledge of the sesquimodular decomposition tree plus the encoding inside that quotient. Then, the whole digraph (resp. 2-structure) can be retrieved by juxtaposing the parts which are thus given by the quotients of the tree. This is a quite common practice in topics around modular decomposition.

Besides, recall that a quotient in the sesquimodular decomposition tree of a 2-structure $G = (X, C)$ has a ground set of the form $Y = \{X_1, X_2, \ldots, X_k\}$, where $\{X_1, X_2, \ldots, X_k\}$ is a partition of the vertex set X (cf. Definition 1.2 in Chapter 1). Basically, from the following property we can obtain the above mentioned encoding when the quotient is such that every X_i $(1 \leq i \leq k)$ contains one and only one vertex $v_i \in X$.

Proposition 3.3 *The family of sesquimodules of a 2-structure $G = (X, C)$ is*

- *complete without guard if and only if the colour function C is partially constant:*

$$C(y, z) = C(y, t) \text{ for all } y \in X \text{ and } z, t \in X \setminus \{y\}.$$

- *complete with a guard $x \in X$ if and only if the colour function C satisfies:*

  1. $C(x, y) = C(x, z)$ *for all* $y, z \in X \setminus \{x\}$,

2. $C(y,z) = C(y,t)$ *for all* $y \in X \setminus \{x\}$ *and* $z,t \in X \setminus \{x,y\}$,
3. $C(y,x) \neq C(y,t)$ *for all* $y \in X \setminus \{x\}$ *and* $t \in X \setminus \{x,y\}$.

- *linear without guard w.r.t. the ordering* $X = (x_1, x_2, \ldots, x_n)$ *if and only if the colour function* C *satisfies:*

$$\forall x_i \in X,\ \exists c \neq d,\ \Big(1 \leq k < i \ \Rightarrow\ C(x_i, x_k) = c\ \Big) \wedge \Big(i < k \leq n \ \Rightarrow\ C(x_i, x_k) = d\ \Big).$$

- *linear with a guard* $x \in X$ *w.r.t. the ordering* $X \setminus \{x\} = (x_1, x_2, \ldots, x_{n-1})$ *if and only if the colour function* C *satisfies:*

1. $C(x,y) = C(x,z)$ *for all* $y,z \in X \setminus \{x\}$,
2. $\forall x_i \in X \setminus \{x\},\ \exists c \neq d,$ *such that*

$$\Big(1 \leq k < i \ \Rightarrow\ C(x_i, x_k) = c\ \Big) \wedge \Big(i < k < n \ \Rightarrow\ C(x_i, x_k) = d\ \Big),$$

3. *for all* $x_i \in X \setminus \{x\}$, *when any among* x_{i-1} *and* x_{i+1} *exists, we have both* $C(x_i, x) \neq C(x_i, x_{i-1})$ *and* $C(x_i, x) \neq C(x_i, x_{i+1})$.

Proof: For each of the four items in the statement, it is straightforward to check that the corresponding condition is sufficient, namely that the right hand-side of the iff implies the left hand-side. We only give the proof for the other direction in each case.

For the first item, if every vertex subset is a sesquimodule, then in particular every vertex subset that is quasi-trivial (i.e. of cardinality $|X| - 1$) satisfies the "in-neighbours condition" of a sesquimodule. From this observation it is straightforward to conclude.

For the second item, if the family of sesquimodules is complete with a guard, then $|X| \geq 4$ otherwise the family would be complete without guard. By examining $G[X \setminus \{x\}]$ using what has just been proved in the last paragraph, we deduce the condition number two. By observing that $X \setminus \{x\}$ is a sesquimodule whenever x is a guard we deduce the condition number one. Now suppose that the condition number three is violated. We can deduce a contradiction to the fact that x is a guard as follows. There exist vertices y and t such that x, y, t are pairwise distinct and $C(y,x) = C(y,t)$. Let z be any other vertex: $z \in X \setminus \{x,y,t\}$, which exists since $|X| \geq 4$. The "out-neighbours condition" of a sesquimodule, when applied on the sesquimodule $\{y,z\}$, states that as long as $C(y,x) = C(y,t)$, we must have $C(z,x) = C(z,t)$. Combining this and the conditions number one and two we have so far that the colour function C is almost partially constant, except for t: for every vertex $u \neq t$ and for all $v,w \in X \setminus \{u\}$, we have $C(u,v) = C(u,w)$. As for vertex t, we have that $C(z,x) = C(z,y)$ from the previous

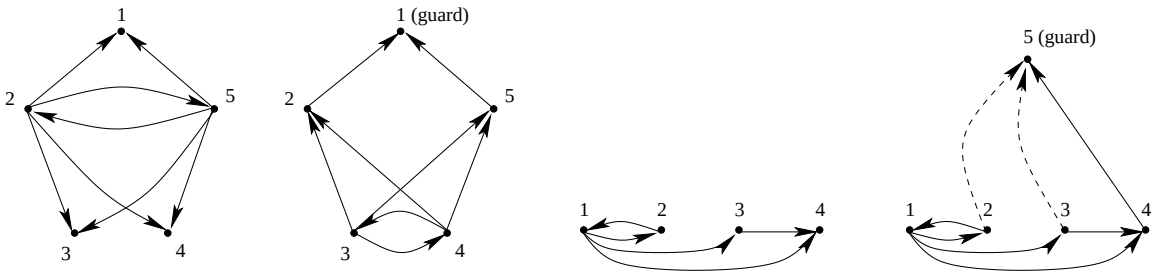
sentence, then, applying the "out-neighbours condition" on the sesquimodule $\{z, t\}$ allows to conclude that $C(t, x) = C(t, y)$. But then we are exactly in the case of the first item of the statement of the proposition, among other things x is not a guard. Contradiction.

For the third item, let us consider vertex x_i. Using the "in-neighbours condition" of a sesquimodule on both $\{x_1, x_2, \ldots, x_{i-1}\}$ and $\{x_{i+1}, x_{i+2}, \ldots, x_n\}$, we obtain the existence of two colours c and d (not necessarily different) such that $(1 \leq k < i \Rightarrow C(x_i, x_k) = c)$ and $(i < k \leq n \Rightarrow C(x_i, x_k) = d)$. This can be done for any vertex x_i. After this step, suppose that for some vertex x_i with $1 < i < n$, the corresponding colours c and d are equal: $c = d$. Then, using the "out-neighbours condition" of the sesquimodule $\{x_2, x_3, \ldots, x_{n-1}\}$, we can conclude that $\{x_1, x_n\}$ is a sesquimodule. This contradicts the fact the family of sesquimodule is linear w.r.t. the ordering $X = (x_1, x_2, \ldots, x_n)$.

For the last item, we prove the conditions number one and number two in a similar way like before. Then, like before again, we can prove condition number three using an argument by contradiction as follows. Suppose that the condition is violated. By symmetry of the ordering $X = (x_1, x_2, \ldots, x_{n-1})$, we can suppose w.l.o.g. that there exists x_i with $1 < i \leq n - 1$ such that $C(x_i, x) = C(x_i, x_{i-1})$. But then we can prove that $\{x_1, x_2, \ldots, x_{i-1}\} \cup \{x\}$ is a sesquimodule using the "out-neighbours condition" of the sesquimodule $\{x_i, x_{i+1}, \ldots, x_{n-1}\}$. Contradiction. □

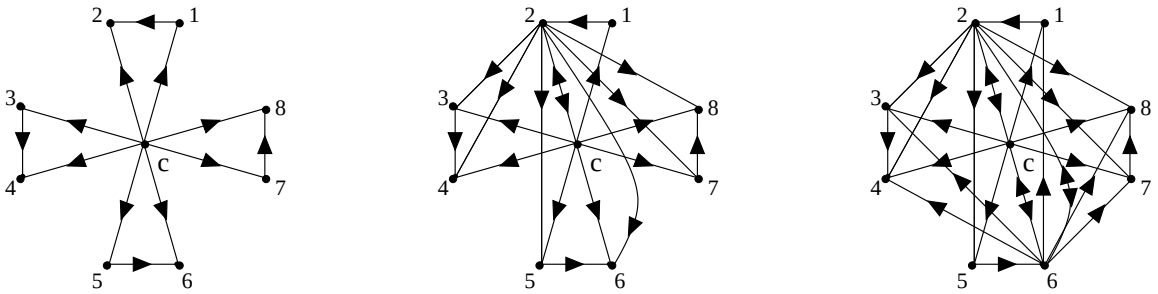
Basically, from Proposition 3.3, it is straightforward to encode a 2-structure using a very efficient space when the family of its sesquimodules is complete or linear (with or without a guard). For instance, if the family is complete without guard, then it suffices to encode for every vertex of the 2-structure the colour of one of its out-going arcs: the colour of all other out-going arc from that vertex is the same, according to the first item of Proposition 3.3. A synopsis is given in Figure 3.10.

Let us now consider a quotient $\mathcal{Q}(u)$ w.r.t. node u of the sesquimodular decomposition tree of a 2-structure $G = (X, C)$. Let $Y = \{X_1, X_2, \ldots, X_k\}$ be the ground set of $\mathcal{Q}(u)$. As previously said, Proposition 3.3 applies when $\mathcal{Q}(u)$ is such that every X_i $(1 \leq i \leq k)$ contains one and only one vertex $v_i \in X$. In the modular decomposition of graphs, such a result is sufficient to answer to similar questions as Questions 1 and 2. The reason is roughly because if A and B are modules of undirected graph G – here viewed as a 2-structure – then all arcs in G with an extremity in A and another extremity in B have the same colour. Then, even when the X_i's get an arbitrary size, each of them can still be represented by a vertex, say any $v_i \in X_i$. After encoding the adjacency of v_i, we can skip encoding the adjacency of the other vertices of X_i: it is the same as that of v_i. Unfortunately, such a property does not necessarily hold for sesquimodular decomposition. For instance, every digraph given in Figure 3.11 has the same family of sesquimodules.

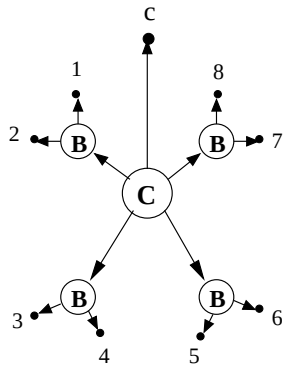


i. **ii.** **iii.** **iv.**

Figure 3.10: 2-structures where the family of sesquimodules is: complete without guard (i.); complete with a guard (ii.); linear without guard (iii.); linear with a guard (iv.). In the three first examples, the 2-structure is also a digraph. The dashed arrows in (iv.) represents a third colour of the 2-structure.



i. A digraph. **ii. A digraph.** **iii. A digraph.**



iv. Their sesquimodular decomposition tree.

Figure 3.11: Three digraphs with the same sesquimodular decomposition tree.

Even so, the colours of the out-going arcs from the sesquimodule $\{1, 2\}$ can have several configurations. Besides, it is straightforward to add an arbitrary number of "petals" to these examples, where the "petals" refer to the sesquimodules of the form $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$, ...

Basically, in these examples, we might end up storing, for every quotient $\mathcal{Q}(u)$, one colour per vertex of each X_i. This would imply $|X|$ colours per quotient $\mathcal{Q}(u)$. As the number of possible quotients could be proportional to $|X|$, we might end up with a total encoding space proportional to $|X|^2$, namely the same space complexity as that of the 2-structure itself. The question of characterizing the quotients of sesquimodular decomposition trees from the scope of obtaining an efficient encoding of the corresponding 2-structures seems to be more difficult. The difficulty seems to remain intact even for the restriction to digraphs.

It is now time to close the current part of the thesis and give way for more applicative assumptions. We have until now discussed on a general technique for obtaining various representation theorems for set families satisfying some closure axioms. We have also seen that those results are crucial to define various decomposition schemes. All of them are modelled using a graph theoretic point of view, with possibly a slight generalization to 2-structures. In the next part of the thesis, among other things, we will give some concrete motivations for the study of decomposition schemes in graph theory. Before this, let us recall that a synopsis of all results on the representation of set families which have been discussed so far can be found in Figures 1.4 and 1.5 at the end of Chapter 1.

Part II

Decomposition and Divide-and-Conquer Algorithms

On magazines for gum chewers:
"Each one encourages you to think you belong to an elite clique,
so advertisers can appeal to your ego and get you to cultivate an image
that sets you apart from the crowd. It's the divide and conquer trick."
Bill Watterson, 1992.
(*The complete Calvin and Hobbes – Book three.* 1st ed., Andrews McMeel Publishing, 2005.)

Overview of Part II

Divide-and-conquer is a general strategic principle which, in common lore, refers back to a long history. It ranges, among others, from Sun Tzu to Niccolò Machiavelli, and more recently, to René Descartes. The practice involves many famous names including Gaius Julius Caesar, the Habsburgs, and also Stargate or Teenage Mutant Ninja Turtles. Even nowadays, it is still widely used as reference to a combination of political, military and economic strategies. It consists in gaining and maintaining one's power by means of breaking up larger concentrations of power into chunks that individually will be less powerful than one's self. However, this approach is scarce in reality since it is difficult to break up existing power structures. Thus, the divide-and-conquer principle, in practice, will also refer to its restriction to the more realizable act of preventing small power groups from linking up together.

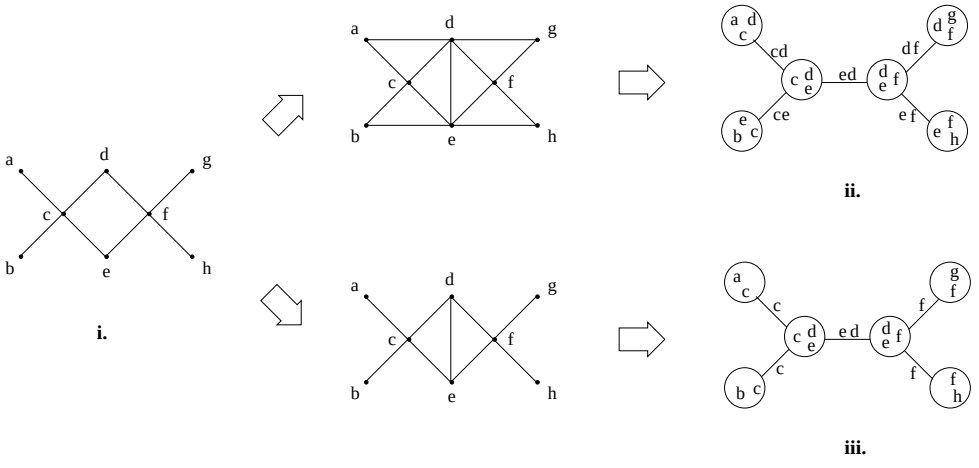
The principle came into computer science in the early 1960s with the publication of Anatolii Karatsuba's fast multiplication algorithm. In this field however, rather than the latter practice, it will be exactly the original divide-and-conquer principle that is solicited. A bit more precisely, the now classical divide-and-conquer algorithmic framework can be summarized as dividing the input problem into some sub-problems; conquering them by making recursive calls; eventually uniting the sub-solutions into a global solution. Thus, it is more of a divide-conquer-and-unite approach, from a certain standpoint. Also from the same standpoint, one might want to consider the consequences of omitting the unification process in the general divide-and-conquer principle: the decline and fall of the Roman Empire could be seen as a motivation for such a consideration. However, in computer science, such a discussion is forfeit: the need of a unification process is final, and sometimes it is even the very core of the question to be solved. Such an example could be the very standard merge-sort. According to this, it is quite intriguing why the paradigm is still widely denoted by the name of divide-and-conquer algorithms.

Beside those vocabulary matters, it is quite popular that one of the major drawbacks of the divide-and-conquer algorithmic paradigm comes up when, for some problems, many of the sub-problems overlap. If such a situation occurs, it is more interesting to use again

the sub-solutions, as many times as possible. Such a practice is so-called memoization and can also be seen as the basic idea of a dynamic programming. Also, the best known examples of divide-and-conquer algorithms, beside some standard sorting algorithms, are probably dynamic programming algorithms.

In algorithmic graph theory as in any other field, the divide-and-conquer paradigm has the prior requirement of being able to divide an input graph instance, not in an arbitrary way, but in such a way that a unification stage is possible after the corresponding "conquests". For this aim many tools have been developed, with the most notorious being, in lexicographic order: articulation vertex, block, bridge, cut, cut set, edge-connectivity, flow, isthmus, maxflow, mincut, separator, vertex-connectivity. Here, the two last notions are related to a more or less academic yet certainly popular and celebrated notion, the so-called tree-decomposition. It is the close associate of the notion of a graph treewidth, and is roughly the completion of an undirected graph G into a chordal graph having a clique number as small as possible. Then, the chordal graph can be bijectively mapped into a tree, a so-called clique tree. In such a tree, the node labels correspond to the maximal cliques of the chordal graph, while the edge labels are the intersection of the labels of the two incident nodes. To that minimum clique number (of the chordal graph) minus one is given the name of the treewidth of G, to the clique tree of such a chordal graph is given the name of a (possible) tree-decomposition of G.

Actually, from any tree-decomposition of G, there is a straightforward transformation to result in another tree-decomposition of G holding the following fact, which is essential for algorithmic purposes: every edge label of the tree is a minimal separator of G (see, e.g., [8] for a recent and general survey, or [12, 109, 110] for more foci on separators). An example could be:



Example of tree-decomposition: i. A graph G. ii. A tree-decomposition of G. iii. A tree-decomposition of G which displays only minimal separators of G.

Now, a separator by definition is a vertex subset whose removal splits a connected graph into several disconnected parts. This turns out to be a convenient way to enhance the use of divide-and-conquer techniques. As a result, many optimization problems on G can be solved by those techniques if a tree-decomposition of G is given (see, e.g., [8], and also [79, Chapter 10. Extending the Limits of Tractability] for a more detailed overview). In this topic, the dividing stage of the divide-and-conquer algorithms follows directly from the input tree-decomposition. On the other hand, the unification stage is usually more complex, and constitutes the core question to be solved, like with the case of merge-sort. Finally, it is important to highlight that the same discourse actually applies for various other graph decomposition schemes as well. For instance, we have seen in the previous part of the thesis in Section 2.3.1 that modular decomposition is also a good candidate for this perspective. Other major examples include split decomposition, branch decomposition, as well as the relatively recent clique decomposition.

At the same time, an original aspect of studying divide-and-conquer algorithms could come from the converse of what has just been said. Actually, it is a quite common fact that some algorithms, while theoretically efficient, may be notorious for being complicated and impractical. For instance, in order to compute the modular decomposition tree of a given graph, the first algorithms with a linear runtime were found more than one decade ago [37, 88]. Nonetheless, there still are ongoing large efforts in simplifying or giving alternatives to those already optimal algorithms [108]. To give another example, for the enumeration of all common intervals of two permutations, the first linear time algorithm [111] was already found (in 1996) almost one decade before the proposition of an alternative [4]. In some of those situations, the goal would be to improve the robustness of the existing algorithm and/or to lead to a better understanding of its combinatorial structure. An original approach here is to consider the algebraic aspects of the existing algorithms in order to find insights in some combinatorial properties related to the statements of the initial problem. In other words, a deeper study of some (infamous) algorithms may help in the discovery of unexpected combinatorial results. In particular, if the corresponding algorithm follows a divide-and-conquer approach, then the probability to run into a decomposition result will be increased. It is, e.g., the case for what will be presented in Chapter 5, where we in fact revisit the aforementioned enumeration of common intervals [111]: the corresponding decomposition scheme was proposed in [19], long after the discovery of the algorithm.

The current part of the thesis is meant to emphasize the above mentioned duality between combinatorial decompositions and algorithm design. We investigate three cases. In each case, we insist on the combinatorial aspects of the corresponding problem, before effectively giving an algorithmic solution. A leading idea here could be seen as the fact

these algorithms rely on some underlying structural properties of the instances defined by the problem to be solved. In reality it may happen that the very structural properties are found after the discovery of the corresponding algorithm (e.g., Lemma 5.2 in Chapter 5). However, even in those cases, we still address combinatorial issues first, and skip the discussion on the converse view of the duality, namely on how analysing the algorithm gives hints in pointing out those combinatorial issues.

Let us specify more the structure of the current part.

We address in Chapter 4 two graph problems arising from computational biology: the so-called common connected component enumeration and the so-called cograph sandwich problem. For the first problem we give a unique algorithm which has various runtime depending on the data-structure we use. This runtime, for all configurations of the data-structures to be used, is sub-quadratic on the number of vertices and edges of the input graphs. Besides, our solution can be seen as a unified viewpoint of various approaches for solving the common connected component enumeration using the divide-and-conquer paradigm. Also, unlike the case with merge-sort, the unification stage of our algorithm is straightforward, while the dividing stage is more tricky. To achieve the sub-quadratic runtime we introduce a new graph search, the so-called competitive graph search, which has a sub-linear runtime on the size of its input graph. Then, for the second problem, namely the cograph sandwich problem, instead of designing an algorithm from scratch, we show how one can derive an efficient solution from the solution of the first problem and from some structural analysis of the involved notions.

In Chapter 5, we deepen a particular instance of the common connected component enumeration, the so-called common interval enumeration. We revisit a tricky algorithm designed by T. Uno and M. Yagiura [111], which can be made to run in linear time on the number of vertices (by the addition of a simple one-line routine). While simple to implement and having a very fast performance, Uno-Yagiura algorithm has been quite notorious for its tricky correctness and complexity analysis. Our study discusses in detail the correctness and complexity issues of this algorithm. At the same time, we point out strong structural properties of the involved notions. Among other things, we establish in Lemma 5.2 an intersecting submodular property for a very general framework of modular decomposition. Various decomposition issues around common intervals are also discussed in the same chapter. We also give an application of Uno-Yagiura algorithm in the modular decomposition of graphs (if a so-called factoring permutation is given).

We close the thesis with an attempt in Chapter 6 of giving a unified viewpoint of, at the same time, modular decomposition, split decomposition, bijoin decomposition, as well as various recent advances in algorithmic graph theory including clique decomposition,

NLC-decomposition, and the newly introduced rank decomposition. However, we restrict our discussion on this topic to some algorithmic issues around the new notion of an H-join decomposition, as well as those around the restriction of H-join decompositions to rank decomposition. We point out how, in a so-called fixed-parameter tractable (FPT) single exponential time, one can enhance our decomposition scheme into an object amenable to the divide-and-conquer paradigm. When applied to rank decomposition, our enhancing process runs in FPT time with single exponential on the rankwidth of the input graph. We also exemplify our approach on a dynamic programming solution for the NP-hard problem of finding the clique number of a graph, first under our framework in general, then under the framework of rank decomposition in particular. Similarly as in the cases of tree-decompositions and merge-sort, the dividing stage of our method is straightforward while the unification stage is the crux of the algorithm.

This part of the composition is based on [19, 20, 22].

Chapter 4

Common Connectivity

This chapter is based on [20].

The algorithmics series begins with a special focus on connected components of graphs. More precisely, we investigate two problems, both arising from computational biology. The first problem consists of, given two graphs over the same vertex set, finding the coarsest partition of the vertex set into subsets which induce connected subgraphs in both input graphs. The second problem is an instance of sandwich graph problems: given a partial subgraph G_1 of G_2, find a completion of G_1 into a partial subgraph G of G_2 (sandwich) such that G is a cograph. They will be called *common connected component* and *cograph sandwich* problems, respectively.

The first problem was studied in [13] for its connection with some gene structure. There, one graph can be obtained by the distance between the genes of some genomic sequence, taken with respect to a given threshold. The other graph can be any graph on the same set of genes, for example one generated by some chemical reaction. Besides, this problem arises also from comparative genomics, such as in the search of the so-called gene-teams [2]. In this topic, both graphs are obtained by the former method, namely by the distance w.r.t. a threshold among the genes in a genomic sequence. As for the second focus of the chapter, the class of sandwich graph problems was defined in [63]. However, most instances of this class are NP-complete, and only some few polynomial cases are known. They include cograph sandwiches [63] and module sandwiches [11, 26].

Beside applicative purposes, the work presented in this chapter has also a theoretical motivation. Indeed, it is folklore that, without specific assumptions, the divide-and-conquer approach helps with designing algorithms running in quadratic worst case time. Moreover, most of the classical speed up techniques in the literature consist in holding some condition on the "conquer" part (i.e. the recursive calls). This is the case for, e.g., the standard merge-sort and more complex procedures such as the median computation [7] or algorithms derived from the planar separator theorem [84]. Actually, even when no

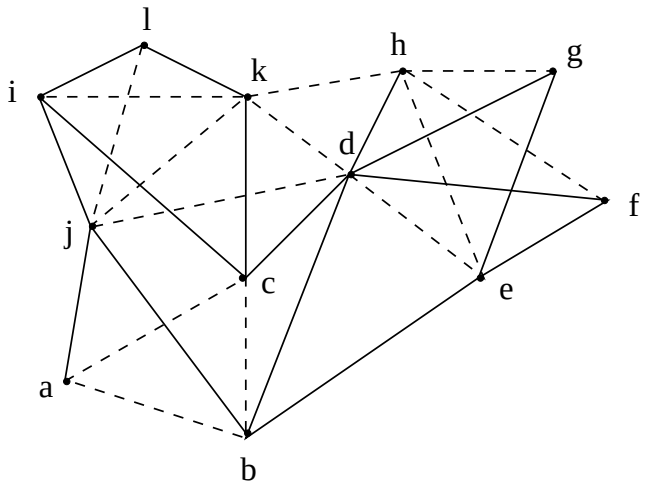


Figure 4.1: Two graphs over the same vertex set. Non-trivial common connected sets are: $\{d, e, f, h\}$, $\{d, e, g, h\}$, $\{d, e, f, g, h\}$, and $\{i, j, k, l\}$. Solution for common connected component problem: $\{\{a\}, \{b\}, \{c\}, \{d, e, f, g, h\}, \{i, j, k, l\}\}$.

condition is placed on the recursive calls, it is also established that cutting down the "divide" and "unite" part improves the global computing time [1, 82]. However, applied examples of this paradigm are scant in the literature. Our computations turn out to be such examples. More details on those matters will be given in Section 4.2.1.

The chapter is organized as follows. We will begin with the study of some structural behaviour of the so-called common connected components of two graphs. Then, using a new and sub-linear graph search, the so-called *competitive graph search*, we depict a sub-quadratic enumeration of common connected components, and solve the first problem. For cograph sandwiches, we study some structural properties in relation with the common connected component problem. After this, we will see how an efficient computation for cograph sandwiches follows directly from structural analysis.

4.1 Some Structural Aspects of Common Connected Components

We first focus on structural issues of the first problem, on the so-called common connected components.

Definition 4.1 (Common Connected Component) Given two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, a *common connected set* A of (G_1, G_2) is a vertex subset of V such that both $G_1[A]$ and $G_2[A]$ are connected. A *common connected component* is a maximal common connected set.

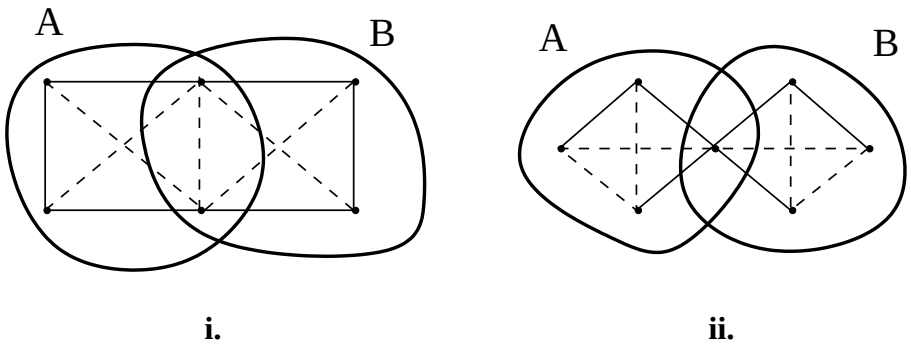


Figure 4.2: i. One graph is not cycle free and the common connected sets fail the intersection closure. ii. One graph is not a path and the difference closure fails.

Let us begin with an observation which, though simple, has many consequences.

Proposition 4.1 *Clearly, the family of common connected sets of a pair of two given graphs over the same vertex set is closed under the union of its overlapping members. Moreover, if both graphs are forests, then the family is an intersecting family. Finally, if both forests are forests of paths, then the family is weakly partitive. This hierarchy is tight.*

Proof: Let G_1 and G_2 be two forests over the same vertex set. Let A and B be two common connected sets of (G_1, G_2). We first consider two vertices $a, b \in A \cap B$. By definition there exist a chain from a to b in $G_1[A]$, and a chain from a to b in $G_1[B]$. Since G_1 is a forest this chain is unique and therefore included in $A \cap B$.

Now, let us suppose that both G_1 and G_2 are forests of paths, and that A and B overlap. We consider $x, y \in A \setminus B$. Suppose that x, y are not connected in $G_1[A \setminus B]$. Clearly, both of them belong to A, which is a common connected set. Then, the (unique) chain in G_1 linking x to y goes through a vertex z in $G_1[A \cap B]$. By definition, it exists at least a vertex $t \in B \setminus A$. Clearly, both z and t belong to B, which is a common connected set. But then the connected component of G_1 containing x, y, z, and t is not a chain. Contradiction.

Finally, Figure 4.2 proves that the corresponding hierarchy over the graph classes is tight. □

From Proposition 4.1 and what has been said in the first part of the thesis (more precisely in Section 2.2.1), there exists a decomposition scheme of the input graphs into their common connected sets when both graphs are forests of paths. Then, recursiveness can be conducted easily through the quotients of the corresponding decomposition tree. In fact, this corresponds to a well-studied case and the computation of the decomposition

tree can be done in $O(|V|)$ time by adding some slight modifications to the computation of common intervals of two permutations [111]. (Here, the paths of each forest can be put together to result in a permutation.) The next chapter will deepen those matters.

A step forward in generality, when dealing with arbitrary forests, the structure of the intersecting family of common connected sets will be instructive. However, we will mostly use the following partitioning lemma, which, though still simple, can be seen as a stronger structural property of common connected sets. Also, the forest case will be the most basic case of the algorithm depicted in this chapter, as we will show how our forest-based algorithm turns out to be suited for even the general case of arbitrary graphs, thanks to tree spanners. Accordingly, an important idea throughout the chapter is that, we will roughly "think forest", even in non-forest cases.

Lemma 4.1 *([61]) If there are no edges in the cut $\{X, V \setminus X\}$ w.r.t. one graph among $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, then, the common connected components of (G_1, G_2) are exactly those of $(G_1[X], G_2[X])$ plus those of $(G_1[V \setminus X], G_2[V \setminus X])$.*

Basically, the lemma has the two following algorithmic consequences. Unless the solution is trivial (both graphs are connected), one can always divide an instance of the common connected component problem into several parts. Moreover, in order to find this division, it suffices to determine *some* connected components, and not necessarily *all* of them. The first observation leads directly to a solution based on a divide-and-conquer approach. More interestingly, the second observation is the basic idea for our speed up result on common connected components. Indeed, in order to find the above mentioned division, we will consider the problem of, given a graph and a list of one representative vertex per connected component, visiting all connected components but the largest. We will depict how a so-called *competitive graph search* can solve the problem in linear time on the size of the visited vertices and edges. This complexity is sub-linear in the sense that it is sub-linear on the size of the problem, namely the total size of the input and the output. Of course, the complexity is clearly not sub-linear on the size of the output. Also, note that the size of the largest component might be very close to that of the initial graph. In this case the competitive graph search will record a small time complexity.

4.2 Some Algorithmic Aspects

4.2.1 Divide-and-Conquer Algorithmic Framework – the Basics

Divide-and-conquer is a now standard algorithmic framework and can be found in any classical algorithmic handbook, e.g., [29, 90]. In this thesis, we address the following

formalism.

Definition 4.2 (Divide-and-Conquer Algorithm) Let $\mathcal{P}$ be a problem on a set $\mathcal{S}$ of data structures, and let *Size* be a function from $\mathcal{S}$ to $\mathbb{R}^+$. $\mathcal{H}$ is a *divide-and-conquer algorithm with respect to Size solving* $\mathcal{P}$ if:

- there exists a set $\mathcal{T} \subseteq \mathcal{S}$ of trivial inputs on which $\mathcal{H}$ solves $\mathcal{P}$ in $O(1)$ time;
- any $S \in \mathcal{S}$ with $Size(S) \leq 1$ is a trivial input, namely $S \in \mathcal{T}$;
- for all $S \notin \mathcal{T}$, $\mathcal{H}(S)$ proceeds as follows.
  - (divide) first it divides the input S into some sub-instances $S_1, S_2, \ldots, S_k$ with $Size(S_i) > 0$ for all i and $Size(S_1) + Size(S_2) + \cdots + Size(S_k) \leq Size(S)$,
  - (conquer) then it recurses with $\mathcal{H}(S_1), \mathcal{H}(S_2), \ldots, \mathcal{H}(S_k)$,
  - (unite) finally it combines the results in order to provide the output of $\mathcal{H}(S)$.

Let $C(S)$ be the total computing time of $\mathcal{H}(S)$, $Div(S)$ be the time for finding $S_1, S_2, \ldots, S_k$, and $Uni(S)$ for uniting the sub-solutions into the output of $\mathcal{H}(S)$. Then, for all $S \notin \mathcal{T}$,

$$C(S) = Div(S) + \sum_{i=1}^{k} C(S_i) + Uni(S)$$

straight from definition. Let $n = Size(S)$. If $Div(S) + Uni(S) = O(n)$, then there is a naive bound $C(S) = O(n^2)$ (cf. e.g., [29, 90]). Well-known speed up techniques divide S into two subproblems S_1 and S_2 of equal size. This yields $O(n \log n)$ time algorithms such as the very standard merge-sort (cf. e.g., [29, 90]).

Besides, the naive quadratic bound is known to improve as recursive calls decrease. For instance, most famous algorithms such as the median computation [7] or algorithms deriving from the planar separator theorem [84] reach linear worst case time bound by avoiding a fraction of S on recursive calls, namely by granting

$$\frac{Size(S_1) + Size(S_2) + \cdots + Size(S_k)}{Size(S)} < 1.$$

The success of such examples might explain why minimizing the divide and unite time $Div(S) + Uni(S)$ is usually disregarded in standard speed up approaches. Here, we address the case when recursive calls have to be applied on all parts, namely when $\frac{Size(S_1)+Size(S_2)+\cdots+Size(S_k)}{Size(S)} \leq 1$ with the bound reached. As a result of a larger theorem given in [1, 82], minimizing $Div(S)+Uni(S)$ becomes fruitful here, if it is done according to an "*avoid the largest*" principle. However, the proof given in [1, 82] is complicated as

it addresses a much larger situation. We give in the below an alternative proof. Within our terminology, the statement of the property could be as follows.

Proposition 4.2 *([1, 82]) Let $\mathcal{H}$ be a divide-and-conquer algorithm, and α be such that, for all $S \in \mathcal{S} \setminus \mathcal{T}$,*

$$Div(S) + Uni(S) \leq \alpha \times \left(Size(S) - \max_{i \in [\![1,k]\!]} Size(S_i) \right),$$

where $\{S_1, S_2, \ldots, S_k\}$ is the partition of S given by $\mathcal{H}(S)$. Then, for every input $S \in \mathcal{S}$, $\mathcal{H}(S)$ runs at most in $\alpha \times Size(S) \log Size(S)$ time. The bound is tight.

Proof: We proceed by induction on $s = Size(S)$. If S is not trivial and $S_1, S_2, \ldots, S_k$ are such that $s_k = Size(S_k)$ is greater than any $s_i = Size(S_i)$, then

$$\begin{aligned} Div(S) + Uni(S) + \sum_{i=1}^{k} C(S_i) &\leq \alpha \times \left(\sum_{i=1}^{k-1} s_i + \sum_{i=1}^{k} s_i \log s_i \right) \\ &\leq \alpha \times \left(\sum_{i=1}^{k-1} s_i + \sum_{i=1}^{k-1} s_i \log \frac{s}{2} + s_k \log s \right) \\ &\leq \alpha \times s \log s. \end{aligned}$$

Now, let $\mathcal{P}$ and $\mathcal{H}$ be such that there exist $S_0 \in \mathcal{T}$ and S_q ($q \geq 1$) where $\mathcal{H}$ divides S_q into two sub-instances that are both identical to S_{q-1}. Then, $\mathcal{H}$ computes at least in $\alpha \times Size(S_q) \log Size(S_q)$ time on S_q. □

Remark 4.1 *The standard speed up technique used in merge-sort results in the same bound. However, the size of the input given to merge-sort is granted to geometrically decrease (by half) as inductive levels grow, implying that the induction depth is smaller than* $\log Size(S)$*. On the other hand, the above result still holds even when the induction depth is linear on $Size(S)$.*

Though it may be straightforward to avoid the largest part for linear data structures such as ordered arrays, it is less easy in other cases, in particular when dealing with graphs. The challenge will be to avoid some "largest" graph component without exploring the whole graph. We exemplify the practical potential of Proposition 4.2 on graphs with a so-called *competitive graph search* technique.

4.2.2 Competitive Graph Searches

Let $G = (V, E)$ be a graph. We define the size of G as its number of vertices and edges: $Size(G) = |V| + |E|$. By abuse in the terminologies, we refer to the size of a vertex subset

as the size of the subgraph it induces. For more vocable precision, we say in the following that all vertices of a vertex subset A belong to the induced subgraph $G[A]$, and an edge of G belongs to $G[A]$ if both its extremities belong to A. We address two problems.

Exploring Connected Components

Let *Rep* be a list of pointers to one representative vertex per connected component of G. The first problem consists of, given G and *Rep*, visiting all connected components of G but the largest. To this aim, a *competitive graph search* proceeds as follows. At the beginning, all components are competitors via their corresponding representative vertex in *Rep*. Then, each step of the search visits one new element – vertex or edge (the "or" is exclusive) – of each competitor. The competitors for which no new element is found are discarded. This process continues as long as there are at least two remaining competitors. Obviously the last competitor C is the largest and has not been entirely visited. Indeed, if s' is the size of the second largest competitor C', then only s' elements of C have been visited, which leads to the following result.

Proposition 4.3 *Given a graph G and a list of pointers to one representative vertex per connected component of G, a* competitive graph search *visits all connected components of G but the largest component C in time bounded by $2 \times (s_G - s_C)$ with s_G the number of vertices and edges of G, and s_C the number of vertices and edges of $G[C]$.*

Proof: The exact visiting time is $(s_G - s_C) + s'$ with s' the size of the second largest component. □

Exploring Induced Subgraphs

Let $\{V_1, V_2, \ldots, V_k\}$ be a vertex partition of G, for example described by k lists. Let `oracle`(v, w) be an oracle box stating whether the vertices v and w belong to the same V_i. The second problem consists of, given G, $\{V_1, V_2, \ldots, V_k\}$, and `oracle`, visiting all induced subgraphs $G[V_1], G[V_2], \ldots, G[V_k]$ but the largest. Here, let *Rep* be a list of pointers to the first element of each V_i. Note that *Rep* is not required as input, but we have a much stronger requirement, namely the input of $\{V_1, V_2, \ldots, V_k\}$. We still start with the list *Rep* representing the competitors $V_1, V_2, \ldots, V_k$. Then, each step still tries to visit one new element (vertex or edge) of each competitor using any standard graph search on the corresponding vertex list V_i and the adjacency list of G. (This is why we need the input of $\{V_1, V_2, \ldots, V_k\}$.) The hitch is that inter-edges, namely those in $IE = \{vw \in E \mid \exists i \neq j \text{ s.t. } v \in V_i \text{ and } w \in V_j\}$, belong to none of the competitors. However, thanks to `oracle`, the search can check at any moment whether an edge is

inter-edge, and avoid leaving the current $G[V_i]$. To sum up, for each competitor, each step of the graph search either discovers a new vertex, or checks the outgoing edges until one edge belonging to that competitor is found. The remaining of the search behaves like before.

Proposition 4.4 *Given a graph $G = (V, E)$, a partition $\{V_1, V_2, \ldots, V_k\}$ of V, and an oracle function* `oracle` *stating whether two vertices belong to the same V_i, a* competitive graph search *visits the subgraphs $G[V_1], G[V_2], \ldots, G[V_k]$ but the largest in time bounded by $2 \times (s_G - s_C) + M$ with M the number of inter-edges between the subgraphs, s_G the number of vertices and edges of G, and s_C the number of vertices and edges of the largest subgraph.*

Proof: The exact visiting time is $(s_G - s_C) + s' + M'$ with s' the size of the second largest subgraph, and M' the number of visited inter-edges. □

To conclude, the main technical difficulty of a competitive graph search is to manage an entry to each competitor before starting and to maintain this as an invariant during the recursive process. Notice that this generic competitive search can be applied to other discrete structures such as directed graphs, hypergraphs or matroids. Let us examine the paradigm on the two foci of this chapter.

4.3 Common Connected Component Enumeration

Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be the input graphs of an instance of the common connected component problem. Firstly, we can can suppose w.l.o.g. that $E_1 \cap E_2 = \emptyset$ by recursively merging together vertices x and y if $(x, y) \in E_1 \cap E_2$ [60, 61]. Besides, if none of G_1 and G_2 is connected, a preliminary standard graph search can build the sub-instances $(G_1[X], G_2[X])$ for all connected components X of G_1. Lemma 4.1 states that computing the common connected components of (G_1, G_2) results in computing those of the latter sub-instances. Therefore, we can also suppose w.l.o.g. that G_1 is connected. Finally, another preliminary graph search can compute a list of one representative vertex per connected component of G_2 before launching our main recursive algorithm.

Concisely, the main algorithm addresses the problem of finding the common connected components of two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, given along with a list *Rep* such that $E_1 \cap E_2 = \emptyset$, G_1 is connected, and *Rep* has exactly one representative vertex per connected component of G_2. It proceeds as follows.

- If $k = |Rep| = 1$ then return V.

- Otherwise, let $V_1, V_2, \ldots, V_k$ be the connected components of G_2. Then, for all $1 \leq i \leq k$, we compute $G_1[V_i]$, $G_2[V_i]$, and a list Rep_i containing one representative vertex per connected component of $G_1[V_i]$.

- By inverting G_1 and G_2, we make recursive calls on $(G_2[V_i], G_1[V_i], Rep_i)$ and return all results.

The correctness follows from Lemma 4.1. Clearly, all the above operations can be done using standard graph searches, which would yield a naive $O(n(n+m))$ solution. However, we can benefit from competitive graph searches to improve the bound. Let $s(G) = Size(G) = |V| + |E|$ for any graph $G = (V, E)$. For convenience, we also note $s_i^1 = s(G_1[V_i])$, $s_i^2 = s(G_2[V_i])$, and $s_i = s_i^1 + s_i^2$. Let the "sum of all but the max" $\text{sam}_{i \in I} s_i$ be a shortcut for $\sum_{i \in I} s_i - \max_{i \in I} s_i$.

Lemma 4.2 *If s_i^1, s_i^2 are positive and $s_i = s_i^1 + s_i^2$ for all $i \in I$, then:*

$$\text{sam}_{i \in I}\ s_i^p \quad \leq \quad \text{sam}_{i \in I}\ s_i, \qquad \text{with } p \in \{1, 2\}.$$

Proof: Let i_0 and i_p be such that $s_{i_0} = \max_{i \in I} s_i$ and $s_{i_p} = \max_{i \in I} s_i^p$. Obviously, $s_{i_0}^p \leq s_{i_p}^p \leq s_{i_p}$. Besides, $\sum_{i \in I \setminus \{i_0, i_p\}} s_i^p \leq \sum_{i \in I \setminus \{i_0, i_p\}} s_i$. Adding the two inequalities allows to conclude. □

As already mentioned a competitive graph search *on the connected components* of G_2 computes all $G_2[V_i]$ except for $G_2[V_{i_2}]$ with $s_{i_2}^2 = \max_{1 \leq i \leq k} s_i^2$, as well as all V_i, except for V_{i_2}. During the search, we label the vertices in V_i ($i \neq i_2$) so that they can be distinguished afterwards. Those in V_{i_2} keep their old label so that they also come as a distinct k^{th} class. We define `oracle` which tests whether two vertices have same labels. By removing from G_2 vertices and edges of the $k-1$ computed graphs, we compute $G_2[V_{i_2}]$. Likewise, by removing from V vertices of the other V_i, we compute V_{i_2}. The operations so far run in $O(\text{sam}_{1 \leq i \leq k} s_i^2)$ time.

Using the computed $V_1, V_2, \ldots, V_k$ and the function `oracle`, a competitive graph search *on the induced subgraphs* of G_1 computes all $G_1[V_i]$ except for $G_1[V_{i_1}]$ with $s_{i_1}^1 = \max_{1 \leq i \leq k} s_i^1$. Let IE contain all inter-edges in G_1 between the subgraphs $G_1[V_i]$. By removing from G_1 vertices and edges of $G_1[V_i]$ ($i \neq i_1$), plus the inter-edges in IE, we compute $G_1[V_{i_1}]$. These operations take $O(|IE| + \text{sam}_{1 \leq i \leq k} s_i^1)$ time.

As $G_1[V_i]$ ($i \neq i_1$) are of small enough size, we simply compute Rep_i ($i \neq i_1$) thanks to standard searches on those graphs (such as the breath-first graph search). At this point, we have computed the recursive input for every $(G_2[V_i], G_1[V_i], Rep_i)$ such that $i \neq i_1$. The operations of this paragraph take $O(\text{sam}_{1 \leq i \leq k} s_i^1)$ time.

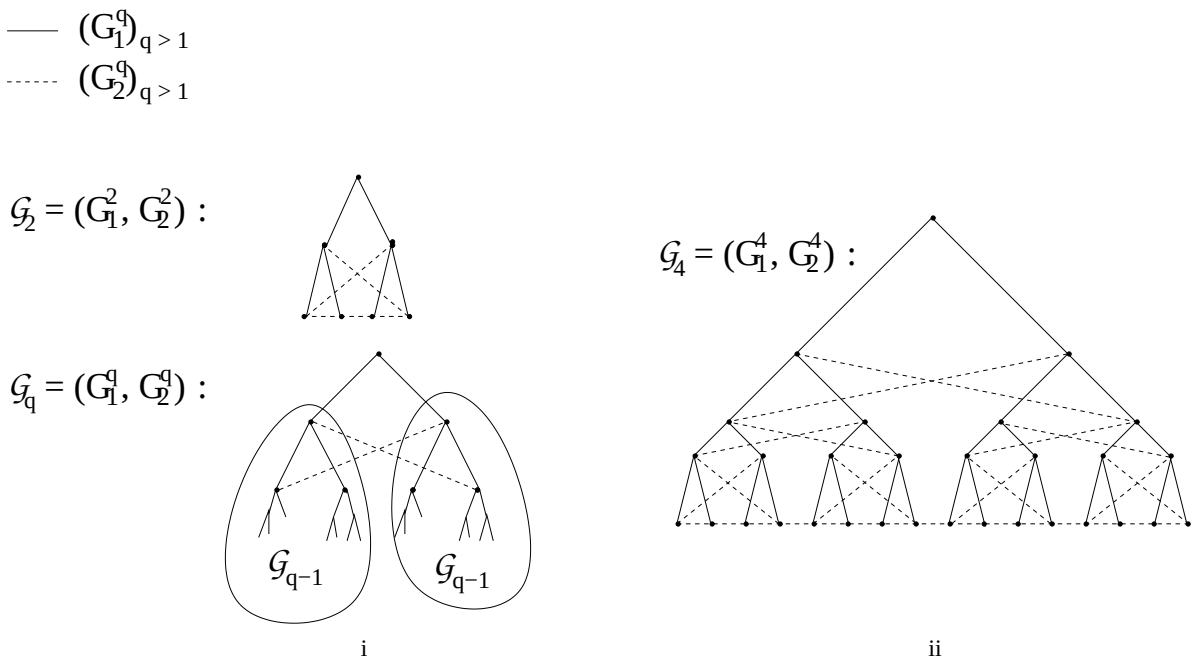


Figure 4.3: i. A sequence $(\mathcal{G}_q)_{q>1} = ((G_1^q, G_2^q))_{q>1}$ of instances for which our common connected component computation runs in $\Theta(n \log n)$. ii. Details of $\mathcal{G}_4$. Notice that all common connected sets/components of this sequence are and only are singletons.

From Lemma 4.2 all the operations so far run in $O(|IE| + \text{sam}_{1 \le i \le k} s_i)$ time.

The biggest hitch will be the computation of Rep_{i_1}, in order to provide the recursive input for $(G_2[V_{i_1}], G_2[V_{i_1}], Rep_{i_1})$. For this, let us assume that Rep_{i_1} is computed by some routine $\mathcal{R}$. We result in the following main theorem.

Definition 4.3 (Routine $\mathcal{R}$) Given a connected graph $G_1 = (V, E_1)$ and a partition $\{V_1, V_2, \ldots, V_k\}$ of V, the routine $\mathcal{R}$ computes a list Rep_{i_1} containing one representative vertex per connected component of $G_1[V_{i_1}]$, where $G_1[V_{i_1}]$ is the largest among $G_1[V_1], G_1[V_2], \ldots, G_1[V_k]$.

Theorem 4.1 (Main Theorem) *Given a routine $\mathcal{R}$ as defined above, the common connected components of two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ can be computed in $O(n + m \log n + t_{\mathcal{R}})$ time, where $n = |V|$, $m = |E_1| + |E_2|$, and $t_{\mathcal{R}}$ stands for the global computing time (through recursive calls) of the routine $\mathcal{R}$.*

Proof: The preliminary operations for computing the first list *Rep* and for rendering $E_1 \cap E_2 \neq \emptyset$ and G_1 connected run in $O(n + m)$. Now, our main algorithm follows the divide-and-conquer paradigm. Therein, the "unite" time of each step is $O(1)$. Moreover, except for the "$|IE|$" terms due to inter-edges and the cost of calls to the routine $\mathcal{R}$, the divide time fulfils requirements of Proposition 4.2. According to this, we split the global

complexity analysis of the main algorithm into three parts. The first counts the "$|IE|$" terms, the second the total cost of $\mathcal{R}$, and the third the remaining. Let $G'_1 = (V', E'_1)$ and $G'_2 = (V', E'_2)$ be the input graphs given to the main algorithm. Let $n' = |V'|$ and $m' = |E'_1| + |E'_2|$. Then, the "$|IE|$" part is in $O(m') = O(m)$ since an edge can be "inter-edge" only once throughout the running of the main algorithm. The second part was denoted by $t_{\mathcal{R}}$. From Proposition 4.2, the third part is in $O((n' + m') \log(n' + m')) = O(m \log n)$ time (also because $m' \leq m \leq n^2$, and G'_1 connected implies $n' = O(m')$). Whence, the whole running is in $O(n + m \log n + t_{\mathcal{R}})$. □

Implementation of Routine $\mathcal{R}$

The idea of computing Rep_{i_1} is the following. Let OG be the outgoing vertices in G_1 from $G_1[\bigcup_{i \neq i_1} V_i]$ to $G_1[V_{i_1}]$, namely $OG = \{y \in V_{i_1} \mid \exists x \notin V_{i_1} \text{ s.t. } (x, y) \in E_1\}$. Since G_1 is connected, $Rep_{i_1} \subseteq OG$. Computing OG only takes $O(|IE| + \text{sam}_{1 \leq i \leq k} s_i^1)$ time. Our idea is to filter OG efficiently until we obtain Rep_{i_1}. To this aim, we will use two tool boxes.

Definition 4.4 (Tool boxes B_1 and B_2) *Tool box* B_1 computes a spanning-forest of a given graph G, and for each vertex in G a pointer to the identifier of the spanning tree it belongs to. Given a graph G and one such spanning-forest representation of G, plus an edge e in G, *tool box* B_2 computes the spanning-forest representation of $G \setminus \{e\}$, and updates the pointers to spanning tree identifiers.

Thanks to B_1, right before launching the main recursive algorithm of the common connected component problem, we compute the spanning-forest representations of the two input graphs (namely the input graphs for the main algorithm, *after* the preliminary step). At each recursive step we use the `oracle` function to compute the inter-edge set IE in $O(|IE| + \text{sam}_{1 \leq i \leq k} s_i^1)$ time. Using B_2 we delete all edges of IE of the corresponding spanning-forest representation. Then, each vertex in OG has a pointer to the identifier of its spanning tree of the current G_1. We then sort those identifiers using standard sorting. Finally we scan the sorted identifiers and only keep one vertex of OG per identifier, which will form the list Rep_{i_1}.

Sorting the identifiers takes $O(|OG| \log |OG|) = O(|IE| \log |IE|) = O(|IE| \log m)$ time. The sum of all the "$|IE|$" terms throughout the computation is bounded by m. The remaining complexity is bounded by $O(m \log n)$ from Proposition 4.2 (the counting is the same as the one in the proof of Theorem 4.1). Hence, except for the cost of calls to B_1 and B_2, the complexity is still in $O(n + m \log n)$.

Forests: If the input graphs given to the main algorithm are forests, they form their own spanning forests. The only thing to be taken care off is keeping a pointer for each vertex

	best so far	our algorithm	conjecture
forests	$O(n \log n)$ [31]	$O(n \log n)$	$O(n)$
interval graphs	$O(m + n \log n)$ [32]	$O(n + m \log n)$	$O(n + m)$
unit interval graphs	$O(n \log \Delta \log n)$ [2]	$O(n\Delta \log n)$	$O(n + m)$
planar graphs	$O(n \log^2 n)$ [61]	$O(n \log n)$	$O(n)$
permutation graphs	$O(n \log n + m \log^2 n)$ [61]	$O(n + m \log^2 n)$	$O(n + m)$
arbitrary graphs	$O(n \log n + m \log^2 n)$ [61]	$O(n + m \log^2 n)$	$O(n + m \log n)$

Figure 4.4: Common connected component enumeration time, with n the number of vertices, m the total number of edges, and Δ the maximum vertex degree.

to the identifier of the spanning tree it belongs to. This, for B_1 can be done easily in $O(n+m)$ time. For B_2, let the edge to be deleted be $e = (x, y)$. We only need to update the identifiers of the spanning tree that has contained e before the deletion. The deletion of the edge e will split the old spanning tree into two parts, x and y could be seen as representatives for each part. Then, a competitive graph search will update the identifiers of vertices of the smaller part, while those of the bigger part keep their old label. This is in time proportional to the size of the smaller part. Then, the task of B_2 is complete. The complexity of B_2 is bounded by $O(m \log m)$ from Proposition 4.2. Finally, $m = O(n)$ in case of forests. We conclude that $t_{\mathcal{R}} = O(n \log n)$.

Corollary 4.1 *One can enumerate the common connected components of two forests in* $O(n \log n)$ *time.*

Non-forest cases: For arbitrary graphs, we benefit from results of [73] on the so-called *ET-tree* data structure [70]. Let m' be the number of edges in the two graphs given as input to the main algorithm, and n' be the number of their vertices. Here, the implementation of B_1 can be done in $O((n' + m') \log^2 n')$ time [73], or $O(m' \log^2 n')$ as m' is higher than n' (one of the two graphs is connected). The implementation of B_2 can be done in $O(\log^2 n')$ time per operation [73]. Like before, we note that an edge can be "inter-edge" only once during the whole computation, and conclude the total cost for calls to B_2 is in $O(m' \log^2 n')$. Hence, $t_{\mathcal{R}} = O(n + m \log^2 n)$.

Likewise, we use results on *edge-ordered dynamic trees* [49] for planar graphs. The corresponding B_1 and B_2 run in $O(m' \log n')$ and $O(\log n')$, respectively. Therefore, the

total running time of both tool boxes is $O(m' \log n')$. Notice that the number of edges in a planar graph is bounded by three times the number of vertices, and $t_{\mathcal{R}} = O(n \log n)$. For interval graphs, the same idea can be done using *clique-path representations* [67] for an $O(m')$ B_1, an $O(\log n')$ B_2, and a total $t_{\mathcal{R}} = O(n + m \log n)$.

Corollary 4.2 *One can enumerate the common connected components of two arbitrary graphs in $O(n+m \log^2 n)$ time. This runtime is in $O(n \log n)$ if the two graphs are planar. It is in $O(n + m \log n)$ if the two graphs are interval graphs.*

Our algorithm turns out to be a generic algorithm for all the related graph classes. As a consequence, mixing different classes is allowed, and yields the computing time equals to the upper one. For instance, the common connected computing time for a planar graph G_1 and an interval graph G_2 is in $O(n + m \log n)$.

As for performance analysis, our algorithm equals the best known so far for arbitrary graphs [61] and forests [31]. For planar graphs, we improve the performance by a $\log n$ factor, namely with an $O(n \log n)$ computing time. Our complexity for interval graphs is in $O(n+m \log n)$, while a recent result improved this to $O(m+n \log n)$ [32] (see Figure 4.4).

4.4 Application to Cograph Sandwiches

We now address the sandwich graph problems defined by M. Golumbic, H. Kaplan, and R. Shamir (1995).

Definition 4.5 (Sandwich Graph Problem [63]) A *sandwich graph problem* is a problem which addresses the following question:
INPUT: $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and Π, where G_1 and G_2 are two undirected graphs such that $E_1 \subseteq E_2$, and Π is a property over graphs.
QUESTION: Does there exist a graph $G = (V, E)$ satisfying both property Π and $E_1 \subseteq E \subseteq E_2$?

In this class of problems, the edges of E_1 are called *forced edges*, those of E_2 *optional edges*, and those of $E_3 = \overline{E_2}$ *forbidden edges*. Unfortunately, most problems therein are NP-complete, for example with the property $\Pi(G)$: G is a comparability graph (resp. chordal, resp. strongly chordal graph). Actually the problem remains NP-complete even for $\Pi(G)$ being G belongs to the quite small class of split graphs [63]. Only few polynomial cases are known, among which cograph sandwich [63], and the related problem of module sandwich [11, 26]. Therefore it is a natural question to ask for efficient algorithms for these polynomial cases.

We show in this section that, instead of designing from scratch a specific algorithm for cograph sandwiches, structural analysis can directly give an efficient solution for this problem. To this aim, we first exhibit a strong relationship between the cograph sandwich problem and the common connected component problem. Let us recall several folklore facts on cographs (for their very definition refer to Definition 2.18 in Section 2.3.1). The class of cographs is the smallest class of graphs containing the one vertex graph and closed under series and parallel composition. Therefore, any cograph can also be seen as a modular decomposition tree without prime nodes. Equivalently, cographs can be defined as the class of graphs excluding P_4 as induced subgraph [107], where P_4 denotes the path of *size* 4 (hence of length 3).

Theorem 4.2 *Let $G_1 = (V, E_1)$ be a graph of required edges, and $G_2 = (V, E_2)$ with $E_1 \subseteq E_2$ be a graph of possible edges. The complement graph of G_2 is defined as $G_3 = \overline{G_2}$, the graph of forbidden edges. Then, there exists a sandwich $G = (V, E)$ between G_1 and G_2 (meaning $E_1 \subseteq E \subseteq E_2$) which is a cograph if and only if every common connected component of G_1 and G_3 is a singleton.*

Proof: Suppose there is a cograph $G = (V, E)$ with $E_1 \subseteq E \subseteq E_2$. G cannot be both connected and co-connected, since the root of the modular decomposition tree of G is not a prime node. Let $V_1, V_2, \ldots, V_k$ be the partition of V into connected components of: G if it is not connected; $\overline{G}$ otherwise. That $E_1 \subseteq E \subseteq E_2$ implies there are no inter-edges between the vertex subsets V_i in one graph among G_1 and G_3. Then, using the partitioning Lemma 4.1, the common connected components of G_1 and G_3 are exactly the union of those of $G_1[V_i]$ and $G_3[V_i]$ for all i. Obviously, $G[V_i]$ is a sandwich of $G_1[V_i]$ and $G_2[V_i]$. Furthermore, $G[V_i]$ is a cograph, otherwise it would contain an induced P_4, and so would G. Hence, an inductive argument on the vertex subsets V_i will allow to conclude that all common connected components of G_1 and G_3 are singletons.

Conversely, suppose that every common connected component of G_1 and G_3 is a singleton. Let us build a graph $G = (V, E)$ as follows. If $|V| = 1$, then $E = \emptyset$. Otherwise, the instance can be divided into two cases. If G_1 is not connected, let $V_1, V_2, \ldots, V_k$ be its connected components. We define E such that any vertex pair (x, y) satisfying $x \in V_i$, $y \in V_j$, and $i \neq j$ implies $(x, y) \notin E$. If G_1 is connected, then necessarily G_3 is not connected (otherwise V is a common connected component). Let $V_1, V_2, \ldots, V_k$ be the connected components of G_3. We define E such that any pair (x, y) satisfying $x \in V_i$, $y \in V_j$, and $i \neq j$ implies $(x, y) \in E$. In both cases (G_1 not connected or G_3 not connected), the definition of E within each V_i follows inductively on $V_1, V_2, \ldots, V_k$. The fact that all common connected components of G_1 and G_3 are singletons guarantees

that, for all pairs $(x, y) \in V^2$ with $x \neq y$, we have chosen whether (x, y) belongs to E or not without contradictory definitions. Hence, G is well-defined. Then, using standard cograph characterizations, G can be proved to be a cograph. (We actually have built the modular decomposition tree of the cograph.) One can also verify that G is a sandwich between G_1 and G_3 by its construction. □

The above proof is constructive: if all common connected components of G_1 and G_3 are singletons, an algorithm was depicted to compute a cograph that is sandwich of G_1 and G_2. Therein, each step divides the graph into subgraphs induced by some $V_1, V_2, \ldots, V_k$, then decides whether edges between the V_i exist, and finally recurses in the subgraphs. This actually follows a divide-and-conquer scheme, with a $O(1)$ uniting time. Moreover, deciding the adjacency between the V_i results in labelling the corresponding node in the modular decomposition tree with series or parallel, which can be done in $O(1)$ time. Finally, identifying the subgraphs induced by the V_i can be cared off by a competitive graph searching. Hence, when a sandwich cograph exists, we can build one such in $O(n + m \log n)$ time, where n denotes the number of vertices, and m the number of edges of G_1 and G_3. We will use this algorithm as an intermediary result in order to prove the following one.

Corollary 4.3 *The sandwich cograph problem can be solved by a robust – in the sense of certifying – algorithm in $O(n+m \log^2 n)$ time, where n is the number of involved vertices, and m the total number of forced edges and forbidden edges.*

Proof: We first compute in $O(n + m \log^2 n)$ time the common connected components of the graph G_1 of forced edges and the graph G_3 of forbidden edges. Suppose that all common connected components of G_1 and G_3 are singletons. Then, a sandwich cograph can be build in $O(n + m \log n)$ as depicted in the proof of Theorem 4.2. We now suppose that there is some common connected component C that is not a singleton. Then, any sandwich G of G_1 and G_2 holds that $G[C]$ is both connected and co-connected ($G_1[C]$ is partial subgraph of $G[C]$ and $G_3[C]$ is partial subgraph of $\overline{G}[C]$). We deduce that $G[C]$ is not a cograph and must contain a P_4, and so must G. Thus, C is our certificate to state that no sandwich of G_1 and G_2 can be a cograph. In this case, one can verify in linear time that both $G_1[C]$ and $G_3[C]$ are connected and deduce that every sandwich of G_1 and G_2 must contain a P_4. □

The above result improves the $O(n(n + m))$ complexity of the algorithm proposed in [63]. However, we think that:

Conjecture: *There exists a linear time algorithm to solve the sandwich cograph problem.*

The conjecture has also implications in the common connected component problem. Firstly, such an algorithm would imply a linear time algorithm for the characterization of the totally degenerate case of the common connected component problem, namely when all components are singletons. Besides, but quite similarly, the P_4-structure of common connected components is worth being further studied as the following proposition shows. It is highly related to Theorem 4.2, and states that common connected components must contain many P_4's.

Proposition 4.5 *Let C be a common connected component of two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ with $E_1 \cap E_2 = \emptyset$. Then, both $G_1[C]$ and $G_2[C]$ contain a P_4.*

Proof: Consider the root of the modular decomposition tree of $G_1[C]$. It cannot be a parallel node since $G_1[C]$ is connected, nor it can be a series node since $E_1 \cap E_2 = \emptyset$ and $G_2[C]$ is connected. Therefore it is a prime node. Hence, $G_1[C]$ is not a cograph and must contain a P_4. Similar argument holds for $G_2[C]$. □

To conclude, this chapter gives a generic common connected component enumeration. It also exemplifies an infrequent technique for speeding up divide-and-conquer algorithms. Since divide-and-conquer is a very basic method, our algorithm is simply structured while holding some efficient performances (Figure 4.4). We also improve the computation of cograph sandwiches as a corollary of this algorithm.

In general, as soon as some dynamic data structure satisfying our requirements on the tool boxes B_1 and B_2 (see Definition 4.4) is provided, our general algorithmic scheme will apply. We hope that this technique could be helpful to solve other problems, such as that of common strongly connected components.

Now, we have seen in Section 4.1 that the case when both input graphs are forests of paths is special. Indeed, the family of common connected sets of the graphs will then satisfy the axioms of a weakly partitive family (cf. Proposition 4.1). Actually, this restricted case corresponds to another topic arising quite recently from computational biology with some high challenges and specific interests. The next chapter will focus on this restriction.

Chapter 5

Uno-Yagiura Algorithm Revisited

This chapter is based on [19].

The common connected component problem presented in Chapter 4, when restricted to graphs that are collections of disjoint paths, turns out to have important connections to some other problems in computational biology as well. This is a quite promising field with high challenges (a so-called "hot-topic"), and is currently under intensive studies as having applications in the enumeration of gene clusters, the computation of evolutionary distance between species, and that of evolutionary conservations in a given set of genomic sequences (some further precisions are upcoming). In the previous chapter, we have seen in Proposition 4.1 how to connect this restricted case of forests of paths to the structural properties of modular decomposition. The general guideline for this chapter is to revisit a related algorithm designed by T. Uno and M. Yagiura.

In computational biology, the latter restriction to forests of paths is more known via the notion of a common interval. Here, let us restrict the modelling of a genomic sequence to the case of a permutation over the set V of all genes it contains. Therein, an interval is a set of genes which come successively in the sequence (the permutation can be seen as a total order). Then, a set C of genes is called a common interval of a given group of several genomic sequences if C is an interval for every sequence in the group. For instance, the set $\{5, 6, 7, 8\}$ is a common interval of the example given in Figure 5.2. This is among the first attempts to formalize the notion of a gene cluster [111].

In that same paper, T. Uno and M. Yagiura depicted also an algorithm that computes all the K common intervals of two given permutations of length n in $O(n + K)$ time, which is linear on the size of the problem [111]. Afterwards, F. de Montgolfier pointed out strong relationships between the modules of a permutation graph and the common intervals of every realizer of the graph (made of two permutations) [94]. This allows to define a kind of common interval decomposition tree (further details will be given in Section 5.1). From recent works, the tree turns out to own some important biological

Uno-Yagiura general scheme:
1. Let `Potential` be an empty list
2. **For** $i = n$ down to 1 **Do**
3. (Filter): Remove all known boundaries r in `Potential` such that for all $l \leq i$, (l, r) is not a common interval
4. (Add): Add i to the head of `Potential`
5. (Extract): While there still is some boundary r of `Potential` such that (i, r) is a common interval, output (i, r)
6. **End of for**

Figure 5.1: A list `Potential` is used. It contains at each step i all boundaries $r \geq i$ such that there is some $l \leq i$ with (l, r) a common interval. Then, `Potential` is scanned to output the common intervals of the form (i, r). The main difficulty of such an approach is to prove the linear time complexity, as the algorithm is sketched by a double iteration.

meaning [3, 80]. Particularly, common intervals help in finding evolutionary distances between the corresponding species [5, 53, 80], and they can be interpreted as pieces of each genome that have been conserved all along an evolutionary scenario between the involved species and their common ancestor [3].

The seminal algorithmic result on common intervals is due to T. Uno and M. Yagiura (Figure 5.1). This is really a masterpiece among combinatorial algorithms as it uses a unique scan on one of the two permutations and could be seen as an application of a sweep plane paradigm as used in computational geometry [44]. However, even if Uno-Yagiura algorithm is based on the very standard and conventional dynamic programming scheme, its correctness proof is tough to understand. Later, S. Heber and J. Stoye pointed out a smaller and generating sub-family, a so-called family of irreducible common intervals. They succeeded in adapting Uno-Yagiura algorithm to enumerate all irreducible common intervals of d permutations in $O(d \times n)$ time [69]. Besides, generating all the K common intervals from this sub-family is in $O(K)$ time [69]. While T. Uno and M. Yagiura's scheme was used as a crucial part of their algorithm, S. Heber and J. Stoye did not give further explanations for the correctness proof. More recently, some authors bypassed this difficult issue by proposing an alternative algorithm together with its combinatorial proof [4].

In this chapter, we propose a complete invariant-based proof for both correctness and complexity analysis of Uno-Yagiura algorithm. We also show how it can be adapted in a straightforward manner to compute in $O(n)$ time a tree representation of all common intervals of two permutations on n elements. Then, Section 5.3 generalizes Uno-Yagiura algorithm, and uses it as a central step for the computation of the modular decomposition tree of an undirected graph (if a so-called factoring permutation is given).

5.1 Some Structural Aspects of Common Intervals

Before addressing algorithmic issues, we first focus on some combinatorial aspects around the notion of a common interval. Let us denote $\mathbb{N}_n = [\![1, n]\!] = \{1, 2, \ldots, n\}$.

Definition 5.1 (Common Interval) A permutation π over a set V will be regarded indifferently as a bijection from $\mathbb{N}_{|V|}$ to V, a total order on V, or a word $\pi \in V^*$ without multiple occurrence over alphabet V. The support of a factor of π is called an *interval of* π, namely it is of the form $\pi([\![l, r]\!]) = \{\pi(l), \pi(l+1), \ldots, \pi(r)\}$, where $l, r \in \mathbb{N}_{|V|}$ are called its left and right boundaries. A subset of V is a *common interval* of two permutations over V if it is interval of each permutation.

Notice that there could be a quadratic number of common intervals, for example when the two permutations are equal. Let us give a decomposition scheme for common intervals, based on weakly partitive families.

5.1.1 Common Interval Decomposition

We briefly recall results of Section 2.2.1. Let $\mathcal{F} \subseteq 2^V$ be a weakly partitive family over ground set V. The subfamily of $\mathcal{F}$ containing all overlap-free members of $\mathcal{F}$ is denoted by $\mathcal{S}$. The members of $\mathcal{S}$ can be organized by inclusion order into a tree, so-called the *overlap-free decomposition tree* of $\mathcal{F}$. Let us denote it by $\mathcal{T}$. Then, an internal node in $\mathcal{T}$ satisfies one and only one of the following:

- it has two children and every union of children belongs to $\mathcal{F}$, and we say that the node is *linear* in this chapter (unlike elsewhere in the thesis);
- it has at least three children and no union of children belongs to $\mathcal{F}$, except for the node itself, and we say that the node is *prime*;
- it has at least three children and every union of children belongs to $\mathcal{F}$, and we say that the node is *complete*;
- it has at least three children and there is an ordering over the children such that a union of children belongs to $\mathcal{F}$ if and only if they are consecutive w.r.t. this order, and we say that the node is *linear*.

Roughly, the tree $\mathcal{T}$, with the internal nodes labelled as above, is a compact encoding of $\mathcal{F}$: from this all members of $\mathcal{F}$ can be generated in a straightforward manner. The following notion was first introduced by C. Capelle under a restricted framework. However, our extension to set families is absolutely straightforward. It will act as a crux in our point of view when dealing with Uno-Yagiura algorithm.

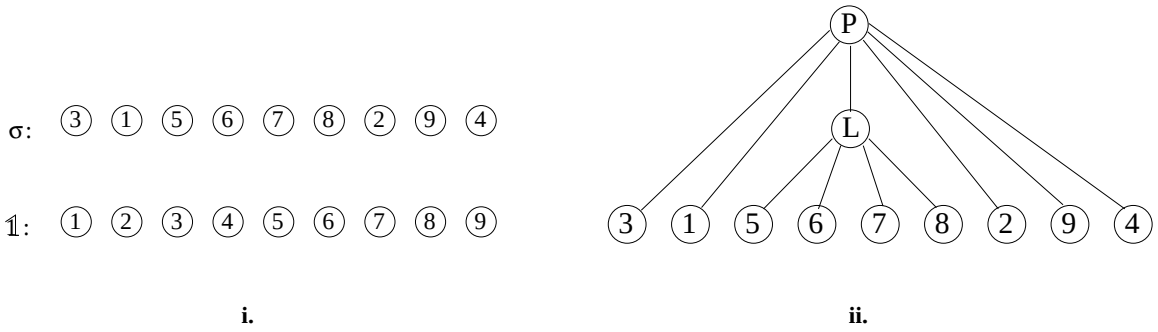


Figure 5.2: Common interval decomposition. "L" stands for *linear* and "P" for *prime*.

Definition 5.2 (Factoring Permutation) *([24])* A permutation σ is *factoring* for a family $\mathcal{F} \subseteq 2^V$ if and only if every overlap-free member of $\mathcal{F}$ is an interval of σ.

In other words, a factoring permutation of $\mathcal{F}$ is the visit-order of the leaves of $\mathcal{T}$ given by a depth-first graph search. Though the following property is straightforward, it gives a formal decomposition framework for common intervals.

Proposition 5.1 (Common Interval Decomposition) *The family $\mathcal{CI}$ of common intervals of two given permutations σ_1 and σ_2 over a same set V always satisfies the three following properties: $\mathcal{CI}$ is weakly partitive; its overlap-free decomposition tree $\mathcal{T}$ has no* complete *nodes; and both σ_1 and σ_2 are factoring.*

Before going to computational purposes, we continue the discussion around combinatorial issues with an essential property. This, in our opinion, is responsible for the nice runtime of Uno-Yagiura algorithm.

5.1.2 Intersecting Submodularity of Common Intervals, with Generalization to Modules and to Genuine-Modules

To our knowledge the first submodular-like property around the topic of common intervals is given by T. Uno and M. Yagiura under the name of the *reverse Monge property* in their seminal paper [111].

Lemma 5.1 (Uno-Yagiura [111]) *Let $\sigma = \sigma_1$ and σ_2 be two permutations over a same set V with $n = |V|$. For all $1 \leq i \leq j \leq n$, we define $l(i,j) = \min\{k \mid \sigma_2(k) \in \sigma([\![i,j]\!])\}$, $r(i,j) = \max\{k \mid \sigma_2(k) \in \sigma([\![i,j]\!])\}$, and $f(i,j) = r(i,j) - l(i,j) - (j-i)$. Then,*

$$f(i',j) + f(i,j') \geq f(i',j') + f(i,j) \quad \textit{for all} \quad i' < i \leq j < j'.$$

This property is fundamental. Moreover, for purposes in Section 5.3, we will use the following extension. We recall the definition of a submodular function. A set function

$f : 2^V \to \mathbb{R}$ is *submodular* if

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B), \quad \text{for all} \quad A, B \subseteq V.$$

Our proposition to extend Uno-Yagiura lemma requires the introduction of the notion of a splitter. We begin with explaining how to define this under the framework of common intervals. After this crucial step, we will see how to extend the notion to graph splitters, and also to 2-structure splitters.

Definition 5.3 (Common Interval Splitter) Let $\sigma = \sigma_1$ and σ_2 be two permutations over a same set V with $n = |V|$. Let $C_2(i, j)$ refer to the convex hull in σ_2 of $\sigma(\llbracket i, j \rrbracket)$, namely $C_2(i, j) = \sigma_2(\llbracket l, r \rrbracket)$ where boundaries l, r are $l = \min\{k \mid \sigma_2(k) \in \sigma(\llbracket i, j \rrbracket)\}$ and $r = \max\{k \mid \sigma_2(k) \in \sigma(\llbracket i, j \rrbracket)\}$. We define the *splitter set* $\mathcal{S}_{\sigma(\llbracket i,j \rrbracket)}$ of $\sigma(\llbracket i, j \rrbracket)$ as $\mathcal{S}_{\sigma(\llbracket i,j \rrbracket)} = C_2(i, j) \setminus \sigma(\llbracket i, j \rrbracket)$.

Roughly, the existence of a splitter is a guarantee that an interval is not a common interval. Let $s(\sigma(\llbracket i, j \rrbracket)) = |\mathcal{S}_{\sigma(\llbracket i,j \rrbracket)}|$ be the function counting the number of splitter of an interval. Then,

Remark 5.1 *The function s counting the number of splitter of an interval is exactly the function f defined in Uno-Yagiura Lemma 5.1. Moreover, $\sigma(\llbracket i, j \rrbracket)$ is a common interval if and only if $s(\sigma(\llbracket i, j \rrbracket)) = f(i, j) = 0$, for all $i \leq j$.*

Let us briefly recall the notion of a genuine-module of a 2-structure (cf. Section 2.3.2 for further details). A 2-structure $G = (V, C)$ is a vertex set V along with a colour function C mapping every arc (x, y) (with $x, y \in V$ and $x \neq y$) to some value in $\mathbb{N}$. A digraph is a 2-structure where the colour function has only two values: present and absent arcs. An undirected graph is a digraph where the colour function has the same value on (x, y) and (y, x), for every pair $\{x, y\} \subseteq V$ with $x \neq y$. A genuine-module M of a 2-structure $G = (V, C)$ is a non-empty vertex subset such that for all $x, y \in M$ and $s \notin M$, the colour function satisfies $C(s, x) = C(s, y)$.

Definition 5.4 (Module Splitter and Genuine-Module Splitter) Let $G = (V, C)$ be a 2-structure. Let $A \subseteq V$ be a vertex subset. A vertex $s \notin A$ outside of A is called a *splitter* of A if there exist $x, y \in A$ such that $C(s, x) \neq C(s, y)$. In particular, the definition applies on the restriction of G to the case of an undirected graph.

Straight from definition, a vertex subset $M \subseteq V$ is a genuine-module if and only if it has no splitters. Let $\mathcal{S}_A$ refer to the splitter set of a vertex subset $A \neq \emptyset$ and $s(A) = |\mathcal{S}_A|$ count their number. Then, we have the following observation, which can be found in [16]. It is a very straightforward generalization (from undirected graphs to 2-structures) of a more crucial observation, given in [19].

Lemma 5.2 (Intersecting Submodularity of Genuine-Module Splitters) *The function s counting the splitters of vertex subsets of a 2-structure is an* intersecting submodular function*, that is*

$$s(A) + s(B) \geq s(A \cap B) + s(A \cup B), \quad \textit{for all} \quad A, B \subseteq V \quad \textit{verifying } A \cap B \neq \emptyset.$$

Moreover, the family of non-empty minimizers of this function is exactly the family of genuine-modules of the 2-structure. In particular, the lemma holds for undirected graphs, where the notion of a genuine-module coincides with that of a module.

Proof: That the family of non-empty minimizers of this function is exactly the family of genuine-modules of the 2-structure follows directly from definition. The intersecting submodularity can be proved as follows. If $A \subseteq B$ or $B \subseteq A$, the inequality is trivial. If $A \otimes B$ then $A \neq \emptyset$ and $B \neq \emptyset$. Let $\mathcal{S}_A$ denote the set of splitters of A. If $\{X_1, X_2, \ldots, X_k\}$ is a partition of X, we will note $X = \{X_1, X_2, \ldots, X_k\}$.

Clearly, $\mathcal{S}_{A\cap B} = \{\mathcal{S}_{A\cap B} \setminus B,\ \mathcal{S}_{A\cap B} \cap B\}$. Also, that $\mathcal{S}_A \cap A = \emptyset$ implies the partition $\mathcal{S}_{A\cup B} = \{\mathcal{S}_{A\cup B} \setminus \mathcal{S}_A,\ \mathcal{S}_{A\cup B} \cap \mathcal{S}_A\}$ can be reduced to $\mathcal{S}_{A\cup B} = \{\mathcal{S}_{A\cup B} \setminus \mathcal{S}_A,\ \mathcal{S}_A \setminus (A \cup B)\}$. Similarly, $\mathcal{S}_B = \{\mathcal{S}_B \setminus \mathcal{S}_{A\cap B},\ \mathcal{S}_{A\cap B} \setminus B\}$.

Finally, that $\mathcal{S}_A = \{\mathcal{S}_A \setminus B,\ (\mathcal{S}_A \cap B) \setminus \mathcal{S}_{A\cap B},\ (\mathcal{S}_A \cap B) \cap \mathcal{S}_{A\cap B}\}$ can be reduced to $\mathcal{S}_A = \{\mathcal{S}_A \setminus (A \cup B),\ (\mathcal{S}_A \cap B) \setminus \mathcal{S}_{A\cap B},\ \mathcal{S}_{A\cap B} \cap B\}$. Hence,

$$|\mathcal{S}_A| + |\mathcal{S}_B| - |\mathcal{S}_{A\cup B}| - |\mathcal{S}_{A\cap B}| = |(\mathcal{S}_A \cap B) \setminus \mathcal{S}_{A\cap B}| + |\mathcal{S}_B \setminus \mathcal{S}_{A\cap B}| - |\mathcal{S}_{A\cup B} \setminus \mathcal{S}_A|.$$

To achieve proving the lemma, we prove that $\mathcal{S}_{A\cup B} \setminus \mathcal{S}_A \subseteq \mathcal{S}_B \setminus \mathcal{S}_{A\cap B}$. Indeed, let $s \in \mathcal{S}_{A\cup B} \setminus \mathcal{S}_A$. Then, $s \notin A \cup B$ and $C(s, x) = C(s, y)$ for all $x, y \in A$. Now, suppose that $s \notin \mathcal{S}_B$. Since s does not belong to B, we deduce $C(s, x) = C(s, y)$ for all $x, y \in B$. Furthermore, as A and B overlap, we deduce $C(s, x) = C(s, y)$ for all $x, y \in A \cup B$ and $s \notin A \cup B$, which is by definition $s \notin \mathcal{S}_{A\cup B}$. Contradiction. Finally, supposing $s \in \mathcal{S}_{A\cap B}$ would imply $s \in \mathcal{S}_A$. □

5.1.3 Right-Free Intervals

For computational purposes, the main drawback of the raw definition of a common interval is that there is potentially a quadratic number of them. This is responsible for the term "K" in Uno-Yagiura runtime. (Uno-Yagiura algorithm computes all the K common intervals of two given permutations of length n in $O(n + K)$ time.) In order to rectify this situation, S. Heber and J. Stoye introduced a smaller, but generating, subfamily of the family of all common intervals. It is as follows.

Definition 5.5 (Reducible vs. Irreducible Common Interval) *([69])* A common interval is *reducible* if it is the union of consecutively overlapping non-trivial common intervals. A common interval is *irreducible* when not reducible.

There is a straightforward manner to enumerate all the K common intervals from the only knowledge of the subfamily of irreducible common intervals, in time $O(K)$ [69]. Also, this can easily be generalized to any weakly partitive family, and is very similar to the notion of a 2-graph (Definition 2.12 in Section 2.2.2). Now, it follows directly from definition that the irreducible common intervals correspond exactly to *prime* nodes and pairs of consecutive children of *linear* nodes of the decomposition tree. Besides, the decomposition tree, which is an inclusion tree, can be seen as a collection of its nodes, namely the subfamily of overlap-free common intervals. Then, the difference between the subfamily of irreducible common intervals and the decomposition tree, viewed as the subfamily of overlap-free common intervals, is thin. Firstly, the difference only concerns linear nodes. Moreover, even for the linear nodes, going from one to the other is easy: an overlap-free common interval bound to a linear node can be seen as a pair (l, r) of left and right boundaries on one of the two input permutations; while the irreducible common intervals bound to the same linear node are exactly the pairs $(l, l+1)$, $(l+1, l+2)$, $\ldots$, $(r-1, r)$. In other words, one can compute in $O(n)$ time the subfamily of irreducible common intervals from the decomposition tree and conversely.

For convenience, we would rather focus on the decomposition tree and some related notions that have been introduced throughout the first part of the thesis. We will also have to introduce a new notion in order to adapt Uno-Yagiura algorithm to obtain a computation of the decomposition tree in $O(n)$ time (without the "K" term). From what has been said on weakly partitive families it is clear that the decomposition tree is a generating object for the enumeration of all common intervals of the initial permutations. The intermediary notion we need is so-called *right-free interval*, where right-free is a shortcut for overlap-free-on-the-right.

Let $\sigma = \sigma_1$ and σ_2 be two permutations over V. Let $\mathcal{CI}$ refer to the family of their common intervals. Then, σ is factoring for $\mathcal{CI}$. W.l.o.g., from now on, *intervals* will stand for intervals of σ. By definition, a common interval is an interval.

Definition 5.6 (Right-Free Interval) Let σ be a factoring permutation of a (weakly) partitive family $\mathcal{F} \subseteq 2^V$. Let $\sigma(\llbracket i, j \rrbracket) \in \mathcal{F}$ be *at the same time* an interval of σ and a member of $\mathcal{F}$. Then, we call the interval $\sigma(\llbracket i, j \rrbracket)$ *right-free* if it does not overlap on its right any other interval of σ that belongs to $\mathcal{F}$, namely if

$$\text{for all } \; i < i' \leq j < j', \quad \text{we have} \quad \sigma(\llbracket i', j' \rrbracket) \notin \mathcal{F}.$$

Roughly, a right-free interval of $\mathcal{CI}$ is a member of $\mathcal{CI}$ that does not overlap any other member of $\mathcal{CI}$ on its right in the order σ. From definition, it is clear that there are more right-free intervals than overlap-free intervals. Fortunately enough, their number is still bounded by the same value as that for overlap-free intervals, namely $2 \times n$ (cf. Corollary 5.1 below). This is a crucial fact as it roughly explains why we can allow to compute the right-free intervals as a pre-processing step before computing the overlap-free intervals. In order to formalize the computation of right-free intervals, let us define

$$\texttt{Select}(i) = \{j \mid \sigma([\![i, j]\!]) \text{ is a right-free interval}\}, \quad \text{for all } n \geq i \geq 1.$$

Note that $\texttt{Select}(i)$ always includes at least one member, namely i.

Definition 5.7 (Useless Boundary) While inspecting σ from n down to 1, $\sigma([\![l, r]\!])$ is *visited* at step i if $i < l$, *unvisited* otherwise. Then, $r \in [\![i, n]\!]$ is *useless* w.r.t. i if none of the unvisited right-free intervals is of the form $\sigma([\![l, r]\!])$.

Lemma 5.3 *Let m_i be the maximum boundary such that $\sigma([\![i, m_i]\!]) \in \mathcal{F}$. Clearly, m_i is well defined and $i \leq m_i$. Moreover, $m_i = \max \texttt{Select}(i)$ and, when exists, every r such that $i < r < m_{i+1}$ also verifies that r is useless w.r.t. i.*

Proof: If $\sigma([\![i, m_i]\!])$ overlaps $\sigma([\![i', m']\!])$ on its right (i.e. if $i < i' < m_i < m'$), then $\sigma([\![i, m']\!]) \in \mathcal{F}$ (partitivity) and m_i is not maximum. Therefore, $m_i \in \texttt{Select}(i)$, and $m_i = \max \texttt{Select}(i)$ follows directly from the definition of m_i. Besides, for all $l < i + 1 \leq r < m_{i+1}$, $\sigma([\![l, r]\!])$ overlaps $\sigma([\![i + 1, m_{i+1}]\!])$ on its right. □

Corollary 5.1 $|\texttt{Select}(1)| + |\texttt{Select}(2)| + \ldots + |\texttt{Select}(n)| \leq 2 \times n$.

Proof: From Lemma 5.3, the sets $\texttt{Select}(i) \setminus \{\max \texttt{Select}(i)\}$ $(1 \leq i \leq n)$ are pairwise disjoint and their total cardinal is bounded by n. □

When dealing with computational issues in the upcoming Section 5.2 we will first output all right-free intervals before computing the decomposition tree. As well, the enumeration of right-free intervals will be the main part of the difficulty. Fortunately, it fits into T. Uno and M. Yagiura's sweep paradigm.

5.2 Common Interval Enumeration

With a slight modification, namely by adding an one-line routine, Uno-Yagiura algorithm computes in $O(n)$ time the family of *right-free intervals* of two given permutations $\sigma = \sigma_1$

Uno-Yagiura algorithm revisited:
1. Let `Potential` be an empty list and $\texttt{Select}(n+1) = \emptyset$
2. **For** $i = n$ down to 1 **Do**
3. (Update-Detect): Collect all known useless boundaries w.r.t. i
4. (Pre-Filter): If there are some $r < r_0 (= \max \texttt{Select}(i+1))$ in `Potential`, remove them and mark r_0 as *Eaten*
5. (Customized Filter): Remove all known useless boundaries w.r.t. i
6. (Add): Add the boundary i to the head of `Potential`
7. (Extract): Find the right-most r_q in `Potential` with $s_i(r_q) = 0$ and output $\texttt{Select}(i) = \{r_1 \dots r_q\}$
8. **End of for**

and σ_2 over V, where $n = |V|$. However, we will discuss in detail its correctness, since the original version is tough to understand. The sets $\texttt{Select}(i)$ $(n \geq i \geq 1)$ will be computed using a list `Potential`. At each step i, this list contains the right boundaries $r \geq i$ of every unvisited right-free interval, and possibly some extra boundaries.

`Potential` is initialized as an empty list. Each step $n \geq i \geq 1$ aims at removing from `Potential` as many useless boundaries w.r.t. i as possible. For this purpose, let $s_i(j) = s(\sigma([\![i, j]\!])) = |\mathcal{S}_{\sigma([\![i,j]\!])}|$ denote the number of splitters of interval $\sigma([\![i, j]\!])$. We have seen that a splitter makes an interval not a common interval.

Proposition 5.2 *([111]) $\sigma([\![i, j]\!])$ is a common interval if and only if $s_i(j) = 0$.*

We define $\delta_i(p_j) = s_i(p_{j+1}) - s_i(p_j)$ if a member p_j of `Potential` has a successor p_{j+1}. Otherwise, $\delta_i(p_j) = +\infty$. Then, Corollary 5.2 below, which is a direct application of Uno-Yagiura Lemma 5.1, is fundamental and most results hereafter rely on it.

Corollary 5.2 *([111]) $\delta_i(p_j) < 0$ implies p_j is useless w.r.t. i.*

To simplify the presentation of how to handle the list `Potential`, let us assume some routines. At each step i, assume that some Update-Detect routine provides

- for each p_j in `Potential` a pointer to the value of $s_i(p_j)$; and
- a list `Detected` of pointers to all p_j with $\delta_i(p_j) < 0$, and possibly to some other useless boundaries w.r.t. i. Besides, assume that the pointed $p_{j_1} < p_{j_2} < \cdots < p_{j_h}$ are organized increasingly.

Then, `Potential` is filtered twice. The first filtering (routine Pre-Filter) is our only addition to the original algorithm. It follows from Lemma 5.3, which states that it is

possible to move apart some useless boundaries w.r.t. i even before considering $\sigma(i)$. Concisely, a pointer to $r_0 = \max \texttt{Select}(i+1)$ is maintained. Then, if r_0 has some predecessors in `Potential`, they are removed and r_0 receives the mark *Eaten*, which will be for later use in the final construction of the decomposition tree. The second filtering (routine Customized Filter) backtracks `Detected` from p_{j_h} down to p_{j_1}. Each p_{j_k} is removed from `Potential` if still there. If some removing makes the next-left boundary p' have $\delta_i(p') < 0$, p' is also removed and so on. Thus, only useless boundaries w.r.t. i are removed, and all remaining boundaries have positive δ_i. Both filtering takes linear time on the number of removed boundaries. The boundary i is then added to the head of `Potential` (routine Add) and the update of step i is complete. Notice that $\delta_i(i) \geq 0$.

Invariant 5.1 *After the update of step i, let p_{j_0} be the first member of* `Potential` *with $s_i(p_{j_0}) \neq 0$. Then,* $\texttt{Select}(i) = \{r < p_{j_0} \mid r$ *is a member of* `Potential`$\}$.

Proof: After the update, all p_j have $\delta_i(p_j) \geq 0$. If $r \in \texttt{Select}(i)$, then $s_i(r) = 0$ and $r < p_{j_0}$. Besides, $\sigma([\![i, r]\!])$ is unvisited at step i. Hence, r still is a member of `Potential`, and it is strictly before p_{j_0}. Conversely, any member $r < p_{j_0}$ of `Potential` after the update satisfies $s_i(r) = 0$. If $\sigma([\![i, r]\!])$ overlaps some $\sigma([\![i', r']\!])$ on its right, then $i < i' \leq r < r'$, $\sigma([\![i', r]\!]) \in \mathcal{CI}$, $\sigma([\![i', r']\!]) \in \mathcal{CI}$ and the Pre-Filter at step i' would remove r from `Potential` if it was still there. □

Outputting $\texttt{Select}(i)$ from the list `Potential` (routine Extract) follows from Invariant 5.1. Its computing time is clearly linear on the size of the output. Finally, Corollary 5.1 and the fact that each boundary is inserted exactly once in `Potential` imply the following.

Proposition 5.3 *The computing time to enumerate all right-free intervals is in $O(n)$ if routine Update-Detect runs in linear time on the size of the outputted list* `Detected` *at each iteration step i. Such a runtime is sublinear on the size of the input.*

Outputting `Dectected` in linear time on the size of this list is not trivial. T. Uno and M. Yagiura's solution to this problem [111] is as follows. Let us denote list `Potential` at the beginning of step i as $\texttt{Potential} = [p_1(= i+1), p_2, \ldots, p_l]$. The routine updates two lists $\texttt{Min} = [Min_1, Min_2, \ldots, Min_s]$ and `Max`. Each $1 \leq Min_j \leq n$ is a boundary with two pointers $\texttt{first}(Min_j)$ and $\texttt{last}(Min_j)$ to two members of `Potential`. All p_j between these two members satisfy $Min_j = \min\{k \mid \sigma_2(k) \in \sigma([\![i, p_j]\!])\}$. Besides, each p_j in `Potential` has a pointer $\texttt{Min}(p_j)$ to the corresponding member of `Min`. It is analogous for `Max`. By supposing $V = [\![1, n]\!]$, computing $s_i(p_j)$ from this structure is in $O(1)$ time.

We recall the notation of a convex hull given in Definition 5.3: the convex hull in σ_2 of $\sigma([\![i, j]\!])$ is $C_2(i, j) = \sigma_2([\![l, r]\!])$, where boundaries l, r are $l = \min\{k \mid \sigma_2(k) \in \sigma([\![i, j]\!])\}$ and $r = \max\{k \mid \sigma_2(k) \in \sigma([\![i, j]\!])\}$.

Let $\texttt{Min} = [Min'_1, Min'_2, \ldots, Min'_{s'}]$ and $\texttt{Max} = [Max'_1, Max'_2, \ldots, Max'_{t'}]$ at the beginning of step i. Suppose inductively that $C_2(i+1, p_j) = \sigma_2([\![\texttt{Min}(p_j), \texttt{Max}(p_j)]\!])$ for all p_j and that `Min`, resp. `Max`, is strictly decreasing, resp. increasing. Notice that $\sigma_2(Min'_1) = \sigma_2(Max'_1) = \sigma(i{+}1)$. Now, i' with $\sigma_2(i') = \sigma(i)$ can be obtained in $O(1)$ time. Then, either $i' < Min'_1$ and `Max` will be unchanged, or $Max'_1 < i'$ and `Min` unchanged. We trace `Min`, resp. `Max`, from $j = 1$ until finding the first j^* with $Min'_{j^*} \leq i' < Max'_1$, resp. $Min'_1 < i' \leq Max'_{j^*}$. Notice that $j^* > 1$ and let $p_{j_0} = \texttt{first}(Min'_{j^*-1})$, resp. $p_{j_0} = \texttt{first}(Max'_{j^*-1})$.

Lemma 5.4 *([111]) p_j is useless w.r.t. i if $s_i(p_j) - s_{i+1}(p_j) > s_i(p_{j+1}) - s_{i+1}(p_{j+1}) \geq 0$.*

Proof: This lemma follows from the application of Uno-Yagiura Lemma 5.1. □

Invariant 5.2 *(equivalent to Lemma 5.4) p_j with $1 \leq j < j_0$ is useless w.r.t. i.*

W.l.o.g. $Min'_{j^*} \leq i' < Max'_1$, we set Min'_{j^*-1} to i'; point $\texttt{first}(Min'_{j^*-1})$ to p_1; and for all $1 \leq j < j_0$, point $\texttt{Min}(p_j)$ to Min'_{j^*-1}. Thus, each p_j satisfies $C_2(i, p_j) = \sigma_2([\![\texttt{Min}(p_j), \texttt{Max}(p_j)]\!])$. It is straightforward to maintain this fact until the end of step i, and the inductive hypothesis for the next step holds. Finally, `Detected` is defined as a list of pointers to $p_1 < p_2 < \cdots < p_{j_0-1}$. Now, the only member of `Potential` where δ_i can be negative that is not pointed by `Detected` is $p_{j_1} = \texttt{last}(Min_{j^*-1})$. Thus, if $\delta_i(p_{j_1}) < 0$, we add a pointer to p_{j_1} to the end of `Detected`. The running time is $O(j_0 + j^*) = O(j_0) = O(|\texttt{Detected}|)$.

We have proved the tricky part of the revisited Uno-Yagiura algorithm, that is

Proposition 5.4 *The enumeration of the right-free intervals of two given permutations over a same set of n elements takes $O(n)$ time.*

Remark 5.2 *Ideally, at each step i,* `Potential` *would contain* only *the right boundaries $r \geq i$ of all unvisited right-free intervals. Is it true ?*

After the enumeration of right-free intervals, a symmetric sweep from left-to-right generates the overlap-free common intervals (i.e. both right-free and left-free). We recall that those are the nodes of the decomposition tree. Moreover, the sweep organizes them by interval inclusion. Hence, constructing the tree is in $O(n)$ time. Then, the labelling can use the following remarks. Since there are only *prime* and *linear* nodes, the overlap-free common intervals that are marked *Eaten* by the enumeration of right-free intervals have also this mark in the computation of left-free intervals. Besides, a node has *Eaten* if and only if it is *linear*.

Finally, Proposition 5.2, Corollary 5.2, and Lemma 5.4 can be generalized to the case of d permutations if one replaces $C_2(i, p_j)$ with $C_j = \mathcal{S}_{\sigma([\![i,p_j]\!])} \uplus \sigma([\![i, p_j]\!]) = \bigcup_{h=2}^{d} C_h(i, p_j)$. Then, at each step i in the new Update-Detect, one has to maintain C_j rather than just $C_2(i, p_j)$. The hitch lays on the fact that $C_h(i, p_j)$ $(2 \leq h \leq d)$ are not pairwise disjunctive. However, as an element $\sigma(i')$ can be added to some $C_h(i'', p''_j)$ only once throughout the computation, the total maintenance can be done in $O(d \times n)$ time. We have proved that

Theorem 5.1 *The common interval decomposition tree of d given permutations can be computed in $O(d \times n)$ time. From this tree all common intervals can be enumerated in a straightforward manner in linear time on their number.*

5.3 Application to Modular Graph Decomposition

We briefly recall some aspects already presented in Section 2.3.1.

Let $G = (V, E)$ be a loopless simple undirected graph with $n = |V|$ and $m = |E|$. A vertex $v \in V \setminus A$ exterior to $A \neq \emptyset$ is *adjacent to* A if it is adjacent to every vertex of A, *non-adjacent to* A if non-adjacent to every vertex of A. In both cases, v is *uniform to* A. Otherwise, v is a *splitter of* A. The vertex subset A is a *module* if it has no splitters. Roughly, the family $\mathcal{M}$ of modules of G refers to the set of subgraphs of G that behave as one single vertex. It is well-known that $\mathcal{M}$ is partitive [47, 62], and finding efficient algorithms for computing $\mathcal{T}_{\mathcal{M}}$ from G has been an important challenge of the last two decades [25, 37, 43, 65, 87, 89, 108]. The *factoring permutations of* G refer to those of $\mathcal{M}$. Linear time algorithms to output such a permutation are available for chordal graphs [74], inheritance graphs [66], and even for arbitrary graphs [65].

C. Capelle initiated in [24] a "graph lay-out" approach for the computation of the modular decomposition tree of a given graph. It consists of first finding a factoring permutation [65], then constructing the modular decomposition tree [25]. Both computations run in $O(n + m)$ time even if the latter [25] is somewhat a complicated procedure. One of the aspects of this section is to give an alternative to this algorithm.

Definition 5.8 (Permutation Graph and Realizer) A graph is a *permutation graph* if it has an intersection model consisting of straight lines (one per vertex) between two parallels. Equivalently, a graph $G = (V, E)$ is a permutation graph if and only if it has a so-called realizer, whose definition is as follows. A *realizer* of G is a pair (σ_1, σ_2) of permutations over the vertex set V such that there is an edge xy in G if and only if:

- either x is on the right of y in σ_1 while y is on the right of x in σ_2, or
- x is on the left of y in σ_1 while y is on the left of x in σ_2.

In order to link common intervals to modular decomposition, the following observation given by F. de Montgolfier (cf. [94, Proposition 73, page 217]) will be important. Its proof is a straightforward case analysis.

Lemma 5.5 *([94]) Both permutations of a realizer of a permutation graph are factoring for the graph.*

Corollary 5.3 *The family $\mathcal{CI}$ of common intervals of two permutations is included in the family $\mathcal{M}$ of modules of the permutation graph where the two permutations form a realizer. Besides, the overlap-free members of $\mathcal{CI}$ are exactly the overlap-free members of $\mathcal{M}$. Finally, the decomposition tree $\mathcal{T}_{\mathcal{CI}}$ of $\mathcal{CI}$ is isomorphic to the decomposition tree $\mathcal{T}_{\mathcal{M}}$ of $\mathcal{M}$, with* complete *labels in $\mathcal{T}_{\mathcal{M}}$ replaced by* linear *labels in $\mathcal{T}_{\mathcal{CI}}$, and conversely.*

The proof of Corollary 5.3 is straightforward. From this, the algorithm of the previous section is an $O(n)$ time modular decomposition algorithm for an n-vertex permutation graph given by one realizer. Notice that the number of edges of such a graph can be in $m = \Theta(n^2)$.

Corollary 5.4 *The modular decomposition tree of an n-vertex permutation graph given by one of its realizers can be computed in $O(n)$ time.*

There exists another common interval decomposition algorithm of two permutations which runs in $O(n)$ time [86], and hence it is also an alternative to Corollary 5.4 above. Unfortunately, the algorithm therein is not that simple and relies on a rather sophisticated one [42]. Moreover that approach does not seem to be extendible to arbitrary modular graph decomposition. To this aim one can use the algorithm proposed in [25]. However, this latter produces a rather heavy sequence of trees. On the other hand, we will show that revisiting the Uno-Yagiura approach results in using a unique paradigm for both computations of the common interval decomposition tree of two given permutations and the modular decomposition tree of an arbitrary graph. Then, not only we unify the two corresponding topics but also provide very efficient algorithms.

Accordingly, we address the following problem: given a factoring permutation σ of an arbitrary graph $G = (V, E)$, compute the modular decomposition tree of G. Let us first adapt Proposition 5.2 and Corollary 5.2. Let $\mathcal{S}_A$ refer to the splitter set of a vertex subset $A \neq \emptyset$ and $s(A) = |\mathcal{S}_A|$ count the number of its splitters (see Definition 5.4). Proposition 5.5 and Corollary 5.5 below are our graph versions of Proposition 5.2 and Corollary 5.2, respectively.

Proposition 5.5 *$\sigma([\![i, j]\!])$ is a module if and only if $s_i(j) = s(\sigma([\![i, j]\!])) = 0$.*

Corollary 5.5 *Let $i \leq p_j < p_{j+1}$, and $\delta_i(p_j) = s_i(p_{j+1}) - s_i(p_j)$. Then, $\delta_i(p_j) < 0$ implies there is no $k \leq i$ such that $\sigma([\![k, p_j]\!])$ is a module.*

Proof: If $\delta_i(p_j) < 0$, then the submodularity of Lemma 5.2 on the subsets $\sigma([\![k, p_j]\!])$ and $\sigma([\![i, p_{j+1}]\!])$ for all $k \leq i$ implies that $s_k(p_j) > s_k(p_{j+1}) \geq 0$. □

Basically, the two above generalizations imply that the algorithmic scheme of the previous section can be used in the case of modules if one adapts the involved routines accordingly. Actually, the adaptation is straightforward, except for the Update-Detect routine and labelling the decomposition tree.

The Update-Detect Routine for Modular Decomposition

Let $G = (V, E)$ be a graph, and σ a factoring permutation of G. Let N_X (resp. $\overline{N}_X$) be the set of adjacent (resp. non-adjacent) vertices to X. We implement here the Update-Detect routine to be used for modular decomposition. Let us recall what has to be done in this routine.

Let $[p_1, p_2, \ldots, p_k]$ be the value of `Potential` at the beginning of iteration step i. After the computation of Update-Detect, from each p_j, one has to be able to compute the value of $s_i(p_j)$ or $\delta_i(p_j)$ in $O(1)$ time. Besides, the routine is to output a list `Detected` of pointers to all members p_j of `Potential` with $\delta_i(p_j) < 0$, plus some other useless boundaries w.r.t. i. In this case of modular decomposition, we will compute `Detected` such that the pointed boundaries are exactly those that have a strictly negative δ_i. Finally, `Detected` has to organize increasingly the pointed boundaries $p_{j_1} < p_{j_2} < \cdots < p_{j_h}$.

To obtain this, we need another data structure from the one used in the common interval decomposition. Our implementation aims at an $O(n + m)$ runtime, and follows the rule: each step i only considers the neighbourhood of $\sigma(i)$. Some notions have to be introduced.

Let $N_{i,j}$ and $\overline{N}_{i,j}$ refer to $N_{\sigma([\![i,j]\!])}$ and $\overline{N}_{\sigma([\![i,j]\!])}$ for short. Let $[p'_1, p'_2, \ldots, p'_l]$ be the value of `Potential` at the end of step i in the selection phase. Then, the fact that $p'_1 (= i) < \cdots < p'_l$ implies $N_{i,p'_1} \supseteq \cdots \supseteq N_{i,p'_l}$. Therefore, the neighbourhood of $\sigma(i)$ in G can be partitioned into l *neighbour wings* $N_{i,i} = (NW_{i,p'_1}, NW_{i,p'_2}, \ldots, NW_{i,p'_l})$, where each neighbour wing is defined as $NW_{i,p'_j} = N_{i,p'_j} \setminus N_{i,p'_{j+1}}$ for all $1 \leq j \leq l$ and $N_{i,p'_{(l+1)}} = \emptyset$. The definition of the *non-neighbour wings* such that $\overline{N}_{i,i} = (\overline{N}W_{i,p'_1}, \overline{N}W_{i,p'_2}, \ldots, \overline{N}W_{i,p'_l})$ is analogous. Then, the *level* L_{i,p'_j} is defined as $L_{i,p'_j} = NW_{i,p'_j} \uplus \overline{N}W_{i,p'_j}$. We define $H_i = \{\sigma(i)\}$ and trivially deduce that $V = (L_{i,p'_1}, L_{i,p'_2}, \ldots, L_{i,p'_l}, H_i)$. The *level threshold* does not depend on i and is defined as $\theta(p'_j) = p'_{j+1} - p'_j$ for all $1 \leq j \leq l$ and $p'_{l+1} = p'_l$.

Notice that its obtaining does not require any data structure since it can be directly computed from p'_j and its successor in $O(1)$ time.

Proposition 5.6 $s_i(p'_j) = n - (|[\![i, p'_j]\!]| + |N_{i,p'_j}| + |\overline{N}_{i,p'_j}|)$ *for all* $1 \leq j \leq l$.

Proof: A splitter of a vertex subset is an exterior non-uniform vertex. □

Corollary 5.6 $\delta_i(p'_j) = s_i(p'_{j+1}) - s_i(p'_j) = |L_{i,p'_j}| - \theta(p'_j)$ *for all* $1 \leq j \leq l$.

Proposition 5.7 *A data structure representing* $V = (L_{i,p'_1}, L_{i,p'_2}, \ldots, L_{i,p'_l}, H_i)$ *with respect to partition refinements techniques [68] allows to implement the Update-Detect routine for graph modules to compute in* $O(|N_{i,i}|)$ *time per step* i.

Indeed, from Corollary 5.6, the value of $\texttt{delta}(p'_j) = \delta_i(p'_j)$ can be obtained in $O(1)$ from the one of $|L_{i,p'_j}|$. According to this, we use the partition refinement techniques [68] to maintain the partition of V into $V = (L_{i,p'_1}, L_{i,p'_2}, \ldots, L_{i,p'_l}, H_i)$ at the end of each step i.

Let us assume a partition refinement function `Refine`, which takes as input a pivot set $\texttt{S} \subseteq V$ and a data structure `P` representing a partition of $(X_1, X_2, \ldots, X_p)$. Each X_i $(1 \leq i \leq q)$ has a pointer $\texttt{Size}(X_i)$ to its cardinal. Then, $\texttt{Refine}(\texttt{S}, \texttt{P})$ proceeds in $O(|\texttt{S}|)$ and splits any X_i with $X_i \cap \texttt{S} \neq \emptyset$ to an *intersection subset* $I_i = X_i \cap \texttt{S}$ and a *different subset* $D_i = X_i \setminus \texttt{S}$. The two subsets have pointers to each other. The details of `Refine` are in [68].

We then define a data structure holding the following. At the beginning of each step i, besides the list $\texttt{Potential} = [p_1, p_2, \ldots, p_k]$, a data structure `Partition` is maintained with respect to the partition refinement techniques to represent $\texttt{Partition} = \big(N_{p_1}, \overline{N}_{p_1}, N_{p_2}, \overline{N}_{p_2}, \ldots, N_{p_k}, \overline{N}_{p_k}, H\big)$. Each p_j points to both N_{p_j} and $\overline{N}_{p_j}$. The pointer $\texttt{delta}(p_j)$ does not exist: it is replaced by the addition of $\texttt{Size}(N_{p_j})$ and $\texttt{Size}(\overline{N}_{p_j})$, to which p_j can access in $O(1)$ time.

Let us assume that an inductive hypothesis provides $N_{p_j} = NW_{i+1,p_j}$, $\overline{N}_{p_j} = \overline{N}W_{i+1,p_j}$ and $H = H_{i+1}$. We will prove the inductive hypothesis for the next step by describing the Update-Detect routine. To begin with, $\{\sigma(i)\}$ can be removed from `Partition` with a call to $\texttt{Refine}(\{\sigma(i)\}, \texttt{Partition})$. Indeed, there only is one single intersection subset $HTemp = \{\sigma(i)\}$, which is temporally stored apart. All the remaining is redefined `Partition`, where any member (wings or H) is the old one excluded $\sigma(i)$. This takes $O(1)$ time. Besides, an empty N'_i is created with a pointer $\texttt{Size}(N'_i)$ to 0, as well as an empty $\overline{N'}_i$ with $\texttt{Size}(\overline{N'}_i)$ to another 0. Furthermore, `Modified` is initialized to be an empty list (of pointers). This takes $O(1)$ time.

After this, `Refine`$(N_{i,i}, \texttt{Partition})$ is called with some extra rules. When it splits a neighbour wing $N_{p_j} = N_{i+1,p_j} \setminus N_{i+1,p_{j+1}} \setminus \{\sigma(i)\}$ in `Partition` into two subsets, it also perform the following. First, as the intersection subset holds $N_{p_j} \cap N_{i,i} = N_{i,p_j} \setminus N_{i,p_{j+1}}$, the old neighbour wing in `Partition` is replaced by this. Second, as the difference subset $N_{p_j} \setminus N_{i,i} = L^1_{p_j} \cap \overline{N}_{i,i}$ is included in $\overline{N}_{i,i} \setminus \overline{N}_{i,p_1}$, it is concatenated to $\overline{N'}_i$ and the involved pointers `Size` are updated. Last, a pointer to p_j is added to the end of the list `Modified`. It is analogous when a non-neighbour wing $\overline{N}_{p_j} = \overline{N}_{i+1,p_j} \setminus \overline{N}_{i+1,p_{j+1}} \setminus \{\sigma(i)\}$ is split since the difference subset holds $\overline{N}_{p_j} \setminus N_{i,i} = \overline{N}_{p_j} \cap \overline{N}_{i,i} = \overline{N}_{i,p_j} \setminus \overline{N}_{i,p_{j+1}}$ and the other holds $\overline{N}_{p_j} \cap N_{i,i} \subseteq N_{i,i} \setminus N_{i,p_1}$. When splitting H, we replaced it by $HTemp = \{\sigma(i)\}$, and concatenate $H \cap N_{i,i}$ to N'_i and $H \setminus N_{i,i}$ to $\overline{N'}_i$. All these operations are in $O(1)$ time per splitting operation. Therefore, the refinement is in $O(|N_{i,i}|)$. At this point, the value $\left(NN_{p_1}, \overline{NN}_{p_1} \ldots, NN_{p_k}, \overline{NN}_{p_k}, HH\right)$ of `Partition` holds $NN_{p_j} = NW_{i,p_j}$, $\overline{NN}_{p_j} = \overline{N}W_{i,p_j}$, and $HH = H_i$. Thus, acceding to $|L_{i,p_j}| - \theta(p_j) = \texttt{Size}(NN_{p_j}) + \texttt{Size}(\overline{NN}_{p_j}) - (p_{j+1} - p_j) = \delta_i(p_j)$ from each p_j takes $O(1)$ time, which is one of the two main results of Update-Detect. Now, we deduce by elimination that $N'_i = N_{i,i} \setminus N_{i,p_1}$ and $\overline{N'}_i = \overline{N}_{i,i} \setminus \overline{N}_{i,p_1}$. Therefore, the routine outputs $\texttt{s} = \texttt{Size}(N'_i) + \texttt{Size}(\overline{N'}_i) - (p_1 - i) = s_i(i+1)$ for further use in the Add routine. This takes $O(1)$ time.

Finally, it is obvious to state that `Modified` contains the pointers to all p_j such that $\delta_{i+1}(p_j) = \delta_i(p_j)$. Therefore, `Detected` is a sublist of `Modified` since δ_i is positive elsewhere. Besides, the pointed boundaries by `Modified` are increasing since `Partition` has a specific order of increasing levels (the increasing monotonicity of boundaries pointed by `Modified` is not necessarily strict though: a boundary might be introduced twice when either N_{p_j} and $\overline{N}_{p_j}$ have been split). Hence, obtaining `Detected` by tracing `Modified` is straightforward in $O(|\texttt{Modified}|) = O(|N_{i,i}|)$. This is the second of the two main results of Update-Detect.

Now, we have to prove the inductive hypothesis for the next step. To obtain this, the filtering routines (Pre-Filter, Customized Filter) have to perform some extra works: when a boundary p_j is removed, if its predecessor p_{j-1} in `Potential` exists, we concatenate the involved neighbour wings with respect to $N_{i,p_{j-1}} \setminus N_{i,p_{j+1}} = N_{i,p_{j-1}} \setminus N_{i,p_j} \uplus N_{i,p_j} \setminus N_{i,p_{j+1}}$. It is similar for the involved non-neighbour wings. If p_j is at the head of `Potential`, the wings are concatenated to N'_i and $\overline{N'}_i$. Besides, the values of `delta` have to be updated accordingly. If p_j is at the head of `Potential`, $\texttt{delta}(p_j)$ is added to `s` so that we always have $\texttt{s} = s_i(p_j)$ with p_j at the head of `Potential`. The cost of each removing still is $O(1)$. Besides, for later use, when the Pre-Filter removes some $r < r_0 (= \max \texttt{Select}(i+1))$ and gives r_0 the mark *Eaten*, we mark r_0 as *Adjacency* if and only if $NW_{i,r}$ is not empty. Notice that either $NW_{i,r}$ is not empty and $\overline{N}W_{i,r}$ is empty or $NW_{i,r}$ is empty and $\overline{N}W_{i,r}$

is not empty. This helps distinguishing *series* from *parallel* nodes. Finally, when the Add routine inserts i to the head of `Potential`, it also has to insert N'_i and $\overline{N'}_i$ to the head of `Partition`, and make i point to both of them. Let $p_1 = i$ and p_2 be the two first members of `Potential` at this state, the Add routine uses the value of $\mathtt{s} = s_i(p_2) = \delta_i(p_1)$ to create the pointer $\mathtt{delta}(p_1)$ for this boundary $p_1 = i$. The cost of the Add routine still is $O(1)$. By doing so, it is straightforward to deduce the inductive hypothesis for the next step.

As for complexity issues, at each step i, the computing time of the Update-Detect routine is $O(|N_{i,i}|) = O(d(\sigma(i)))$, where $d(\sigma(i))$ is the degree of vertex $\sigma(i)$. Then, the global complexity cost for calls to this routine is in $O(n+m)$, while the other complexity costs are like before, namely in $O(n)$.

Now, let us show how to label the decomposition tree. First, it is well-known that a modular decomposition tree has no *linear* nodes, and its *complete* nodes are divided into *series* nodes – adjacency guaranteed between all children – and *parallel* nodes – non-adjacency between children [47, 62]. Then, by analogous remarks as in the previous section, nodes marked *Eaten* are *complete*, others are *prime*. Besides, thanks to *Adjacency* marks, which state the adjacency between the children of the node, we can differ the *series* and *parallel* nodes in the labelling.

We have showed that

Proposition 5.8 *Given an arbitrary graph along with one of its factoring permutation, we can compute the modular decomposition tree of the graph in $O(n+m)$ time by following Uno-Yagiura approach.*

To conclude, this chapter has displayed the importance of graph layout approaches, e.g., the approach via factoring permutations. Such an approach can also be seen as a gateway between string algorithms and graph algorithms. Finally, we have showed strong potentials of generalizing Uno-Yagiura algorithm to broader perspectives, and also its connection to weakly partitive families.

We believe that the use of Uno-Yagiura algorithm would be an important crux for designing future algorithms. For instance, it would be interesting to adopt the same philosophy conducted throughout this chapter to other combinatorial problems such as the decomposition in $O(n + m)$ time of an inheritance graph into "inheritance-blocks". This would yield an alternative to the algorithms proposed in [24, 66]. Another example would be the question whether the modular decomposition tree can be computed in $O(n)$ time for a bounded tolerance graph – trapezoid graph with solely parallelograms [9, 51] – when an intersection model is provided. Then, it would be very interesting to have an $O(n)$ modular decomposition time for an interval or trapezoid graph given by one of its intersection models. This would give interesting connections to works on gene-teams [2].

Chapter 6

H-join Decomposition and Dynamic Programming

This chapter is based on [22].

We close the thesis with an attempt to give a generalized standpoint of, at the same time, modular graph decomposition, split decomposition, and the recently introduced bijoin decomposition (see, e.g., Section 2.3.1 and Section 2.3.3 for their definition). In fact, it is now quite widely known that both notions of a split and a bijoin are strict generalizations of that of a module (see Figure 6.1). This chapter goes one step further in the same generalization stream, and addresses a unifying graph composition operation, a so-called H-join operation. It is indexed by a bipartite graph H (see Figure 6.2). We then investigate a so-called H-join decomposition scheme, which gives a broader view than all the above mentioned decompositions.

At the same time, H-join decomposition has also connections to a series of recent advances in algorithmic graph theory, initiated by the important establishment of strong algorithmic results on the so-called clique decomposition of a graph [35]. In this topic, the list of related notions to H-join decomposition includes no fewer than, in lexicographic order: c-decomposition [81], clique decomposition [33, cited in [10]], decomposition into bimodule partitions* [103], k-HB decomposition [77], NLC-decomposition [112], and rank decomposition [96, 100]. This is a currently evolving field with ongoing large efforts for advancements. In this chapter, we will restrict our discussion to some algorithmic issues.

As a matter of fact, several graph decompositions define an associated notion of width parameter. Most of them are already mentioned sparsely in the thesis, even though none of them has really been discussed in detail so far. From an algorithmic point of view, the most important among those are, in order of discovery: treewidth, branchwidth, cliquewidth and rankwidth. The first two of these parameters are "less powerful" than the last two,

*This is different from the bimodular decomposition of bipartite graphs [55].

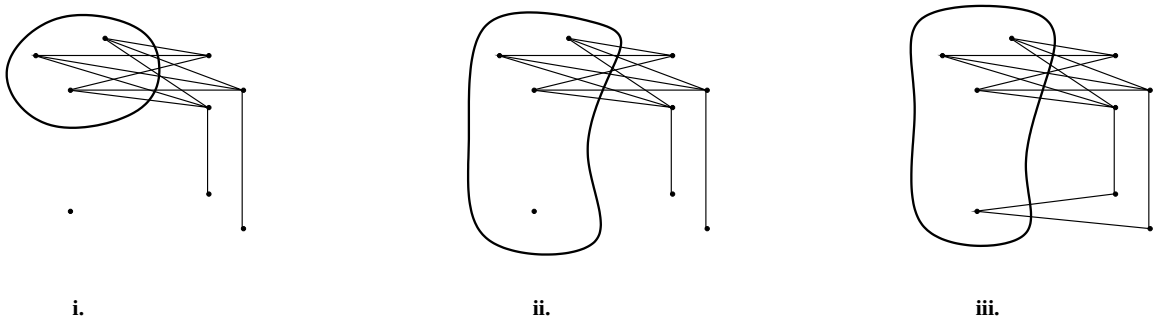


Figure 6.1: i. A module that is also a split and a bijoin. ii. A split that is not a module, nor a bijoin. iii. A bijoin that is not a module, nor a split.

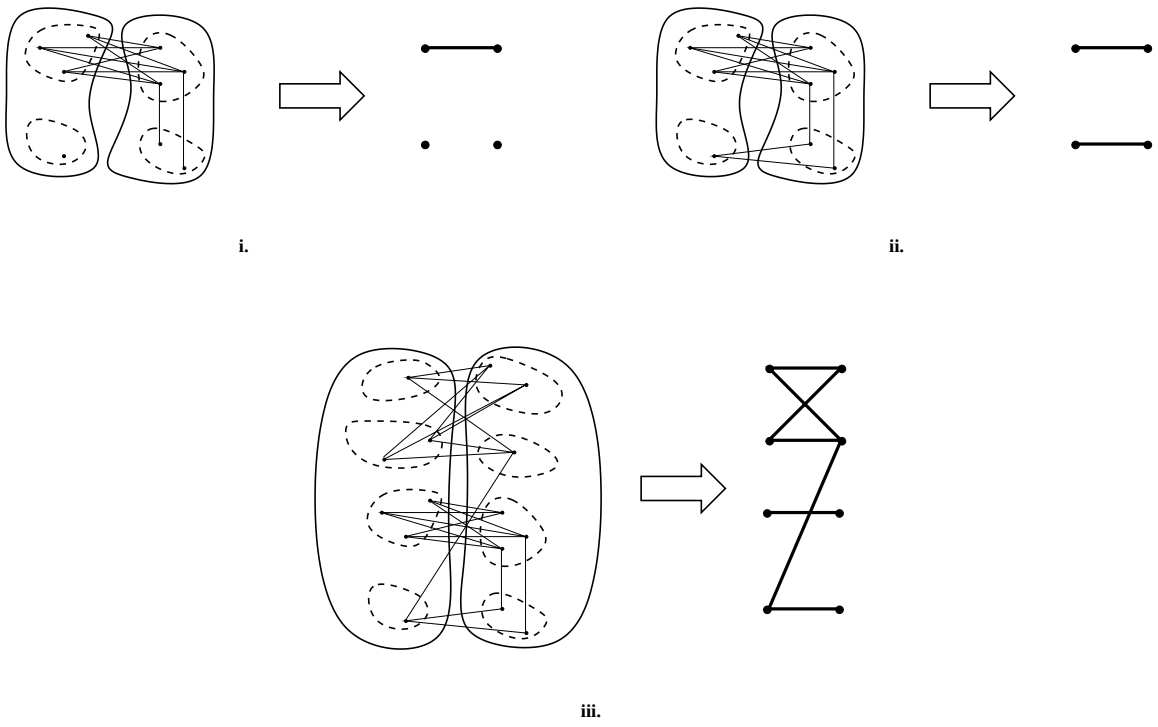


Figure 6.2: i. A split. ii. A bijoin. iii. An H-join.

in the sense that a graph class has bounded treewidth if and only if it has bounded branchwidth [104], it has bounded cliquewidth if and only if it has bounded rankwidth [100], and if it has bounded treewidth then it has bounded cliquewidth but not the other way around [30]. Moreover, the rankwidth of a graph is never larger than its cliquewidth, nor its branchwidth , nor its treewidth plus one [99]. In this sense, rankwidth is the most powerful of the four parameters. Also, since its recent introduction [96], the notion has been quite intensively investigated [34, 36, 71, 97, 98, 99, 100]. In our composition also, rank decomposition will figure in one of the major foci of this chapter.

Given a parameter $\pi(I)$ for every object I belonging to a set $\mathcal{I}$, a fixed-parameter tractable (FPT) algorithm on an instance made by an element of $\mathcal{I}$, parameterized by π is an algorithm running in $O(f(k)|I|^{\alpha})$ time on every instance $I \in \mathcal{I}$ with $\pi(I) \leq k$, where α is a constant. Many NP-hard graph optimization problems have FPT algorithms when parameterized by the above width parameters (see [72] for an overview). Some examples could be: maximum independent set/clique, minimum dominating set, and Hamiltonian path/circuit. FPT algorithms for such problems usually have two stages:

- a first stage computing the right decomposition of the input graph, and
- a second stage solving the problem using the decomposition.

For a long time there was no good first stage algorithm for cliquewidth, and originally rankwidth was in fact introduced, among other things, as a tool to help with computing a decomposition for cliquewidth [100].

Recently, an FPT algorithm was found that given an n-vertex graph G and a parameter k will decide if G has rankwidth at most k and if so output a rank decomposition of width k in time $O(f(k)n^3)$ [71]. Between rankwidth $rw(G)$ and cliquewidth $cw(G)$ we have the connection $rw(G) \leq cw(G) \leq 2^{rw(G)+1}$ and such a width k rank decomposition can be turned into a clique decomposition of width 2^{k+1} of the graph G. Because of the jump from rankwidth k to cliquewidth 2^{k+1} it is clear that based on the first stage rankwidth should be better suited than cliquewidth for getting the fastest possible algorithms.

The problem with rankwidth has been that the rank decomposition has until now not been known to be amenable to the second stage that involves a divide-and-conquer approach along the decomposition (for example with a dynamic programming). In this chapter we rectify the situation by showing how to enhance a rank decomposition, in order to make it amenable to dynamic programming. To this aim we proceed as follows.

We first revise in Section 6.1 some local operations on the rank decomposition of a given graph, which, roughly, results in considering some structural properties of cuts of the graph. Then, by noticing strong connections of those properties to modular graph

decomposition, we define the more general framework of H-join decomposition of graphs. After this, we point out how poor information, a rank decomposition in particular, and an H-join decomposition in general, could give for divide-and-conquer purposes. In order to counteract the lack, we propose in Section 6.2 a computation of further information for those decompositions, making them amenable to the divide-and-conquer paradigm (more specifically, the statement is given in Lemma 6.2). Finally, in Section 6.3 we illustrate the technique by an FPT dynamic programing algorithm solving the NP-hard problem of finding the clique number of a graph, given along with one of its H-join decompositions. We also discuss speed up issues when the H-join decomposition is restricted to the case of a rank decomposition of the input graph.

6.1 Some Structural Aspects of Cuts

The formal definition of a rank decomposition will be given in Section 6.3.2. In the following, let us give an informal discussion on rank decomposition in order to introduce the more general framework of H-join decomposition. Roughly, the basic idea of rank decomposition is to evaluate a *cut* $\{X, V \setminus X\}$ of a graph $G = (V, E)$ by the help of some ranking function $\rho_G : 2^V \to \mathbb{N}$, which is symmetric: $\rho_G(X) = \rho_G(V \setminus X)$. Then, the ranking function is sometimes considered, by abusive notations, as a function over the cuts of G, namely we abusively denote $\rho_G(\{X, V \setminus X\}) = \rho_G(X)$. A rank decomposition of G is then defined in a way analogous to branch decompositions as follows.

A *subcubic tree* is an unrooted tree where all internal nodes have degree three. If T and δ are such that T is a subcubic tree over $|V|$ leaves, and δ a bijection between V and the leaves of T, then for every edge uv of T, the removal of uv from T clearly induces a 2-partition of V. Finally, if we consider the 2-partition as a cut $\{X, V \setminus X\}$ of G, then we can associate every edge of T to a cut of G. The subcubic tree can then be evaluated w.r.t. the ranking value ρ_G, taken over all the edges of the tree. The minimum value of such an evaluation, taken over all possible subcubic trees, will then define the rankwidth of G, while any subcubic tree which has that minimum rankwidth value will be called a rank decomposition of G. A formal definition is given in Section 6.3.2. Nonetheless, according to this informal discussion, studying this kind of branch-like decomposition of graphs turns out to study a certain collection of cuts of the initial graph.

At the same time, we will see that the behaviour of the function ρ_G on a vertex subset X does not change much after the contraction of a *bi-twin* in X, where $x \in X$ is a *bi-twin* of $y \in X$ w.r.t. the cut $\{X, V \setminus X\}$ if, for every $s \in V \setminus X$, the adjacency between x and s is the same as that between y and s. An informal illustration is given in Figure 6.3. This

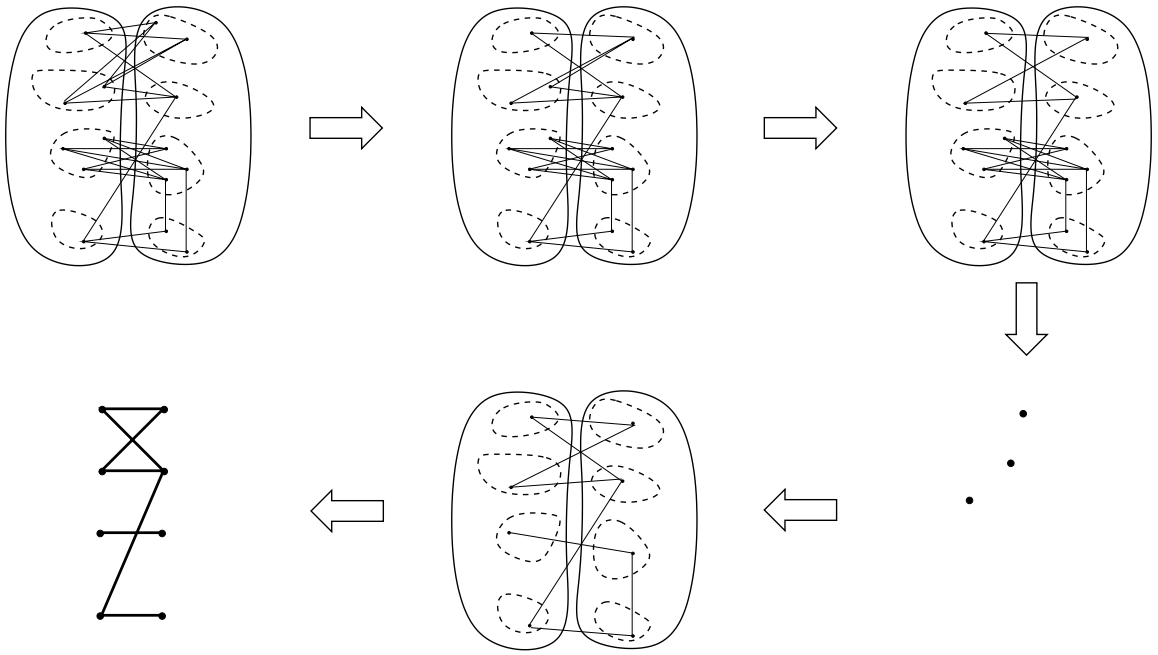


Figure 6.3: A sequence of cuts which preserve the value of the cut-rank function ρ_G.

leads us to the following more general formalism.

Definition 6.1 (*H*-join Operation) Let H be a bipartite graph over colour classes $V(H) = V_1 \cup V_2$. Let $V_1 = (a_1, a_2, ..., a_l)$ and $V_2 = (b_1, b_2, ..., b_r)$ be two orderings of the colour classes. Let A, B be two graphs with $P_A = (A_0, A_1, ..., A_l)$ and $P_B = (B_0, B_1, ..., B_r)$ ordered partitions of $V(A)$ and $V(B)$ respectively, with empty subsets allowed. We define H-join(A, B, P_A, P_B) as the graph having vertex set $V(A) \cup V(B)$ and edge set

$$E(A) \cup E(B) \cup \{uv : u \in A_i \wedge v \in B_j \wedge a_i b_j \in E(H)\}$$

for $1 \leq i \leq l, 1 \leq j \leq r$. Moreover, a graph G is said to be an *H-join of A and B* if $G = H$-join(A, B, P_A, P_B) for some P_A and P_B.

Roughly, the idea of an H-join operation is to associate the first configuration with the last configuration in Figure 6.3. Notice that since A_0 and B_0 can be equal to $V(A)$ and $V(B)$, the disjoint union of graphs A and B is an H-join of A and B for any graph H. Note also that since subsets A_i and B_j are allowed to be empty, we have for any induced subgraph F of H that any F-join of A and B is also realizable as an H-join of A and B. On the other hand, if (an induced subgraph) K is obtained from H after a twin contraction, or after the deletion of an isolated vertex, then any H-join of A and B is also realizable as a K-join of A and B.

The H-join operation is very general and quite powerful. If we decompose graphs through H-join operations in a way analogous to modular decomposition, then any graph

G can be recursively decomposed into trivial one-vertex graphs already by P_2-joins, where P_2 denotes the 2-vertex path. To see this let $V(G) = \{v_1, v_2, ..., v_n\}$, and note that we can construct the graph $G_i = G[\{v_1, v_2, ..., v_i\}]$ inductively starting from the trivial graph G_1, with G_i being the P_2-join of the trivial graph B on vertex v_i with the partition $P_B = (\emptyset, \{v_i\})$ and the graph G_{i-1} with the partition $P_{i-1} = (V(G_{i-1}) \setminus S_{i-1}, S_{i-1})$, where $S_{i-1} = \{N_G(v_i) \cap \{v_1, v_2, ..., v_{i-1}\}\}$. We will instead decompose graphs by H-joins in a way analogous to branch decompositions.

Definition 6.2 (H-join Decomposition) Let T be a subcubic tree and δ a bijection between the leaf set of T and the vertex set of a graph G. We say that (T, δ) is an *H-join decomposition of G* if for any edge uv of T we have G being an H-join of $G[S_u]$ and $G[S_v]$, where S_u, S_v is the 2-partition of $V(G)$ induced by the leaf sets of the two subtrees we get by removing uv from T. A graph having an H-join decomposition will be called an H-join decomposable graph.

A potential drawback of defining H-join decompositions simply as the pair (T, δ) is that, for every edge uv of T, we *a priori* do not know how to obtain ordered partitions P_u and P_v such that $G = H\text{-join}(G[S_u], G[S_v], P_u, P_v)$. Moreover, for dynamic programming we would need to build the graph G bottom-up and also for this it is not clear how to proceed. Notice that this is the main bottleneck for doing dynamic programming along a rank decomposition (cf. further in Section 6.3.2). This being said, we show in the upcoming section how to compute all the needed information. The starting idea is from the straightforward fact that, if such P_u and P_v as mentioned above exist, unordering each of them will result in two partitions (of S_u and S_v) which are both what we call external module partitions. Additionally, this can also be viewed as a link from H-join decomposition to modular decomposition.

Definition 6.3 (External Module Partition) Let G be a graph and let $A \subseteq V(G)$ be a vertex subset. An *external module partition of A* is a partition P_A of A such that, for every $s \in V(G) \setminus A$ and pair of vertices x, y belonging to the same part in P_A, we have x adjacent to s if and only if y adjacent to s.

6.2 Computing Enhanced Information for an H-join Decomposition

Throughout this section, we consider that only G, H, T, and δ are given as input for some computation, in such a way that (T, δ) is an H-join decomposition of graph G. The aim of this section is to output a rooted and labelled tree T_r which will ease a bottom-up approach

for solving problems on an instance defined by G using a dynamic programming. Among most important information, the tree T_r will come equipped with the ordered partitions of vertex sets used in the various H-join operations to incrementally build G bottom-up, starting with the trivial one-vertex graph. Let $n = |V(G)|$, $m = |E(G)|$, and $h = |V(H)|$.

We first focus on a useful subroutine which has input a vertex subset $A \subseteq V(G)$. It should output a maximum external module partition of A, which is well-defined by Lemma 6.1 below. We can process using partition refinement techniques: just initialize an unordered partition as $P = \{A\}$; then, for every exterior vertex $s \in V(G) \setminus A$, refine P using the neighbourhood of s as pivot. Those operations can be done in $O(m)$ time since each refinement operation can be done in time proportional to the size of the pivot set. (Partition refinement is a now standard algorithmic operation, we would refer the reader with further interests to, e.g., [68] for implementation details.) The correctness is stated in the following lemma.

Lemma 6.1 *Let G be a graph and A be a vertex subset of $V(G)$. The maximum (with respect to coarseness) external module partition of A is well-defined and can be computed in $O(|E(G)|)$ time.*

Proof: Let P be a maximal external module partition of A. Suppose it is not maximum, then there exists an external module partition Q of A such that there are some parts $X \in P$ and $Y \in Q$ such that X and Y overlap. Then, replace all X_i in P which overlap (or included in) Y by $\bigcup_i X_i \cup Y$, and obtain P'. Using the transitivity of the relation on x, y for a given s: "x and y are linked to s the same way", we can prove that P' is an external module partition that is coarser than P. Contradiction.

To finish, it suffices to prove that the subroutine described in the above text computes correctly a maximum external module partition. Here, the fact the computation results in an external module partition is straightforward from an argument by contradiction. Moreover, that partition is maximum since, for every external module partition P_A of A, for every $s \in V(G) \setminus A$, the neighbourhood of s does not overlap any part in P_A. □

We now address the problem of finding an order over the colour classes $V(H) = V_1 \cup V_2$ of H, $V_1 = (a_1, a_2, ..., a_l)$ and $V_2 = (b_1, b_2, ..., b_r)$, and for every edge uv of T a quadruplet (c_u, c_v, P_u, P_v) satisfying the following. Both c_u and c_v are pointers stating which colour class of H will be mapped to u, and respectively to v. Supposing w.l.o.g. that c_u maps u to V_1, then P_u and P_v will be two ordered partitions of S_u and S_v respectively. Moreover, we have $P_u = (A_0, A_1, A_2, \ldots, A_l)$ and $P_v = (B_0, B_1, B_2, \ldots, B_r)$, in such a way that $G = H\text{-join}(G[S_u], G[S_v], P_u, P_v)$, where S_u, S_v is the 2-partition of $V(G)$ induced by the leaf sets when removing uv from T. The following property will be important.

Proposition 6.1 *Let (T, δ) be an H-join decomposition of G. Let S_u, S_v be the 2-partition of $V(G)$ induced by the deletion of an edge uv in T. Let M_u, M_v be the (unordered) maximum external module partitions of respectively S_u and S_v. Let R_u and R_v be two sets containing exactly one representative vertex per part in respectively M_u and M_v. Let H_{uv} be the bipartite graph defined by the bipartite adjacency in G between R_u and R_v.*

Then, there is at most one isolated vertex in each colour class of H_{uv}. Furthermore, removing all isolated vertices from the graph H_{uv} results in an induced subgraph H' of H.

Proof: If H_{uv} has more than one isolated vertex in any of the two colour classes, then either M_u or M_v is not maximum. Let H' be the graph we get by removing all isolated vertices from H_{uv}. Now, since (T, δ) is an H-join decomposition of G, G can be obtained from $G[S_u]$ and $G[S_v]$ through an H-join operation. This H-join operation defines two external module partitions of S_u and S_v: say M'_u and M'_v. From Lemma 6.1, M_u (resp. M_v) is coarser than M'_u (resp. M'_v). We deduce that H' is an induced subgraph of H obtained by some successive twin contractions. □

Based on this property the algorithm proceeds as follows. Firstly, choose an order over the colour classes of H: $V_1 = (a_1, a_2, ..., a_l)$ and $V_2 = (b_1, b_2, ..., b_r)$. For every edge uv of the subcubic tree T, process all the following operations. Call the subroutine computing the maximum external module partitions of respectively S_u and S_v. Then, in both partitions, only keep one representative vertex per part, thus obtaining the sets R_u and R_v. Check in R_u and R_v whether there are (up to one each, from Proposition 6.1) isolated vertices w.r.t. the bipartite adjacency induced in G. If there are any isolated vertices, define A_0 and B_0 as the parts in the maximum external module partitions of S_u and S_v corresponding to them, then remove them definitively from R_u and R_v. Again from Proposition 6.1, the bipartite induced subgraph of G by colour classes R_u and R_v is an induced subgraph of H. Find a mapping of vertices in $R_u \cup R_v$ to those in $V(H)$ such that the bipartite adjacency between R_u and R_v (in G) coincides with that of $V(H)$ (in H) as follows.

Remark 6.1 *Given k-vertex graph F and n-vertex graph H such that $k \leq n$, one can brute-force solve the induced subgraph isomorphism problem on the pair (F, H) by first arbitrary ordering the vertices of F into $(v_1, v_2, \ldots, v_k)$, and then checking all possibilities of mapping an k-uplet $(u_1, u_2, \ldots, u_k)$ of vertices of $V(H)$. In case of success a mapping can also be outputted. The resulting time is in $O(k^2 \times n(n-1)(n-2)\ldots(n-k+1))$. This action will hereafter be referred to as brute-force finding an* induced subgraph matching.

After the brute-force mapping, we can determine whether R_u is mapped to vertices of V_1 or not, and assign the pointers c_u and c_v accordingly.

Convention: Since the pointers c_u and c_v are rather technical details and their use in the remaining of the chapter is quite marginal, we will skip referring to them and suppose, whenever we write $G = H\text{-join}(A, B, P, Q)$, that we know exactly which partition P and Q is mapped to which colour class of H.

Finally, we output the ordered partitions P_u and P_v made of: firstly the already defined A_0 and B_0, then parts in the maximum external module partitions of S_u and S_v, w.r.t. the order defined by $(a_1, a_2, \ldots, a_l)$, $(b_1, b_2, \ldots, b_r)$, and the latter mapping, with possibly the completion of absent positions by empty sets. The correctness is straightforward from the definition of an external module partition and an H-join operation. For complexity issues, the runtime of the brute-force induced subgraph matching is bounded by $O(h^2 2^{h \log h})$. Accordingly, the running time of the whole process is in $O(nm + nh^2 2^{h \log h})$.

Furthermore, for a dynamic programming perspective, we will use a rooted version: subdivide any edge in the subcubic tree T, set a root r in the subdivision and obtain T_r. Here, the same idea as in Proposition 6.1 can be used to prove a more useful result for dynamic programming. Actually, the main point here is to focus on computing labels for nodes of the rooted tree T_r that will tell us how to build up the graph G by H-join operations bottom-up on T_r.

Proposition 6.2 *Let (T, δ) be an H-join decomposition of G. Let T_r be the rooted tree obtained from setting a root r in a subdivision of some edge of T. Let w be an internal node of T_r, and a, b the children of w. In general, let S_x be the vertex subset of G mapped to the leaves of the subtree of T_r rooted at a node x; let M_x be the maximum external module partition of S_x in G; let $\overline{M}_x$ be the maximum external module partition of $V(G) \setminus S_x$ in G. Finally, let M'_a (resp. M'_b) be the restriction of $\overline{M_b}$ (resp. $\overline{M_a}$) to S_a (resp. S_b).*

Then, M'_a (resp. M'_b) is a coarser partition than M_a (resp. M_b). More specifically, in the graph $G[S_w]$, both M'_a and M'_b are maximum external module partitions of respectively S_a and S_b. A consequence is that there is a way to order M'_a and M'_b into Q_a and Q_b such that $G[S_w] = H\text{-join}(G[S_a], G[S_b], Q_a, Q_b)$. Finally, M_w is a coarser partition than $M_a \cup M_b$.

Proof: We first prove that, in the induced subgraph $G[S_w]$, M'_a is a maximum external module partition of S_a. Firstly, in that induced subgraph, M'_a is an external module partition otherwise $\overline{M_b}$ fails, in G, to be an external module partition of $V(G) \setminus S_b$. Secondly, if M'_a is not maximum, we can deduce that $\overline{M_b}$ is not maximum using the transitivity of the relation on x, y for a given s: "x and y are linked to s the same way". By symmetry, M'_b is a maximum external module partition of S_b in $G[S_w]$. Then, M'_a (resp. M'_b) is a coarser partition than M_a (resp. M_b) since the latter is an external

module partition in $G[S_w]$. To finish, in the graph G, if $M_a \cup M_b$ is not an external module partition of S_w, then either M_a or M_b fails to be an external module partition of S_a or S_b, respectively. Then, using Lemma 6.1 and the maximality of M_w, we can conclude that M_w is a coarser partition than $M_a \cup M_b$. □

Let w be an internal node of T_r, and a, b the two children of w. In general, let S_x be the vertex subset of G mapped to the leaves of the subtree of T_r rooted at a node x. Moreover, unless x is the root, let P_x and $\overline{P_x}$ be the ordered partitions computed by the previous computation for the edge of T linking x to its father in T_r. Finally, let P'_a (resp. P'_b) be the restriction of $\overline{P_b}$ (resp. $\overline{P_a}$) to S_a (resp. S_b). Our main focus will be to compute, for all w, two re-orderings Q_a and Q_b of respectively P'_a and P'_b such that $G[S_w] = H\text{-join}(G[S_a], G[S_b], Q_a, Q_b)$.

Notice that P'_a and P'_b are not already known. However, from the above proposition, unordering them results in coarser partitions M'_a and M'_b of those partitions M_a and M_b we get by unordering P_a and P_b. We can compute a representation for P'_a (and analogously for P'_b) as follows. W.l.o.g. assume $\overline{P}_b = (P_0, P_1, P_2, \ldots, P_l)$ (otherwise just replace l by r). Create a table P'_a of size $l + 1$. Pick one representative vertex per non-empty part in P_a (up to h vertices). For every such vertex: determine which P_i it belongs to; then create a double-pointer between the i^{th} cell of the table P'_a and the part in P_a which corresponds to the representative vertex. Each membership computation can be done in constant time because we can store both P_a and $\overline{P}_b$ as tables of pointers to the same table of vertices in $V(G)$, and subsequently follow the pointers. After the scanning, for every cell P in P'_a, the position of its (present/absent) pointers to some part of P_a can be viewed under a bit-vector representation I w.r.t. the order defined by P_a, namely with $P = \bigcup_{i \in I} A_i$ and $P_a = (A_0, A_1, A_2, \ldots, A_{|P_a|})$. This can be computed in $O(h)$ time for each P. The bit-vector, in turn, can be viewed as a number written in binary: we will not use the pointers in the following so assign that number to the cell instead. Thus, for each w, we can compute P'_a and P'_b in $O(h^2)$ time.

By a similar technique, we add information to P_w by performing the same representation for every part P of the partition P_w, following the statement where P_w is a coarser partition than $P_a \cup P_b$. Here, we would rather have two numbers for each part P: one to the positions in P_a and the other to the positions in P_b.

Let us head back to P'_a and P'_b. From Proposition 6.2, unordering each of P'_a and P'_b results in a maximum external module partition in $G[S_w]$. Besides, $G[S_w]$ is an H-join of $G[S_a]$ and $G[S_b]$, completing to form similar conditions as in the claim of Proposition 6.1. (To see that $G[S_w]$ is an H-join of $G[S_a]$ and $G[S_b]$, we can for instance consider the restriction of $G = H\text{-join}(G[S_a], G[V(G) \setminus S_a], P_a, \overline{P}_a)$ to the partitions P_a and P'_b). Then,

with a similar approach as before, we can compute in $O(h^2 2^{\frac{h}{2}})$ time a new order for parts of P'_a and P'_b such that (w.l.o.g.) $Q_a = (A'_0, A'_1, A'_2, \ldots, A'_l)$, $Q_b = (B'_0, B'_1, B'_2, \ldots, B'_r)$, and $G[S_w] = H\text{-join}(G[S_a], G[S_b], Q_a, Q_b)$. From what has been said above, each A'_i and each B'_j is assigned a number which represents the positions of parts in respectively P_a and P_b it is made of. We have shown the following.

Lemma 6.2 (Main Tool) *Let (T, δ) be an H-join decomposition of G. We can in time $O(nm + nh^2 2^{h \log h})$ output a rooted tree T_r, an order over the colour classes of H, $V_1 = (a_1, a_2, ..., a_l)$ and $V_2 = (b_1, b_2, ..., b_r)$, and for every node w of T_r with children a, b ordered partitions $Q_a, Q_b, P_w, \overline{P}_w$ such that*

$$G[S_w] = H\text{-join}(G[S_a], G[S_b], Q_a, Q_b) \quad \text{and} \quad G = H\text{-join}(G[S_w], G[V(G) \setminus S_w], P_w, \overline{P}_w).$$

Moreover, each part P in Q_a (resp. Q_b) is assigned a number representing the set of indices of the parts in P_a (resp. P_b) that P is made of. Besides, each part P in P_w is assigned two numbers representing the two sets of indices of the parts in P_a and those in P_b that P is made of as union of the two.

6.3 Dynamic Programming

The dynamic programming algorithm solving a problem $\mathcal{P}$ on an H-join decomposable graph G given with its H-join decomposition (T, δ) will first compute the rooted tree T_r and associated information as described in Lemma 6.2. It will then follow a bottom-up traversal of T_r. With each node w of T_r we associate a data structure *table*. The table of a leaf is initialized according to the base case, usually by a brute-force strategy. The table of an interior node is filled based on previously filled tables of its children. The overall solution is obtained from the table at the root of T_r. The table associated with a node w of T_r contains optimal solutions to a subproblem, a constrained version of the problem $\mathcal{P}$, restricted to the graph $G[S_w]$, where S_w is the set of vertices of G mapped to leaves of the subtree of T_r rooted at w. The subproblem constraints focus on the subsets of S_w that interact in specific ways with the external module partitions of the H-join operations.

6.3.1 MaxClique of H-join Decomposable Graphs

In this section we illustrate the technique for the NP-hard problem of computing the size of the maximum clique in a graph G. We will prove the following result.

Theorem 6.1 *Given an n-vertex m-node graph G, and an H-join decomposition (T, δ) of G, we can in $O(nm + nh^2 2^{h \log h} + nh\beta_H 2^{\max(l,r)})$ time find the clique number of G, where β_H is the number of maximal bicliques of the (l, r)-bipartite graph H, and $h = l + r$.*

In order to deal with H-join operations, which will be viewed as cuts, we will need the following formalism.

Definition 6.4 (Biclique) A *biclique* (L, R) of a graph G is defined as two vertex subsets L and R such that the bipartite adjacency in G between L and R forms a complete bipartite graph (note that we allow adjacencies inside L and R). If A, B is a 2-partition of $V(G)$, we denote by $\beta_G(A, B)$ the set of maximal bicliques of G among the bicliques (L, R) for which $L \subseteq A$ and $R \subseteq B$ (maximal in the ordering where (L_1, R_1) is defined to be smaller than (L_2, R_2) if we have both $L_1 \subseteq L_2$ and $R_1 \subseteq R_2$, giving a relation which is a poset over the bicliques of G).

Note that (L, R) is a biclique if and only if (R, L) is a biclique, and that $(L, \emptyset)$ and $(\emptyset, R)$ are bicliques for all L and R. Finally, for a graph G, let $cl(G)$ be the size of the maximum clique of G, and for $S \subseteq V(G)$, let $cl(S) = cl(G[S])$.

Proposition 6.3 *Let G be a graph. Let A, B be a 2-partition of $V(G)$ and let $S \subseteq V(G)$. Then,*

$$cl(S) = \max\{cl(K_A \cap S) + cl(K_B \cap S), \quad (K_A, K_B) \in \beta_G(A, B)\}.$$

Proof: That $cl(S)$ is greater than or equal to the value of the right hand-side is straightforward since each situation in the right hand-side corresponds to a clique of $G[S]$. We prove the converse by exhaustive checking over the cliques of $G[S]$. Let K be a clique of $G[S]$. Clearly, $(K \cap A, K \cap B)$ is a biclique of G: let $(K_A, K_B) \in \beta_G(A, B)$ be such that $K \cap A \subseteq K_A$ and $K \cap B \subseteq K_B$. Then $K \cap A$ is a clique of G with $K \cap A \subseteq K_A \cap S$: we deduce $|K \cap A| \leq cl(G[K_A \cap S])$. To conclude, notice that the same holds for B, and that $|K| = |K \cap A| + |K \cap B|$. □

Note that all bicliques (L, R), with L and R nonempty, of a bipartite graph H with colour classes $V_1 \cup V_2 = V(H)$ follow the cut $\{V_1, V_2\}$. Accordingly, the set of maximal bicliques of H can be viewed as $\beta_H(V_1, V_2)$. The following proposition defining the set of bicliques $\beta_G(A, B)$ of a graph G resulting from an H-join of $G[A]$ and $G[B]$ follows directly from the definitions.

Proposition 6.4 *Let G be a graph, and A, B a 2-partition of $V(G)$. Let H be a bipartite graph with colour classes $V_1 \cup V_2 = V(H)$ with given ordering $V_1 = (a_1, a_2, \ldots, a_l)$ and $V_2 = (b_1, b_2, \ldots, b_r)$. Let $P_A = (A_0, A_1, A_2, \ldots, A_l)$ and $P_B = (B_0, B_1, B_2, \ldots, B_r)$ be two ordered partitions of A and B. If $G = H$-join$(G[A], G[B], P_A, P_B)$, then*

$$\beta_G(A, B) = \left\{ \left(\bigcup_{i \in I} A_i, \bigcup_{j \in J} B_j \right), \quad \text{where } \left(\bigcup_{i \in I} a_i, \bigcup_{j \in J} b_j \right) \in \beta_H(V_1, V_2) \right\}.$$

Following Lemma 6.2 we have computed information on T_r such that for any node w of T_r with children a and b the incident labels of a, b, and w give us the 8 ordered partitions P_a, P_b, $\overline{P}_a$, $\overline{P}_b$, Q_a, Q_b, P_w, and $\overline{P}_w$ such that

- $G = H\text{-join}(G[S_a], G[V(G) \setminus S_a], P_a, \overline{P}_a)$,
- $G = H\text{-join}(G[S_b], G[V(G) \setminus S_b], P_b, \overline{P}_b)$,
- $G = H\text{-join}(G[S_w], G[V(G) \setminus S_w], P_w, \overline{P}_w)$, and
- $G[S_w] = H\text{-join}(G[S_a], G[S_b], Q_a, Q_b)$.

W.l.o.g. let

- $Q_a = (A'_0, A'_1, A'_2, ..., A'_l)$ and $Q_b = (B'_0, B'_1, B'_2, ..., B'_r)$,
- $P_a = (A_0, A_1, A_2, ..., A_{|P_a|})$ and $P_b = (B_0, B_1, B_2, ..., B_{|P_b|})$, and
- $P_w = (W_0, W_1, W_2, ..., W_{|P_w|})$.

We now describe the bottom-up dynamic programming on T_r to solve MaxClique. Let us consider that we are computing at the level of node w having children a, b and that $cl(S)$ is known for all combinations S of parts that are all in P_a or that are all in P_b, in other words we have $I \subseteq \{1, 2, ..., |P_a|\}$ and $S = \bigcup_{i \in I} A_i$ or $I \subseteq \{1, 2, ..., |P_b|\}$ and $S = \bigcup_{i \in I} B_i$. Let us briefly describe the data structure needed to store and access $cl(S)$. If $S = \bigcup_{i \in I} A_i$ is given by a bit-vector representation of I w.r.t. the order of the A_i defined by P_a, then one can view I as a number written in binary, and $cl(S)$ can then be obtained in constant time by accessing the I^{th} cell of a table data-structure (this table is of size $2^{|P_a|} \leq 2^{\max(l,r)}$). Let us first describe how to compute the maximum clique of the graph $G[S_w]$, i.e. the graph induced by the vertices of G mapped to leaves of the subtree rooted at node w of T_r. This value $cl(S_w)$ will be stored in a table at node w and computed based on the cl values already computed and stored in tables at its children a and b. For simplicity we do not distinguish explicitly between tables in the formulae that follow, since it is clear from context where the cl values are stored.

$$cl(S_w) = \max \left\{ cl(\bigcup_{i \in I} A'_i) + cl(\bigcup_{j \in J} B'_j) \right\}, \quad \text{where } \left(\bigcup_{i \in I} a_i, \bigcup_{j \in J} b_j \right) \in \beta_H(V_1, V_2).$$

The proof that this correctly computes $cl(S_w)$ follows from a combination of Proposition 6.3, Proposition 6.4, and the fact that $G[S_w] = H\text{-join}(G[S_a], G[S_b], Q_a, Q_b)$. Note that from Lemma 6.2 we know that all the needed values have already been computed.

For example, A'_1 is the union of some A_i, and we have already computed a numeral representation for that union. Then, every value of $cl(\bigcup_{i\in I} A'_i)$ can be obtained from the inductive hypothesis by adding all numeral representations of A'_i ($i \in I$): the global sum will represent the set Σ_I with $\bigcup_{i\in\Sigma_I} A_i = \bigcup_{i\in I} A'_i$, and we can read the corresponding cl-value in the ${\Sigma_I}^{th}$ cell of the cl-table data structure given by the inductive hypothesis on P_a. (More precisely: the xor of two bit-vectors is exactly the symmetric difference of the corresponding sets; in the case (here) where the sets are pairwise disjoint, the symmetric difference is exactly the union while the xor is exactly the sum.) For complexity issues, note that the implicit ordering of $V(H)$, Q_a, and Q_b, implies that for all I, each access to the numerical representation of each A'_i (for i varying in I) takes constant time. Then, the computation of Σ_I for each I requires $O(h)$ time since $|I| \leq |V(H)|$. Accordingly, it takes $O(h\beta_H)$ time to compute $cl(S_w)$.

Now we need to provide the inductive hypothesis for the parent of w. Namely, we will need to compute $cl(S)$ for any subset S such that $S = \bigcup_{k\in K} W_k$ for any $K \subseteq \{1, 2, ..., |P_w|\}$. Again, from a combination of Proposition 6.3, Proposition 6.4, and the fact that $G[S_w] = H\text{-join}(G[S_a], G[S_b], Q_a, Q_b)$, the following statement is correct: for all $K \subseteq \{1, 2, \ldots, |P_w|\}$, if $S = \bigcup_{k\in K} W_k$, then

$$cl(S) = \max\left\{ cl(\bigcup_{i\in I} A'_i \cap S) + cl(\bigcup_{j\in J} B'_j \cap S) \right\}, \quad \text{where } \left(\bigcup_{i\in I} a_i, \bigcup_{j\in J} b_j\right) \in \beta_H(V_1, V_2).$$

Now, from Lemma 6.2, every W_k is the union of some A_i and some B_j. Moreover, we have assigned, for each W_k, two numeral representations of respectively I_k and J_k such that $W_k = W_k(a) \cup W_k(b)$, where $W_k(a) = \bigcup_{i\in I_k} A_i$ and $W_k(b) = \bigcup_{j\in J_k} B_j$. Let us have a closer look to the right hand side of the above equality. Firstly, for the maximal biclique $(V_1, \emptyset)$ we need $cl(S_a \cap S)$ and note that $S_a \cap S = \bigcup_{k\in K} W_k(a)$ and we can access its cl-value via $\Sigma_{K,a} = \sum_{k\in K} I_k$ and the cl-table data structure provided by the inductive hypothesis on P_a (like before with the union of some of the A'_i, those $W_k(a)$ are pairwise disjoint). The same goes for maximal biclique $(\emptyset, V_2)$ and $cl(S_b \cap S)$. Consider $\bigcup_{i\in I} A'_i \cap S = \bigcup_{i\in I} A'_i \cap \bigcup_{k\in K} W_k(a)$. Recall that we have just computed the value of $\Sigma_{K,a}$ such that $X = \bigcup_{k\in K} W_k(a) = \bigcup_{i\in\Sigma_{K,a}} A_i$. Also, from the previous computation of $cl(S_w)$, we have computed Σ_I such that $Y = \bigcup_{i\in I} A'_i = \bigcup_{i\in\Sigma_I} A_i$. We now want the value $\Sigma_{I,K,a}$ such that $X \cap Y = \bigcup_{i\in\Sigma_{I,K,a}} A_i$. This can be done by an $O(h)$ scanning. Accordingly, we can access the cl-value of $\bigcup_{i\in I} A'_i \cap S$ via $\Sigma_{I,K,a}$ and the cl-table data structure provided by the inductive hypothesis on P_a. The same can then be done for $\bigcup_{j\in J} B'_j \cap S$. For the running time, we need $O(h\beta_H)$ time for each $K \subseteq \{1, 2 \ldots, |P_w|\}$.

Initializing at the leaves of T_r is trivial, since $cl(v) = 1$ for a vertex v and $cl(\emptyset) = 0$. The maximum clique of G is found as $cl(V(G))$ at the root r of T. Summing up, the runtime for each node w is in $O(h\beta_H 2^{\max(l,r)})$. Hence, the global runtime of the dynamic programming stage is $O(nh\beta_H 2^{\max(l,r)})$. From Lemma 6.2, the runtime for computing the enhanced information needed to perform the dynamic programming is $O(nm+nh^2 2^{h \log h})$ and from this Theorem 6.1 follows.

6.3.2 MaxClique of Graphs of Bounded Rankwidth

Let us first give the definition of rankwidth.

Definition 6.5 (Rank-Decomposition) For any graph G, the cut-rank function ρ_G is defined over every vertex subset $X \subseteq V(G)$ as the rank of the $X \times V(G) \setminus X$ submatrix of the adjacency matrix of G. For any pair (T, δ) with T a subcubic tree and δ a bijection between vertices of G and leaves of T, (T, δ) is defined as a width r rank decomposition of G if for every edge uv in T, the cut-rank of S_u is at most r, where S_u, S_v is the 2-partition of $V(G)$ induced by the leaf sets of the two subtrees we get by removing uv from T. The rankwidth of G is the minimum integer r such that there exists a width r rank decomposition of G.

Definition 6.6 (Bipartite Graph R_k) For a positive integer k we define a bipartite graph R_k having for each nonempty subset S of $\{1, 2, ..., k\}$ a vertex $a_S \in A$ and a vertex $b_S \in B$, with $V(R_k) = A \cup B$. This gives $2^k - 1$ vertices in each of the colour classes A and B. Two vertices a_S and $b_{S'}$ are adjacent if and only if $|S \cap S'|$ is odd.

Lemma 6.3 *The function $\sigma_G : 2^{V(G)} \to \mathbb{N}$ defined by*

$$\sigma_G(X) = \min\{k : G \text{ is an } R_k\text{-join of } G[X] \text{ and } G[V(G) \setminus X]\}$$

is equal to the cut-rank function ρ_G.

Proof: Let $k = \rho_G(X)$ be the cut-rank value of X. There are several ways to view the graph R_k. Before proving the lemma, note the following, where we slightly abuse the notation of Definition 6.6 by denoting the vertices arising from a one-element subset $S = \{i\}$ simply as a_i and b_i. We denote by M_k the bipartite adjacency matrix of the bipartite graph R_k, meaning that its rows correspond to the vertices of one colour class and the columns to those of the other colour class. Suppose that the vertices $a_1, a_2, \ldots, a_k$ are mapped to rows in M_k: again by abuse on the notation, we can view vertex of R_k as a row/column it is mapped to in M_k. Let a_S be a vertex of R_k with $S = i_1, i_2, \ldots, i_p$.

We can prove that in M_k, the row a_S is the $GF(2)$-sum of the rows $a_{i_1}, a_{i_2}, \ldots, a_{i_p}$: for every column $b_{S'}$ of M_k, $|S \cap S'|$ is odd iff there is an odd number of the i_q $(1 \leq q \leq p)$ which belong to S', that is M_k has a 1 in the row a_{i_p} and column $b_{S'}$. The same holds for $b_1, b_2, \ldots, b_k$. Note also that an arbitrary bipartite adjacency matrix is not necessarily symmetric but it is clear here that

Claim: There is a way to swap the columns and rows of M_k to result in a symmetric matrix. Also, adding one column and one row to M_k with all 0's inside will result in a matrix of rank k of maximum size.

Moreover, let us w.l.o.g. define M_k in such a way that $\{a_1, a_2, \ldots, a_k\}$ are mapped (in this order) to the first k rows of M_k while $\{b_1, b_2, \ldots, b_k\}$ are mapped to the k first columns. This way, the first $k \times k$ block of M_k is equal to the identity matrix of size k. We define L_k as the block of M_k made of the first k rows. Clearly, L_k has $2^k - 1$ columns and has no column with only 0's.

We now come to the actual proof of the lemma. We first prove that $\sigma_G(X) \leq k$. Let M be the bipartite adjacency matrix induced by X and $V(G) \setminus X$ in G. A *valid elimination* in a matrix is a deletion of a column (resp. a row) when, either the column (resp. row) is a 0-vector, or the matrix has another column (resp. row) identical to the one we delete. Let us obtain N from M through a maximal sequence of valid eliminations. This operation corresponds more or less to the contraction with respect to some external module partition (to the absence of 0-vectors). Then, in order to prove that G is an R_k-join of $G[X]$ and $G[V(G) \setminus X]$, it suffices to prove that the bipartite graph G_N with bipartite adjacency matrix N is an induced subgraph of R_k. This will be proved in two steps.

There cannot be less than k rows in N. If the number of rows in N is exactly k, then we look at N as a collection of columns. By maximality of the sequence of valid eliminations, all the latter columns are pairwise distinct and are all non-null. Besides, if we look at L_k as a collection of columns, then by definition L_k contains all possible non-null k-bit vectors. Therefore, N (as a collection of columns) is a subset of L_k. Hence, G_N is an induced subgraph of the bipartite graph defined by L_k, and consequently it is an induced subgraph of R_k. If the number of columns in N is exactly k, then by transposition we can conduct a similar argument to conclude.

Otherwise we take k rows of N which induce a k-basis of the matrix N. Putting those k rows together results in a matrix Z of k rows. Besides, the other rows of N are linear combinations of those k rows. Therefore, the columns of Z are pairwise distinct otherwise there would be identical columns in N, which contradicts the maximality of the sequence of valid eliminations. Then, the previous argument applies, and every column of Z is a

column of L_k: w.l.o.g. suppose Z is a block of L_k (otherwise swap columns). Let T be a set of rows which contains all linear combinations of rows of Z. Now, the set of rows of M_k contains every linear combination of rows of L_k, and Z is a block of L_k. Consequently, we can suppose w.l.o.g. that T is a block of M_k (otherwise just swap rows). Then, the bipartite graph G_T defined by T is an induced subgraph of R_k. Besides, it is clear that every row of N belongs to T and G_N is an induced subgraph of G_T. Hence, G_N is an induced subgraph of R_k.

We now prove that $\rho_G(X) \leq \sigma_G(X)$. Let $l = \sigma_G(X)$. The R_l-join operation of $G[X]$ and $G[V(G) \setminus X]$ defines two external module partitions P and Q of X and $V(G) \setminus X$. Let Y and Z contain one representative vertex per part in respectively P and Q. If there are isolated vertices in the bipartite graph defined by the bipartite adjacency in G between Y and Z, then remove them from Y or Z accordingly. Then, the cut-rank value $\rho_G(X)$ is equal to the rank of the bipartite adjacency matrix M between the remaining Y and Z. Clearly, the graph defined by M is an induced subgraph of R_l from Proposition 6.1. Hence, the cut-rank value $\rho_G(X)$ cannot exceed the rank of the bipartite adjacency matrix of R_l, which is equal to l. □

Theorem 6.2 *(T, δ) is a width k rank decomposition of G if and only if (T, δ) is an R_k-join decomposition of G. Thus G is a graph of rankwidth at most k if and only if G is an R_k-join decomposable graph.*

Theorem 6.2 follows from Lemma 6.3 and that R_l is an induced subgraph of R_k for all $l \leq k$. A consequence of Theorem 6.2 is that one can apply the generic dynamic programming for H-join graphs on graphs of rankwidth bounded by k. Moreover, R_k-join decompositions are obviously particular cases of H-join decompositions. Then, there is for example a runtime improvement of Lemma 6.2 for R_k, stated in Lemma 6.4 below. Actually, the complexity bottle-neck in the generic runtime given in Lemma 6.2 is the brute-force induced subgraph matching, which runs in $O(2^{2k+(k+1)2^{k+1}})$ for R_k. Let us improve the bound to $O(k^2 2^{2k^2})$. Recall that R_k is a bipartite graph having for each nonempty subset S of $\{1, 2, ..., k\}$ a vertex $a_S \in A$ and a vertex $b_S \in B$, with $V(R_k) = A \cup B$. Two vertices a_S and $b_{S'}$ are adjacent iff $|S \cap S'|$ is odd. Let us solve the induced subgraph matching of a bipartite graph F to R_k, when we know that F is indeed an induced subgraph of R_k.

The general idea is to first brute-force compute a $GF(2)$-basis of the bipartite adjacency matrix M_F of F. To find one, let us use the characterization where a basis is a *maximal* set of linearly independent vectors. We first check the columns: $\mathcal{C}$ is initialized to be an empty list. For every column c of M_F do: if $\mathcal{C} + c$ is linearly independent

then add c to $\mathcal{C}$. The test whether a set $S = \{c_1, c_2, \ldots, c_p\}$ is linearly independent is as follows: for every $\{i_1, i_2, \ldots, i_p\}$ with $i_q \in \{0, 1\}$ and at least one $i_q \neq 0$ check if $i_1c_1 + i_2c_2 + \cdots + i_pc_p = 0$; if there is none of such then the answer is positive. The correctness follows directly from the above characterization of a basis. For complexity issues, there are at most 2^k columns in M_F, each of size at most 2^k. Besides, the rank of M_F is known to be at most k (i.e. the number $p = |S|$ in the above is at most k). Finally, checking if $i_1c_1 + i_2c_2 + \cdots + i_pc_p = 0$ requires at most $p2^k$ time. Hence, we can compute a basis $\mathcal{C} = \{c_1, c_2, \ldots, c_l\}$ of the columns of M_F in $O(k2^{3k})$ time. Likewise, we compute a basis $\mathcal{R} = \{r_1, r_2, \ldots, r_l\}$ of the rows of M_F in the same runtime.

Each element of $\mathcal{C}$ and $\mathcal{R}$ actually corresponds to a unique vertex in F: by abuse in the terminology we says that $\mathcal{C}$ and $\mathcal{R}$ are vertex subsets of F. The idea now is to solve the induced subgraph matching of $F[\mathcal{C} \cup \mathcal{R}]$ and R_k using the brute-force algorithm described in Remark 6.1. Then, the property where a basis is a spanning set of vectors can be used to map the remaining vertices of M_F to R_k as follows. For every vertex of F that is not in $\mathcal{C} \cup \mathcal{R}$, say w.l.o.g. a column c in M_F, we look at the adjacency defined between c and $\mathcal{R}$, and obtain an l-bit vector. From the bipartite adjacency matrix of $F[\mathcal{C} \cup \mathcal{R}]$ and the latter l-bit vector, find the (unique) linear combination such that $i_1c_1 + i_2c_2 + \cdots + i_lc_l = c$. Let $(a_{S_1}, a_{S_2}, \ldots, a_{S_l})$ be the vertices of R_k mapped to $(c_1, c_2, \ldots, c_l)$. Then, map c to a_S with $S = i_1S_1\Delta i_2S_2\Delta \ldots \Delta i_lS_l$, where Δ is the symmetric difference operation (also-called xor). The correctness follows from the spanning property of the basis. For complexity issues, brute-force matching induced subgraph $F[\mathcal{C} \cup \mathcal{R}]$ (of F hence) of R_k requires $O(l^2 \times C^l_{2^k} \times C^l_{2^k}) = O(k^2 2^{2k^2})$ runtime. Then, finding the l-bit vector requires l time; finding the linear combination by exhaustive checking all possibilities requires $l \times 2^l$ time; computing a_S requires at most l xor operations on sets of size k, taking at most $l \times k$ time. Hence, the overall complexity is in $O(k^2 2^{2k^2})$. We have shown that

Lemma 6.4 *When $H = R_k$, the runtime given in Lemma 6.2 to find an enhanced version of R_k-decomposition can be improved to $O(nm + nk^2 2^{2k^2})$.*

Besides, an arbitrary bipartite graph H over h vertices can have $2^{\frac{h}{2}}$ pairwise distinct maximal bicliques, but

Proposition 6.5 *For every $k \geq 1$, R_k has no more than 2^{2k^2} maximal bicliques.*

Proof: The cock-tail party graph $CP(j, j)$ ($j \geq 1$) is the bipartite complement graph of an (j, j)-matching bipartite graph. This $2j$-vertex graph has exactly 2^j maximal bicliques [102]: a maximal biclique is obtained for each combination of a vertex subset of the first colour class with the set of all vertices of the other colour class which are not

hit by former vertices through the (j, j)-matching. By a theorem of E. Prisner [102], if a (p, q)-bipartite graph H does not contain $CP(j + 1, j + 1)$ as an induced subgraph, then H has at most $(pq)^j$ maximal bicliques. Now, if R_k has an induced $CP(k+1, k+1)$, then the rank of the bipartite adjacency matrix of R_k would exceed k. □

Eventually, as a combination of Theorem 6.1, Theorem 6.2, Lemma 6.4, and Proposition 6.5, we obtain a dynamic programming solving the MaxClique problem on graph of bounded rankwidth. The solution directly follows along the tree of the rank decomposition of the graph.

Corollary 6.1 *Given an n-vertex m-edge graph G along with a width k rank decomposition (T, δ) of G, one can solve MaxClique on G in $O(nm + n2^{2k^2+k+2^k})$ time.*

To conclude, rankwidth is a quite recent graph parameter, introduced by S. Oum and P. Seymour [96, 100], whose value for a graph is lower than both its cliquewidth and its branchwidth [99]. Algorithms for NP-hard problems on graphs of bounded rankwidth have so far relied on dynamic programming on graphs of bounded cliquewidth and the fact that the cliquewidth of a graph is no more than an exponential function of its rankwidth. In this chapter we have shown that using the connection to H-join decompositions we can do dynamic programming directly on the rank decomposition.

There are various questions that can be asked for future research. Can the equivalence between R_k-join decompositions and rank decompositions of width k be used to find vertex minors [97] for bounded rankwidth? Can the equivalence be used to find a logical formula describing rank decompositions of width k? Can the equivalence be used to solve further problems in FPT time for graphs of bounded rankwidth? Since the rankwidth is relatively small for large classes of graphs, the most interesting algorithmic problem may be to use the equivalence to carefully design dynamic programming algorithms for specific problems, and classes of problems, on graphs of bounded rankwidth. For example, we believe that with some effort the MaxClique problem could be solved for rankwidth k graphs in time singly exponential in k.

Concluding Remarks

In the first part of the thesis, we have presented a study on some combinatorial issues around set families. In order to estimate the number of set families satisfying some closure axioms, we have developed new tools and techniques to find tree-like representations for them, with the most interesting tool being probably the general framework for representing set families using cross-free decomposition trees. It was presented in Chapter 1. Under this framework, we succeeded in obtaining polynomial results for two new classes of set families. Each of them could be seen as responsible for a new combinatorial decomposition, with one being a strict generalization of the modular decomposition of digraphs, and the other being a strict generalization of the clan decomposition of 2-structures. Both decompositions are polynomially computable.

In the second part of the thesis, we have presented various algorithmic results on discrete structures, mostly modelled into graph theoretic problems. The leading idea was the duality between the divide-and-conquer algorithmic framework and some structural decomposition of the corresponding input instances. We pointed out how one can find noteworthy structural properties of the family of vertex subsets which induce connected subgraphs in a given set of graphs. Using this we gave efficient algorithmic solutions for, not only the proper computation of the previously mentioned common connected sets themselves, but also a related computation of the so-called cograph sandwiches. Still using the same properties, we revisited an algorithm given by T. Uno and M. Yagiura, (in-)famous for its tough correctness proof.

At the same time, we deepened some structural aspects of this Uno-Yagiura algorithm and established an intersecting submodular property for a very general framework of modular decomposition. This was captured in Lemma 5.2 in Chapter 5, and could be seen as the most representative example of the idea behind the second part of the composition, namely to derive combinatorial results from the analysis of some tricky algorithms.

Finally, we have, in the closing chapter, attempted to give a general standpoint over various graph decompositions, ranging from modular, split, and bijoin decompositions, to clique, NLC-, and rank decompositions. However, on this (hot-)topic, we have restricted

our discussion to only some algorithmic issues. We showed how a rank decomposition in particular, and an H-join decomposition in general, can be made amenable to dynamic programming solutions for NP-hard optimization problems, such as the problem of finding the clique number of a given graph.

Turning our attention to perspectives, the purpose of the last chapter is manifold. Even though our composition was restricted to algorithmic aspects, the framework we present opens the way for various directions, ranging from basic speed up algorithmic questions to more complex issues such as those related to graph logic theory and those related to the well-quasi ordering of graphs w.r.t. the vertex minor relation. The end of Chapter 6 gave more precisions on those questions.

Besides, fixed-parameter tractable (FPT) algorithmics also presents highly challenging questions. This research field has close connections to width-based decompositions of graphs, such as those related to the treewidth, the cliquewidth, and also the rankwidth measures. In the latter list, rankwidth, though being a very recent notion, owns some important algorithmic results. It should be interesting to explore more on the impact of rank decomposition on FPT algorithmic graph theory.

Let us head back to the first part of the thesis, namely to some discrete bijections over set families in general. A simple, yet hitting, critique of our composition could be its ubiquitous obsession on two *and only two* set operations: the overlapping and the crossing relations. A broader perspective could be the examination of other binary set operations under a similar framework as that presented in Chapter 1. From a similar point of view, a second simple remark on our composition is that, not only the classes of families we address are defined using closure axioms, but also (and more importantly) all the corresponding axioms are based on *only* binary set operations. For instance, a union-difference family is a family which is closed under the union and the difference of its overlapping members, where the overlapping relation is a relation over pairs of members of the family. A broader perspective could be the examination of closure axioms using more arity.

We continue the perspective list with a few words in connection with the very first paragraph of our composition, in the introduction section. There, we have seen that collections of collections of objects – so-called set families – allow broader modelling perspectives than just collections of objects. We have also seen that a large branch of combinatorics in fact does not address set families in general, but their restriction to simple graphs, i.e. families of 2-element sets. Accordingly, much as set families go one step further in modelling than elementary sets, collections of collections of collections of elementary objects go one step further than set families. This formalism comes under

the name of a class of set families. It could be interesting to study such a notion. In other words, if we have seen how fruitful it is to upgrade from an element to a set, and subsequently to a family, then it could be interesting to go one step further and upgrade from a family to a class. Beside this, much as graphs are interesting instances of set families in combinatorics, classes of graphs could be interesting instances of classes of set families. Finally, and according to all what has been said, studying problems with input made by a class of graphs will give a broader modelling perspective than the study of problems with input made by one graph. It could be interesting to study such problems. They have connections to the modelling of dynamics of graphs and are related to many topics of current interest, including some problems in ad-hoc networks, social networks, and data-mining.

Eventually, we would like to close the thesis in connection with the very last word of the introduction section, namely the underlined word written using italic font: "*finite*". Actually, it is a matter of fact that *all* discrete structures which have been addressed in the composition are finite. It could be interesting to investigate the infinite case, e.g., the study of how to generalize the sesquimodular decomposition scheme to infinite graphs.

Bibliography

[1] L. Alonso, E. Reingold, and R. Schott. Multidimensional Divide-and-Conquer Maximin Recurrences. *SIAM Journal on Discrete Mathematics*, 8(3):428–447, 1995.

[2] M.-P. Béal, A. Bergeron, S. Corteel, and M. Raffinot. An algorithmic view of gene teams. *Theoretical Computer Science*, 320(2-3):395–418, 2004.

[3] S. Bérard, A. Bergeron, and C. Chauve. Conservation of combinatorial structures in evolution scenarios. In *8th Annual International Conference on Computational Molecular Biology (RECOMB'04)*, volume 3388 of *LNCS*, pages 1–14, 2004.

[4] A. Bergeron, C. Chauve, F. de Montgolfier, and M. Raffinot. Computing Common Intervals of K Permutations, with Applications to Modular Decomposition of Graphs. In *13th Annual European Symposium on Algorithms (ESA'05)*, volume 3669 of *LNCS*, pages 779–790, 2005.

[5] A. Bergeron and J. Stoye. On the Similarity of Sets of Permutations and its Applications to Genome Comparison. In *9th Annual International Conference on Computing and Combinatorics (COCOON'03)*, volume 2697 of *LNCS*, pages 68–79, 2003.

[6] A. Bernáth. A note on the directed source location algorithm. Technical Report TR-2004-12, Egerváry Research Group, Budapest, 2004. `www.cs.elte.hu/egres`.

[7] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan. Time bounds for selection. *Journal of Computer and System Science*, 7(2):36–44, 1973.

[8] H. Bodlaender. Treewidth: Characterizations, Applications, and Computations. In *32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'06)*, volume 4271 of *LNCS*, pages 1–14, 2006.

[9] K. Bogart, P. Fishburn, G. Isaak, and L. Langley. Proper and unit tolerance graphs. *Discrete Applied Mathematics*, 60:99–117, 1995.

[10] R. Borie, R. Parker, and C. Tovey. Solving problems on recursively constructed graphs. *To appear in ACM Computing Surveys*.

[11] C. Bornstein, C.M.H. de Figueiredo, and V.G.P. de Sá. The pair completion algorithm for the homogeneous set sandwich problem. *Information Processing Letters*, 98(3):87–91, 2006.

[12] V. Bouchitté and I. Todinca. Treewidth and Minimum Fill-in: grouping the Minimal Separators. *SIAM Journal on Computing*, 31(1):212–232, 2001.

[13] F. Boyer, A. Morgat, L. Labarre, J. Pothier, and A. Viari. Syntons, metabolons and interactons: an exact graph-theoretical approach for exploring neighbourhood between genomic and functional data. *Bioinformatics*, 21(23):4209–4215, 2005.

[14] B. Bui Xuan, A. Ferreira, and A. Jarry. Computing Shortest, Fastest, and Foremost Journeys in Dynamic Networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.

[15] B.-M. Bui-Xuan and M. Habib. A Representation Theorem for Union-Difference Families and Application. In *8th Latin American Theoretical Informatics (LATIN'08)*, volume 4957 of *LNCS*, pages 492–503, 2008.

[16] B.-M. Bui-Xuan, M. Habib, V. Limouzy, and F. de Montgolfier. Algorithmic Aspects of a General Modular Decomposition Theory. *Discrete Applied Mathematics: special issue of the 3rd conference on Optimal Discrete Structures and Algorithms (ODSA'06), to appear*.

[17] B.-M. Bui Xuan, M. Habib, V. Limouzy, and F. de Montgolfier. Homogeneity vs. Adjacency: generalising some graph decomposition algorithms. In *32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'06)*, volume 4271 of *LNCS*, pages 278–288, 2006.

[18] B.-M. Bui-Xuan, M. Habib, V. Limouzy, and F. de Montgolfier. Unifying two Graph Decompositions with Modular Decomposition. In *18th Annual International Symposium on Algorithms and Computation (ISAAC'07)*, volume 4835 of *LNCS*, pages 52–64, 2007.

[19] B.-M. Bui Xuan, M. Habib, and C. Paul. Revisiting T. Uno and M. Yagiura's Algorithm. In *16th Annual International Symposium of Algorithms and Computation (ISAAC'05)*, volume 3827 of *LNCS*, pages 146–155, 2005.

[20] B.-M. Bui-Xuan, M. Habib, and C. Paul. Competitive Graph Searches. *Theoretical Computer Science*, 393(1-3):72–80, 2008.

[21] B.-M. Bui-Xuan, M. Habib, and M. Rao. Representation Theorems for two Set Families and Applications to Combinatorial Decompositions. *Extended abstract in Proceedings of the International Conference on Relations, Orders and Graphs: Interaction with Computer Science (ROGICS'08), Nouha editions,* pages 532–546, 2008.

[22] B.-M. Bui-Xuan and J. A. Telle. H-join and dynamic programming on graphs of bounded rankwidth. *Abstract presented in the Workshop on Graph Decomposition: Theoretical, Algorithmic and Logical Aspects,* 2008.

[23] C. Capelle. Block Decomposition of Inheritance Hierarchies. In *23rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'97)*, volume 1335 of *LNCS*, pages 118–131, 1997.

[24] C. Capelle. *Décomposition de Graphes et Permutations Factorisantes*. PhD thesis, Université Montpellier II, 1997.

[25] C. Capelle, M. Habib, and F. de Montgolfier. Graph decomposition and factorizing permutations. *Discrete Mathematics and Theoretical Computer Science*, 5(1):55–70, 2002.

[26] M. Cerioli, H. Everett, C.M.H. de Figueiredo, and S. Klein. The homogeneous set sandwich problem. *Information Processing Letters*, 67(1):31–35, 1998.

[27] M. Chein, M. Habib, and M.-C. Maurer. Partitive hypergraphs. *Discrete Mathematics*, 37(1):35–50, 1981.

[28] D. Cohen, M. Cooper, and P. Jeavons. Generalising Submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. Technical Report CS-RR-06-06, Oxford University, 2006.

[29] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[30] D. Corneil and U. Rotics. On the Relationship Between Clique-Width and Treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005.

[31] F. Coulon and M. Raffinot. Identification of maximal common connected sets of interval graphs and tree forests. In *1st International Conference on Algorithms and Computational Methods for Biochemical and Evolutionary Networks (CompBioNets'04)*, 2004. *To appear.*

[32] F. Coulon and M. Raffinot. Fast algorithms for identifying maximal common connected sets of interval graphs. *Discrete Applied Mathematics*, 154(12):1709–1721, 2006.

[33] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993.

[34] B. Courcelle and M. Kanté. Graph Operations Characterizing Rank-Width and Balanced Graph Expressions. In *33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'07)*, volume 4769 of *LNCS*, pages 66–75, 2007.

[35] B. Courcelle, J. Makowsky, and U. Rotics. Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[36] B. Courcelle and S. Oum. Vertex-minors, monadic second-order logic, and a conjecture by Seese. *Journal of Combinatorial Theory, Series B*, 97(1):91–126, 2007.

[37] A. Cournier and M. Habib. A new linear algorithm for modular decomposition. In *Trees in algebra and programming (CAAP'94)*, volume 787 of *LNCS*, 1994.

[38] C. Crespelle. *Représentation dynamiques de graphes.* PhD thesis, Université Montpellier II, 2007.

[39] W. Cunningham. *A combinatorial decomposition theory.* PhD thesis, University of Waterloo, 1973.

[40] W. Cunningham. Decomposition of submodular functions. *Combinatorica*, 3(1):53–68, 1983.

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

[42] E. Dahlhaus. Parallel algorithms for hierarchical clustering, and applications to split decomposition and parity graph recognition. *Journal of Algorithms*, 36(2):205–240, 2000.

[43] E. Dahlhaus, J. Gustedt, and R. McConnell. Efficient and practical algorithms for sequential modular decomposition. *Journal of Algorithms*, 41(2):360–387, 2001.

[44] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry*. Springer-Verlag, 1991.

[45] E. Dinitz, A. Karzanov, and M. Lomonosov. On the structure of a family of minimal weighted cuts in a graph. In *A. Pridman (Ed.), Studies in Discrete Optimization, Nauka, Moscow*, pages 290–306, 1976. *(in Russian)*.

[46] J. Edmonds and R. Giles. A min-max relation for submodular functions on graphs. *Annals of Discrete Mathematics*, 1:185–204, 1977.

[47] A. Ehrenfeucht, T. Harju, and G. Rozenberg. *The Theory of 2-Structures - A Framework for Decomposition and Transformation of Graphs*. World Scientific, 1999.

[48] A. Ehrenfeucht and G. Rozenberg. Theory of 2-structures. *Theoretical Computer Science*, 3(70):277–342, 1990.

[49] D. Eppstein, G. Italiano, R. Tamassia, R. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *Journal of Algorithms*, 13:33–54, 1992.

[50] M. Everett and S. Borgatti. Regular Equivalence: General Theory. *Journal of Mathematical Sociology*, 18:29–52, 1994.

[51] S. Felsner. Tolerance graphs and orders. *Journal of Graph Theory*, 28(3):129–140, 1998.

[52] J. Fiala and D. Paulusma. A complete complexity classification of the role assignment problem. *Theoretical Computer Science*, 349(1):67–81, 2005.

[53] M. Figeac and J.-S. Varré. Sorting by reversals with common intervals. In *4th International Workshop on Algorithms in Bioinformatics (WABI'04)*, volume 3240 of *LNBI*, pages 26–37, 2004.

[54] T. Fleiner and T. Jordán. Coverings and structure of crossing families. *Mathematical Programming*, 84(3):505–518, 1999.

[55] J.-L. Fouquet, M. Habib, F. de Montgolfier, and J.-M. Vanherpe. Bimodular Decomposition of Bipartite Graphs. In *30th International Workshop on Graph-Theoretic*

Concepts in Computer Science (WG'04), volume 3353 of *LNCS*, pages 117–128, 2004.

[56] S. Fujishige. Canonical decompositions of symmetric submodular systems. In *Graph Theory and Algorithms*, volume 108 of *LNCS*, pages 53–64, 1981.

[57] S. Fujishige. Structures of polyhedra determined by submodular functions on crossing families. *Mathematical Programming*, 29(2):125–141, 1984.

[58] H. Gabow. Centroids, Representations, and Submodular Flows. *Journal of Algorithms*, 18(3):586–628, 1995.

[59] J. Gagneur, R. Krause, T. Bouwmeester, and G. Casari. Modular decomposition of protein-protein interaction networks. *Genome Biology*, 5(8), 2004.

[60] A.-T. Gai. *Algorithmes de Partitionnement: Minimisation d'Automates et Applications aux Graphes*. MSc thesis, Université Montpellier II, 2003.

[61] A.-T. Gai, M. Habib, C. Paul, and M. Raffinot. Identifying Common Connected Components of Graphs. Technical Report RR-LIRMM-03016, LIRMM, Université de Montpellier II, 2003.

[62] T. Gallai. Transitiv orientierbare Graphen. *Acta Mathematica Academiae Scientiarum Hungaricae*, 18:25–66, 1967.

[63] M. Golumbic, H. Kaplan, and R. Shamir. Graph Sandwich Problems. *Journal of Algorithms*, 19:449–473, 1995.

[64] M. Habib. *Substitution des structures combinatoires. Théorie et algorithmes*. Thèse d'état, Université Pierre et Marie Curie, 1981.

[65] M. Habib, F. de Montgolfier, and C. Paul. A simple linear-time modular decomposition algorithm. In *9th Scandinavian Workshop on Algorithm Theory (SWAT'04)*, volume 3111 of *LNCS*, pages 187–198, 2004.

[66] M. Habib, M. Huchard, and J. Spinrad. A linear algorithm to decompose inheritance graphs into modules. *Algorithmica*, 13(6):573–591, 1995.

[67] M. Habib, C. Paul, and M. Raffinot. Common connected Components of Interval Graphs. In *15th Annual Symposium on Combinatorial Pattern Matching (CPM'04)*, volume 3109 of *LNCS*, pages 347–358, 2004.

[68] M. Habib, C. Paul, and L. Viennot. Partition refinement techniques: An interesting algorithmic tool kit. *International Journal of Foundations of Computer Science*, 10(2):147–170, 1999.

[69] S. Heber and J. Stoye. Finding all common intervals of k permutations. In *12th Annual Symposium on Combinatorial Pattern Matching (CPM'01)*, volume 2089 of *LNCS*, pages 207–218, 2001.

[70] M. Henzinger and V. King. Randomized dynamic graph algorithms with polylogarithmic time per operation. In *27th Annual ACM Symposium on Theory of Computing (STOC'95)*, volume 787, pages 519–527, 1995.

[71] P. Hliněný and S. Oum. Finding Branch-Decompositions and Rank-Decompositions. In *15th Annual European Symposium on Algorithms (ESA'07)*, volume 4698 of *LNCS*, pages 163–174, 2007.

[72] P. Hliněný, S. Oum, D. Seese, and G. Gottlob. Width Parameters Beyond Tree-width and Their Applications. *The Computer Journal*, 51(3):326–362, 2008.

[73] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2−edge, and biconnectivity. In *30th Annual ACM Symposium on Theory of Computing (STOC'98)*, pages 79–89, 1998.

[74] W.-L. Hsu and T.-M. Ma. Substitution decomposition on chordal graphs and applications. In *2nd International Symposium on Algorithms (ISA'91)*, volume 557 of *LNCS*, pages 52–60, 1991.

[75] W.-L. Hsu and R. McConnell. PC-trees and circular-ones arrangements. *Theoretical Computer Science*, 296:99–116, 2003.

[76] M. Huchard. *Sur quelques questions algorithmiques de l'héritage multiple*. PhD thesis, Université Montpellier II, 1992.

[77] J. Johnson. *Polynomial time recognition and optimization algorithms on special classes of graphs*. PhD thesis, Vanderbilt University, 2003.

[78] T. Király. *Edge-connectivity of undirected and directed hypergraphs*. PhD thesis, Eötvös Loránd University, 2003.

[79] J. Kleinberg and É. Tardos. *Algorithm Design*. Pearson/Addison-Wesley, 2005.

[80] G. Landau, L. Parida, and O. Weimann. Using PQ trees for comparative genomics. In *16th Annual Symposium on Combinatorial Pattern Matching (CPM'05)*, volume 3537 of *LNCS*, 2005.

[81] J.-M. Lanlignel. *Autour de la décomposition en coupes*. PhD thesis, Université Montpellier II, 2001.

[82] Z. Li and E. Reingold. Solution of a divide-and-conquer maximin recurrence. *SIAM Journal on Computing*, 18(6):1188–1200, 1989.

[83] V. Limouzy. *Thesis in preparation*, Université Paris VII Diderot.

[84] R. Lipton and R. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, 1980.

[85] F. Maffray and M. Preissmann. A translation of Tibor Gallai's paper: Transitiv orientierbare Graphen. In *Perfect Graphs*, pages 25–66. J. Wiley, 2001.

[86] R. McConnell and F. de Montgolfier. Algebraic Operations on PQ Trees and Modular Decomposition Trees. In *31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG'05)*, volume 3787 of *LNCS*, pages 421–432, 2005.

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

[88] R. McConnell and J. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*, pages 536–545, 1994.

[89] R. McConnell and J. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201:189–241, 1999. Extended abstract at *SODA'94*.

[90] K. Mehlhorn. *Data Structures and Efficient Algorithms*. Springer Verlag, EATCS Monographs, 1984.

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

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

[93] F. de Mongolfier and M. Rao. The bi-join decomposition. 2005. `http://hal.archives-ouvertes.fr/hal-00132862/`.

[94] F. de Montgolfier. *Décomposition modulaire des graphes. Théorie, extensions et algorithmes*. PhD thesis, Université Montpellier II, 2003.

[95] S. Nikolopoulos and L. Palios. Minimal separators in P_4-sparse graphs. *Discrete Mathematics*, 306(3):381–392, 2006.

[96] S. Oum. *Graphs of Bounded Rank-width*. PhD thesis, Princeton University, 2005.

[97] S. Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79–100, 2005.

[98] S. Oum. Rank-width and Well-quasi-ordering. *SIAM Journal on Discrete Mathematics*, 22(2):666–682, 2008.

[99] S. Oum. Rank-width is less than or equal to branch-width. *Journal of Graph Theory*, 57(3):239–244, 2008.

[100] S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

[101] C. Paul. *Aspects algorithmiques de la décomposition modulaire*. Habilitation thesis, Université Montpellier II, 2006.

[102] E. Prisner. Bicliques in Graphs I: Bounds on Their Number. *Combinatorica*, 20(1):109–117, 2000.

[103] M. Rao. *Décomposition de graphes et algorithmes efficaces*. PhD thesis, Université Paul Verlaine, Metz, 2006.

[104] N. Robertson and P. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

[105] A. Schrijver. Proving total dual integrality with cross-free families - A general framework. *Mathematical Programming*, 29(1):15–27, 1984.

[106] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer-Verlag, 2003.

[107] D. Seinsche. On a property of the class of n-colorable graphs. *Journal of Combinatorial Theory, Series B*, pages 191–193, 1974.

[108] M. Tedder, D. Corneil, M. Habib, and C. Paul. Simple, Linear-Time Modular Decomposition. In *35th International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5125 of *LNCS*, pages 634–645, 2008.

[109] I. Todinca. *Aspects algorithmiques des triangulations minimales des graphes*. PhD thesis, École Normale Supérieure de Lyon, 1999.

[110] I. Todinca. *Décompositions arborescentes de graphes : calcul, approximations, heuristiques*. Habilitation thesis, Université d'Orléans, 2006.

[111] T. Uno and M. Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26(2):290–309, 2000.

[112] E. Wanke. *k-NLC* Graphs and Polynomial Algorithms. *Discrete Applied Mathematics*, 54(2-3):251–266, 1994.

[113] D. White and K. Reitz. Graph and Semigroup Homomorphisms on Networks of Relations. *Social Networks*, 5:193–234, 1983.

Index

Notation and Abbreviation

$\{x_1, x_2, \ldots, x_k\}$	set of size k, in particular $\{x_1, x_2\} = \{x_2, x_1\}$
$(x_1, x_2, \ldots, x_k)$	k-uplet, in particular $(x_1, x_2) \neq (x_2, x_1)$
$[x_1, x_2, \ldots, x_k]$	linked list containing k elements, in particular $[x_1, x_2] \neq [x_2, x_1]$
$\lvert A\rvert$, or $\#A$	cardinality of set A
$A \subseteq B$	A is a subset of B, in particular $B \subseteq B$
$A \subsetneq B$	A is a subset of B and $A \neq B$
2^X	$\{A,\ A \subseteq X\}$
$A \Delta B$	symmetric difference, namely $A \Delta B = (A \setminus B) \cup (B \setminus A)$
$A \uplus B$	only when $A \neq B$, we denote $A \uplus B = A \cup B$
$A \Cap B$	true if $A \cap B \neq \emptyset$, $A \setminus B \neq \emptyset$, and $B \setminus A \neq \emptyset$
C_n^k, or $\binom{n}{k}$	binomial coefficient: $C_n^k = \frac{n!}{k!(n-k)!}$ for $0 \leq k \leq n$
$\mathbb{Z}$	set of integers
$\mathbb{N}$	set of integers greater or equal to 0
$\mathbb{R}$	set of real numbers
$\mathbb{R}^+$	set of real numbers greater or equal to 0
$\lfloor x \rfloor$	when $x \in \mathbb{R}$, the floor function is $\lfloor x \rfloor = \max\{n \in \mathbb{Z} \mid n \leq x\}$
$\lceil x \rceil$	when $x \in \mathbb{R}$, the ceiling function is $\lfloor x \rfloor = \min\{n \in \mathbb{Z} \mid n \geq x\}$
$V(G)$	when G is a graph, $V(G)$ is the vertex set of G
$E(G)$	when G is a graph, $E(G)$ is the edge set of G
uv, or (u, v)	usually denotes the edge between vertices u and v of some graph
$N^-(v)$, or $N_G^-(v)$	in-neighbours of vertex v in graph G: $N^-(v) = \{u,\ uv \in E(G)\}$
$N^+(v)$, or $N_G^+(v)$	out-neighbours of vertex v in graph G: $N^+(v) = \{u,\ vu \in E(G)\}$
$N(v)$, or $N_G(v)$	when G is undirected, neighbours of v: $N(v) = N^-(v) = N^+(v)$
$d(v)$, or $d_G(v)$	when G is undirected, degree of vertex v: $d(v) = \lvert N(v)\rvert$

$G[A]$	when G is a graph, $G[A] = (\ A,\ E(G)\ \cap\ A \times A\ )$
$\text{sam}_{i \in I} x_i$	sum of all but the max: $\text{sam}_{i \in I} x_i = \sum_{i \in I} x_i - \max_{i \in I} x_i$
a.k.a.	also known as
cf.	*confer*
e.g.	*exempli gratia*
i.e.	*id est*
iff	if and only if
resp.	respectively
s.t.	such that
vs.	*versus*
w.l.o.g.	without loss of generality
w.r.t.	with respect to

Résumé de la thèse

Ce manuscrit de thèse développe certains aspects autour de trois thèmes généraux, sur la représentation arborescente des familles d'ensembles, les décompositions de graphes, et les algorithmes de graphes. Les thèmes abordés vont de la combinatoire théorique à l'algorithmique en bio-informatique, en passant par plusieurs décompositions de graphes et aussi par l'optimisation combinatoire.

Le manuscrit commence par l'étude de certaines propriétés combinatoires des familles d'ensembles en général. Afin d'estimer le nombre de familles satisfaisant certains axiomes de clôture, de nouveaux outils et techniques pour en obtenir des représentations arborescentes ont été développés. Un point intéressant ici est l'établissement d'une technique générale pour représenter une famille quelconque par un arbre. Ceci est rendu possible en unifiant et étendant plusieurs concepts fondamentaux venant de différents domaines de la combinatoire, allant de l'arborescence des familles sans croisement d'Edmonds-Giles (*cross-free family* en anglais) aux techniques de représentation d'Ehrenfeutch-Harju-Rozenberg, en passant par deux décompositions de graphes assez connues, dites de Gallai, et de Cunningham.

Puis, l'étude se poursuit avec une des applications des propriétés ci-dessus : celle concernant les décompositions de graphes. Pas moins de sept schémas de décomposition de graphes, dont deux nouveaux, sont développés dans le manuscrit : décomposition modulaire, décomposition par clans, décomposition en coupes, décomposition en bi-joints, décomposition de fonctions symétriques et sousmodulaires, décomposition en "modules véritables" (*genuine-modules* en anglais), et décomposition en "modules désordonnés" (*unordered-modules* en anglais). Ces deux niveaux d'étude (représentation de familles d'ensembles – décomposition de graphes) occupent entièrement la première moitié du manuscrit.

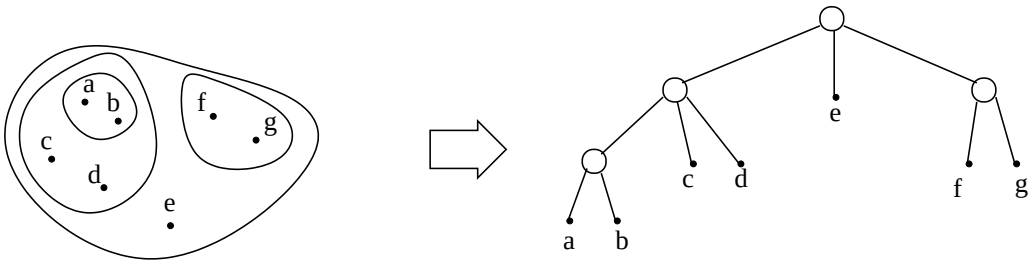
La deuxième moitié du manuscrit est consacrée aux applications des décompositions de graphes dans l'algorithmique de graphes. Trois problèmes algorithmiques seront à l'étude. Dans chacun des trois, il est montré pourquoi et comment on peut appliquer l'idée de la décomposition de graphes pour résoudre le problème posé de manière efficace.

Il est également montré comment appliquer les trois solutions proposées pour résoudre trois autres problèmes d'algorithmique de graphes. Le premier cas étudie les parties connexes communes à plusieurs graphes, son cas d'application concerne les cographes sandwich. Le deuxième cas étudie les intervalles communs à plusieurs permutations, son cas d'application concerne la décomposition modulaire des graphes. Le troisième cas, enfin, étudie une autre décomposition de graphes, appelée décomposition en H-joints, son cas d'application concerne la programmation dynamique sur les décompositions par largeur de rang.

Finalement, notons sur ce dernier point que nous avons introduit récemment le cadre de décomposition en H-joints comme étant une généralisation unificatrice de plusieurs décompositions existant en théorie des graphes. Celles-ci comprennent : décomposition modulaire, décomposition en coupes ou en bi-joints, décomposition par largeur de clique, par largeur NLC, et par largeur de rang.

Partie 1 : Entre la représentation de familles d'ensembles et les décompositions

L'objectif principal ce cette partie est de trouver des codages efficaces pour représenter une famille d'ensembles donnée. Une des motivations théoriques pourrait être la suivante. A partir d'un ensemble X contenant n éléments distincts, on a clairement 2^{2^n} choix de familles de sous-ensembles de X. Pourtant, si la famille satisfaisait quelques axiomes simples, la situation pourrait être complètement différente. Par exemple, on dit que deux sous-ensembles A et B se chevauchent si elles ont une intersection non-vide $A \cap B \neq \emptyset$, ainsi que des différences non-vides $A \setminus B \neq \emptyset$ et $B \setminus A \neq \emptyset$. Ensuite, une famille est dite laminaire (ou sans chevauchement) s'il n'y a pas deux éléments de la famille qui se chevauchent. Par élimination, deux éléments disjoints de cette famille ne peuvent qu'être inclus l'un dans l'autre. Ainsi, en ordonnant les éléments de la famille par inclusion, on obtient un sous-graphe partiel d'un arbre, dont la racine correspond à X, dont les feuilles correspondent aux singletons $\{x\}$ (pour tout $x \in X$), et dont les noeuds internes ont tous au moins deux fils. Cela implique clairement qu'une famille laminaire sur X contient au

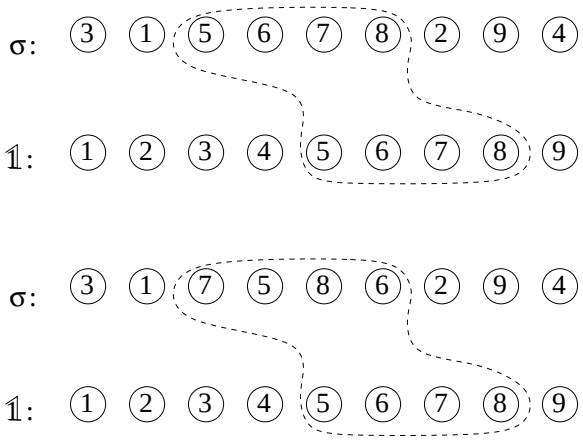


Une famille laminaire.

plus $2n$ éléments (et, par conséquent, il y a au plus 2^{2n} choix d'une telle famille). Dans des théories de complexité classiques en informatique, un tel saut exponentiel est important pour des raisons computationelles. Pour cette raison, nous nous intéressons à l'étude de certaines de ces situations, où une famille d'ensembles admet une représentation, non pas exponenetielle, mais polynomiale en espace.

L'application principale de notre étude vient de la théorie des graphes. Cela pourrait paraître assez étrange, étant donné que les graphes ne sont que des cas particuliers de familles d'ensembles : un graphe n'est guère plus qu'une famille de sous-ensembles à deux éléments, alors pourquoi le détour par les familles quelconques ? Cependant, on peut premièrement constater que de nombreux schémas de décomposition de graphes font appel de manière très naturelle à diverses familles de sous-ensembles de sommets du graphe en question. C'est par exemple le cas pour la décomposition modulaire et la décomposition en coupes : les deux notions associées, celle d'un module et celle d'une coupe, ne sont essentiellement que des sous-ensembles de sommets répondant à un certain axiome d'homogénéïté. Deuxièmement, il est maintenant assez bien connu que l'ensemble des modules d'un graphe quelconque définit toujours une famille dite partitive, tandis que celui des coupes d'un graphe connexe définit une famille dite symétrique et "à croissement" (*symmetric crossing family* en anglais). A partir de là, la décomposition modulaire et la décomposition en coupes découlent toutes les deux des résulats de représentation assez connus pour les familles correspondantes. Pour donner un autre exemple, les familles d'ensembles interviennent aussi dans la décomposition arborescente et la décomposition par largeur de clique par l'intermédiare de leurs alternatives : la décomposition par largeur de branche (d'un graphe) et la décomposition par largeur de rang (d'un graphe). En effet, ces dernières décompositions sont toutes deux des cas particuliers de la décomposition par largeur de branche d'une fonction de connectivité. Celle-ci s'adresse à la famille des points où la fonction admet des valeurs inférieures à un certain seuil fixé. Finalement, l'arbre de décomposition associé pourra être vu comme une représentation d'une sous-famille de la famille précédente.

Par ailleurs, l'optimisation combinatoire est aussi un domaine important de la combinatoire. Une question fondamentale de ce domaine consiste à calculer l'ensemble des racines d'une fonction sous-modulaire. Cela permet, entre autres, de résoudre le problème du flot maximum dans des problématiques liés à l'étude des réseaux (l'équivalence correspondante est bien connue sous le nom de la théorie de la dualité, *min-max duality* en anglais). Dans cette thématique, les racines non-vides d'une fonction sous-modulaire définissent toujours une famille dite à intersection (*intersecting family* en anglais). Alors, l'objectif de représenter efficacement une telle famille a joué un rôle principal dans une



L'ensemble $\{5, 6, 8, 7\}$ est un intervalle commun dans ces deux exemples.

classe de solutions au problème ci-dessus, à savoir celui de minimiser les fonctions sous-modulaires. C'est par exemple le cas pour la décomposition dite d'arbre des ensembles partiellement ordonnés (*tree-of-posets decomposition* en anglais).

Les familles d'ensembles ont aussi des applications en bio-informatique. Par exemple, elles interviennent dans des cas particuliers de problèmes liés à la notion de cluster de gènes. En effet, limitons-nous au cas où une séquence génomique peut être modélisée par une permutation sur l'ensemble X contenant tous les gènes de la séquence. Alors, un intervalle est défini comme un ensemble de gènes se succédant dans l'ordre défini par la permutation. Ensuite, étant donné un ensemble de plusieurs séquences génomiques sur le même ensemble de gènes X, un intervalle commun à ces séquences est défini comme un sous-ensemble de X qui est intervalle pour chacune des séquences considérées. Cette notion est considérée comme étant une des premières tentatives afin de formaliser la notion d'un cluster de gènes. Ici, il s'avère que les intervalles communs à plusieurs séquences données définissent toujours une famille dite faiblement partitive. Une conséquence est que l'on peut utiliser un résultat de représentation bien connu de ces familles pour définir un schéma de décomposition d'un ensemble quelconque de séquences génomiques en clusters de gènes de cet ensemble. A partir de là, on peut non seulement dériver un calcul efficace des intervalles communs, mais aussi étudier leur comportement dynamique vis à vis des modifications successives sur les séquences génomiques. Ainsi, le premier fait s'applique directement au calcul des clusters de gènes, tandis que le dernier fait s'avère fondamental pour le calcul de la distance dite de renversements dans l'évolution des espèces.

Quelques autres exemples de décompositions de structures discrètes où un résultat de représentation efficace d'un certain type de familles d'ensembles joue un rôle central pourraient être : décomposition canonique de fonctions de connectivité [56], décomposition de

fonctions sous-modulaires et en particulier décomposition de matroïdes [40], décomposition par clans de 2-structures [48], décomposition par blocs de graphes d'héritage [23, 66, 76], décomposition bimodulaire de graphes bipartis [55], décomposition en bi-joints de graphes [93], et décomposition en modules non-ordonnés de tournois [18].

Dans cette première partie du manuscrit, une des questions centrales est d'appréhender la distance (en terme de complexité en espace) entre une famille sur un ensemble X et l'ensemble X lui-même. Rappelons ici qu'une famille laminaire sur X a le même comportement (en complexité en espace) que X lui-même, à savoir qu'elle ne peut avoir plus de $2 \times |X|$ éléments. Alors, une manière équivalente de poser la question est d'établir les familles laminaires comme référence et d'évaluer les autres familles en fonction de celles-ci. Par ailleurs, d'après un théorème assez connu en optimisation combinatoire, dû à J. Edmonds et R. Giles, les familles laminaires sont en bijection avec des arbres. Ainsi, une troisième manière de poser la question est aussi d'étudier la distance entre une famille d'ensembles et une structure d'arbre. Outre cette question à triple formulation, cette partie du manuscrit développe également ce qui a été dit précédemment à propos du lien entre le problème de représentation des familles d'ensembles et celui de décomposition des graphes.

Un bref résumé du contenu de cette partie du manuscrit pourrait être comme suit.

Le chapitre d'ouverture construit la base pour la méthode de représentation que nous allons utiliser à travers le manuscrit entier. A cette fin, nous présentons de nouveaux outils et techniques pour trouver une représentation arborescente d'une famille d'ensembles quelconque. Ceux-ci englobent et étendent certains concepts fondamentaux venant des domaines différents de la combinatoire, allant de l'arborescence de familles sans croisement d'Edmonds-Giles (*cross-free family* en anglais) [46] aux techniques d'Ehrenfeucht-Harju-Rozenberg pour décomposer les graphes en général [47]. Outre ceci, ce chapitre comporte également une brève discussion sur certains résultats simples pour le problème de la représentation des familles d'ensembles. Enfin, le chapitre se termine avec deux tableaux récapitulatifs de tous les résultats de représentation abordés dans l'ensemble du manuscrit.

Le Chapitre 2 commence par rappeler quelques résultats de représentation déjà établis, ainsi que certaines de leur applications en décomposition de graphes. En particulier, le chapitre détaille un des résultats de représentation les plus efficaces pour les familles à intersection. Ces familles sont importantes pour minimiser les fonctions sous-modulaires. Par ailleurs, le chapitre passe en revue de manière détaillée un résultat très classique à propos des familles partitives (cf. Chein-Habib-Maurer [27] et Möhring-Radermacher [92]), ainsi qu'un autre résultat classique à propos des familles symétrique et à croisement (cf. Cunningham-Edmonds [39, 41]). Alors que les familles partitives sont fondamentales pour

la décomposition modulaire des graphes (cf. Gallai [62]), les familles symétrique et à croisement sont importantes pour minimiser les fonctions de connectivité (ceci est folklore). En même temps, le chapitre passe une deuxième fois sur le résultat à propos des familles partitives et présente une approche alternative pour obtenir la même représentation. La motivation est que, contrairement à l'approche classique, l'alternative en question suit le cadre général décrit dans le Chapitre 1. Quant aux familles symétriques et à croisement, le cadre présenté au Chapitre 1 englobe déjà l'approche classique de W. Cunningham et J. Edmonds. A partir de là, ce qui a été développé dans le Chapitre 1 pourrait être vu comme une unification sous le même formalisme de diverses approches pour représenter les familles d'ensembles. Le Chapitre 2 se termine par une série d'applications des résultats de représentation ci-dessus en décomposition de graphes. Ici, la liste des applications comprend pas moins de sept schémas de décomposition de diverses structures discrètes (on peut aussi dériver cette liste de la table des matières). Bien que la plupart de ces schémas proviennent de travaux antérieurs sur le sujet, deux d'entre eux proviennent de nos activités au cours de la période de thèse. Cependant, nous n'allons pas les développer dans le manuscrit (voir [83] pour plus de détails).

Le troisième et dernier chapitre de cette partie est consacré à deux résultats récents. Pour les introduire, rappelons que deux ensembles se chevauchent s'ils ont une intersection non-vide et s'ils ne sont pas inclus l'un dans l'autre. Ici, nous disons de plus que deux sous-ensembles d'un ensemble X se croisent s'ils se chevauchent ainsi que leur complémentaires dans X. Ensuite, une famille d'ensembles est appelée faiblement à croisement partitif si elle est close par union, intersection, et différence d'éléments se croisant. Par ailleurs, une famille d'ensembles est appelée à union-différence si elle est close par union et différence d'éléments se chevauchant. Ce chapitre présente une représentation en espace linéaire, resp. quadratique, pour les familles faiblement à croisement partitif, resp. les familles à union-différence. Ces résultats suivent l'approche devéloppée dans le Chapitre 1. Le chapitre se termine avec la présentation de deux nouveaux schémas de décomposition combinatoire. Par abus de langage, nous les appellerons par le même nom de décomposition en sesquimodules. L'un des deux est une généralisation stricte de la décomposition modulaire de graphes orientés, tandis que l'autre est une généralisation stricte de la décomposition par clans de 2-structures. Les deux notions associées d'arbre de décomposition peuvent être calculées en temps polynomial.

Partie 2 : Entre les décompositions et les algorithmes diviser-pour-régner

Diviser-pour-régner est un principe stratégique qui a une longue histoire dans la littérature populaire. Il apparaît dans plusieurs oeuvres allant de Sun Tzu à Niccolò

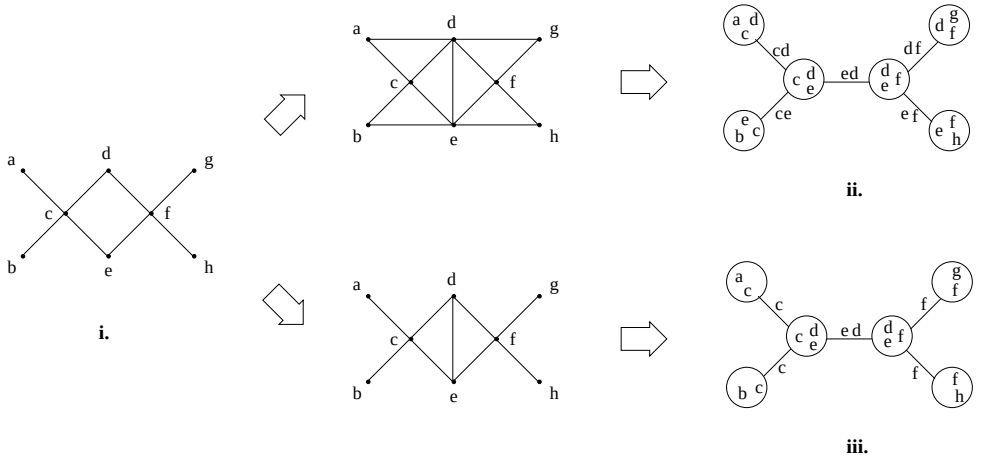
Machiavelli, et plus récemment aussi dans les travaux de René Descartes. Même de nos jours, il est encore largement utilisé pour se référer à une combinaison de stratégies à la fois politiques, militaires et économiques. Il consiste à gagner et maintenir le pouvoir en réduisant des concentrations de pouvoir en éléments qui, pris individuellement, ont moins de puissance que soi-même. Cependant, cette pratique est rare en réalité, car il est difficile de briser les structures de pouvoir déjà existantes. Ainsi, le principe diviser-pour-régner, dans la pratique, se réfère également à sa restriction qui consiste à empêcher les petits groupes de pouvoir de s'agglomérer.

Ce principe a été introduit en informatique avec la publication de l'algorithme de multiplication rapide de Anatolii Karatsuba dans les années soixantes. Dans ce domaine cependant, ce sera exactement l'aspect originel du principe qui est utilisé. Plus précisément, le schéma algorithmique dit *diviser-pour-régner* consiste à diviser le problème à étudier en plusieurs sous-problèmes, "conquérir" les sous-problèmes par des appels récursifs, et finalement unir les solutions à ces sous-problèmes en une solution au problème de départ. Ici, notons qu'il est assez connu que l'un des principaux défauts de l'approche diviser-pour-régner se manifeste lorsque, pour certains problèmes, un grand nombre de sous-problèmes coïncident. Si une telle situation se produit, il est plus intéressant d'utiliser à nouveau la solution commune à ces sous-problèmes autant de fois que possible. Une telle pratique est appelée en anglais *memoization* et peut être considérée comme l'idée fondamentale de la programmation dynamique. Aussi, les exemples les plus connus des algorithmes diviser-pour-régner, outre des algorithmes de tri, sont probablement les algorithmes de programmation dynamique.

Dans l'algorithmique de graphes comme dans tout autre domaine de l'algorithmique, l'approche diviser-pour-régner exige la condition préalable de savoir comment diviser une instance de graphe donnée, pas de manière arbitraire, mais d'une manière telle que l'étape d'unification, après les "conquêtes" des sous-problèmes correspondants, reste possible. Pour cet objectif, de nombreux outils ont été développés, les plus notoires étant, par ordre alphabétique : arête-connexité, bloc, coupe, *cut-set*, flot, isthme, *max-flow*, *min-cut*, séparateur, sommet d'articulation, sommet-connexité. Dans cette liste, les notions de séparateur et de sommet-connexité sont liés à la notion notoire de décomposition arborescente. Cette dernière, que l'on appelle quelques fois décomposition par largeur arborescente, s'obtient en complétant un graphe G en un graphe triangulé H ayant une clique maximum aussi petite que possible. Ensuite, le graphe triangulé H peut être associé de manière bijective à un arbre, appelé l'arbre de clique de H. Dans un tel arbre, les étiquettes des noeuds correspondent aux cliques maximales du graphe triangulé, tandis que les étiquettes des arêtes s'obtiennent en prenant l'intersection des étiquettes de ses

deux noeuds ajacents. En fin de compte, la largeur arborescente de G est définie comme la taille maximum d'une clique de H moins un, tandis que l'arbre de clique d'un tel graphe H définit ce que l'on appelle une décomposition arborescente de G.

En fait, à partir de n'importe quelle décomposition arborescente de G, il y a une transformation simple pour obtenir une décomposition T satisfaisant la propriété suivante, qui est essentielle pour l'algorithmique : l'étiquette d'une arête de T est toujours un séparateur minimal de G (voir, e.g., [8] pour une vue d'ensemble, ou [12, 109, 110] pour plus de détails sur les séparateurs). Un exemple pourrait être:



i. Un graphe G. ii. Une décomposition arborescente de G. iii. Une décomposition arborescente de G qui n'induise que des séparateurs minimaux de G.

Or, un séparateur d'un graphe connexe G est par définition un sous-ensemble de sommets dont la suppression déconnecte G en plusieurs morceaux. Cela s'avère être un moyen pratique pour permettre l'application des techniques diviser-pour-régner. Ainsi, de nombreux problèmes d'optimisation sur G peuvent être résolus par ces techniques si une décomposition arborescente de G est donnée (voir, e.g., [8], et aussi [79, Chapter 10. Extending the Limits of Tractability] pour plus de détails). Dans cette thématique, l'étape de division dans l'approche diviser-pour-régner découle directement de la décomposition arborescente, tandis que l'étape d'unification est généralement plus complexe, et constitue souvent la question à résoudre, comme dans le cas très académique du tri-fusion. Enfin, il est important de souligner que le même discours s'applique à bien d'autres schémas de décomposition de graphes. Parmi ceux-ci, notons les cas de la décomposition modulaire, de la décomposition en coupes, de la décomposition par largeur de branche, ainsi que la décomposition par largeur de clique.

En même temps, un aspect original dans l'étude des algorithmes diviser-pour-régner

vient d'un point de vue inverse de ce qui a été dit. En fait, il arrive que certains algorithmes, bien que théoriquement efficaces, s'avèrent compliqués et, ainsi, peu pratiques. Par exemple, afin de calculer l'arbre de décomposition modulaire d'un graphe donné, les premiers algorithmes en temps linéaires ont été trouvés il y a plus d'une décennie [37, 88]. Néanmoins, d'énormes efforts de recherche sont encore faits pour simplifier et/ou donner des alternatives à ces algorithmes déjà optimaux [108]. Pour donner un autre exemple, afin d'énumérer tous les intervalles communs à deux permutations, le premier algorithme en temps linéaire [111] a été trouvé (en 1996) presque une décennie avant la proposition d'un autre algorithme [4]. Dans certaines de ces situations, l'objectif est d'améliorer la robustesse de l'algorithme existant et/ou de conduire à une meilleure compréhension de la structure combinatoire du problème de départ. L'approche originale ici consiste à examiner les propriétés "algébriques" des algorithmes existants dans le but de trouver des propriétés combinatoires du problème à résoudre. En d'autres termes, une étude plus approfondie de certains algorithmes pourrait contribuer à la découverte de résultats combinatoires nouveaux et inattendus. En particulier, si l'algorithme en question suit l'approche diviser-pour-régner, alors la probabilité de tomber sur un résultat de décomposition sera augmentée. Ce sera, par exemple, le cas pour ce qui sera présenté dans le Chapitre 5, où nous allons en fait réviser le susdit algorithme d'énumération des intervalles communs [111] : le schéma de décomposition associé a été proposé dans [19], longtemps après la découverte de l'algorithme.

La deuxième partie du manuscrit met l'accent sur la dualité mentionnée ci-dessus entre les décompositions combinatoires et l'établissement des algorithmes. Nous étudions trois cas. Dans chacun des cas, nous insistons sur les aspects combinatoires du problème, avant de donner la solution algorithmique correspondante. Une idée-clef de cette partie, comme expliqué précédemment, réside dans le fait que ces algorithmes reposent sur certaines propriétés structurelles qui sont sous-jacentes au problème à résoudre. En réalité, il se peut que ces propriétés structurelles aient été trouvées après la découverte de l'algorithme en question (e.g., Lemme 5.2 du Chapitre 5). Toutefois, même dans ces cas, nous addressons les questions combinatoires d'abord, et omettons le débat sur la façon dont l'analyse de l'algorithme a conduit aux propriétés combinatoires du problème de départ.

La structure de la deuxième partie du manuscrit pourrait se résumer comme suit.

Nous abordons dans le Chapitre 4 deux problèmes de graphes qui viennent de la bioinformatique : celui de l'énumération des composantes connexes communs à deux graphes et celui du cographe sandwich. Pour le premier problème nous donnons un algorithme unique ayant un temps d'exécution qui varie en fonction de la structure de données utilisée. Ce temps d'exécution, pour toutes les configurations de structures de données à utiliser,

est sous-quadratique en le nombre de sommets et d'arêtes des deux graphes à considérer. Contrairement au cas du tri-fusion, l'étape d'unification de notre algorithme diviser-pour-régner est simple, tandis que l'étape de division est plus complexe. Pour atteindre le temps d'exécution sous-quadratique, nous introduisons un nouveau parcours de graphe, appelé le parcours compétitif. Ce dernier s'exécute en temps sous-linéaire en la taille de la donnée de l'algorithme. Ensuite, pour le deuxième problème, à savoir le problème du cographe sandwich, au lieu d'écrire un algorithme à partir de zéro, nous montrons comment on peut dériver une solution efficace à ce problème de l'algorithme précédent, par le biais d'une analyse structurelle des notions correspondantes.

Dans le Chapitre 5, nous détaillons un cas particulier du problème des composantes connexes communes, celui concernant ce que l'on appelle les intervalles communs à deux permutations. Nous revisitons un algorithme conçu par T. Uno et M. Yagiura [111], qui se modifie de manière très simple pour fonctionner en temps linéaire en le nombre d'éléments des permutations (par l'ajout d'une simple ligne de commande). Bien que l'algorithme soit simple à implémenter et ait un temps d'exécution très rapide, les aspects théoriques de l'algorithme Uno-Yagiura ont été assez peu mis en évidence, ce qui a rendu difficile l'analyse de sa correction et sa complexité. Notre étude examine en détail la correction et la complexité de cet algorithme. En même temps, nous mettons en évidence de puissantes propriétés structurelles des notions impliquées. Entre autres, nous démontrons dans le Lemme 5.2 une propriété de sous-modularité à intersection (*intersecting submodularity* en anglais), applicable dans un contexte très général de la décomposition modulaire. Plusieurs questions liées à la décomposition des intervalles communs sont également discutées dans le même chapitre. Nous donnons également une application de l'algorithme Uno-Yagiura dans le calcul de l'arbre de décomposition modulaire d'un graphe quelconque en temps linéaire (quand une permutation dite factorisante est donnée).

Dans le Chapitre 6, le manuscrit se termine par une tentative de donner un point de vue unifié de la décomposition modulaire, la décomposition en coupes, la décomposition en bi-joints, ainsi que de divers progrès récents dans l'algorithmique de graphes dont la décomposition par largeur de clique, par largeur NLC, et par largeur de rang. Cependant, nous limitons notre discussion sur ce sujet à certains problèmes algorithmiques autour de la nouvelle notion de décomposition en H-joints, ainsi que quelques problèmes autour de la restriction de celle-ci à la décomposition par largeur de rang. Nous montrons comment, en un temps d'exécution dit à paramètre fixé (*FPT-runtime* en anglais) avec un seul tour exponentiel, on peut améliorer la décomposition en H-joints à un objet permettant l'usage des techniques diviser-pour-régner. Lorsqu'il est appliqué à la décomposition par largeur de rang, ce temps de calcul admet un seul tour exponentiel en la largeur de rang

du graphe de départ. Nous implémentons également notre approche sur l'exemple d'une programmation dynamique pour résoudre le problème NP-difficile du calcul de la taille d'une clique maximum d'un graphe quelconque, d'abord en utilisant la décomposition en H-joints, puis en utilisant son cas particulier d'une décomposition par largeur de rang. A l'instar des cas de la décomposition arborescente et du tri-fusion, l'étape de division de notre méthode est simple tandis que celle d'unification consitue le coeur de l'algorithme.

Nos activités jusqu'à ce jour ont amené à des références [14, 15, 16, 17, 18, 19, 20, 21, 22] (détails ci-dessous). Dans ce manuscrit, nous développons [15, 19, 20, 21, 22]. Nous mentionnons également [16, 17, 18] sans rentrer dans les détails. Nous n'évoquons pas [14] dans ce manuscrit.

La plupart des idées présentées dans le Chapitre 1, l'approche alternative présentée dans le Chapitre 2 pour représenter les familles partitives, deux schémas de décomposition de ceux présentés à la fin du Chapitre 2, ainsi que tous les résultats présentés dans le Chapitre 3, sont basés sur [15, 16, 17, 18, 21]. Les Chapitres 4, 5, 6 sont basés respectivement sur [19, 20, 22].

---

[14] B. Bui Xuan, A. Ferreira, and A. Jarry. Computing Shortest, Fastest, and Foremost Journeys in Dynamic Networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.

[15] B.-M. Bui-Xuan and M. Habib. A Representation Theorem for Union-Difference Families and Application. In *8th Latin American Theoretical Informatics (LATIN'08)*, volume 4957 of *LNCS*, pages 492–503, 2008.

[16] B.-M. Bui-Xuan, M. Habib, V. Limouzy, and F. de Montgolfier. Algorithmic Aspects of a General Modular Decomposition Theory. *Discrete Applied Mathematics: special issue of the 3rd conference on Optimal Discrete Structures and Algorithms (ODSA'06), to appear.*

[17] B.-M. Bui Xuan, M. Habib, V. Limouzy, and F. de Montgolfier. Homogeneity vs. Adjacency: generalising some graph decomposition algorithms. In *32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'06)*, volume 4271 of *LNCS*, pages 278–288, 2006.

[18] B.-M. Bui-Xuan, M. Habib, V. Limouzy, and F. de Montgolfier. Unifying two Graph Decompositions with Modular Decomposition. In *18th Annual International Symposium on Algorithms and Computation (ISAAC'07)*, volume 4835 of *LNCS*, pages 52–64, 2007.

[19] B.-M. Bui Xuan, M. Habib, and C. Paul. Revisiting T. Uno and M. Yagiura's Algorithm. In *16th Annual International Symposium on Algorithms and Computation (ISAAC'05)*, volume 3827 of *LNCS*, pages 146–155, 2005.

[20] B.-M. Bui-Xuan, M. Habib, and C. Paul. Competitive Graph Searches. *Theoretical Computer Science*, 393(1-3):72–80, 2008.

[21] B.-M. Bui-Xuan, M. Habib, and M. Rao. Representation Theorems for two Set Families and Applications to Combinatorial Decompositions. *Extended abstract in Proceedings of the International Conference on Relations, Orders and Graphs: Interaction with Computer Science (ROGICS'08), Nouha editions,* pages 532–546, 2008.

[22] B.-M. Bui-Xuan and J. A. Telle. H-join and dynamic programming on graphs of bounded rankwidth. *Abstract presented in the Workshop on Graph Decomposition: Theoretical, Algorithmic and Logical Aspects,* 2008.

Abstract

We address some issues around three main topics: on the representation of set families by a tree, on decompositions of graphs, and on algorithms on graphs. Our study ranges from theoretical questions in combinatorics to the design of algorithms in computational biology, and also includes several decomposition schemes of graphs, as well as some issues in combinatorial optimization.

The first half of the thesis has two foci. Firstly, in order to estimate the number of set families satisfying some closure axioms, we have developed new tools and techniques to find tree-like representations for them. Then, we have given some applications of the previous results in a branch of graph theory called graph decomposition.

The second half of the thesis is devoted to algorithmic applications of set representations and decompositions to three graph problems. For each of them we show how the philosophy of decomposition can help to provide efficient solutions. We also show how to apply our three solutions to solve three other graph problems.

Keywords: set system, cross-free family, graph decompositions, submodular function, sub-linear representation, divide-and-conquer algorithms, sub-linear complexity, fixed-parameter tractable algorithms

Résumé

Ce manuscrit de thèse développe certains aspects autour de trois thèmes généraux, sur la représentation arborescente des familles d'ensembles, les décompositions de graphes, et les algorithmes de graphes. Les thèmes abordés vont de la combinatoire théorique à l'algorithmique en bio-informatique, en passant par plusieurs décompositions de graphes et aussi par l'optimisation combinatoire.

La première moitié du manuscrit développe deux études. D'abord, afin d'estimer le nombre de familles d'ensembles satisfaisant certains axiomes de clôture, de nouveaux outils et techniques pour obtenir des représentations arborescentes de celles-ci ont été développés. Puis, l'étude se poursuit avec une des applications des propriétés ci-dessus : celle concernant les décompositions de graphes.

La deuxième moitié du manuscrit est consacrée aux applications des décompositions de graphes dans l'algorithmique de graphes. Trois problèmes algorithmiques seront à l'étude. Dans chacun des trois, il est montré pourquoi et comment on peut appliquer l'idée de la décomposition de graphes pour résoudre le problème posé de manière efficace. Il est également montré comment appliquer les trois solutions proposées pour résoudre trois autres problèmes d'algorithmique de graphes.

Mots clés : famille d'ensembles, famille sans croisement, décompositions de graphes, fonction sous-modulaire, représentation sous-linéaire, algorithmes "diviser-pour-régner", complexité sous-linéaire, algorithmes à paramètre fixé

Lirmm, 161 Rue Ada 34392 Montpellier Cedex 05, France