Benemérita Universidad Autónoma de Puebla

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

POSGRADO EN CIENCIAS MATEMÁTICAS

HYPERTREE DECOMPOSITIONS

Tesis
presentada para obtener el título de
Maestría en Ciencias (Matemáticas)

Presenta:

Angeles Carranza Cisneros

Directores de Tesis:
Dr. Carlos Guillén Galván
Prof. Dr. Georg Gottlob

Puebla, Puebla. Noviembre de 2021

To my family and friends.

Acknowledgments

This project would not have been possible without the support of many people. Special thanks to the Consejo Nacional de Ciencia y Tecnología for providing me with the financial means to complete this project.

Thank you Dr. Carlos Guillén Galván, for your patience, guidance, and support. My full gratitude to Professor Georg Gottlob for your invaluable feedback on my writing. Thanks to my committee members, who so generously took time to read my work, your detailed feedback have been very important to me.

I also want to express my gratitude to the Database and Artificial Intelligence (DBAI) Group of TU Wien for inviting me for a research stay in Vienna.

Last but not the least, I would also like to thank all of my friends and family members for encouraging and supporting me.

Content

List of Figures

Symbols

$\mathbb{N}$	The set $\{0, 1, 2, 3, \ldots\}$.
$\mathbb{N}[x]$	The set of all polynomials with coefficients in $\mathbb{N}$.
Σ	An alphabet.
$\mathcal{L}$	A language.
τ	An assignment.
G	A graph.
$V(G)$	Set of vertices of G.
$E(G)$	Set of edges of G.
$\|G\|$	Number of vertices of G.
$\mathcal{P}$	Path.
$d(v)$	The degree of vertex v.
T	Tree.
T_p	Tree T rooted at p.
$tw(G)$	Treewidth of G.
H	A hypergraph.
H_I	Hypergraph of the CSP instance I.
$\mathcal{H}$	A class of hypergraphs.
$\wp(A)$	Power set of A.
$ghw(H)$	Generalized hypertree-width of H.
$hw(H)$	Hypertree-width of H.

$fw(H)$	Fractional-width of H.
$subw(H)$	submodular-width of H.
$bw(H)$	branch-width of H.

Introduction

The use of the computer has become an instrument of great value in all areas of knowledge and, as expected, computing devices cannot be separated from the software, that is, from the algorithmic part. By using a software system, one hopes to get answers in " real time ", this means, obtaining the desired results when the user requires it. The response in real time is related to the concepts of efficiency and tractability of the algorithms, concepts that are directly related to the time and space that the algorithms need to perform a given task. Time is specially considered very important in many problems that require algorithmic solutions. Space, in many cases, can pass to second term due to the technological advance of the memory devices that day to day offer greater capacity of storage, in contrast to the speed of the more advanced processors that do not keep the same growth rate.

Within the class of computational problems, there are mainly three groups; decision problems, counting problems, and optimization problems. So each group is divided into classes of

complexity not necessarily disjoint. For example, the classes **P, NP**, **NP-complete** and **NP-hard** are decision problems; **FP**, **#P**, **# P-complete** and **# P-hard** are located in counting problems; and optimization problems include the classes **PO**, **NPO** and **NPO- complete.**

In many classes there are problems that can be chosen as "representative" of the entire class, in the sense that every problem in the class is *computationally reduced* to it or to a part of it. For example the **SAT** problem which refers to the problem of deciding whether a given Boolean formula has an assignment that satisfies it, is representative of the **NP** class is, as a matter of fact, it is a problem which belongs to the **NP-complete** class [10]. The counting version of the **SAT** problem, known as the **# SAT**-problem, is located in the **# P-complete** class and has the same characteristic for the class **# P** [35]. The problem of maximizing the benefit that can be obtained from placing objects of a certain weight inside a container is known as the knapsack problem, this problem is located in the **NPO-complete** class.

Many problems of different kinds of complexity can be studied through problems that arise within the theory of graphs and hypergraphs, for example, determining if a given graph has a Hamiltonian circuit is a problem located in the **NP-complete** class. The power of synthesis that graphs (hypergraphs) have makes them attractive for study and analysis, for this reason there is a natural inclination to study many problems of the

theory of the complexity and analysis and design of algorithms through graph or hypergraph problems.

The concepts of tree decomposition and tree-width as well as the widths related to cliques, bands, paths and branches, among others, have proven to be of great importance in computational complexity theory [6, 22]. However, as mentioned in [36], the practical usefulness of the methods based on tree decomposition has been limited since calculating the width of a graph is difficult. Determining whether the treewidth of a given graph G is **NP-complete**. Even if the treewidth is determined, good performance cannot be guaranteed, since the time complexity of most algorithms is exponential in the treewidth.

One of the fundamental interests when studying the complexity of many problems is to find conditions or restrictions for which the tractability of the problem is guaranteed, in this sense the **parameterized complexity**, second-order languages and the concept of **(hyper)tree decomposition** have a very important role to solve difficult problems efficiently. The methods based on (hyper)tree decomposition under the condition that the (hyper)graph has a bounded (hyper)tree width and can be expressed in monadic second order logic provide the necessary conditions so that many intractable problems can be solved in polynomial time and even in linear time. In this direction we can highlight the works presented in [2, 31] and mainly those related to the width concepts discussed in [11, 12, 13].

In [22], there is a review of the concepts of decomposition in hypertrees and hypertree width from a perspective of graph theory and a very significant result is proved: computing hypertree decompositions is fixed-parameter intractable. More recent works such as [28], [33], and [23] tackle certain computational problems through different hypergraph invariants. The purpose of this work is to present different hypergraph decomposition methods developed in the aforementioned papers, and some important results related to tractability, particularly when solving constraint satisfaction problems.

This thesis is structured as follows. Chapter 1 gives a general introduction to complexity theory as well as the explicit description and classification of some specific problems. Before moving to the next chapter, we take a brief look at constraint satisfaction problems. In Chapter 2, basic notions about graphs and hypergraphs are discussed, a few results regarding tree decompositions are presented as a prelude to decompositions in hypergraphs. In the following chapter, Chapter 3, we move into the main part of this thesis, we introduce hypertree decompositions and related hypergraph invariants: their definitions, examples, results and relationships among them. Chapter 4 is meant to serve as a brief examination of the great benefits of decompositions in hypergraphs, specifically when coping with CSPs and equivalent problems. Additionally, open problems in the area are described. The last chapter contains a summary of the study and suggestions for future work.

Chapter 1

Basic computational complexity theory

Two algorithms that execute the same task can be compared for their efficiency, that is, for the amount of computational resources they use, which means the time and space required by the algorithm for its execution. Given a problem that requires a computational algorithm, there is a need to classify it based on the computational resources of the known algorithms for its solution. In this direction, the theory of computational complexity makes a classification of the problems, placing them in *complexity classes* based on the computational resources used by the " best " known algorithms that solve them.

1.1 Complexity Classes

In computational complexity theory, a decision problem is one which provides a *yes* or *no* answer. There are decision problems that lend themselves to efficient algorithms. However, for there also exist decision problems for which it is not obvious how to get the answer. The complexity class P refers to the class of problems that can be solved by a deterministic algorithm in polynomial time, which means that its execution time is bounded by a polynomial expression on the size of the input. Problems that can be solved by a polynomial-time algorithm are called tractable problems. On the other hand, the solution of problems in the *NP* class might be very difficult to find but once found, it is easy to verify. The *P versus NP problem* asks whether these two classes are actually the same.

Around 1970, Stephen Cook defined a decision problem that was probably as hard or harder than every other *NP* problem, the so called SAT problem. An efficient solution for the SAT problem could be used to create an efficient solution to any other problem in *NP.* Promptly, Richard Karp used Cook's results to show that a number of other decision problems could serve the same purpose. These problems belong to the *NP-complete* classs. Many people have tried for many years to find polynomial solutions for *NP-complete* problems, all have failed. If we could solve any *NP-complete* problem quickly (polynomial time), we could solve all *NP* problems quickly. If

that is the case, then $NP = P$. Let us give some examples of such problems.

- *Hamiltonian path/cycle problem.* Determine whether a Hamiltonian path or Hamilton cycle exists in a given graph (either directed or undirected).
- *Exact Cover Problem.* Consider a set of elements A and let $F \subset \wp(A)$ be family of subsets. Is there a subset of F such that each element of A appears exactly once?
- *Decision Traveling Salesman Problem (TSP).* Given a graph with weighted vertices, and a cost bound k. Is there a path that contains all vertices in the graph, that has a total cost less than k?
- *Graph Coloring.* Given a graph and a number of colors k, can we color every vertex using no more than k colors, such that all adjacent vertices have different colors?
- *Vertex cover.* Given graph G and value k, is there a vertex cover S for G such that $|S| \leq k$?
- *Subgraph isomorphism problem.* Given two graphs G_1 and G_2, determine whether G_1 contains a subgraph that is isomorphic to G_2.
- *Clique problem.* Deciding whether a graph contains a clique larger than a given size.
- *Subset sum problem.* Given a set of integers, does any non-empty subset of them add up to zero?

- *Tree width at most k.* Determine whether the tree width of a given graph is at most a given integer k.

An NP-intermediate problem, *NPI* for short, is a decision problem belonging to NP but it is not in P, and it is not *NP-complete.* So far we have not found any *NPI* problem. If we could find one NP problem that is neither P not *NP-complete,* it would mean that $P \neq NP$ since we would have a problem in NP that is not in P.
If we assume $P = NP$, then the set of NP-intermediate problems would be empty because we cannot have a problem in NP but not in P. On the other side of the coin, if $P \neq NP$, then Ladner's theorem guarantees at least one NP-intermediate problem exists. As a result of that, $P = NP$ if and only if *NPI* is empty.

Two of the most well known *NPI* candidates are the following:

- *Decision version of factoring.* Given integers n and k, is there an integer m with $1 < m < k$ such that m divides n?
- *Graph isomorphism problem.* Given two finite graphs, determine whether they are isomorphic.

Additional problems that might be *NPI* are listed below.

1. *Factoring.* Finding the prime factorization of an integer.

2. *Linear divisibility problem.* Given two integers m and n, determine whether there exists an integer of the form $mx + 1$ that divides n.

3. *Minimum dominating set in tournament.* Given a tournament G, (a directed graph where every two vertices are connected by one edge) find a dominating set $S \subset V(G)$ of minimum cardinality.

4. *Discrete logarithm problem.* Given a group $\mathcal{G}$, an element g of $\mathcal{G}$ and a generator a of the group, find the discrete logarithm of g to the base a in the group $\mathcal{G}$, that is, find an integer x such that $a^x = g$.
   Several important algorithms in public-key cryptography base their security on the assumption that the discrete logarithm problem over certain groups has no efficient solution.

5. *Graceful labeling in graphs.* Deciding whether a graph admits a graceful labeling (a special labeling for graphs).

As already said, *NP* is only a class of decision problems. However, there are problems aren't naturally stated in this manner. Examples of such are: "list all maximal cliques ", "find the shortest path in the graph G that visits every vertex" and, "give a set of variable assignments that makes a boolean expression true". Some of these problems might even have the same sort of difficulty as an *NP-complete* problem or even more. A problem like this is called *NP-hard.* Informally

stated, this class contains problems which are at least as hard as the hardest problems in *NP*. Hence, all *NP-complete* problems happen to be *NP-hard* as well. Now, some *NP-hard* but not *NP-complete* problems include:

- *Optimization version of TSP.* Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?
- *Minimum Vertex Cover.* Given graph G find the minimum vertex cover for G.
- *Knapsack problem.* Given weights and values of n items, put these items in a knapsack of capacity w to get the maximum total value in the knapsack.
- *Optimization version of some NP-complete problems.*

One way to classify a problem is to measure the time and space of an algorithm designed in a Turing machine to recognize or transduce its associated language.

For our purpose, an *alphabet* is a finite non-empty set $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$. The elements of Σ are called the symbols of the alphabet. A *string* or *word* is a finite sequence of symbols of the alphabet, $x = (\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_n})$, which we will rather write as $\sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_n}$. The empty word is denoted by e. The expression $|x|$ represents the number of symbols of the word x and it is called the *length* of x. For instance, $|e| = 0$. The

infinite set of words over an alphabet Σ is denoted by Σ^* and a subset $\mathcal{L} \subseteq \Sigma^*$ is referred to as a *language* over Σ.

Definition 1.1. A *deterministic turing machine* is tuple $\langle Q, \Sigma, I, q_0, F \rangle$ where

- Q is a finite set of states,
- Σ is an alphabet which contains the special symbol B, called the blank symbol,
- I is a finite set of 5-tuples $\langle q, \sigma, \sigma', m, q' \rangle$ called the set of instructions where $q, q' \in Q$, $\sigma, \sigma' \in \Sigma$, and $m \in \{L, S, R\}$, in such a way that there are no pairs of 5-tuples in the set with the same first two elements,
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is the set of final states.

Each $i \in I$ is interpreted as a command for the head of the tape of the turing machine. The m represents where the head will move after processing the step, it can be one of three values: R means move right one square, L means move left one square, and S means to stay put and do not move the head.

The definition of *nondeterministis turing machines (NTM)* is very similar to the definition of DTM, however in this case 5-tuples with the same first two elements are allowed.

We have already mentioned the P and NP classes. Now let us formalize these terms. Let $f : \mathbb{N} \to \mathbb{N}$ be function. The set of all languages accepted by a deterministic Turing machine within the time $f(n)$ is called the complexity class $DTIME(f(n))$. Similarly, the set of all languages accepted by a non-deterministic Turing machine is known as the complexity class $NTIME(f(n))$.

Definition 1.2. (a) The class of languages accepted in polynomial time by a DTM, denoted by P, is defined as

$$P = \bigcup_{k \geq 0} DTIME(n^k).$$

(b) The class of languages accepted in polynomial time by a NTM, denoted by NP, is defined as

$$NP = \bigcup_{k \geq 0} NTIME(n^k)$$

1.2 Fixed Parameter Tractability

It is well know that classical complexity theory analyzes and classifies problems according to the amount of resources necessary to solve a problem, two commonly considered resources are time and space. Parameterized complexity theory is a branch of computational complexity theory that not only considers the complexity with respect to the size of the inputs, it also considers complexity in terms of a parameter, which is a

numerical value which depends on the input in an arbitrary way. The central notion of this theory is the fixed parameter tractability.

Basic Definitions

Definition 1.3. (a) Let Σ be a finite alphabet. A *parameterization of* Σ^* is a mapping $\kappa : \Sigma^* \to \mathbb{N}$ with κ computable in polinomial time.

(b) A *parameterized problem* over Σ is a couple (Q, κ) where $Q \subseteq \Sigma^*$ and κ is a parameterization.

(c) If (Q, κ) is a parameterized problem over Σ, then the strings $x \in Q$ will be called *instances* of Q, and the corresponding values $\kappa(x)$ will be called *parameters.*

The languages $\mathcal{L}$ of the cartesian product $\Sigma^* \times \mathbb{N}$, $(\mathcal{L} \subseteq \Sigma^* \times \mathbb{N})$ are called *parameterized languages.* Let (x, k) be an element of a parameterized language $\mathcal{L}$, the second input of this couple, k, will be called *parameter.*

In parameterized complexity theory it is possible to define a parameter that makes, in some cases, a difficult problem treatable. This is why it is important to introduce the next definition.

Definition 1.4. Let Σ be a finite alphabet and $\kappa : \Sigma^* \to \mathbb{N}$ a parameterization.

(a) Given an algorithm $\mathbb{A}$ and an input alphabet Σ, we say that an algorithm is *fixed parameter tractable* (FPT algorithm) with respect to κ if there exist a computable function $f : \mathbb{N} \to \mathbb{N}$ and a polynomial $p \in \mathbb{N}[x]$ such that for every $x \in \Sigma^*$, the running time of $\mathbb{A}$ in an input x is at most $f(\kappa(x)) \cdot p(|x|)$.[1]

(b) A parameterized problem (Q, κ) is *fixed parameter tractable* if there exists a *FPT* algorithm with respect to κ which decides Q.

For example, for the vertex cover problem, the parameter can be the number of vertices in the cover, say k. Moreover, it can be shown that this problem parameterized by k can be solved in time $2^k|G|$ [16].

1.3 Constraint satisfaction problems

On occasions we can find problems subject to a set of restrictions that must be satisfied in order to find a solution. Such problems are known as *constraint satisfaction problems*, abbreviated *CSPs*, and although they have always naturally existed, the first works related to constraint satisfaction date back to the 1960s in the field of Artificial Intelligence.

[1] Since $p \in \mathbb{N}[x]$, $p(|x|)$ stands for the polynomial p evaluated in the lenght of the word x.

In this work we will focus mainly on problems on a finite Boolean domain. Let us recall first that an assignment for a Boolean formula with set of variables $\{v_1, v_2, v_3 \ldots, v_n\}$, is a function $\tau : \{v_1, v_2, v_3, \ldots, v_n\} \to \{0, 1\}$, that is, one possible configuration of the input variables and the result of the operation for those values.

Definition 1.5. A *constraint* is a Boolean function

$$f : \{0, 1\}^k \to \{0, 1\},$$

where $k \in \mathbb{N}$ is called the arity of f.

In this sense, the domain of a Boolean function is the set of all possible assignments $\{\tau : \{x_1, x_2, \ldots, x_k\} \to \{0, 1\}\}$, which is the same as the set whose elements are the words that are made up of the binary values 0 or 1 and whose length is k. Thus, each assignment τ can be identified with a k-tuple.

Definition 1.6. A Boolean function f is *satisfiable* if there exists an assignment τ such that $f(\tau) = 1$. If such assignment does not exist then it is *unsatisfiable.*

Example 1.1. Let us consider the Boolean variables x, y and z.

1. The negation of x, $f(x) = \bar{x}$, is a unary constraint function.

2. The disjunction and conjunction of x and y, $f_1(x, y) = x \vee y$ and $f_2(x, y) = x \wedge y$, respectively, are binary constraint functions.

3. The function defined by $f(x, y, z) = x \wedge y \vee z$ is a ternary constraint function.

As expected, constraints may be applied to more than three variables, in fact, they can be applied to a large number of variables but not necessarily to all the variables in the domain. An example of this is the Boolean function $f : \{0,1\}^4 \to \{0,1\}$ defined by $f(w, x, y, z) = x \wedge \bar{y} \vee z$.

Definition 1.7. Given the Boolean variables $v_1, v_2, v_3, \ldots, v_n$, a constraint f of arity k, and the indices $i_1, i_2, i_3, \ldots, i_k \in \{1, 2, 3, \ldots, n\}$, the pair $\langle f, (i_1, i_2, i_3, \ldots, i_k)\rangle$ is referred to as the *application of the constraint* f to $v_1, v_2, v_3, \ldots, v_n$, and it is denoted by $f(v_{i_1}, v_{i_2}, v_{i_3}, \ldots, v_{i_k})$.

A finite collection of satisfiable constraints $\mathcal{F}$ is called a *constraint family* or a *constraint set.*

Definition 1.8. Let $v_1, v_2, v_3, \ldots, v_n$ be Boolean variables and $m \in \mathbb{N}$. A collection of constraint applications of the form $\{\langle f_j, (i_1(j), \ldots, i_{k_j}(j))\rangle\}_{j=1}^{m}$, where $f_j \in \mathcal{F}$ and k_j is the arity of f_j, is referred to as an $\mathcal{F}$-collection of constraints.

If it possible to find an assignment which satisfies every element of the $\mathcal{F}$-collection of constraints, then such collection is satisfiable.

Example 1.2. Given the $\mathcal{F}$-collection of constraint applications $\{\langle f_j, (i_1(j), \ldots, i_{k_j}(j))\rangle\}_{j=1}^{m}$ for $m = 3$, on the Boolean

variables v_1, v_2, v_3, v_4, v_5, where $f_1 : \{0,1\}^2 \to \{0,1\}$, $f_2 : \{0,1\}^4 \to \{0,1\}$, and $f_3 : \{0,1\} \to \{0,1\}$ are defined by $f_1(v_2, v_3) = v_2 \vee \bar{v_3}$, $f_2(v_2, v_3, v_4, v_5) = (v_2 \wedge v_3) \vee (v_4 \wedge v_5)$, and $f_3(v_1) = \bar{v_1}$, respectively, let us find an assignment which satisfies it.

Explicitly, the $\mathcal{F}$-collection of constraints is the set

$\{\langle f_1, (i_1(1), i_2(1))\rangle, \langle f_2, (i_1(2), i_2(2), i_3(2), i_4(2))\rangle, \langle f_3, (i_1(3))\rangle\}$

$= \{f_1(v_{i_1(1)}, v_{i_2(1)}), f_2(v_{i_1(2)}, v_{i_2(2)}, v_{i_3(2)}, v_{i_4(2)}), f_3(v_{i_1(3)})\}$

$= \{f_1(v_2, v_3), f_2(v_2, v_3, v_4, v_5), f(v_1)\}$.

We are looking for an assignment which satisfies f_1, f_2, and f_3 simultaneously, that is, an assignment which satisfies the Boolean formula $(v_1 \vee \bar{v_3}) \wedge ((v_2 \wedge v_3) \vee (v_4 \wedge v_5)) \wedge \bar{v_1}$. Clearly, some assignments which satisfy the given $\mathcal{F}$-collection of constraints are $(0,1,0,1,1)$, $(0,1,1,0,1)$, and $(0,0,0,1,1)$.

The constraint satisfaction problem can be stated in different manners, depending on the class of problems that we study. For instance, the constraint satisfaction version of the class NP is the SAT($\mathcal{F}$) problem: given an $\mathcal{F}$-collection of constraints, decide whether there exists an assignment that satisfies it. For the class $\#P$, we may define $\#$SAT($\mathcal{F}$), the counting version of SAT($\mathcal{F}$,) which asks to find the number of distinct assignments that satisfy a given $\mathcal{F}$-collection of constraints.

Particularly, the $3SAT$ problem, which asks whether it is possible to solve the SAT problem provided that there are n variables and 3 literals in each clause of a Boolean formula in

conjunctive normal form, can be interpreted as a *CSP* problem where the constraints correspond to the clauses, thus the arity of each constraint is 3.

When it comes to optimization, there is more than one way to obtain the constraint satisfaction version. One of them, the MAX constraint satisfaction problem denoted MAX SAT($\mathcal{F}$), is the problem of finding an assignment that satisfies a maximum number of constraints in a given $\mathcal{F}$-collection of constraints. Its analogous version, MIN SAT($\mathcal{F}$), asks to find an assignment that minimizes the number of unsatisfied constraints in a given $\mathcal{F}$-collection of constraints.

Chapter 2

Graphs and Hypergraphs

2.1 Graphs

The origin of graph theory dates back to 1735 with the problem of the Königsberg bridges, however, it was until 1878 when the term "graph" was introduced. In numerous areas of study, mathematical properties of graphs have been exploited to study a huge series of phenomena. Next, let us mention some concepts that will eventually lead us to the definition of tree decomposition.

Definition 2.1. A *simple graph* G is a pair of sets (V, E), where V is the set of *vertices* (or *nodes*) and $E \subset [V]^2$ is the set of *edges*.

For simplicity, we use the symbols $V(G)$ and $E(G)$ to represent the set of nodes and the set of edges of G, respectively.

Also, by definition, the edges of G are sets of the form $\{u, v\}$ such that $u, v \in V(G)$, yet we will write uv to refer to the edge $\{u, v\}$. If $V(G) = \emptyset$, then $E(G) = \emptyset$, in such case $G = \emptyset$ is called the *empty graph*.
Typically, a graph is represented through a series of points (the vertices) joined by lines (the edges).

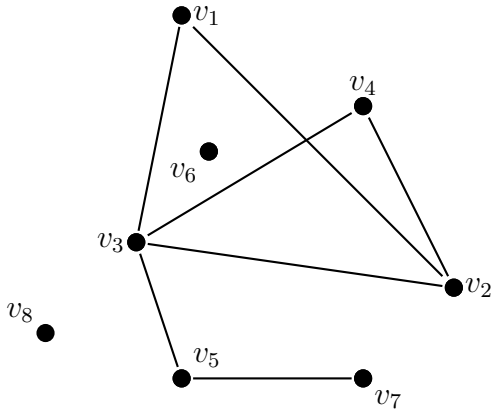


Figure 2.1: Simple graph $G = (V, E)$.

For this section only, the term "graph" will be used to refer to a simple graph, unless stated otherwise.

Definition 2.2. A graph G' is a *subgraph* of graph G, written $G' \subseteq G$, if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$.

Definition 2.3. Let G be a graph. The cardinality of $V(G)$ is called the *order* of G and it is denoted by $\mid G \mid$. The number of edges is called the *size* of G and is denoted by $\parallel G \parallel$.

Definition 2.4. Two vertices of a graph G, say u and v, are said to be *neighbors* or *adjacent* nodes if $uv \in E(G)$. Similarly, two edges are adjacent if they share a vertex.

Notice that $v \in V(G)$ may have no neighbors at all, it may have one, or even more. If the order or G is n, then the maximum possible number of adjacent nodes of v is $n-1$, that is because a vertex can form an edge with all other vertices except by itself. The number of adjacent vertices of a given $v \in V(G)$ is known as the *degree* of v.

Definition 2.5. A *clique* is subgraph C of a graph G such that every two distinct vertices in C are adjacent.

Given a clique C with n vertices, the degree of every vertex of C is the same: $n-1$.

Definition 2.6. A non-empty graph $\mathcal{P} = (V, E)$ where $V = \{x_0, x_1, \ldots, x_n\}$, $E = \{x_0x_1, x_1x_2, \ldots, x_{n-1}x_n\}$ and $x_i \neq x_j$ for every $i \neq j$, is called a *path*. The vertices x_0 and x_n are the *ends* of $\mathcal{P}$ and the rest of the vertices are its *inner vertices*. The number of edges in a path is called its *length*.

Definition 2.7. A *cycle* is a path which starts and ends at the same vertex.

Since we are considering only simple graphs, it is possible to have paths of length 0, and the least possible length of a cycle is 3.

Definition 2.8. If there is a path joining any two vertices of a non-empty graph G, then G is *connected*. Otherwise it is *disconnected*. A maximal connected subset of $V(G)$ is called a *component*.

A special kind of graphs, trees, provide a range of useful applications, it is of our interest the one in data structures of computer science.

Definition 2.9. A connected graph with no cycles is called a *tree*, the disjoint union of them is a *forest*. Every vertex having one adjacent edge in a tree is called a *leaf* node. The tree is said to be *rooted* if a particular vertex is designated as the root.

In a tree, the node which is a predecessor of any node is called a *parent* and the node which is descendant of any node is known as a *child node*. Without doubt, all the nodes except the root are child nodes.

As one would expect, given a tree G a *subtree* of G is a subgraph of G that is also a tree. The importance of trees lies in the fact that they are graphs that connect all vertices using the fewest edges possible. A tree with n vertices has exactly $n - 1$ edges, and a cycle is formed if any edge is added to the graph. Moreover, a tree is connected but would become disconnected if any single edge is removed from it.

Trees are considered the most simple objects in graph theory, their properties are of great benefit when trying to solve certain problems. Outstandingly, some problems considered to be *NP-hard* for general graphs can be solved in linear time for trees.

2.2 Tree decompositions

In [15], Diestel points out that the concept of tree decompositions and tree-width were originally introduced (under diferente names) by Rudolf Halin in a 1976 paper titled *S-functions for graphs.* These concepts were later reintroduced by Robertson and Seymour led by the attempt to solve Wagner's conjecture, quoting the words of Professor Seymour [30] "we had no idea that Halin had thought about tree-width already."

As specified by Courcelle's theorem, if a graph problem can be expressed in the logic of graphs using monadic second order logic, then it can be solved in linear time on bounded tree-width graphs. Case in point: the 3-coloring problem for graphs, which asks whether it is possible to assign one of 3 colors to each node of a graph in such a way that the adjacent vertices do not share the same color, is NP-complete for general graphs, but since it can be writen in monadic second order logic, by Courcelle's results, it can be solved in linear time for graphs with bounded constant tree-width [11].

Definition 2.10. (*a*) A *tree decomposition* of a graph G is a tree T along with a labeling function $\varphi : V(T) \longrightarrow \wp(V(G))$, such that the following conditions hold:

1. $V(G) = \bigcup_{t \in V(T)} \varphi(t)$.

2. For every edge e of G, there is some $t \in V(T)$ such that $e \subseteq \varphi(t)$.

3. (Interpolation property.) If t is an inner vertex in the only path in T from x to y, then $\varphi(x) \cap \varphi(y) \subseteq \varphi(t)$.

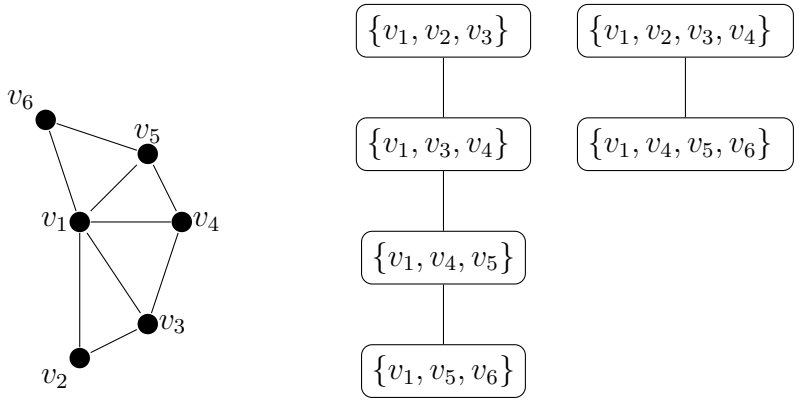
(b) If $\langle T, \varphi \rangle$ is a tree decomposition, its *width* is the maximum value of $|\varphi(x)| - 1$ taken over all the vertices $x \in V(T)$.

The labeling function φ is actually a family of subsets of $\wp(V(G))$, each element of this family is called a *bag*. According to the definition, it is possible to have more than one tree decomposition for the same graph. As a matter of fact, every graph G has a trivial tree decomposition which has only one vertex which contains all the elements of $V(G)$. However, this is ineffective for the purpose of solving *NP-hard* problems.

Example 2.1. In figure 2.2 (b), we present two possible tree decompositions for the graph in figure 2.2(a). The width of the tree decomposition on the left is equal to 2, and the width of the other tree decomposition is 3.

Definition 2.11. The minimum width over all tree decompositions of a graph G is called the *tree-width* of G, is it usually denoted as $tw(G)$.

The problem of determining the tree-width of a graph is unfortunately NP-hard [21]. However, when we consider a graph



(a) Graph $G = (V, E)$ **(b)** Tree decompositions of G

Figure 2.2: Two possible tree decomposition for the same graph.

G and a parameter k (an integer), there are algorithms that determine whether G has tree-width at most k in time $O(n^k)$ [34], where n is the order of G.

Now, let us mention some facts related to tree-width.

Theorem 2.1. *If a graph G has tree-width k then it has a vertex of degree at most k.*

Proof. Let $\langle T, \varphi \rangle$ be a tree decomposition of G whose width is k and consider a leaf ℓ of T. Since ℓ is a leaf, it only has one neighbor t. Then the corresponding bag of ℓ, B_ℓ, contains a vertex v such that v is not an element of B_t, or else $B_\ell \subseteq B_t$. Besides that, v is not a member of any other bag either, otherwise by the interpolation property $v \in B_t$. Considering that $|B_\ell| \leq k + 1$, it follows that the degree of v in G is at most k. □

Theorem 2.2. *A graph with n vertices is a clique if and only if it has tree-width $n-1$.*

Proof. Let G be a n-vertex clique and assume $tw(G) \neq n-1$, then $tw(G) \leq n-2$. By virtue of Theorem 2.1, G has a vertex of degree at most $n-2$, which is a contradiction.
For the converse, let G be graph with n nodes having treewidth equal to $n-1$ then, we can say that its tree decomposition is $\langle T, \phi \rangle$ where T is a tree whose only bag contains all vertices of G. If G is not a clique then there exists a pair of distinct vertices in $V(G)$, say v_1 and v_2, such that they are not neighbors. Hence, we can take a tree decomposition with two nodes: $V \setminus \{v_1\}$ and $V \setminus \{v_2\}$. In this case, the treewidth is $n-2$, contradicting the hypothesis.

□

Theorem 2.3. *A graph G is a forest if and only if G has tree-width at most 1.*

Even though the most well-known width notion in graphs is tree-width, there exist many more: path-width, branch-width, and clique-width, to mention a few. If the tree decomposition of a graph is a path we say that the decomposition is a *path decomposition* and the term tree-width is replaced by *path-width.* Then, it is straight forward that the path-width of a graph G, $pw(G)$, is at least its tree-width:

$$pw(G) \geq tw(G).$$

Now, defining branch width is a different matter. Let $G = (V, E)$ be a graph and consider a partition of its edges $E = E_1 \cup E_2$. Although, E_1 and E_2 have no common edges they might have some vertices that are incident both with an edge in E_1 and with an edge in E_2. The number of vertices satisfying this condition is called the *connectivity value* and is denoted by $\lambda(E_1)$.

Definition 2.12. A *branch decomposition* of a graph G is a subcubic tree T, which means that each of its nodes has degree 1 or 3, such that each leaf of T is labeled by an edge of G.

Remark that, in a branch-decomposition T of a graph $G = (V, E)$, for any edge e of T, $T - e$ disconnects T into two components, that is, e sets out a partition $E = E_1 \cup E_2$ of the edges of the graph such that the edges in E_1 and E_2 are mapped to one component of $T - e$, respectively. The *width of the edge* e in T is the connectivity value $\lambda(E_1) = \lambda(E_2)$, the *width of* T is the maximum over the widths of its edges and, the *branch-width* of G, $bw(G)$, is the minimum width over all its branch decompositions.

Example 2.2. The following illustration shows a branch decomposition for the graph $G = (E, V)$ given in figure 2.2, where we take the partition $E = E_1 \cup E_2$ with $E_1 = \{v_1v_4, v_1v_6, v_5v_6\}$, consequently $E_2 = \{v_1v_2, v_1v_3, v_1v_5, v_2v_3, v_3v_4, v_4v_5\}$. As illustrated in figure 2.3 the edge e in orange separates E_1 from E_2; the vertices in common are v_1, v_4, and v_5, thereupon the width of e is 3.

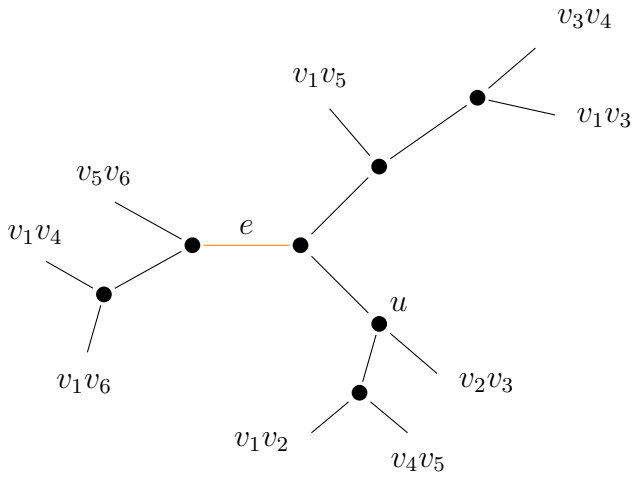


Figure 2.3: A branch decomposition for G from figure 2.2.

Branch-width has similarities with tree-width, in fact, the following result formulates an explicit relationship between them.

Theorem 2.4. *[29] Let G be a graph with tree-width a and branch-width $b > 1$, then*

$$b \leq a + 1 \leq \lfloor \frac{3}{2} b \rfloor.$$

Proof. Let G be a graph and consider a branch decomposition for it having width b. By definition every vertex of such decomposition has degree either 1 or 3. Let x be a node of the decomposition such that $d(x) = 3$, then there must exist three subsets $B_1(x), B_2(x), B_3(x) \subseteq V(G)$ determined by

the three edges incident to x.[1] Therefore, for a vertex $v \in G$ appearing in one of the three subsets of $V(G)$ that we previously mentioned, it is also a member of another of those. Under that circumstances, $|B_1(x) \cup B_2(x) \cup B_3(x)| \leq \lfloor \frac{3}{2}b \rfloor$. If we set $B(x) = B_1(x) \cup B_2(x) \cup B_3(x)$ and we take the tree decomposition whose bags are the $B(x)$, it follows that $tw(G) \leq \lfloor \frac{3}{2}b \rfloor - 1$.

□

Definition 2.13. Let G be a graph whose set of nodes is $\{v_1, v_2, \ldots, v_n\}$. The adjacency matrix of G is the $n \times n$ matrix $A(G) = (a_{ij})$ defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if } v_i v_j \in E(G) \\ 0 & \text{if } v_i v_j \notin E(G). \end{cases}$$

Example 2.3. The adjacency matrix for the graph G in Figure 2.2 is shown below.

[1] Going back to figure 2.3, vertex u has degree 3 and the three subsets set out by such node are $B_1(u) = \{v_2, v_3\}, B_2(u) = \{v_1, v_2, v_4, v_5\}$, and $B_3(u) = \{v_1, v_3, v_4, v_5, v_6\}$

$$A(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

2.3 Hypergraphs

The origin of hypergraph theory dates back to the second half of the twentieth century, it is a recent field mainly developed in France by Claude Berge in the 1960s [4]. Of course, there were other characters such as Paul Erdös, László Lovász, and Paul Turán, who developed studies in this branch of mathematics as well [9]. Hypergraphs are mathematical tools which can be used to model relationships among more than two objects. They are more general structures than graphs. In this section we formally define hypergraphs and introduce basic notions about them.

Definition 2.14. A *hypergraph* is a couple $H = (V, E)$ where V the set of vertices or nodes of H and the set of edges E is a collection of subsets of $\wp(V)$.

Hypergraph theory was originally developed as an extension of graph theory, as a result, a great deal of the terminology

used for graph is the same for hypergraphs. For instance, the *order* of the hypergraph H is the cardinality of $V(H)$ and its *size* is the cardinality of $E(H)$. In adition, two vertices are *adjacent* if they lie in the same edge and two edges are called *incident* (rather than adjacent) if they share a vertex. A vertex of a hypergraph incident to no edge is called *isolated.* If both $V(H)$ and $E(H)$ are empty then H is the *empty hypergraph;* if $V(H) \neq \emptyset$ and $E(H) = \emptyset$, H is a *trivial hypergraph.* Again we will use the symbol $d(v)$ to denote the degree of vertex $v \in V(H)$, that is the number of edges incident to v.

When drawing a hypergraph, points are used to represent the vertices and each edge is represented by a simple closed curve enclosing its corresponding elements. Other authors might have different ways of drawing hypergraphs.

Example 2.4. The hypergraph H shown in figure 2.4 contains an isolated node, the order of H is 10, and its size is 7.

Definition 2.15. Let u and v two distinct vertices of a hypergraph H, a *path* from vertex u to vertex v is a sequence of distinct edges $(e_{p_1}, e_{p_2}, \ldots, e_{p_k})$ such that $u \in e_{p_1}$, $v \in e_{p_k}$, $e_{p_i} \cap e_{p_{i+1}} \neq \emptyset$, and $e_{p_i} \cap e_{p_j} = \emptyset$ if $|j - i| \geq 2$.

The sequence $(e_{p_1}, e_{p_2}, \ldots, e_{p_k})$ is called an *edge path* (or just a path when no confusion arises) from e_{p_1} to e_{p_k}. Two nodes of a hypergraph are *connected* if there is a path from one to the other and two edges are connected if there is an edge path

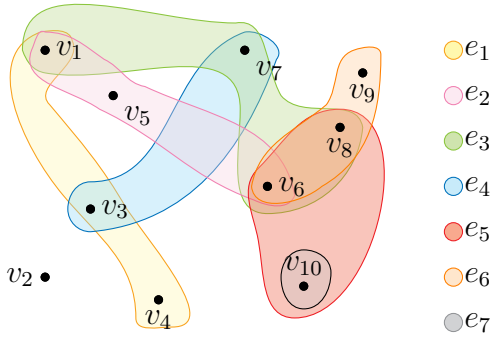


Figure 2.4: A hypergraph H.

from one to the other. The maximal connected sets of edges of a hypergraph are known as its *components.*

For a hypergraph H and a set $S \subseteq V(H)$, we say that two elements of $V(H)$, a and b, are $[S]$-*adjacent* if there is an edge $e \in E(H)$ such that $\{a, b\} \subseteq e \setminus S$. An $[S]$-*path* from a to b is a sequence $a = x_0, x_1, x_2, \ldots, x_n = b$ of vertices such that x_i is $[S]$-adjacent to x_{i+1}, for every $i \in \{0, 1, 2, \ldots, n-1\}$. A subset X of $V(H)$ is $[S]$-*connected* if for all $x, y \in X$ there is a $[S]$-path joining them. An $[S]$-*component* is a maximal $[S]$-connected, non-empty set of vertices $X \subseteq V(H) \setminus S$.

Adjacency in hypergraphs still models pairwise relationships, as it does in graphs. However, since edges in hypergraphs contain $k \geq 1$ nodes, it has not been an easy task to extend the notion of adjacency matrix to hypergraphs. Some approaches have been made on this subject, one of them can be found in [9] where the adjacency matrix is defined as:

Definition 2.16. The *adjacency matrix* of the hypergraph $H = (V, E)$ is the $|V| \times |V|$ matrix $A(H) = (a_{ij})$, where

$$a_{ij} = \begin{cases} |\{e \in E : v_i, v_j \in e\}| & \text{if } i \neq j \\ 0 & \text{if } i = j. \end{cases}$$

In line with this definition, whenever we have an edge of a hypergraph containing only one vertex v_j, we will still get the entry $a_{jj} = 0$ in the matrix. For instance, the adjacency matrix for the hypergraph in Figure 2.4 is written next.

$$A(H) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \end{array} \begin{array}{c} \begin{array}{cccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} \end{array} \\ \left(\begin{array}{cccccccccc} 0 & 0 & 1 & 1 & 1 & 2 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 & 0 & 1 & 3 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 3 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{array} \right) \end{array}.$$

For what comes next, we assume that hypergraphs have no isolated vertices.

Let us point out that the notion of cyclicity and aciclicity in hypergraphs is quite different from the one in graphs, there are differente "degrees" of it, as it happens, a loop (an edge

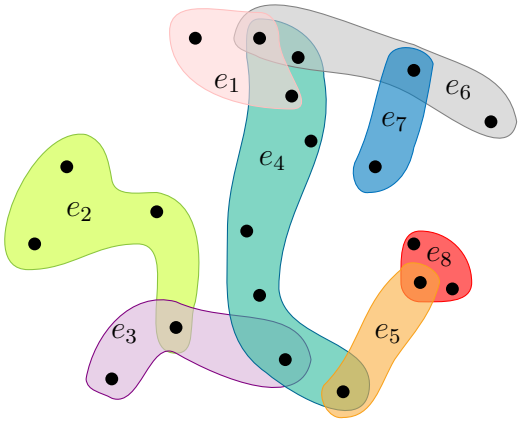
containing a single vertex) is not considered as a cycle in hypergraphs. The most general and common notion of cyclicity in literature is α-cyclicity [18], its definition has not been fully studied in terms of cycles, instead it is given in terms of other concepts.

Definition 2.17. A *join tree* for a hypergraph H is, if it exist, a rooted tree $T = (E(H), J)$ such that, if $v \in V(H)$, belongs to two edges e_i and e_j in $E(H)$, then it is also contained in all the inner nodes of the path in T from e_i to e_j.

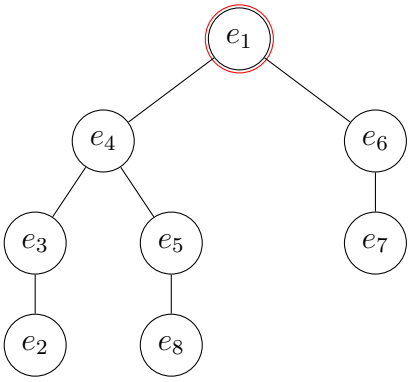
We will use the word *acyclic* to refer to α-acyclicity, unless otherwise stated.

Definition 2.18. A hypergraph is acyclic if it admits a join tree.

Example 2.5. Figure 2.5 shows an acyclic hypergraph.



(a) Hypergraph H.



(b) Rooted tree T.

Figure 2.5: A join tree T for hypergraph H.

Chapter 3

Hypergraph decompositions

In the previous section we saw that treewidth is a measure of the cyclicity of a graph. Likewise, there exits various hypergraph decomposition methods leading to different notions of hypergraph width which are helpful when trying to measure hypergraph cyclicity.

3.1 Hypertree decompositions

Let us first turn our attention to hypertree decompositions, a concept introduced by Gottlob et al. at the ACM Symposium on Principles of Database Systems (PODS) 1999 [20]. In graph theory, bounded treewidth yields a generalization

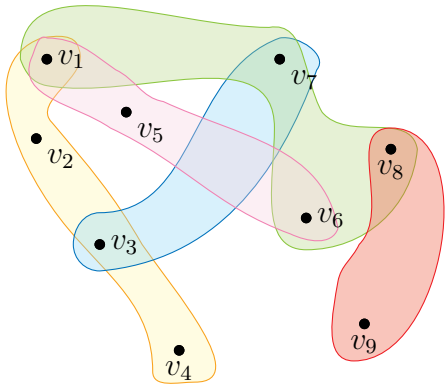
of graph acyclicity. Correspondingly, hypertree width, is a true generalization of acyclicity in hypergraph theory, as the acyclic hypergraphs are precisely those hypergraphs having hypertree width one. [25].

Definition 3.1. Let H be a hypergraph, a *tree decomposition* of H is a pair $\langle T, \varphi \rangle$ where T is a rooted tree, and φ is a labeling function from $V(T)$ to $\wp(V(H))$, such that the following conditions are satisfied:

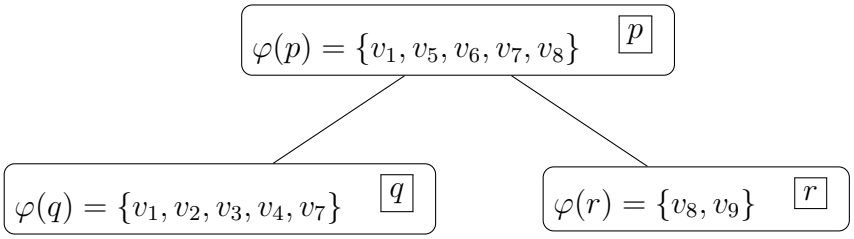
1. For each $x \in V(H)$ there exists $t \in V(T)$ such that $x \in \varphi(t)$;
2. For each edge e of H there is a node $t \in V(T)$ such that $e \subseteq \varphi(t)$;
3. (Interpolation property.) For each node $x \in V(H)$ the set $\{t \in V(T) : x \in \varphi(t)\}$ is non empty and connected in T, this means that there is a path connecting all the vertices of T, called the bags of the tree, which contain the vertex x.

It is not hard to see that a join tree of a hypergraph H, defined in chapter 2, is also a tree decomposition of H. However, the hypergraph in figure 3.1 admits tree decomposition but no join tree.

Definition 3.2. A *hypertree* for a hypergraph H, also called a *generalized hypertree decomposition,* is a triple $\langle T, \varphi, \psi \rangle$,



(a) Hypergraph H



(b) $\langle T, \varphi \rangle$

Figure 3.1: A tree decomposition for H.

where $\langle T, \varphi \rangle$ is a tree decomposition and ψ is a labeling function from $V(T)$ to $\wp(E(H))$ such that for each vertex $t \in V(T)$, $\varphi(t) \subseteq \bigcup_{e \in \psi(t)} e$, that is, the set $\varphi(t)$ is covered by the edges in the ψ-labeling.

For every $t \in V(T)$, let us use the symbol T_t to refer to the subtree of T rooted at t and $\varphi(T_t)$ to denote the set $\bigcup_{s \in T_t} \varphi(s)$. In order to simplify the computation of decomposition a special condition, called *Descendant condition,* was introduced:

for each $t \in V(T)$ and for each edge $e \in \psi(t)$ it holds that $e \cap \varphi(T_t) \subseteq \varphi(t)$.

Definition 3.3. A *hypertree decomposition* of a hypergraph H is a hypertree $\langle T, \varphi, \psi \rangle$ for H which satistifies the Descendant condition.

Example 3.1. Let H be a hypergraph, as pictured in figure 3.2, with vertices $V(H) = \{v_1, v_2, v_3, \ldots, v_{10}\}$ and $E(H) = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, with $e_1 = \{v_1, v_2, v_5, v_9\}$, $e_2 = \{v_3, v_4\}$, $e_3 = \{v_3, v_4, v_7\}$, $e_4 = \{v_1, v_3, v_8, v_{10}\}$, $e_5 = \{v_7, v_8\}$, and $e_6 = \{v_6, v_9\}$. Figure 3.3 shows two different decompositions for H. Notice that the hypertree in (a) is not a hypertree decomposition since it does not satisfy the Descendant condition. In fact, for $e_2 \in \psi(r)$ we have that $e_2 \cap \varphi(T_r) = \{v_3, v_4\} \cap \{v_1, v_3, v_4, v_7, v_8, v_{10}\} = \{v_3, v_4\} \subsetneq \{v_1, v_3, v_7, v_8, v_{10}\} = \varphi(r)$ and; for $e_4 \in \psi(p)$, $e_4 \cap \varphi(T_p) = \{v_2, v_5, v_6, v_7\} \subsetneq \varphi(p) = \{v_1, v_2, v_5, v_7, v_9\}$. On the other hand, the hypertree in (b) is indeed a hypertree decomposition for H.

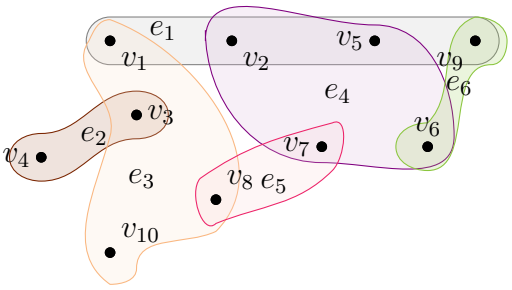
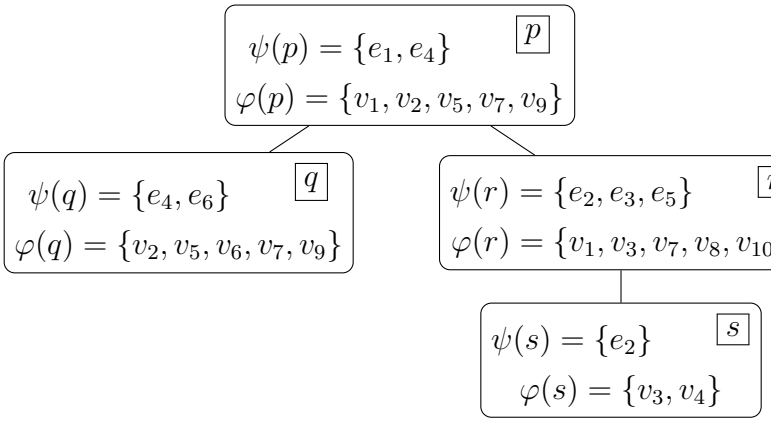
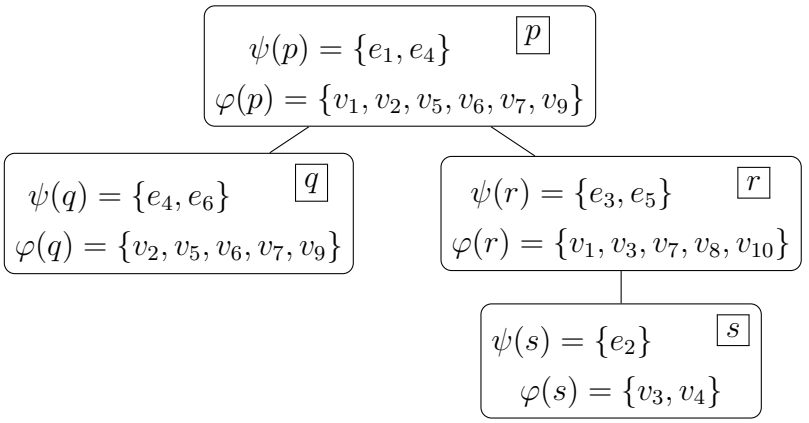


Figure 3.2: A hypergraph H.



(a) A generalized hypertree decomposition for H.



(b) A hypertree decomposition for H.

Figure 3.3: A hypergraph and its decompositions.

Definition 3.4. The *width of a (generalized) hypertree decomposition* $\langle T, \varphi, \psi \rangle$ is given by $max\{|\psi(t)| : t \in V(T)\}$, and the *(generalized) hypertree-width* of a hypergraph is the minimum width over all its (generalized) hypertree decompositions.

Figure 3.3 shows an example of a hypergraph and its hypertree decomposition of width 2.

Notice that since acyclic hypergraphs are those which admit a join tree, and every bag of them contains one and only one edge, then hypertree width generalizes hypergraph acyclicity as stated by the following theorem.

Theorem 3.1. *A hypergraph H is acyclic if and only if*

$$hw(H) = 1.$$

Proof. First, we show that acyclicity implies hypertree width equals to 1. Let H be an acyclic hypergraph, then it admits a join tree T. Also, let T' be a tree and $f : V(T) \longrightarrow V(T')$ be a one-to-one function such that for every $s, t \in V(T)$, it holds that $st \in E(T)$ if only if $f(s)f(t) \in E(T')$. Now, let $\psi : V(T') \longrightarrow \wp(E(H))$ and $\varphi : V(T') \longrightarrow \wp(V(H))$ be the labeling functions defined by $\psi(t') = \psi(f(t)) = \{t\}$ and $\varphi(t')$ equals to the set of vertices of t. Hence, the triple $\langle T', \varphi, \psi\rangle$ is a tree decomposition whose width is 1.
For the converse, let $\langle T, \varphi, \psi\rangle$ be a hypertree decomposition of H having width 1. Without loss of generality, since $hw(H) = 1$, we may assume that ψ associates one hyperedge of H to a vertex of T. It is not hard to show that from $\langle T, \varphi, \psi\rangle$ we may obtain a hypertree $\langle T', \varphi, \psi\rangle$ such that $V(T') \subseteq V(T)$ and $\langle T', \varphi, \psi\rangle$ is a jointree for H. □

Taking into account that all tree decompositions are also generalized hypertree decompositions, it is straightforward that

$ghw(H) \leq hw(H)$ and we know from [1] that $hw(H) \leq 3 \cdot ghw(H) + 1$, thusly:

Theorem 3.2. *Given a hypergraph H, the following inequalities hold:*

$$ghw(H) \leq hw(H) \leq 3 \cdot ghw(H) + 1.$$

This means that for our purposes, hypertreewidth and generalized hypertree width are equivalent. For simplicity, we will work with the latter.

According to the above definitions, certain caracteristics of (generalized) hypertree decompositions might be undesirable. For instance, a decomposition may contain two vertices with exactly the same labels. In order to avoid such kind of redundancies, we define a normal form for (generalized) hypertree decompositions.

For a (generalized) hypertree decomposition $\langle T, \varphi, \psi \rangle$ and a node t of T let us use $[t]$ as a synonym of $[\varphi(t)]$. Viewed in this way, a $[t]$-path is the same as a $[\varphi(t)]$-path; a $[t]$-component is a $[\varphi(t)]$-component, and so forth.

Definition 3.5. Let $\langle T, \varphi, \psi \rangle$ be a (generalized) hypertree decomposition of a hypergraph H, we say that $\langle T, \varphi, \psi \rangle$ is in *normal form* if for each node $t \in V(T)$, and for each child p of t, the conditions below are satisfied:

1. There is exactly one $[t]$-component, say C_t, such that $\varphi(T_p) = C_t \cup (\varphi(t) \cap \varphi(p))$;

2. The $[t]$-component C_t in the first condition is such that $C_t \cap \varphi(p) \neq \emptyset$;
3. $\varphi(t) \cap \Big(\bigcup_{e \in \psi(p)} e \Big) \subseteq \varphi(p)$.

The existence of the normal form is guaranteed in [25] where it is proved that given a hypergraph H with hypertree decomposition of width k there exists a hypertree decomposition of width k in normal form.

Example 3.2. Let us look back at hypergraph H in figure 3.2 and the generalized hypertree decomposition for H displayed in 3.3, we want to check out whether such decomposition is in normal form.
First of all, consider $p, q, r, s \in V(T)$, clearly, p has two children in T: q and r. At the same time, s is a child of r in T. A $[p]$-component in $\langle T, \varphi, \psi \rangle$ is a nonempty maximal $[p]$-connected set $X \subseteq V(H) \backslash \varphi(p) = \{v_3, v_4, v_6, v_8, v_{10}\}$. The only sets which fulfill such conditions are $\{v_6\}$ and $\{v_3, v_4, v_8, v_{10}\}$. Correspondingly, the $[r]$-components for this decomposition are $\{v_4\}$ and $\{v_2, v_5, v_6, v_9\}$.

- For q, child of p,

1. $\varphi(T_q) = \varphi(q) = \{v_2, v_5, v_6, v_7, v_9\}$ and $C_p \cup (\varphi(p) \cap \varphi(q)) = C_p \cup \{v_2, v_5, v_7, v_9\}$, then C_p must be the singleton $\{v_6\}$.
2. $\{v_6\} \cap \varphi(q) \neq \emptyset$.

3. $\varphi(p) \cap \Big(\bigcup_{e \in \psi(q)} e \Big) = \{v_1, v_2, v_5, v_7, v_9\} \cap \{v_2, v_5, v_6, v_7, v_9\}$
$= \{v_2, v_5, v_7, v_9\} \subseteq \varphi(q).$

- For r, child of p,

1. $\varphi(T_r) = \varphi(r) \cup \varphi(s) = \{v_1, v_3, v_4, v_7, v_8, v_{10}\}$ and
$C_p \cup (\varphi(p) \cap \varphi(r)) = C_p \cup \{v_1, v_7\}$, therefore C_p is the set $\{v_3, v_4, v_8, v_{10}\}$.
2. $C_p \cap \varphi(r) = \{v_3, v_4, v_8, v_{10}\} \cap \{v_1, v_3, v_7, v_8, v_{10}\} \neq \emptyset$.
3. $\varphi(p) \cap \Big(\bigcup_{e \in \psi(r)} e \Big) = \{v_1, v_7\} \subseteq \varphi(r)$.

- For s, child of r,

1. $\varphi(T_s) = \varphi(s) = \{v_3, v_4\}$ and
$C_r \cup (\varphi(r) \cap \varphi(s)) = C_r \cup \{v_3\}$, hence $C_r = \{v_4\}$.
2. $C_r \cap \varphi(s) = \{v_4\} \neq \emptyset$.
3. $\varphi(r) \cap \Big(\bigcup_{e \in \psi(s)} e \Big) = \{v_3\} \subseteq \varphi(s)$.

In short, $\langle T, \varphi, \psi \rangle$ is in normal form.

3.2 Other Width Parameters

There is a more general notion that (generalized) hypertree decomposition which we define below.

For a hypergraph H we consider the mappings $\mu : E(H) \longrightarrow \{0,1\}$ and $\eta : E(H) \longrightarrow [0,1]$. If $f \in \{\mu, \eta\}$, then the set of vertices of H covered by f is described as

$$B(f) = \left\{ v \in V(H) | \sum_{e \in E(H), v \in e} f(e) \geq 1 \right\}$$

and its *weight* is

$$weight(f) = \sum_{e \in E(H)} f(e).$$

Definition 3.6. An *edge cover* of a hypergraph H is a function $\mu : E(H) \longrightarrow \{0,1\}$ such that $\sum_{e \in E(H), v \in e} \mu(e) \geq 1$ for all $v \in V(H)$, that is $V(H) = B(\mu)$, and a *fractional edge cover* is a function $\eta : E(H) \longrightarrow [0,1]$ such that $V(H) = B(\eta)$. The *(fractional) edge cover number* is the minimum weight over all (fractional) edge covers and it is denoted by (ρ^*_H) ρ_H, that is $\rho^*_H = \min\{weight(\eta) \,|\, \eta : E(H) \to [0,1] \text{ and } V(H) = B(\eta)\}$.

Standard linear programming results guarantee that this minimum exists and is rational [28].
For $X \subseteq V(H)$, $\rho^*_H(X)$ is the minimum of $\sum_{e \in E(H), v \in e} \eta(e) \geq 1$ taken over all fractional edge covers of H restricted to the vertices $v \in X$.

It is clear that for every hypergraph H we have $\rho^*_H \leq \rho_H$, hence the inequality might be strict or, as shown in the theorem coming next, ρ_H* and ρ_H can be equal.

Theorem 3.3. *Let C be a clique having $2n$ vertices, then $\rho^*_C = \rho_C = n$.*

Proof. Let C be a clique with $2n$ vertices. On one side, we can use a set A of n edges, with $\mu(e) = 1$ for every $e \in A$, to cover the $2n$ vertices of C. Then $\rho_C \leq weight(\mu) = |A| = n$. On the other side, let η be a fractional edge cover of C. By definition, $\sum_{e \in E(H), v \in e} \eta(e) \geq 1$ for every $v \in V(C)$, then the weight on the vertices of C is $\sum_{v \in V(H)} \sum_{e \in E(H), v \in e} \eta(e) \geq 2n \cdot 1$. Since the weight of each edge adds to the weight of at most 2 vertices, we need at least weight n on the edges to achieve at least $2n$ weight on the vertices, that is, $n \leq \rho^*_H$. In short, $n \geq \rho_C \geq \rho^*_C \geq n$. □

Definition 3.7. A *fractional hypertree decomposition* [23] of a hypergraph H is a triple $\langle T, \varphi, \{\eta_t\}_{t \in V(T)} \rangle$, where $\langle T, \varphi \rangle$ is a tree decomposition for H and $\{\eta_t\}_{t \in V(T)}$ is a collection of mappings associating each vertex t of T with a function $\eta_t : E(H) \longrightarrow [0, 1]$ such that $\varphi(t) \subseteq B(\eta_t)$.
The *width* of $\langle T, \varphi, \{\eta_t\}_{t \in V(T)} \rangle$ is the maximum of $weight(\eta_t)$ over all vertices $t \in V(T)$ and the *fractional hypertree width* of H, denoted by $fhw(H)$, is the minimum width over all its fractional hypertree decompositions.

Example 3.3. Let ϵ be an arbitrary but fixed small number greater than zero. Figure 3.4 depicts a hypergraph H with a possible fractional tree decomposition in which $\varphi : V(T) \longrightarrow \wp(V(H))$ is given by $\varphi(p) = \{v_1, v_2, v_3, v_4\}, \varphi(q) =$

$\{v_2, v_5, v_6\}, \varphi(r) = \{v_4, v_5, v_6, v_7, v_8\}, \varphi(p) = \{v_6, v_7, v_9, v_{10}\}$, and the collection $\{\eta_t\}_{t \in V(T)}$ is explicitly described as follows:

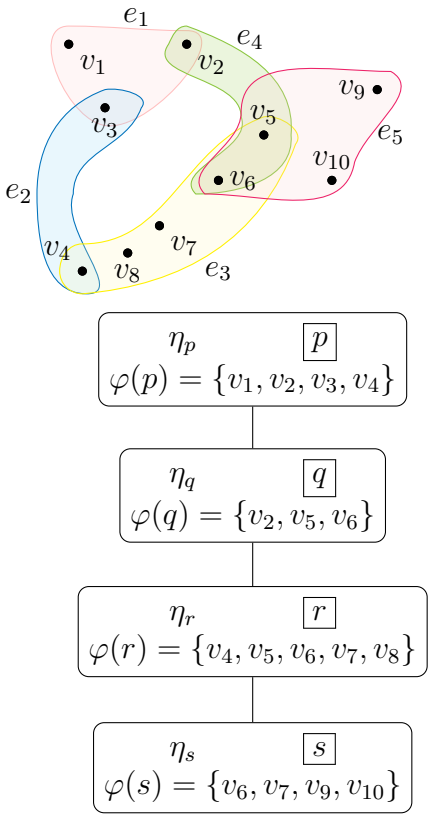


Figure 3.4: A hypergraph H and a fractional hypertree decompositions for it.

$$\eta_p : E(H) \longrightarrow [0,1] \qquad \eta_q : E(H) \longrightarrow [0,1]$$

$$\begin{aligned} e_1 &\longrightarrow 1 & e_1 &\longrightarrow 1 \\ e_2 &\longrightarrow \tfrac{1}{2} & e_2 &\longrightarrow \tfrac{4}{5} \\ e_3 &\longrightarrow \tfrac{\pi}{4} & e_3 &\longrightarrow \epsilon \\ e_4 &\longrightarrow \tfrac{3}{8} & e_4 &\longrightarrow 1 - \epsilon \\ e_5 &\longrightarrow 0 & e_5 &\longrightarrow 0.1 \end{aligned}$$

$$\begin{aligned}\eta_r : E(H) &\longrightarrow [0,1] \\ e_1 &\longrightarrow 0 \\ e_2 &\longrightarrow \tfrac{\sqrt{3}}{3} \\ e_3 &\longrightarrow 1 \\ e_4 &\longrightarrow \tfrac{3}{4} \\ e_5 &\longrightarrow 0\end{aligned} \qquad \begin{aligned}\eta_s : E(H) &\longrightarrow [0,1] \\ e_1 &\longrightarrow 0 \\ e_2 &\longrightarrow 0 \\ e_3 &\longrightarrow 1 \\ e_4 &\longrightarrow 0 \\ e_5 &\longrightarrow 1\end{aligned}$$

See that for $p \in V(T)$,

$$\begin{aligned}B(\eta_p) &= \left\{ v \in V(H) | \sum_{e \in E(H), v \in e} \eta_p(e) \geq 1 \right\} \\ &= \{v_1, v_2, v_3, v_4, v_5, v_6\} \supseteq \{v_1, v_2, v_3, v_4\} = \varphi(p).\end{aligned}$$

Also, $\varphi(q) = \{v_2, v_5, v_6\} \subseteq \{v_1, v_2, v_3, v_5, v_6\} = B(\eta_q)$,
$\varphi(r) = \{v_4, v_5, v_6, v_7, v_8\} = \{v_4, v_5, v_6, v_7, v_8\} = B(\eta_r)$, and
$\varphi(s) = \{v_6, v_7, v_9, v_{10}\} \subseteq \{v_4, v_5, v_6, v_7, v_8.v_9, v_{10}\} = B(\eta_q)$.

Let us also calculate the weight for each element of $\{\eta_t\}_{t \in V(T)}$.

$$\begin{aligned}weight(\eta_p) &= \sum_{e \in E(H)} \eta_p(e) \\ &= \eta_p(e_1) + \eta_p(e_2) + \eta_p(e_3) + \eta_p(e_4) + \eta_p(e_5) \\ &= 1 + \frac{1}{2} + \frac{\pi}{4} + \frac{3}{8} + 0 = \frac{15 + 2\pi}{8}.\end{aligned}$$

Likewise, $weight(\eta_q) = 2.9$, $weight(\eta_r) = \dfrac{21 + 4\sqrt{3}}{12}$, and $weight(\eta_s) = 2$. For this reason, the width of such fractional hypretree decomposition is 2.9.

This notion does not deal with the labeling function ψ which gives us edge covers, instead it involves fractional covers of hypergraph vertices. As a matter of fact, fractional hypertree-width is strictly more general than the hypertree width, hence it is possible to find classes of hypergraphs with bounded fractional hypertree width but no necessarily bounded fractional edge cover number or bounded (generalized) hypertree width. By way of illustration, think of the class of graphs having only disjoint edges, the tree-width and hypertree-width of this class is 1, but the fractional edge cover number is unbounded.

The next theorem enunciates a relationship between general hypertree width and fractional hypertree width.

Theorem 3.4. *For every hypergraph H, $ghw(H) = 1$ if and only if $fhw(H) = 1$.*

Proof. Let H be a hypergraph with $fhw(H) = 1$, then there exists a fractional hypertree decomposition $(T, \varphi, \{\eta_t\}_{t \in V(T)})$ where (T, φ) is a tree decomposition of H and $\{\eta_t\}_{t \in V(T)}$ is a family of mappings from $E(H)$ to $[0, 1]$, such that for every $t \in V(T)$ we have $\varphi(t) \subseteq B(\eta_t)$, and $weight(\eta_{t_0}) = 1$ for some t_0 in $V(T)$.

Claim: If $S \subseteq B(\eta_{t_0})$, then $S \subseteq e$ for all $e \in E(H)$ with $\eta_{t_0}(e) > 0$.

Proof: If we assume $S \subseteq B(\eta_{t_0})$, but $S \nsubseteq e_0$ for some $e_0 \in E(H)$ with $\eta_{t_0}(e) > 0$, then there is a node $v \in S \subseteq B(\eta_{t_0})$ such that $v \notin e_0$ and $\eta_{t_0}(e_0) > 0$. In other words, $v \in V(H)$

and is such that $\sum_{e \in E(H)\setminus\{e_0\}, v \in e} \eta_{t_0}(e) \geq 1$ with $\eta_{t_0}(e_0) > 0$. To put it concisely,

$$\begin{aligned} 1 = weight(\eta_{t_0}) &= \sum_{e \in E(H)} \eta_{t_0}(e) \\ &= \sum_{e \in E(H)\setminus\{e_0\}} \eta_{t_0}(e) + \eta_{t_0}(e_0) \\ &\geq \sum_{e \in E(H)\setminus\{e_0\}, v \in e} \eta_{t_0}(e) + \eta_{t_0}(e_0) \\ &\geq 1 + \eta_{t_0}(e_0) > 1, \end{aligned}$$

and so we have arrived to a contradiction. ■

Distinctively, for the mapping $\eta_{t_0} : E(H) \to [0, 1]$ we have that $\varphi(t_0) \subseteq B(\eta_{t_0})$, as a direct consequence of the claim it follows that $\varphi(t_0) \subseteq e$ for each $e \in E(H)$ with $\eta_{t_0}(e) > 0$. Thereafter, η_{t_0} can be seen as a labeling function which provides us with an edge cover of $V(H)$ and thus (T, φ, η_{t_0}) is a generalized hypertree decomposition and its corresponding generalized hypertree width is 1. For the sufficiency, it is enough to point out that a generalized hypertree decompsition can be seen as a triple $(T, \varphi, \{\eta_t\}_{t \in V(T)})$ where (T, φ) is a tree decomposition and $\{\eta_t\}_{t \in V(T)}$ is a collection of functions from $E(H)$ to $\{0, 1\}$ with $\varphi(t) \subseteq B(\eta_t)$.

□

In [33], Marx introduces a yet more general width concept called submodular width.

Let $\langle T, \varphi \rangle$ be a tree decomposition for a hypergraph H and let $f : \wp(V(H)) \longrightarrow [0, \infty)$ be a function. The *f-width* of $\langle T, \varphi \rangle$ is $\max\{f(\varphi(t)) | t \in T\}$. The *f-width* of H is the minimum of the f-widths of all its tree decompositions.

Example 3.4. We have previously defined tree width and fractional width of a hypergraph H, we may now rewrite such concepts using this setup.

1. Let $r : \wp(V(H)) \longrightarrow [0, \infty)$ be defined by $r(A) = |A| - 1$, then $tw(H) := r\text{-}width(H)$.

2. The fractional hypertree width of H is

$$fhw(H) := \rho^*_H\text{-}width(H),$$

where ρ^*_H is the fractional edge cover number of H.

Let H be a hypergraph and $\mathcal{F}$ be an arbitrary (possibly infinite) class of functions which goes for from $\wp(V(H)) \setminus \{\emptyset\}$ to the set of nonnegative real numbers, the $\mathcal{F}$-*width* of H is

$$\mathcal{F}\text{-}width(H) := \sup\{f\text{-}width(H) | f \in \mathcal{F}\}.$$

As a result, if $\mathcal{F}$-$width(H)$ is bounded by a constant k, then for every $f \in \mathcal{F}$, H has a tree decomposition with f-*width* at most k. Notice that this tree decomposition can be different for the different functions f. For normalization purposes, we consider only functions f on $V(H)$ that satisfy two conditions:

1. $f(\emptyset) = 0$ and
2. f is *edge-dominated,* that is, $f(e) \leq 1$ for every $e \in E(H)$.

Definition 3.8. A function $f : \wp(V(H)) \longrightarrow [0, \infty)$ is *submodular* if for every pair of subsets, X and Y, of $V(H)$ it holds that $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$.

Example 3.5. Let X be a set, $A \subseteq X$, and $\{c_i\}_{i \in X}$ a collection of finite sets. For $A \subseteq X$ let $C_A = \bigcup_{i \in A} c_i$, then the function defined by $f(A) = |C_A|$, where $|\cdot|$ symbolizes the cardinality of a set, is submodular.

Definition 3.9. The *submodular width* of a hypergraph H is

$$subw(H) := \mathcal{F}\text{-}width(H),$$

where $\mathcal{F}$ contains every edge-dominated monotone submodular function f on $V(H)$ with $f(\emptyset) = 0$.

The following result [33] affirms that an edge-dominated submodular function is always bounded by the fractional cover number.

Theorem 3.5. *Let H be a hypergraph and f be a monotone edge-dominated submodular function with $f(\emptyset) = 0$. Then $f(X) \leq \rho^*_H(X)$for every $X \subseteq V(H)$.*

As a direct result of this fact, we get that the submodular width of a hypergraph is at most its fractional hypertree width.

Theorem 3.6. *The inequality* $subw(H) \leq fhw(H)$ *holds for every hypergraph* H.

Proof. Let$\langle T, \varphi \rangle$ be a tree decomposition of H such that its ρ^*_H-width is $fhw(H)$. If f is an edge-bounded monotone submodular function with $f(\emptyset) = 0$, then by virtue of the former proposition, $f(\varphi(t)) \leq \rho^*_H(\varphi(t)) \leq fhw(H)$ for every bag $\varphi(t)$ of the decomposition, which means that $f\text{-}width(H) \leq fhw(H)$. Since this is true for every such function f, it follows that $subw(H) \leq fhw(H)$. □

The following diagram illustrates the relationship among the hypegraph decompositions we have recently discussed.

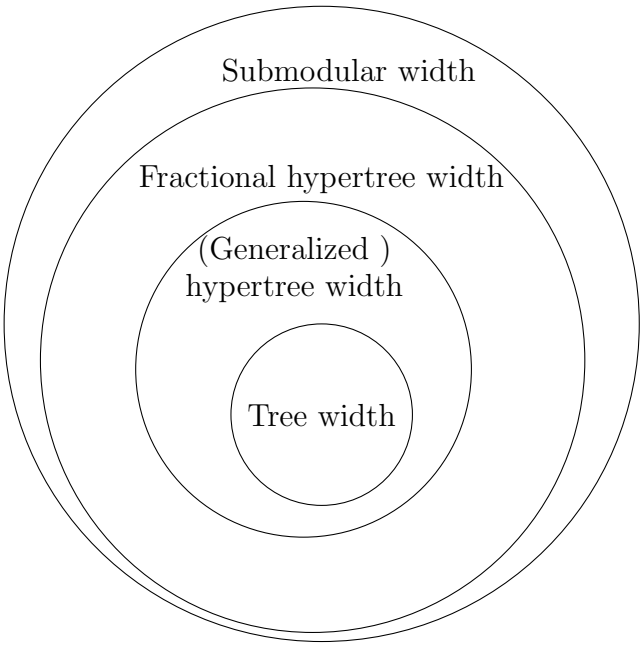


Figure 3.5: Relationship among some hypergraph decompositions.

The concept of branch-width also applies to hypergraphs. As a matter of fact, the definition of branch width given for graphs can be immediately applied to hypergraphs [29], except that the connectivity value of a hyperedge partition is not its number of incident vertices, but the number of hyperedges required to cover its incident vertices. This concept is referred to as *hyperbranch width*, *hbw* for short.
The following theorem [1] tells us that bounded hyperbranch width is equivalent to bounded generalized hypertree width.

Theorem 3.7. *(a) For any hypergraph H it holds that $hbw(H) \leq ghw(H)$.*

(b) For any non-trivial hypergraph H (meaning its edges are not pairwise disjoint), then $ghw(H) \leq 2hbw(H)$.

Of course, the width notions related to hypergraphs that are mentioned here are only a few. Sources like [22, 26, 33] contain information about other types of widths on hypergraphs, including early width parameters, namely *biconnected* width, *cutset* width, $tree - clustering$ width and, $hinge - width$. Moreover, it is reported in [1] that up to a constant factor, hypertreewidth is equilatent to various other hypergraph invariants that are equivalent to graph invariants such as the branch width of a graph, the *bramble number,* and the number of cops required to win the *robber and cops game* (defined by Seymour and Thomas), among others.

Chapter 4

Computational Efficiency

In various areas of Computer science, there are problems whose essential structure can be best described as a hypergraph. Principally, hypergraph decomposition methods provide techniques for solving efficiently large class of instances.

4.1 Efficiency Results for Constraint Satisfaction Problems

As a matter of fact, the structure of a constraint satisfaction problem (*CSP*) instance can be represented by its associated hypergraph, so far, acyclicity is the most basic and most fundamental structural property regarding CSPs. Many important problems can be formulated as CSPs, as it happens, the

constraint satisfaction problem is equivalent to the problem of evaluating Boolean conjunctive queries which are of great importance in database theory and in AI. In general, solving CSPs is *NP-complete,* as a consequence, several researchers have dedicated to studying methods for decomposing cyclic CSPs into acyclic CSPs.

Definition 4.1. An *instance* of a constraint satisfaction problem is a triple (V, U, C), where:

- V is a set of variables,
- U is a domain of values,
- C is a set of constraints, $\{c_1, c_2, \ldots, c_q\}$. Each constraint $c_i \in C$ is a pair $\langle S_i, r_i \rangle$, where S_i is a tuple of variables of length k_i, called the *constraint scope*, and r_i is an k_i-ary relation over U, called the *constraint relation.*

In the first chapter, CSPs were defined in terms of boolean formulas so, in that case, $U = \{0, 1\}$. Also, according to that definition, each pair $\langle S_i, r_i \rangle$ is nothing else than a constraint application and k_i is the arity of the constraint.

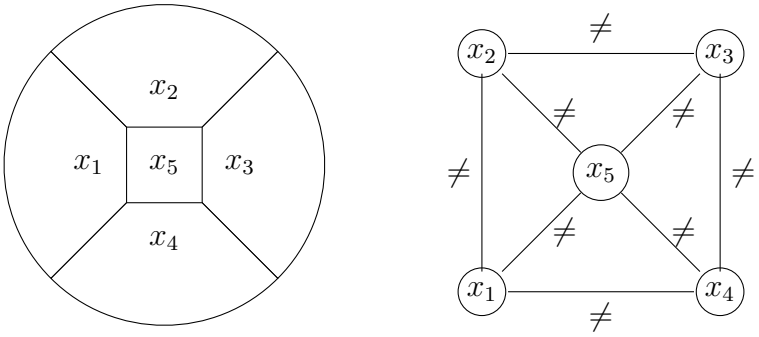
Definition 4.2. A *solution* to a CPS instance is a function $f : V \longrightarrow U$ such that for each $i \in \{1, 2, 3, \ldots, q\}$, the tuple $\langle f(v_{i_1}), f(v_{i_2}), f(v_{i_3}), \ldots, f(v_{i_k}) \rangle \in r_i$.

The hypergraph of a CSP instance I is a hypergraph on the variables of I such that for each constraint there is an edge in

the hypergraph containing exactly the variables that appear in the constraint.

Example 4.1. 1. In the case of figure 4.1 (a) , we have a map of five regions to be colored with green, yellow or blue in such a way that every two neighboring regions have different colors. We may define this as a CSP instance $I = (V, U, C)$ as follows:

- $V = \{x_1, x_2, x_3, x_4, x_5\}$,
- $U = \{g, y, b\}$, where $g = green$, $y = yellow$, $b = blue$ and,
- $C = \{\langle(x_1, x_2), \neq\rangle, \langle(x_1, x_4), \neq\rangle, \langle(x_1, x_5), \neq\rangle, \langle(x_2, x_3), \neq\rangle, \langle(x_2, x_5), \neq\rangle, \langle(x_3, x_4), \neq\rangle, \langle(x_3, x_5), \neq\rangle, \langle(x_4, x_5), \neq\rangle\}$.



(a) A map to be colored. **(b)** Graph associated to I.

Figure 4.1: Map coloring problem.

For this CSP instance we have a set of binary constraints that can be represented by a graph, where the nodes represent the variables and the edges represent

the constraints between thems. Figure 4.1 (b) portrays the graph corresponding to the instance I. Notice that the variables corresponding to adjacent regions are connected by an edge and there are eight constraints in the problem, that is, eight edges in the graph. A solution for I is the assignment $f : V \longrightarrow U$ defined as $f(x_1) = g, f(x_2) = b, f(x_3) = g, f(x_4) = b$ and, $f(x_5) = y$.

2. Figure 4.2 , shows an example of a hypergraph for a non-binary CSP instance with tree constraints: r_1, r_2 and r_3: the first one on the set of variables $\{v_1, v_2, v_3\}$; the second one on the set $\{v_3, v_4, v_5\}$ and; the last one on $\{v_1, v_5, v_6\}$.

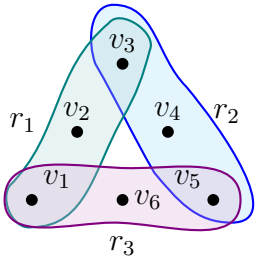


Figure 4.2: A hypergraph for a non-binary CSP instance.

For practical purposes, let $\|A\|$ stand for the size of a data structure A (a specialized format for organizing, processing, retrieving and storing data), and let H_I denote the hypergraph of the CSP instance I. Let us mention that since the following results are taken from different sources, $\|A\|$ is not

necessarily defined exactly the same in all of them, however, the difference among them is only a polynomial factor.

Theorem 4.1. *[29] For a CSP instance I and a generalized hypertree decomposition D of width k of H_I, it takes time*

$$O(\| I \|^{k+1} log(\| I \|) + \| D \|)$$

to check whether I is solvable, and to compute a solution if so.

To prove this theorem, we convert D to a normal form so as to prevent superfluity, and then transform instance I in time $\| I \|^k$ into a CSP instance I' whose associated hypergraph is acyclic and then solve the acyclic instance.

Let H be a hypergraph and k be a fixed constant greater or equal to 0, we know from [25] that there exists a polynomial time algorithm for deciding whether $hw(H) \leq k$ and, if $hw(H) = k$, then a hypertree decomposition of width k can be computed in polynomial time . However, for $k \geq 2$ checking whether $ghw(H) \leq k$ is NP complete and, it has recently been proven in [23], that checking whether $fhw(H) \leq k$ also belongs to the NP-complete complexity class.

For a class of hypergraphs $\mathcal{H}$ we consider $\mathbf{csp}(\mathcal{H})$ as the following decision problem:

$\mathbf{csp}(\mathcal{H})$
Input: A CSP instance with $H_I \in \mathcal{H}$
Output: YES, if I has a solution; NO, if it does not.

An important question is for which classes $\mathcal{H}$ of hypergraphs the problem $\mathbf{csp}(\mathcal{H})$ is tractable. According to the following results, a class $\mathcal{H}$ which makes $\mathbf{csp}(\mathcal{H})$ tractable is that whose elements have bounded fractional edge cover number.

Theorem 4.2. *[28] A CSP instance I has at most $\| I \|^{\rho^*(H_I)}$ solutions.*

Theorem 4.3. *[28] The solutions of a CSP instance I can be enumerated in time $\| I \|^{\rho^*(H_I)+O(1)}$.*

Corollary 4.1. *[28] Let $\mathcal{H}$ be a class of hypergraphs of bounded fractional edge cover number. Then $\mathbf{csp}(\mathcal{H})$ is in polynomial time.*

If $\mathcal{H}$ is a class of hypergraphs having bounded fhw and we want to prove that $\mathbf{csp}(\mathcal{H})$ can be solved in polynomial time, we might need a polynomial algorithm that computes at least an approximately optimal fractional hypertree decompositions for hypergraphs whose fhw is at most a fixed constant k. However, if we are only interested in the parameterized complexity of our CSP, we do not need a decomposition algorithm. Recall our definition of a fixed parameter tractable problem, it suggests that if the parameter is small, then the dependence of the running time on the parameter can be disregarded.

Let us state the parameterization of $\mathbf{csp}(\mathcal{H})$:

p-csp$(\mathcal{H})$
Input: A CSP instance with $H_I \in \mathcal{H}$

Parameter: the number of variables of I.
Output: YES, if I has a solution; NO, if it does not.

Theorem 4.4. *[28] Let $r \geq 1$. Then there exists a polynomial-time algorithm that, given a CSP instance I and a fractional hypertree decomposition D of H_I of width at most r, decides whether I is satisfiable.*

Corollary 4.2. *[28] If $\mathcal{H}$ is class of hypergraphs of bounded fractional hypertree width, then **p-csp**$(\mathcal{H})$ is fixed parameter tractable.*

Besides providing methods for proving problems to be fixed parameter tractable, parameterized complexity theory also gives a framework for dealing with apparently intractable problems. In the midst of the great variety of classes of parameterized intractable problems, $W[1], W[2], W[3], \ldots$ of the W-hierarchy, can be highlighted as the most important of these classes and they are such that $FPT \subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$. These classes were originally defined via the weighted satisfiability problem for Boolean circuits. The complexity classFPT is a generalization of P as FPT also contains parameterized versions of problems that are probably not in P. The class $W[1]$ can be thought of as a parameterized counterpart to NP, even so, $W[1]$ is not known or believed to contain all parameterized versions of problems in NP. The central conjecture of parameterized complexity theory states that $FPT \neq W[1]$, assuming such hypothesis the following theorem holds.

Theorem 4.5. *[33] If $\mathcal{H}$ is a recursively enumerable class (there is an algorithm that enumerates the members of it) of hypergraphs with bounded edge size, then the following are equivalent:*

1. *$\mathcal{H}$ has bounded tree-width*
2. ***csp**($\mathcal{H}$) is solvable in polynomial time.*
3. ***csp**($\mathcal{H}$) is fixed-parameter tractable.*

As already remarked, submodular width is a strictly more general property than bounded fractional hypertree width, the following theorem states that bounded submodular width of $\mathcal{H}$ implies that $\mathbf{csp}(\mathcal{H})$ is fixed parameter tractable.

Theorem 4.6. *[33] Let $\mathcal{H}$ be a class of hypergraphs such that $subw(H) \leq k$ for every $H \in \mathcal{H}$. Then **csp**($\mathcal{H}$) can be solved in time $2^{k \cdot 2^{O(|V(H)|)}} \cdot \|I\|^{O(k)}$.*

4.2 Open problems

In a 2006 paper, Grohe and Marx stated as an open question whether there exist a polynomial-time algorithm that determines, or approximates, the fractional hypertree width and constructs a corresponding decomposition. In 2010, the approximation problem was solved: is was proved that for every $k \geq 1$, and a given hypergraph H having fractional hypertree width at most k, there is a polynomial-time algorithm

that computes a fractional hypertree decomposition of H of width $O(k^3)$ [32]. More recently, results by Gottlob et al. [23] guarantee that, for any fixed constant $k \geq 2$, deciding $fhw(H) \leq k$ is an NP-complete problem.

Let us present a list of relevant problems related to the topic which, to this day, remain unsolved.

• Can the inequality in Theorem 3.2, $hw(H) \leq 3 \cdot ghw(H) + 1$ be improved? [1]

• Are there polynomyal time solvable families of CSP instances having unbounded fractional hypertree width? [28]

• Find new applications that profit from the different types of hypergraph decompositions.

• As stated in [33], it is known that unbounded submodular width is fixed parameter untractable and hence it is not polynomial time solvable. Additionally, bounded fractional hypertree width implies polynomial time solvability. An immediate question is, what happens with the class of hypergraphs whose submodular width is bounded but fractional hypertree width is unbounded? The author mentions four possibilities, however, so far none of them has been proved nor rejected.

Conclusions

The width notions presented in this thesis, as well as those that were not mentioned, are worth studying. Hypertree decomposition was once considered the most general known hypegraph decomposing method leading to large tractable classes of important problems until fractional hypertree decomposition appeared and subsequently the notion of submodular width. We can always go further, researching on methods that help us to solve larger classes of problems. There is a long way to go on this subject, as more invariants are introduced, more questions arise.

Bibliography

[1] Adler, I., Gottlob, G., and Grohe, M. Hypertree width and related hypergraph invariants. *European Journal of Combinatorics 28*, 8 (2007), 2167–2181. 39, 51, 61

[2] Arnborg, S., Lagergren, J., and Seese, D. Easy problems for tree-decomposable graphs. *Journal of Algorithms 12*, 2 (1991), 308–340. xv

[3] Balcázar, J. L., Díaz, J., and Gabarró, J. *Structural Complexity*, 2nd ed. Springer, Verlag Berlin Heidelberg, 1995.

[4] Berge, C. *Graphs and Hypergraphs*, 2 ed. North-Holland mathematical library, volume 6. American Elsevier Publishing Company; Revised edition, 1973. 26

[5] Berge, C. *Hypergraphs*, 2nd ed. Elsevier Science, North-Holland, Amsterdam, 1989.

[6] Bodlaender, H. L. A tourist guide through treewidth. *Acta cybernetica 11*, 1-2 (1994), 1. xv

[7] Bovet, D. P., and Crescenzi, P. *Introduction to the Theory of Complexity*. Prentice Hall, 1994.

[8] Brault-Baron, J. Hypergraph acyclicity revisited. *ACM Computing Surveys (CSUR) 49*, 3 (Dec. 2016), 54:1–54:26.

[9] Bretto, A. *Hypergraph theory*. Springer, 2013. 26, 28

[10] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing* (1971), pp. 151–158. xiv

[11] Courcelle, B. The monadic second-order logic of graphs iii: Tree-decompositions, minors and complexity issues. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications 26*, 3 (1992), 257–286. xv, 19

[12] Courcelle, B. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations*. World Scientific, 1997, pp. 313–400. xv

[13] Courcelle, B., and Engelfriet, J. *Graph structure and monadic second-order logic: a language-theoretic approach*, vol. 138. Cambridge University Press, 2012. xv

[14] Creignou, N., Khanna, S., and Sudan, M. *Complexity classifications of Boolean constraint satisfaction problems*. SIAM, 2001.

[15] Diestel, R. *Graph Theory*, 5th ed. Springer, 2017. 19

[16] Downey, R. G., and Fellows, M. R. *Fundamentals of parameterized complexity*, vol. 4. Springer, 2013. 10

[17] Duris, D. Some characterizations of gamma and beta-acyclicity of hypergraphs. *Information Processing Letters 112*, 16 (2012), 617–620.

[18] Fagin, R. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the Association for Computing Machinery 30*, 3 (1983), 514–550. 30

[19] Gallier, J. H. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. eBook (Revised, 2003).

[20] Gottlob, G., Greco, G., Leone, N., and Scarcello, F. Hypertree decompositions: Questions and answers. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (2016), pp. 57–74. 33

[21] Gottlob, G., Greco, G., and Scarcello, F. Treewidth and hypertree width. *Tractability: Practical Approaches to Hard Problems 1* (2014). 20

[22] Gottlob, G., Grohe, M., Musliu, N., Samer, M., and Scarcello, F. Hypertree decompositions: Structure, algorithms, and applications. In *International Workshop on Graph-Theoretic Concepts in Computer Science* (2005), Springer, pp. 1–15. xv, xvi, 51

[23] Gottlob, G., Lanzinger, M., Pichler, R., and Razgon, I. Complexity analysis of general and fractional hypertree decompositions. *arXiv preprint arXiv:2002.05239* (2020). xvi, 43, 57, 61

[24] Gottlob, G., Leone, N., and Scarcello, F. Hypertree decompositions: A survey. In *International Symposium on Mathematical Foundations of Computer Science* (2001), Springer, pp. 37–57.

[25] Gottlob, G., Leone, N., and Scarcello, F. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences 64* (May 2002), 579–627. 34, 40, 57

[26] Gottlob, G., and Pichler, R. Hypergraphs in Model Checking: Acyclicity and Hypertree-Width versus Clique-Width. *SIAM Journal on Computing 33*, 2 (2004), 351–378. 51

[27] Gottlob, G., and Samer, M. A backtracking-based algorithm for computing hypertree-decompositions. *Journal of Experimental Algorithmics (JEA) 13*, 1 (2008), 1:1–1:19.

[28] Grohe, M., and Marx, D. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms (TALG) 11*, 1 (2014), 1–20. xvi, 42, 58, 59, 61

[29] Hliněný, P., Oum, S.-i., Seese, D., and Gottlob, G. Width parameters beyond tree-width and their applications. *The computer journal 51*, 3 (2008), 326–362. 24, 51, 57

[30] (https://cstheory.stackexchange.com/users/28818/paul seymour), P. S. The origin of the notion of treewidth. Theoretical Computer Science Stack Exchange. URL:https://cstheory.stackexchange.com/q/27317 (version: 2014-11-03). 19

[31] Maheshwari, A., and Zeh, N. I/o-efficient algorithms for graphs of bounded treewidth. *Algorithmica 54*, 3 (2009), 413–469. xv

[32] Marx, D. Approximating fractional hypertree width. *ACM Transactions on Algorithms (TALG) 6*, 2 (2010), 1–17. 61

[33] Marx, D. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *Journal of the ACM (JACM) 60*, 6 (2013), 1–51. xvi, 47, 49, 51, 60, 61

[34] O'Donnell, R. Lecture 17: Treewidth, November 2013. 21

[35] Valiant, L. G. The complexity of computing the permanent. *Theoretical computer science 8*, 2 (1979), 189–201. xiv

[36] Wei, F. Efficient graph reachability query answering using tree decomposition. In *International Workshop on Reachability Problems* (2010), Springer, pp. 183–197. xv