BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
Facultad de Ciencias Físico Matemáticas

A Tractable Syntactic Class

Tesis presentada para obtener el título de

Licenciada en Matemáticas

Presenta:

Angeles Carranza Cisneros

Director de Tesis:
Dr. Carlos Guillén Galván

Julio 2018

To my family and friends, with love.

Acknowledgments

First and foremost, I would like to thank God for his blessings, for giving me the strength and perseverance that were necessary to complete this stage of my life.

I would like to express my sincere gratitude to my thesis director Dr. Carlos Guillén Galván for the continuous support, for his patience and motivation. His guidance has been extremely valuable throughout this time.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Carlos Alberto López Andrade, Dr. César Bautista Ramos, and M.C. Marisol Mares Javier, for their insightful comments.

Last but not the least, I would like to thank my family and friends for supporting me spiritually throughout writing this thesis and my life in general. I place on record, my sincere thanks to one and all who, directly or indirectly, have helped me in this venture.

Introduction

Mathematicians and computer scientist are aware of the relation between boolean formulas and computational complexity. Certainly, one of the most crucial unsolved problems in mathematical logic and theoretical computer science is the P vs NP problem as it has far-reaching consequences to other problems in mathematics, artificial intelligence, game theory, philosophy, economics, and many other fields. There is even a Clay Millennium Prize offering one million dollars for its solution. Basically such problem asks whether or not, for all problems for which an algorithm can verify a given solution quickly (that is, in polynomial time), an algorithm can also find that solution quickly. This is equivalent to ask whether all problems in NP are also in P.

A common example of an NP problem not known to be in P is the Boolean satisfiability problem, also called the SAT problem, which refers to the satisfiability of propositional logic formulas in CNF form. The representation of formulas in its CNF form is extremely important not only because it exacts essential information but also because it deletes superfluous data. Every decision problem in the complexity class NP can be reduced to the SAT problem, hence solving the question whether SAT has a polynomial time algorithm is equivalent to the P versus NP problem.

Given the SAT problem NP one can formulate the corresponding counting problem, asking how many solutions does a propositional formula have. The #SAT problem counts the satisfying assignments of a given CNF formula and it is extremely harder than SAT. One approach for the solution of this counting problem focuses on structural restrictions of the input formula. The idea behind this is to solve #SAT faster on formulas where interaction between the clauses and the variables is restricted. This is done by assigning a hypergraph (the generalization of graph) to the input CNF formula. From this perspective the complexity of

#SAT is then studied on CNF formulas whose associated hypergraph belongs to a restricted class of hypergraphs.

Among the various syntactic classes, we distinguish $2\mu - e3MON$, the class of monotone CNF formulas with clauses having exactly three literals and such that each variable occurs at most twice. It should be mentioned that until now it is not known where the $2\mu - e3MON$ class is located. In this thesis we present a syntactic subclass within $2\mu - e3MON$ for which we obtain results that lead directly to efficient algorithms that compute the number of models of formulas belonging to such subclass. The hypergraphs associated with such formulas allow disjoint branches decomposition, for which #SAT is tractable [CDM14].

The organization of the thesis is as follows. Chapter 1 contains a review of basic concepts on graph theory and boolean expressions. Important definitions are stated and some relevant examples are presented.

Next, in Chapter 2, a superficial description of complexity theory and syntactic classes is explored. Also, hypergraphs are introduced and examples of notions related to hypertree decomposition are detailed.

Unlike for graphs, there are various degrees of cyclicity for hypergraphs. As a direct consequence, there are several reasonable ways of defining acyclicity for hypergraphs, not to mention equivalent concepts for each degree of acyclicity. In Chapter 3 details on this topic are given. In order to properly present the counting algorithms for certain CNF formulas located in the $2\mu - e3MON$ syntactic class, the associated hypergraph representation is illustrated in this chapter.

Lastly, in Chapter 4, the definition of single chain, alternating chain, simple cycle, and alternating cycle is given. Then, matrix operators acting over these structures are presented in order to obtain efficient algorithms that perform the model counting on the identified family. Finally, the conclusions are stated.

Content

List of Figures

Symbols

$\mathbb{Z}$	Set of integers.
$\mathbb{N}$	Natural numbers set.
G	Graph.
V	Set of vertices.
E	Set of edges.
K^n	Complete graph with n nodes.
$\|G\|$	Number of vertices of G.
P	Path.
$d(v)$	degree of vertex v.
$\delta(G)$	Minimum degree of graph G.
$\Delta(G)$	Maximum degree of graph G.
Σ	Alphabet.
Σ^n	Set of all words over and alphabet Σ with length n.
φ	Boolean formula.
CNF	Conjunctive normal form.
$Var(\varphi)$	Set of variables of φ.
$Lit(\varphi)$	Set of literals of φ.
$M(\varphi)$	Set of models of φ.
$\|M(\varphi)\|$	Cardinality of $M(\varphi)$.
H	Hypergraph.
$I(H)$	Incidence graph of hypergraph H.
$\mathcal{P}(A)$	Power set of set A.
T	Tree.
R	Root of a tree.
T_p	Tree T rooted at p.
H_φ	Hypergraph associated with the formula φ.
r	Restriction over variables of a formula.

$\mathbb{S}$	Edge operator.
$\mathbb{D}$	Double-edge operator.

Chapter 1

Preliminaries

This chapter focuses on the basic concepts of graph theory and boolean formulae which are necessary for the develpment of the thesis. The development of the first section is based on [Die10].

1.1 Basic Concepts of Graph Theory

The symbols $\mathbb{Z}$ and $\mathbb{N}$ denote the set of integers and non-negative integers respectively. The set of all subsets of A having k elements is denoted by $[A]^k$. For example, if $A = \{a, b, c\}$, then $[A]^0 = \{\emptyset\}$, $[A]^1 = \{\{a\}, \{b\}, \{c\}\}$, $[A]^2 = \{\{a, b\}, \{a, c\}, \{b, c\}\}$, $[A]^3 = \{A\}$ and $[A]^k = \emptyset$ for all $k \in \mathbb{Z} \setminus \{1, 2, 3\}$.

Definition 1.1. A *simple graph* G consists of a pair of sets (V, E), where V is called the set of *vertices* (or *nodes*) and $E \subset [V]^2$ is the set of *edges*. The symbols $V(G)$ and $E(G)$ are used to represent the set of nodes and the set of edges of G, respectively.

For instance, in Figure 1.1, G is a simple graph where $V(G) = \{v_1, v_2, v_3, v_4, v_5\}$ and $E(G) = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}, \{v_3, v_5\}\}$.

If $V(G) = \emptyset$, then $E(G) = \emptyset$, thereby $G = \emptyset$ is called the *empty graph*.

Definition 1.2. The number of elements of $V(G)$ is called the *order* of the simple graph G and it is denoted by $\mid G \mid$. The number of edges is called the *size* of G and is denoted by $\parallel G \parallel$.

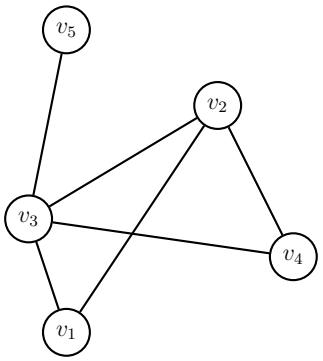


Figure 1.1: Simple graph $G = (V, E)$.

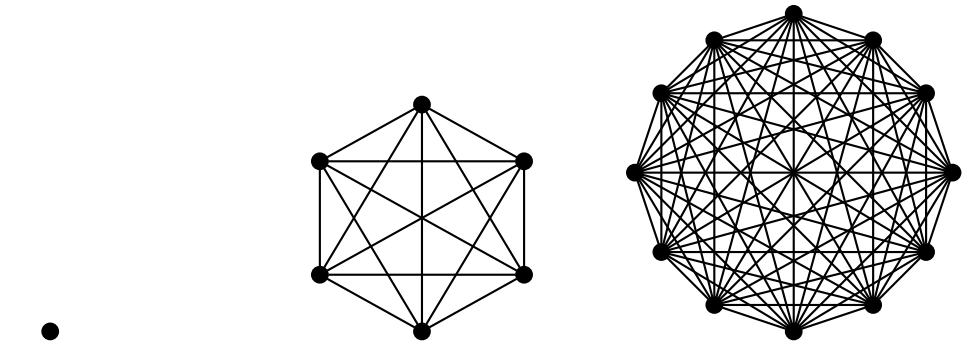
A graph of order 0 or 1 is called a *trivial*.

Definition 1.3. Let G be a simple graph, $v \in V(G)$, and $e \in E(G)$. If vertex v is in edge e, then vertex v is said to be *incident* with e or that e is an edge at v.

Given a simple graph G whose sets of vertices and edges are V and E, respectively, if two nodes v and w belong to $V(G)$, with $vw \in E(G)$ it is meant that $\{v, w\}$ is an edge of G; v and w are called the *ends* of such edge.

Definition 1.4. Vertices v and w of a graph G are said to be *neighbors* or *adjacent* nodes if $vw \in E(G)$. Similarly, two edges are adjacent if they share a vertex.

Let G be a simple graph and $v \in V(G)$. The set of all vertices in $V(G)$ adjacent to vertex v is symbolized by $N(v)$. If every pair of vertices of G are adjacent, then G is *complete*. The symbol K^n stands for a complete graph of n vertices.



(a) K^1 **(b)** K^6 **(c)** K^{12}

Figure 1.2: Complete graphs.

In any simple graph there is at most one edge joining a given pair of vertices. However, many results that hold for simple graphs can be extended to more general objects in which two vertices may have several edges joining them. Actually, the

restriction that an edge always joins two distinct vertices can be modified, allowing the existence of *loops*, edges whose two ends are the same vertex. The resulting object, in which loops and multiple edges are allowed, is formally described as follows.

Definition 1.5. A *multigraph* G consists of a vertex set V, an edge set E and a correspondence $\psi : E \to V \cup [V]^2$ which assigns to every edge either one or two vertices.

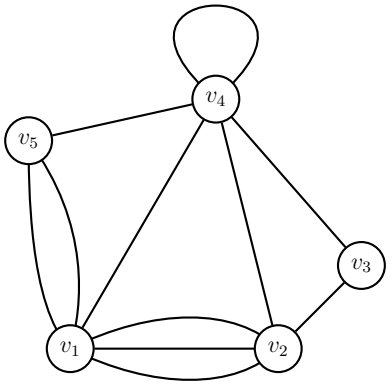


Figure 1.3: A multigraph G.

Thus every simple graph is a multigraph, but not every multigraph is a simple graph.

Definition 1.6. A pair (V, E) of disjoint sets, vertices and edges, together with the mappings $in : E \to V$ and $ter : E \to V$, is called a *directed graph*, or simply a *digraph*. These mappings assign to every edge $e \in E$ an *initial* vertex $in(e)$ and a *terminal* vertex $ter(e)$.

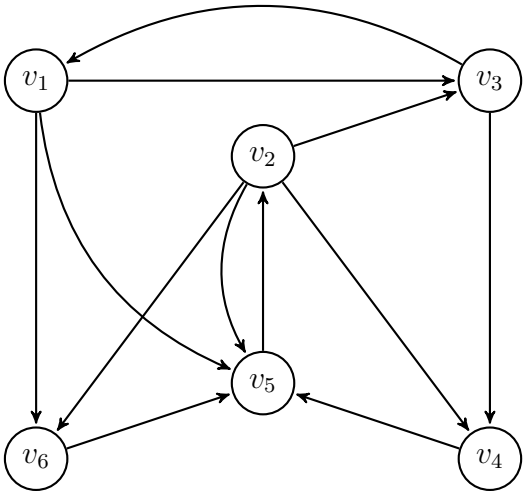


Figure 1.4: A digraph.

The aforementioned definition indicates that edges of a directed graph have a direction associated with them and allows to have more than one edge between the same two vertices. Loops are also permitted since it the initial node and terminal node might be the same. A directed graph can represent asymmetrical relationships between nodes, while an undirected graph, in which edges have no orientation, can represent only symmetrical relationships.

Definition 1.7. Let G_1 and G_2 be two undirected graphs. A function $f : V(G_1) \longrightarrow V(G_2)$ is called a *graph isomorphism* if it satisfies the following conditions:

(i) f is a bijective function.

(ii) For all u and $v \in V(G_1)$, $uv \in E(G_1)$ if and only if $f(u)f(v) \in E(G_2)$.

Two graphs G_1 and G_2 are *isomorphic* if there exists an isomorphism from one to the other. This is written $G_1 \simeq G_2$. Note that the correspondence of vertices of a graph isomorphism preserves the adjacencies and, thereby, the structure of the graphs is maintained.

Definition 1.8. A graph G is a *bipartite graph*, also called a *bigraph*, if its vertices can be divided into two sets V_1 and V_2 with the following properties:

(i) $V_1 \cap V_2 = \emptyset$.

(ii) $V_1 \cup V_2 = V(G)$.

(iii) If $v \in V_1$ then it may only be adjacent to vertices in V_2.

(iv) If $v \in V_2$ then it may only be adjacent to vertices in V_1.

A different way to view a bipartite graph is by coloring the set of vertices with two different colors. For example, if all vertices of set V_1 are colored blue and all vertices of set V_2 are colored purple, then each edge must connect vertices of different colors (see Figure 1.5). Pay attention to the fact that all edges in a bipartite graph go only between V_1 and V_2, there are no edges from V_1 to V_1 or from V_2 to V_2.

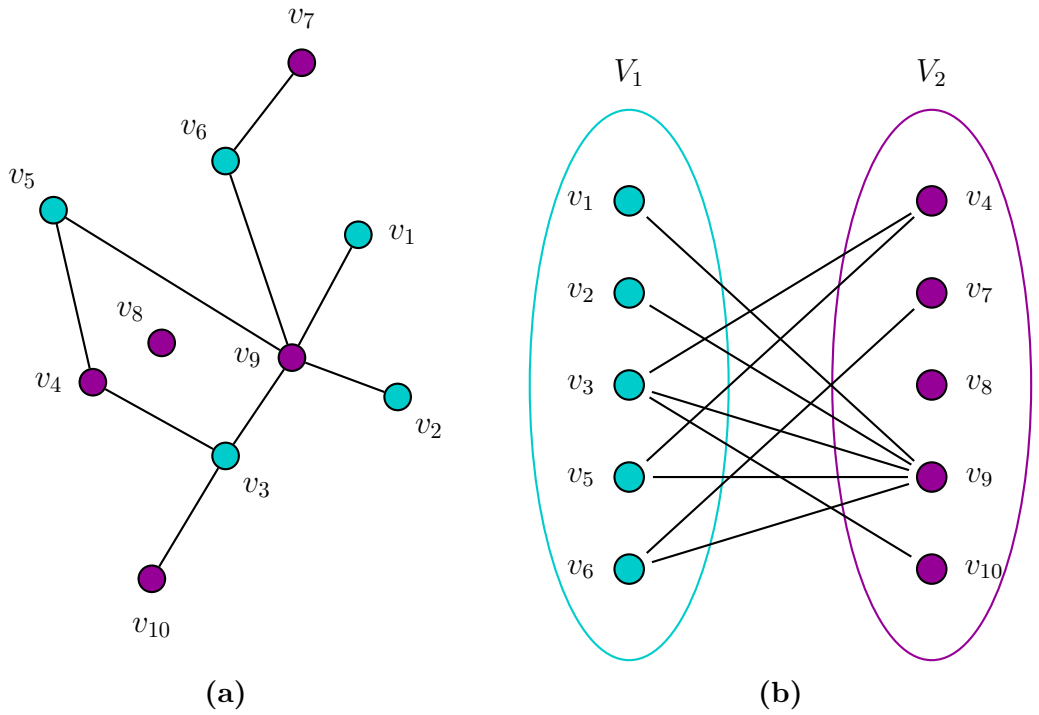


Figure 1.5: Two different ways to represent a bipartite graph.

Definition 1.9. Given an undirected graph G, the cardinality of the set $E(v) = \{e \in E(G) | v \in e\}$ is called the *degree* of vertex v and it is denoted by $d(v)$.

If an edge is a loop, it contributes 2 to the degree of that vertex. Vertex v is *isolated* if $E(v) = \emptyset$, that is if $d(v) = 0$. The possible degrees in a simple graph with n vertices are $0, 1, 2, \ldots, n-1$. Note that no simple graph with n vertices can contain both a vertex of degree 0 and a vertex of degree $n-1$, so in each case there are only $n-1$ possible degrees for n vertices.

When all of its vertices have the same degree, the simple graph G is called *regular*. Under other conditions the minimum degree and maximum degree are defined, respectively, as follows

$$\delta(G) := min\{d(v) | v \in V(G)\}$$

and

$$\Delta(G) := max\{d(v) | v \in V(G)\}.$$

As a matter of fact, complete graphs of order n are regular of degree n - 1, and empty graphs are regular of degree 0. Two further examples are shown next.

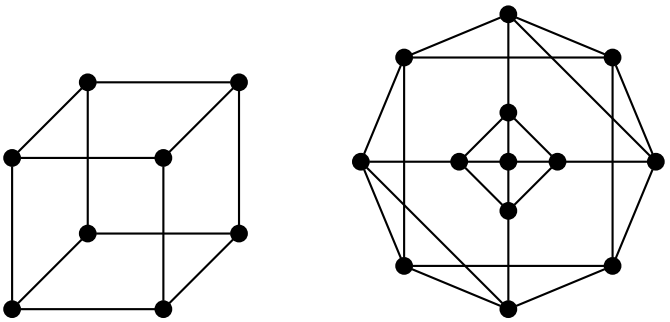


Figure 1.6: Regular graphs.

Definition 1.10. An *independent set* of a simple graph G is a subset $X \subseteq V(G)$ satisfying the following property: if v and w are any two distinct vertices in X, then v and w are not adjacent.

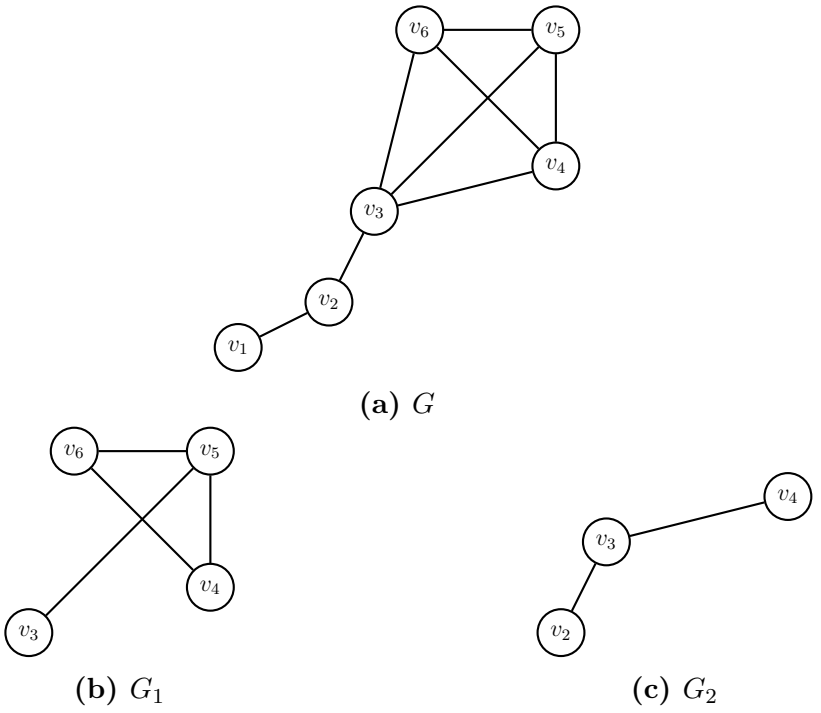
Observe that this condition is trivially satisfied if X contains exactly one vertex since X does not have two distinct vertices in the first place. Hence, every singleton subset $X \subseteq V(G)$ is an independent set of size 1. It follows that the maximum size of an independent set in a complete graph is 1.

Definition 1.11. A graph G' is a *subgraph* of G, written $G' \subseteq G$, if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. If G' consists of all edges of G which have ends in V' ,then G' is called an *induced subgraph* of G and is denoted by $G[V']$. It is also said that V' *induces* G' in G.

So, the construction of an induced subgraph is as simple as removing vertices from $V(G)$ together with all their incident edges, but no more edges. If additional edges are deleted, then G' is still a subgraph of G, but no longer an induced subgraph of G. In particular, the resulting graph after removing only edges but no vertices is not an induced graph.

In Figure 1.7, $V(G_1) = \{v_3, v_4, v_5, v_6\} \subseteq \{v_1, v_2, v_3, v_4, v_5, v_6\} = V(G)$ and $E(G_1) = \{v_4v_5, v_5v_6, v_3v_5, v_4v_6\} \subseteq \{v_1v_2, v_2v_3, v_3v_4, v_4v_5, v_5v_6, v_3v_6, v_3v_5, v_4v_6\} = V(G)$, likewise $V(G_2) = \{v_2, v_3, v_4\} \subseteq V(G)$ and $E(G_2) = \{v_2v_3, v_3v_4 \subseteq E(G)$, therefore $G_1, G_2 \subseteq G$. Moreover, G_2 is an induced subgraph of G.

Definition 1.12. A non-empty simple graph P where $V(P) = \{x_0, x_1, \ldots, x_n\}$, $E(P) = \{x_0x_1, x_1x_2, \ldots, x_{n-1}x_n\}$ and $x_i \neq x_j$ for every $i \neq j$, is called a *path*. Vertices x_0 and x_n are the *ends* of path P and $x_1, x_2, \ldots, x_{n-1}$ are its *inner vertices*. The number of edges in a path is called its *length*. A path of length k is denoted by P^k.



(a) G **(b)** G_1 **(c)** G_2

Figure 1.7: G_1 is a subgraph of G and G_2 is an induced subgraph of G.

Different authors use different terminology, some authors refer to a path as a 'simple' path. It is often said that $P = x_0x_1 \cdots x_n$ is a path from x_0 to x_n or between x_0 and x_n. For example, in Figure 1.7 (a), $P_1 = v_5v_6v_3v_2$ and $P_2 = v_5v_3v_2$ are both paths from v_5 to v_2 whose respective lengths are three and two.

Definition 1.13. A *cycle* or *closed path* is a path which starts and ends at the same vertex.

In general, in simple graphs, it is possible for a path to have length 0, but the least possible length of a cycle is 3.

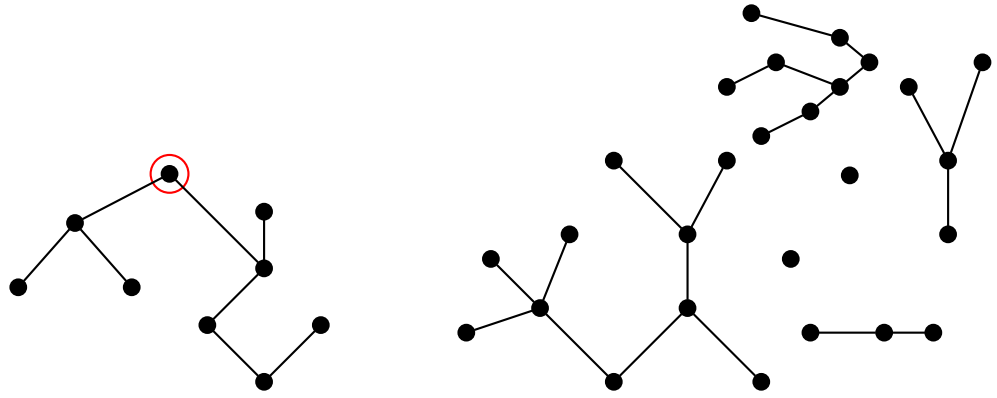
Definition 1.14. Let u, v be two arbitrary vertices of a non-empty graph G. If there is a path from u to v, then G is a *connected* graph. Otherwise it is *disconnected*.

Consider the graphs $G = (V, E)$ and $G' = (V', E')$, set $G \cup G' := (V \cup V', E \cup E')$ and $G \cap G' := (V \cap V', E \cap E')$. Clearly, any disconnected graph G can be expressed as the union of connected graphs.

Definition 1.15. Let G be a graph, a *component* of G is a maximally connected subgraph G' of G.

Components form a partition of the set of vertices of a graph which means that components are non-empty, they are pairwise disjoints, and the union of them forms the set of all vertices of the graph.

Definition 1.16. A connected graph with no cycles is called a *tree*, the disjoint union of them is a *forest*. A tree in which one prominent node have been designated the *root* is a *rooted tree*. Every vertex of degree 1 in a tree is called a *leaf* node.



(a) G_1 **(b)** G_2

Figure 1.8: G_1 is a rooted tree and G_2 is a forest.

In that regard, every connected component in a forest is a tree and a forest with one connected component is a tree. In a tree, there is only one way to get from one node to another. Generally, this is not true in simple graphs as already seen in Figure 1.7 (a). The root of a tree is never considered a leaf, even if it has degree 1.

Let $0 \leq i \leq j \leq n$, the following notations are introduced:

$$
\begin{aligned}
Px_i &:= x_0 \cdots x_i \\
x_iP &:= x_i \cdots x_n \\
x_iPx_j &:= x_i \cdots x_j \\
\mathring{P} &:= x_1 \cdots x_{n-1} \\
P\mathring{x}_i &:= x_0 \cdots x_{i-1} \\
\mathring{x}_iP &:= x_{i+1} \cdots x_n \\
\mathring{x}_iP\mathring{x}_j &:= x_{i+1} \cdots x_{j-1}
\end{aligned}
$$

Definition 1.17. Two *paths* P_1 and P_2 *are independent* if $V(\mathring{P}_1)$ and $V(\mathring{P}_2)$ are disjoint.

Example 1.1. Let G be a graph as shown below. How many paths are there from v_2 to v_4? Are they independent?

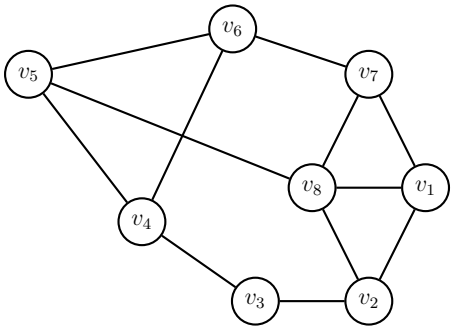


Figure 1.9: $G = (V, E)$.

In this specific problem, all the possible paths between vertices v_2 and v_4 can be easily listed: $P_1 = v_2v_3v_4$, $P_2 = v_2v_8v_5v_4$, $P_3 = v_2v_8v_5v_6v_4$, $P_4 = v_2v_8v_7v_6v_4$, $P_5 = v_2v_8v_7v_6v_5v_4$, $P_6 = v_2v_8v_1v_7v_6v_4$, $P_7 = v_2v_8v_1v_7v_6v_5v_4$, $P_8 = v_2v_1v_7v_6v_4$, $P_9 = v_2v_1v_7v_6v_5v_4$, $P_{10} = v_2v_1v_7v_8v_5v_4$, $P_{11} = v_2v_1v_7v_8v_5v_6v_4$, $P_{12} = v_2v_1v_8v_7v_6v_4$, $P_{13} = v_2v_1v_8v_7v_6v_5v_4$, $P_{14} = v_2v_1v_8v_5v_4$, and $P_{15} = v_2v_1v_8v_5v_6v_4$. Unquestionably, for every $1 < j \leq 15$, P_1 and P_j are independent paths, as well as P_2 and P_8. All the remaining pairs of paths are not independent.

1.2 Boolean Formulae

In this section, some notions and facts on boolean formulas are briefly described.

Definition 1.18. An *alphabet*, say σ, is a nonempty finite set consisting of:

(i) A countable set of symbols or letters, also known as *variables*:
$x_0, x_1, x_2, \ldots$;

(ii) Logical connectives: $\vee, \wedge, \neg, \Rightarrow$, and $\Leftrightarrow$;

(iii) Auxiliary symbols: (,).

A *chain, word* or *string* over σ is a finite sequence of symbols from σ. The set of all words over an alphabet σ is denoted σ^*. Whereas σ^n represents the set of all words over an alphabet σ with length n.

Mappings from $\{0,1\}^n$ to $\{0,1\}$ are called *logical* or *boolean formulas*. If we use a vector notation to represent the input variables of a logical formula denoted by $x_1, x_2, \ldots, x_n$, the output is a function of these variables denoted as $\varphi(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is called an input. Each of the inputs and the output takes one of two possible values which can be denoted by "1 and 0," "yes and no," or "true and false."

The most common logical connectives, previously stated, are defined in terms of their relation to the truth or falsehood of the variable(s) they are operating on. The definition of three logical connectives is presented below.

Definition 1.19. Let $\cdot$ represent the usual product.

(a) *Negation* (NOT, complement, or inversion) of x_1 is a function denoted by $\neg x_1$, also $\overline{x_1}$, such that

$$\neg x_1 = \begin{cases} 1 & \text{if} \quad x_1 = 0 \\ \\ 0 & \text{if} \quad x_1 = 1. \end{cases}$$

(b) A *disjunction* (*OR*, logical sum or union) of x_1 and x_2 denoted by $x_1 \vee x_2$ is a function such that $x_1 \vee x_2 = 1$ if $x_1 = 1$ or $x_2 = 1$.

(c) A *conjunction* (*AND*, logical product or join) of x_1 and x_2 denoted by $x_1 \wedge x_2$ is a function such that $x_1 \wedge x_2 = x_1 \cdot x_2 = x_1 x_2$.

Often, boolean formulas are more complicated than these and require two or more connectives.

Definition 1.20. A *literal* is defined as a variable or its inversion. A conjunction of literals is called a *term*. Similarly, a disjunction of literals is called a *clause*. A *Horn clause* is a clause with at most one unnegated literal. A disjunction of terms is called *disjunctive normal form*, or *DNF* for short. A conjunction of clauses is called a *conjunctive normal form*, or simply *CNF*.

All conjunctions of literals and all disjunctions of literals are in CNF, as they can be seen as conjunctions of one literal clauses and conjunctions of a single clause, respectively. Notice that the only logical connectives that a formula in CNF can contain are $\neg, \vee$, and $\wedge$. In addition, negation can only be used as part of a literal which means that it can only precede a variable. For instance, the following formulas are not in CNF:

$$\neg(x_2 \wedge x_7)$$
$$x_1 \wedge (x_3 \vee (x_2 \wedge x_9).$$

Yet they are respectively equivalent to the following formulas that are in CNF:

$$\neg x_2 \vee \neg x_7$$
$$x_1 \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4).$$

As a matter of fact, every logical formula can be converted into an equivalent formula in conjunctive normal form. This transformation is based on rules about logical equivalences: the double negative law, De Morgan's laws, and the distributive law. Even the connectives $\Rightarrow$ and $\Leftrightarrow$ can be expressed in CNF as follows.

Let p and q be two boolean formulas, then

$$p \Rightarrow q \equiv \neg p \vee q \quad \text{and} \quad p \Leftrightarrow q \equiv (\neg p \vee q) \wedge (p \vee \neg q).$$

In an analogous way, all logical formulas can be converted into an equivalent formula that is in DNF. Nonetheless, in some cases such conversions can lead to an exponential blow up of the formula.

Definition 1.21. If a boolean formula contains only conjunctions and disjunctions as connectives, but no negations, it is a *monotone formula*.

Particularly, a *monotone monomial* is a conjunction of literals with no negations, and a *monotone DNF formula* is a disjunction of monotone monomials. Dually, a *monotone CNF formula* is a conjunction of literals with no negations.

From now on, unless otherwise stated, the word 'formula' will be used as a synonym for 'boolean formula.' The sets of variables and literals of φ will be denoted by $Var(\varphi)$ and $Lit(\varphi)$, respectively. The variable associated with the literal l is denoted by $v(l)$, for example, $v(\overline{x_2}) = x_2$ and $v(x_7) = x_7$.

A table that contains the value of a given logical formula $\varphi(\mathbf{x})$ for every input vector is called the *truth table* of $\varphi(\mathbf{x})$. Each row of the truth table contains a *assignment* for the formula, which is a function $A : Var(\varphi) \longrightarrow \{0, 1\}$, that is, one possible configuration of the input variables and the result of the operation for those values. To get the number of all possible assignments, multiply the number

of options for each variable. In this case, there are always 2 options for each variable. So the total number of rows is 2^n, where n is the number of variables.

An important set of problems in computational complexity involves finding assignments to the variables of a boolean formula expressed in CNF such that the formula is true. This leads to the following definitions.

Definition 1.22. (a) An assignment A *satisfies*

(i) a literal l if and only if

$$A(v(l)) = \begin{cases} 1 & \text{if} \quad l \in Var(\varphi) \\ \\ 0 & \text{if} \quad l \in \neg Var(\varphi), \end{cases}$$

where $\neg Var(\varphi) := \{\overline{x} \,|\, x \in Var(\varphi)\}$;

(ii) a clause c if and only if there is a literal $l \in c$ such that A satisfies it;

(iii) a formula in CNF φ if and only if A satisfies every clause of φ.

(b) A formula φ is *satisfiable* if there exist an assignment A which satisfies φ. If such assignment does not exist then φ is *unsatisfiable*.

Definition 1.23. The set of *models* of a Boolean formula φ, written $M(\varphi)$, is the set of assignments on $Var(\varphi)$ satisfying φ.

Naturally, the number of assignments which satisfy a formula φ is represented by $|M(\varphi)|$.

Example 1.2. 1. Let $\varphi(x_1, x_2) = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$. A satisfying assignment is given by $A(x_1) = A(x_2) = 1$. It can also be verified that $|M(\varphi)| = 2$.

2. Let $\varphi(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$. The truth table method allows to conclude that φ is unsatisfiable since every row ends with φ not satisfied, see Figure 1.10.

3. Determine whether or not $\varphi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$ is satisfiable.

In the truth table every single assignment is checked, therefore such method is complete. Yet, practically, this approach is not feasible for all but very small

x_1	x_2	x_3	$x_1 \vee x_2$	$x_1 \vee \overline{x_2}$	$\overline{x_1} \vee x_2$	$\overline{x_1} \vee x_3$	$\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$	$\varphi(x_1, x_2, x_3)$
0	0	0	0	1	1	1	1	0
0	0	1	0	1	1	1	1	0
0	1	0	1	0	1	1	1	0
0	1	1	1	0	1	1	1	0
1	0	0	1	1	0	0	1	0
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	0	1	0
1	1	1	1	1	1	1	0	0

Figure 1.10: Truth table for $\varphi(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$.

problem instances. In the former example with 3 variables, the total of all possible assignments is $2^3 = 8$, meaning that the corresponding truth table has 8 rows. Since this is a small number, it was easy to check for the satisfiability of the formula. For an instance with 30 variables, the total number of rows ($2^{30} = 1073741824$) is still quite small for a modern computer. However, as the number of variables grows, the number of rows increases all the more, leading to a quick overwhelming of even the fastest computers.

Chapter 2

Hypertree Decomposition

Before proceeding it is important to mention basic notions on complexity theory which are of our interest.

2.1 Complexity Classes

Computational complexity theory emphasizes on classifying certain problems according to their difficulty and it is built on basic sets of assumptions called *model of computation*. For instance Turing machines, recursive functions, and combinatory logic are models of computation. Two of the main goals of this theory are to introduce classes of problems which have the same complexity with respect to an specific computation model and complex measure and to study the essential properties of such classes.

Three types of typical computational problems are defined: decision problems, counting problems and optimization problems.

Decision problems correspond to those which have a boolean formula as an input and admit either a "yes" or "no" as an output. Hence, we are only asked to verify whether the input satisfies a certain property. An example of decision problem is the 3-coloring problem: given an undirected graph, determine whether there is a way to assign a color to each vertex in such a way that no two adjacent vertices have the same color.

Counting problems ask for the number of solutions of a given instance. Many problems in enumerative combinatorics and in statistical physics fall in this category. Given a directed graph and two vertices in it, source v_1 and destination v_n, computing the number of independent paths from v_1 to v_n is a counting problem.

In decision problems all possible solutions are considered equally acceptable. Be that as it may, in many practical situations this consideration is not fulfilled and it is necessary to choose the solutions according to certain criteria. In these situations, a measure is associated with each solution and depending on the application, the best solution is that in which a maximum measurement is reached, or a minimum measurement, if it is the case. This type of problems are called *optimization problems*.

Algorithms whose running time is bounded by a polynomial function are called *polynomial time* algorithms, such algorithms are considered *efficient*.

A *deterministic* computation is a (non necessarily finite) sequence of global states, starting with the initial global state such that each global state in the sequence yields the next.

A *nondeterministic* computation can then be viewed as a tree called a *computation tree*; the nodes correspond to global states while the edges correspond to transitions between global states caused by a single step. Each of the paths of the tree starting from the root is said to be a computation path.

Below there are simple informal descriptions of a few of the most commonly encountered classes.

- **P**: includes all problems that can be solved in polynomial time.
- **NP**: decision problems solvable in polynomial time via nondeterministic algorithms. (The set of problems whose solution can be VERIFIED in polynomial time).
- **NP hard**: problems which are at least as hard as the hardest problems in NP, but are not necessarily in NP.
- **NP complete**: problems which are in NP and are NP-hard.
- **FP**: functions computable in deterministic polynomial time.
- #**P**: counting problems solvable in nondeterministic polynomial time.

There are hundreds of complexity classes. Some of them are a subset of others. For example P is a subset of the NP class. To make the difference clear, if a problem in the P class has solution it can be computed in polynomial time, on the other hand a solution of an NP class problem can be verified in polynomial time if the solution is given.

The first problem shown to be NP-complete is the *SAT problem* in which a formula in CNF is instanced and it yields either a *yes* or a *no*, where *yes* means φ is satisfiable and *no* indicates the formula is unsatisfiable. The extension of the SAT problem to its counting version is denoted as $\#SAT$, the input of which is a formula in CNF, say φ, and its output is the number of assignments which satisfy φ.

2.1.1 Syntactic Classes

In general, the #SAT problem is difficult. This problem is $\#P$-complete and is difficult to approximate. Because of this, there are different ways to treat it. A very common approach is to consider Boolean formulas in CNF as input formulas to the problem. A second case is imposing restrictions on the number of literals or the number of occurrences of variables in the clauses or formulas, respectively.

Having that in mind, the following classes are described:

Name	Description
k-SAT	Formulas in which every clause contains at most k literals.
k-MON	k-SAT formulas with no negations.
k-HORN	Formulas containing HORN clauses of k literals.
$k\mu$-SAT	Each variable in the formula appears at most k times.
$l\mu$-kMON (HORN)	Monotone (HORN) formulas whose clauses contain at most k literals and each variable appears at most l times.
$l\mu$-ekMON (HORN)	Monotone (HORN) formulas whose clauses contain exactly k literals and each variable appears at most l times.

Figure 2.1: Syntactic classes limiting the size of clauses or formulas.

If $\mathcal{C}$ denotes a syntactic subclass of Boolean formulas in CNF, then $\#\mathcal{C}$ denotes its corresponding counting problem, that is, the problem of counting models belonging to class $\mathcal{C}$.

2.2 Hypergraphs

Hypergraphs are the generalization and extension of graphs considered as an efficient tool to represent and model concepts and structures in various areas of computer science and certain areas of mathematics.

Definition 2.1. Consider a set of vertices V and $E \subseteq \mathcal{P}(V) \setminus \{\emptyset\}$, where $\mathcal{P}(V)$ is the power set of V, the ordered pair $H = (V, E)$ is called a *hypergraph*.

Frequently, in literature, the word 'hyperedge' is used to refer to the edges of a hypergraph. That is not our case, we simply use the word 'edge'. A hypergraph H may be drawn as a set of points representing the vertices and simple closed curves enclosing the elements of each edge. Therefore, a simple graph is a hypergraph each of whose edges has cardinality 2, in which a line is drawn to join two vertices instead of drawing a closed curve around them; and a multigraph is a hypergraph in which each edge has cardinality less or equal to 2. Isolated nodes of a hypergraph graph shall not be considered as edges. The next figure shows a hypergraph having an isolated node and four edges whose cardinalities are all different.

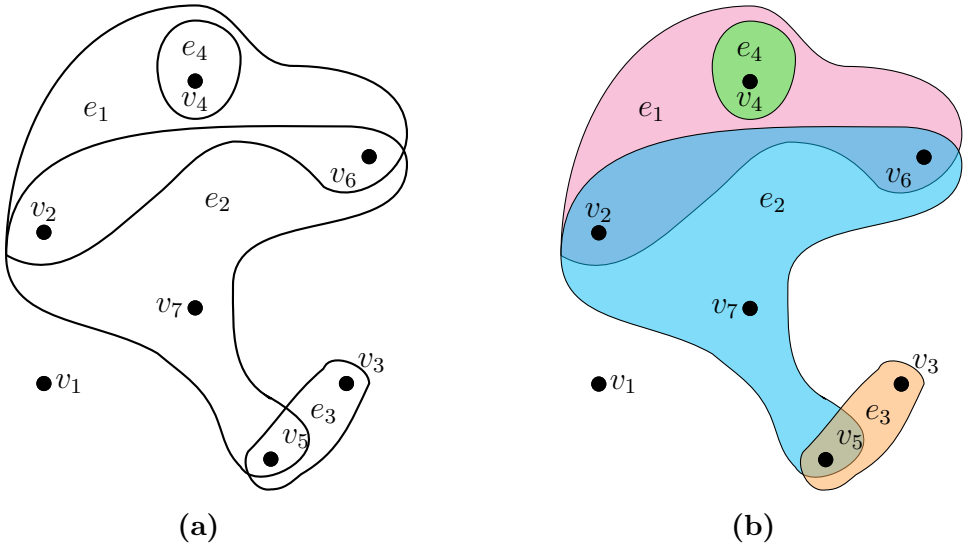


Figure 2.2: A hypergraph H.

The prior definition suggests that edges of a hypergraph can contain an arbitrary nonzero number of vertices. Thus, the possibility of having hypergraphs where all edges have the same cardinality is not excluded.

Definition 2.2. A *k-uniform hypergraph* or *k-graph* H is a pair (V, E) where V is a vertex set and E is a set of edges each consisting of k vertices.

In this sense a 2-uniform hypergraph is a graph, a 3-uniform hypergraph is a set of 3-element subsets, and so on.
A hypergraph can be seen as an incidence structure, that is to say, a family of two sets, "points" and "lines," with an incidence relation between their elements, in which the vertex set plays the role of "points," the collection of edges plays the role of "lines," and the incidence relation is set membership $\in$.

Definition 2.3. The *incidence graph* of a hypergraph H is the bipartite graph $I = (V_I, E_I)$, where $V_I = V(H) \cup E(H)$ and such that there is an edge between $v \in V(H)$ and $e \in E(H)$ if and only if $v \in e$.

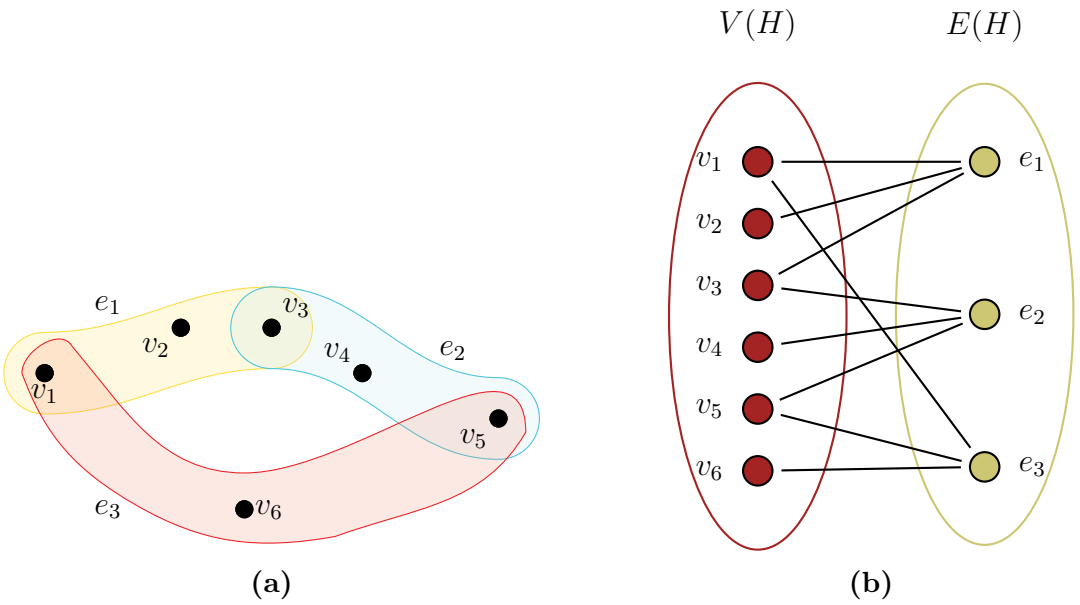


Figure 2.3: Hypergraph H (**a**) and its incidence graph (**b**).

The notation $I(H)$ is used to mean that I is the incidence graph of the hypergraph H. We keep the same notation as in graphs, $V(H)$ stands for the set of vertices of hypergraph H and $E(H)$ represents the set of edges of H.

Another important notion on hypergraphs is independency which is described next.

Definition 2.4. Given a hypergraph H, $X \subseteq V(H)$ is *independent* if X contains no edges of H.

In Figure 2.2, for hypergraph H, $X = \{v_2, v_3, v_6, v_7\}$ is an independent set but if node v_5 is added to E_1, it is no longer independent.

2.2.1 Hypertrees

Informally, a *tree-decomposition* of a graph G is a non-unique representation of G in a tree-like structure with desirable properties that allow it to be used to determine certain information of the original graph. This concept is useful in the study of fundamental questions in graph theory. In some cases, the information obtained from the tree-decomposition can be used to construct efficient algorithms to solve problems on G. Sometimes, problems which are NP-hard might be solvable in polynomial or even linear time when restricted to trees.

The concept of decomposition of a hypergraph and its associated notion of width are presented next.

Definition 2.5. A *hypertree* for a hypergraph H is a triple (T, μ, λ), where $T = (V, E)$ is a tree and $\mu : V \longrightarrow \mathcal{P}(V(H))$ and $\lambda : V \longrightarrow \mathcal{P}(E(H))$ are labeling functions.

Example 2.1. A hypertree for the hypergraph in Figure 2.2 is (T, μ, λ), where $V(T) = \{w_1, w_2, w_3, w_4\}$, $E(T) = \{w_1w_2, w_1w_3, w_3w_4\}$ and the functions μ and λ are defined as follows.

$$\begin{array}{ll} \mu : V(T) \longrightarrow \mathcal{P}(V(H)) & \lambda : V(T) \longrightarrow \mathcal{P}(E(H)) \\ \mu(w_1) = \{v_1, v_2, v_3\} & \lambda(w_1) = \{e_1\} \\ \mu(w_2) = \{v_2, v_4, v_6\} & \lambda(w_2) = \{e_2, e_3\} \\ \mu(w_3) = \{v_3\} & \lambda(w_3) = E(H) \\ \mu(w_4) = \emptyset & \lambda(w_4) = \{e_1, e_2, e_4\}. \end{array}$$

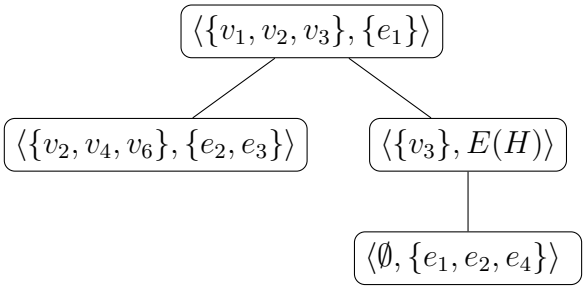
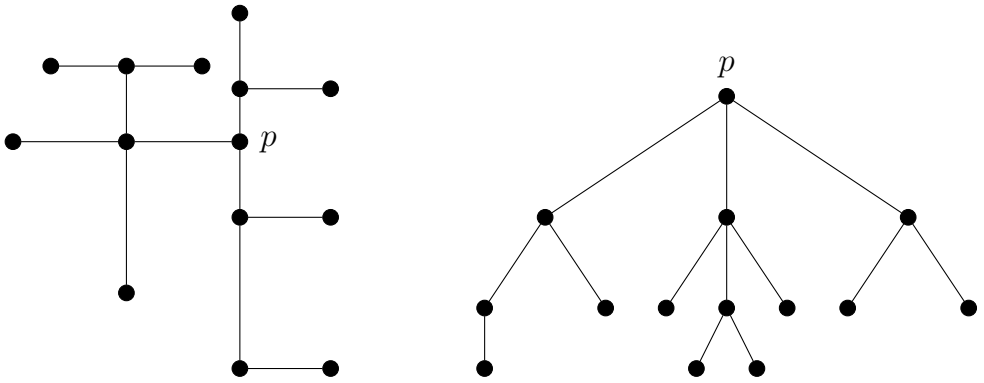


Figure 2.4: A hypertree for the hypergraph in Figure 2.2.

Let $\mathcal{C}(T) = \{A \mid A \text{ is a subtree of } T\}$. If $T' = (V', E')$ is a subtree of T then $\tilde{\mu}(T') : \mathcal{C}(T) \longrightarrow \mathcal{P}(E(H))$ is defined as $\tilde{\mu}(T') = \bigcup_{p \in V'} \mu(p)$.

Recalling the notation used in the first section, the root of T is written R. Now, for every $p \in V(T)$ the subtree of T rooted at p is denoted by T_p. In the following figure T is a tree, $p \in V(T)$ and T_p is shown.

Observe that T and T_p are isomorphic.



(a) A tree T **(b)** T_p

Figure 2.5: A tree rooted at a given vertex.

Definition 2.6. A *hypertree decomposition* of a hypergraph H is a hypertree (T, μ, λ) for H which satisties the following conditions:

(i) For all $e \in E(H)$ there exists $p \in V(T)$ such that $e \subseteq \mu(p)$,

(ii) If $v \in V(H)$ then the set $\{p \in V(T) | v \in \mu(p)\}$ induces a subtree of T,

(iii) For every $p \in V(T)$, $\mu(p) \subseteq \bigcup \lambda(p)$, and

(iv) For any $p \in V(T)$, $\bigcup \lambda(p) \cap \mu(T_p) \subseteq \mu(p)$.

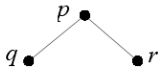
Example 2.2. Let H be a hypergraph such that $V(H) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$ and $E(H) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$ with $e_1 = \{v_1, v_2, v_3\}$, $e_2 = \{v_4, v_5, v_6\}$, $e_3 = \{v_3, v_4, v_7\}$, $e_4 = \{v_1, v_6, v_9\}$, $e_5 = \{v_7, v_9\}$, $e_6 = \{v_2, v_5, v_8\}$, $e_7 = \{v_5, v_{10}\}$, and $e_8 = \{v_1, v_8, v_{10}\}$.
A hypertree decomposition (T, μ, λ) of H is shown in the following figure. Clearly, T is a tree such that $V(T) = \{p, q, r, \}$ and $E(T) = \{\varepsilon_1, \varepsilon_2\}$. The labeling functions μ and λ are defined as

$$\mu(p) = \{v_1, v_2, v_3, v_4, v_5, v_6\} \qquad \lambda(p) = \{e_1, e_2\}$$
$$\mu(q) = \{v_1, v_3, v_4, v_6, v_7, v_9\} \qquad \lambda(q) = \{e_3, e_4\}$$
$$\mu(r) = \{v_1, v_2, v_8, v_{10}\} \qquad \lambda(r) = \{e_6, e_8\}.$$

Indeed, the four conditions of the definition are met.

(i) For $e_1, e_2 \in E(H)$ it holds that $e_1, e_2 \subseteq \mu(p)$.
For $e_3, e_4, e_5 \in E(H)$ it holds that $e_3, e_4, e_5 \subseteq \mu(q)$.
For $e_6, e_7, e_8 \in E(H)$ it holds that $e_6, e_7, e_8 \subseteq \mu(r)$.

(ii)

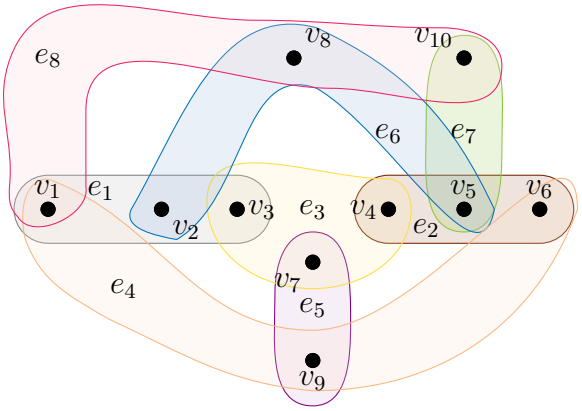
$\mathbf{v} \in \mathbf{H}$	$\{\mathbf{u} \in \mathbf{V(T)} \mid \mathbf{v} \in \mu(\mathbf{u})\}$	**Induced subtree of T**
v_1	$\{p, q, r\}$	
$v_2,\ v_5$	$\{p, r\}$	
$v_3,\ v_4,\ v_6$	$\{p, q\}$	
v_7, v_9	$\{q\}$	
v_8, v_{10}	$\{r\}$	

(iii) For $p \in V(T)$,
$\mu(p) = \{v_1, v_2, v_3, v_4, v_5, v_6\} \subseteq \bigcup \lambda(p) = e_1 \cup e_2 = \{v_1, v_2, v_3\} \cup \{v_4, v_5, v_6\}$.
For $q \in V(T)$,
$\mu(q) = \{v_1, v_3, v_4, v_6, v_7, v_9\} \subseteq \bigcup \lambda(q) = e_3 \cup e_4 = \{v_3, v_4, v_7\} \cup \{v_1, v_6, v_9\}$.
For $r \in V(T)$,
$\mu(r) = \{v_1, v_2, v_5, v_8, v_{10}\} \subseteq \bigcup \lambda(r) = e_1 \cup e_2 = \{v_2, v_5, v_8\} \cup \{v_1, v_8, v_{10}\}$.

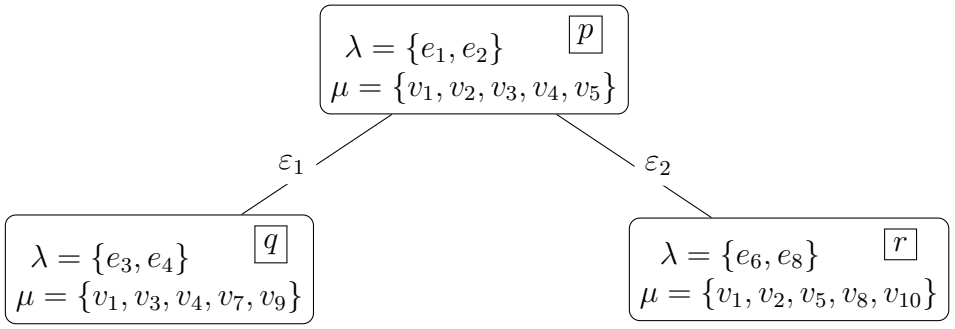
(iv) Given $p \in V(T)$, the set of vertices of T_p is $V(T_p) = \{p, q, r\}$.
Also $\mu(T_p) = \mu(p) \cup \mu(q) \cup \mu(r) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$ and

$\lambda(p) = \{e_1, e_2\}$. Hence $\bigcup \lambda(p) \cap \mu(T_p) = e_1 \cap \mu(T_p) \cup e_2 \cap \mu(T_p) = e_1 \cup e_2 \subseteq \mu(p) = \{v_1, v_2, v_3, v_4, v_5, v_6\}$.

Similarly for $q, r \in V(T)$:
$V(T_q) = \{q\}$, $\mu(T_q) = \mu(q) = \{v_1, v_3, v_4, v_6, v_7, v_9\}$ and $\lambda(q) = \{e_3, e_4\}$. Thus $\bigcup \lambda(q) \cap \mu(T_q) = e_3 \cap \mu(T_q) \cup e_4 \cap \mu(T_q) = e_3 \cup e_4 \subseteq \mu(q) = \{v_1, v_3, v_4, v_6, v_7, v_9\}$;
$V(T_r) = \{r\}$, $\mu(T_r) = \mu(r) = \{v_1, v_2, v_5, v_8, v_{10}\}$ and $\lambda(r) = \{e_6, e_8\}$. Thus $\bigcup \lambda(r) \cap \mu(T_r) = e_6 \cap \mu(T_r) \cup e_8 \cap \mu(T_r) = e_6 \cup e_8 \subseteq \mu(r) = \{v_1, v_2, v_5, v_8, v_{10}\}$.



(a) Hypergraph H



(b) (T, μ, λ)

Figure 2.6: A hypergraph and its tree decomposition.

Definition 2.7. The *width of a hypertree decomposition* (T, μ, λ) is given by $max\{|\lambda(p)| : p \in V(T)|\}$, and the *hypertree-width* of a hypergraph is the minimum width over all its hypertree decompositions.

Figure 2.6 shows an example of a hypergraph and its hypertree decomposition of width 2.

As pointed out in [GLS02], the concept of hypertree decomposition is a natural generalization of the concept of tree decomposition to hypergraphs.

Chapter 3

Cycles in Hypergraphs

3.1 Cyclicity

Empty edges are worthless when studying hypergraph cyclicity and acyclicity notions. This is because the empty edge cannot play a role in a cycle. In the same manner, there is no reason to consider the case where some vertices are contained in no edge. Due to these reasons, from this point on, any hypergraph can be thought as a set of nonempty edges.

Definition 3.1. Consider a hypergraph H and two distinct vertices $u, v \in V(H)$. 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 $|i - j| \geq 2$.

The above sequence of edges is called an *edge path* (or just a *path* when no confusion arises) from e_{p_1} to e_{p_k}.

The concept of connectedness of graphs can be extended in a natural way to hypergraphs. Two nodes are *connected* if there is a path from one to the other. Similarly, two edges are connected if there is an edge path from one to the other. A set of nodes or edges is connected if every pair is connected. The *components* of a hypergraph are its maximal connected sets of edges.

Definition 3.2. A *cycle of a hypergraph* H is a sequence of edges $(e_{p_1}, e_{p_2}, \ldots, e_{p_k})$ satisfying the following conditions:

- $e_{p_1} = e_{p_k}$ and

• for all $2 \leq i \leq k-2$ and an arbitrary $e \in E(H)$,

$$\big((e_{p_{i-1}} \cap e_{p_i}) \cup (e_{p_i} \cap e_{p_{i+1}}) \cup (e_{p_{i+1}} \cap e_{p_{i+2}})\big) \setminus e \neq \emptyset.$$

Such definition of cycles in hypergraphs works for graphs as well. In simple words, a cycle is a sub(hyper)graph, such that after deleting one of its edges, the number of connected components remains the same. However, unlike graphs, in hypergraphs there are different "degrees" of cyclicity which are defined below.

Definition 3.3. A *Berge cycle* in a hypergraph H is a sequence $(e_{p_1}, v_{p_1}, e_{p_2}, v_{p_2}, \ldots, e_{p_n}, v_{p_n}, e_{p_{n+1}})$ such that

(i) $n \geq 2$;

(ii) $e_{p_{n+1}} = e_{p_1}$;

(iii) $v_{p_1}, v_{p_2}, \ldots, v_{p_n}$ are distinct vertices in $V(H)$;

(iv) $e_{p_1}, e_{p_2}, \ldots, e_{p_n}$ are distinct edges in $E(H)$; and

(v) $v_{p_i} \in e_{p_i} \cap e_{p_{i+1}}$, for $1 \leq i \leq n$.

If a hypergraph has a Berge cycle then it is said to be *Berge-cyclic.*

Example 3.1. Consider the hypergraph H shown in the figure below. The sequence $(e_2, v_4, e_3, v_5, e_2)$ is a Berge cycle.

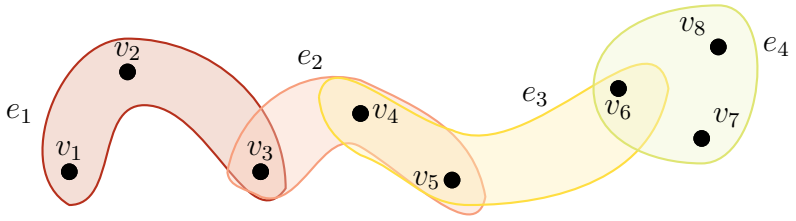


Figure 3.1: A Berge-cyclic hypergraph.

As seen in this example, if there is some pair of edges in a hypergraph H having at least two vertices in common, then H is Berge-cyclic.

Definition 3.4. For $n \geq 3$, let $(e_{p_1}, v_{p_1}, e_{p_2}, v_{p_2}, \ldots, e_{p_n}, v_{p_n})$ be a sequence of distinct edges e_{p_i}s and distinct vertices v_{p_i}s in a hypergraph. If for every $i \in \{1, 2, \ldots, n-1\}$, $v_{p_i} \in e_{p_i} \cap e_{p_{i+1}}$ but does not belong to any other e_{p_j} and $v_{p_n} \in e_{p_n} \cap e_{p_1}$ (it may also be an element of other edges), then the sequence is called a *γ-cycle*.

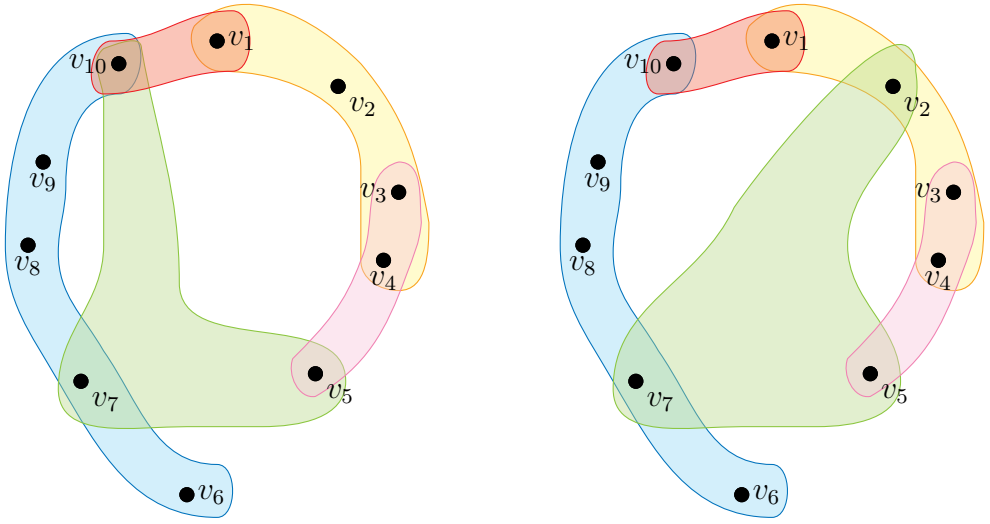
Definition 3.5. A *β-cycle* in a hypergraph is a γ-cycle $(e_{p_1}, v_{p_1}, e_{p_2}, v_{p_2}, \ldots, e_{p_n}, v_{p_n})$ such that v_{p_n} belongs to $e_{p_1} \cap e_{p_n}$ and no other e_{p_j}.

In the very first section, we said that two edges of a graph are *neighbors* if they share a vertex, generalizing this concept to hypergraphs, we now say that two edges e_i and e_j are *neighbors* if they share at least one vertex. Moreover, e_i and e_j are *α-neighboring* if there is no other path allowing to go from one to the other apart from the trivial one, that is (e_i, e_j).

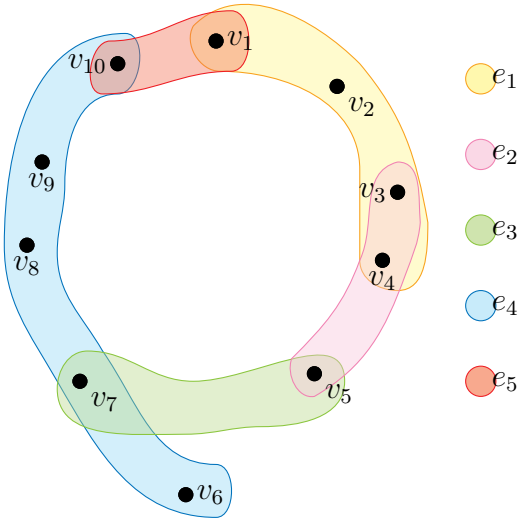
Definition 3.6. An *α-path* in a hypergraph is a sequence of edges $(e_{p_1}, e_{p_2}, \ldots, e_{p_k})$ such that for all i , $1 \leq i < k$, e_{p_i} and $e_{p_{i+1}}$ are α-neighboring.

Definition 3.7. An *α-cycle* in a hypergraph is an α-path $(e_{p_1}, e_{p_2}, \ldots, e_{p_k})$ such that $k > 3$, $e_{p_1} = e_{p_k}$, and there are no $1 \leq i < j < k$ $e_{p_i} \cap e_{p_{i+1}} \subset e_{p_j} \cap e_{p_{j+1}}$.

Example 3.2. Look at Figure 3.3. Consider the sequence $\mathcal{S} = (e_5, v_1, e_1, v_4, e_2, v_5, e_3, v_7, e_4, v_{10})$. In (**a**), $\mathcal{S}$ is a γ-cycle, however this same sequence corresponds to a β-cycle in (**b**). Meanwhile in (**c**), the sequence $(e_1, e_2, e_3, e_4, e_5, e_1)$ represents an α-cycle.



(a) γ-cycle. **(b)** β-cycle.



(a) α-cycle.

Figure 3.3: Three different degrees of cyclicity in hypergraphs.

3.2 Characterizations of acyclicity

Graph acyclicity is defined in a natural way thanks to the notion of cycles in graph theory, meaning that a graph is acyclic if and only if it contains no cycle. Even so, on hypergraphs, there exist different degrees of acyclicity and thus different non-equivalent notions of acyclicity. This section is focused on α, β, γ, and Berge-acyclicity. Each of these notions admits many different characterizations which can be further studied in [BB16] and [Fag83].

Since we have already defined α, β, γ and Berge cycles, let us define the respective different degrees of acyclicity in terms of their absence.

Definition 3.8. A hypergraph is *α-acyclic* (respectively *β-acyclic*, *γ-acyclic*, and *Berge-acyclic*) if it contains no α cycles (respectively β, γ, and *Berge* cycles).

As far as α , β, and γ-acyclicity concerns, we will restrict ourselves to characterizations involving the notion of join trees.

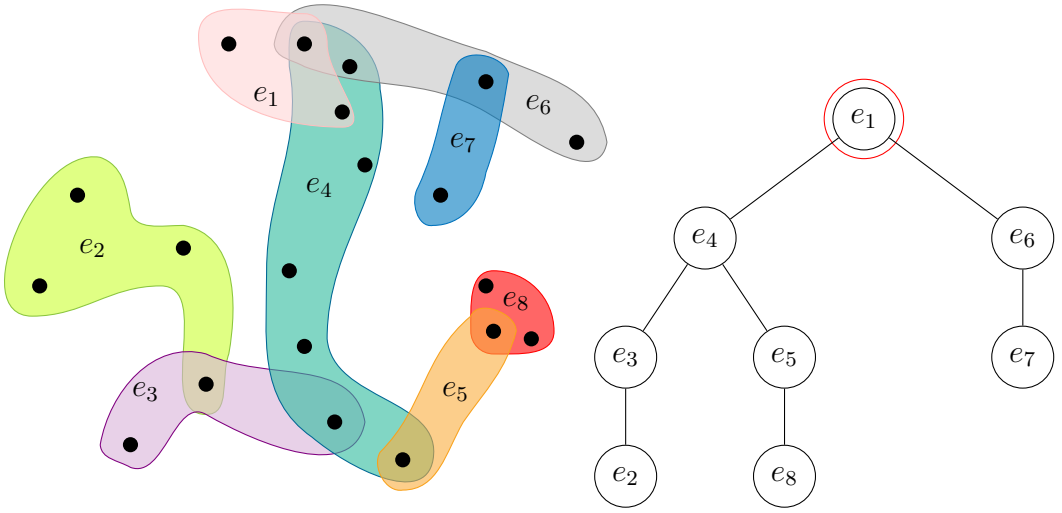
Definition 3.9. A *join tree* for a hypergraph H is, if it exist, a rooted tree $T = (E, J)$ with set of nodes the edges of H, J is its set of edges, and such that if $v \in V(H)$ belongs to two edges e_i and e_j in $E(H)$ then it is also contained in all nodes of $\mathring{e}_i P \mathring{e}_j$ in T.

Hence, the set of nodes of a join tree T that contain a vertex $v \in V(H)$ is connected in T. The *root* of T, denoted R, is some edge of H and a *branch* is a path in T beginning with R and maximal for inclusion. See Figure 3.4.

The notion of α-acyclicity is the most general one known in the literature. However, the definition of α-acyclicity has not been fully studied in terms of cycles on hypergraphs, instead it is usually defined via the Graham reduction process, equivalent definitions based on articulations in hypergraphs or cycles in a graph representation of hypergraphs related to connections of edges. Various authors such as Duris [Dur12], Brault-Baron [BB16], and Jégou [JN09], to name a few, frequently state the next theorem either as a definition or as a equivalent statement of α-acyclicity.

Theorem 3.1. *A hypergraph is α-acyclic if and only if it admits a join tree.*

Other names for α-acyclic hypergraphs are *decomposable hypergraphs* or *hypertrees*.



(a) α-acyclic hypergraph H. **(b)** Rooted tree T.

Figure 3.4: Join tree T of hypergraph H.

The following theorem is given as a definition of β-acyclicity in [Fag83]. It is also proved to be equivalent to the definition we previouly presented.

Theorem 3.2. *A hypergraph H is β- acyclic if and only if every subset of $E(H)$ is α-acyclic.*

Definition 3.10. A join tree T of a hypergraph H has *disjoint branches* if edges of H belonging to different branches of T are disjoint.

In Figure 3.4, e_4 and e_6 belong to different branches but $e_4 \cap e_6 \neq \emptyset$. Consequently, the given join tree does not have disjoint branches. Figure 3.5(**a**) depicts an example of a join tree with disjoint branches.

Theorem 3.3. *If a hypergraph has a join tree with disjoint branches then it is β-acyclic.*

Proof. Let H be a hypergraph having a join tree T with disjoint branches whose root is R. Assume H is not β-acyclic, thus it contains a β-cycle $(e_{p_1}, v_{p_1}, e_{p_2}, v_{p_2}, \ldots, e_{p_n}, v_{p_n})$. Since for every p_i, $v_{p_i} \in e_{p_i} \cap e_{p_{i+1}}$, all edges must belong to the same branch of T. Assume without loss of generality that e_{p_1} is closer to R than e_{p_2}, else the β-cycle $(e_{p_2}, v_{p_1}, e_{p_1}, v_{p_n}, e_{p_n}, v_{p_{n-1}}, e_{p_{n-1}}, \ldots, e_{p_3}, v_{p_2})$ would be considered instead, so every e_{p_i} belongs to the same branch in the order $e_{p_1}, e_{p_2}, \ldots, e_{p_n}$ from the root of T such that e_n is the farthest and $e_{p_2}, \ldots, e_{p_{n-1}}$ lie between e_{p_1} and e_{p_n}. By the definition of β-cycle v_{p_n} belongs to $e_{p_1} \cap e_{p_n}$, and since T is a join tree v_{p_n} should also belong to every other e_{p_i}, which is a contradiction. □

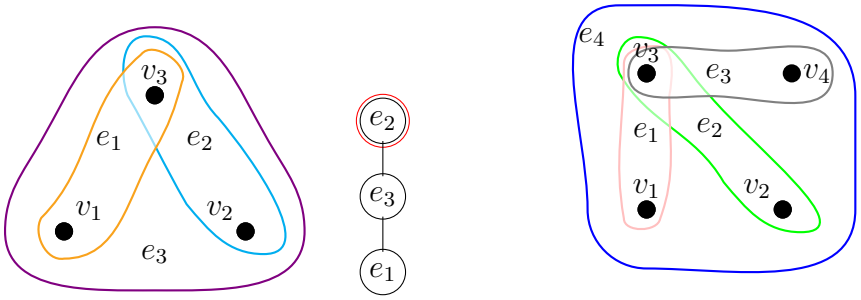
Theorem 3.4. *For every γ-acyclic hypergraph H and for every edge $e \in E(H)$, H has a tree with disjoint branches whose root is e.*

Proof. Mathematical induction on the number of edges is used to prove this. When the hypergraph has only one edge, the statement is trivially valid. Let $H = (V, E)$ be a γ-acyclic hypergraph and assume the induction hypothesis is true for hypergraphs with strictly less edges than H. Consider the hypergraph $(V, E \setminus e)$ splitted in connected components $H_1, ..., H_n$ ignoring the vertices that belong only to e. Clearly, for every $i \in \{1, 2, \ldots, n\}$, $H_i = (V_i, E_i)$ is γ-acyclic and satisfies the induction hypothesis allowing us to choose any edge of H_i as a root. For every i, let T_i be a join tree with disjoint branches for H_i with e_i as a root. The join tree T for H with disjoint branches whose root is e is hereby defined. The root of T is e and each T_i can be connected to T_i with an edge $\{e, e_i\}$. It is know that each T_i has disjoint branches and, by definition of component, the V_is are pairwise disjoint. It follows that T has disjoint branches. It remains to prove that T is a join tree, which means that for every $v \in V$, the set of nodes of T that contain v is connected in T. Inasmuch as $V = \bigcup_{i=1}^{n} V_i \cup e$ there are three cases: $v \in e \setminus \bigcup_{i=1}^{n} V_i$, $v \in \bigcup_{i=1}^{n} V_i \setminus e$, or $v \in e \cap \bigcup_{i=1}^{n} V_i$. If $v \in e$ and v does not belong to any other edge, there is nothing to prove. If the second case happens, there exist $1 \leq i_0 \leq n$

such that v belongs V_{i_0} and the only edges that contain v are in T_{i_0}. Lastly, if $v \in e \cap \bigcup_{i=1}^{n} V_i$ then $v \in e$ and $v \in V_{i_0}$ for some $i_0 \in \{1, \ldots, n\}$. In view of the fact that for every i there exist $e_i \in E_i$ such that $V_i \cap e \subseteq e_i$ [Dur12], particularly, it follows that $v \in e \cap e_{i_0}$. Therefore, e_{i_0} belongs to the set of edges containing v in T_{i_0}. Moreover, the only edge which contains v and is not in T_{i_0} is e, which is connected to e_{i_0} in T. Thus, the set of edges of H that contain v is connected. Since the three cases exhaust all the possibilities, this proves that for any γ-acyclic hypergraph H and every $e \in E(H)$, there exist a join tree with disjoint branches whose root is e. □

Corollary 3.1. *If a hypergraph is γ-acyclic then it has a join tree with disjoint branches.*

In both preceding statements, Corollary 3.1 and Theorem 3.3, the converse is not true. Below, Figure 3.5 (**a**) depicts a γ-cyclic hypergraph which can be writen as the sequence $(e_2, v_2, e_3, v_1, e_1, v_3) = (\{v_2, v_3\}, v_2, \{v_1, v_2, v_3\}, v_1, \{v_1, v_3\}, v_3\ )$ and it has a join tree with disjoint branches which is shown next to it. The hypergraph shown in Figure 3.5 (**b**) is β-acyclic and it is not hard to check that it has no join tree with disjoint branches.



(a) **(b)**

Figure 3.5: (**a**) A γ-cyclic hypergraph and its join tree with disjoint branches. (**b**)A β-acyclic hypergraph having no join tree with disjoint branches.

It can be proved that the reverse of Theorem 3.4 is true, obtaining the following characterization of γ-acyclicity [Dur12].

Theorem 3.5. *A hypergraph H is γ-acyclic if and only if, for every edge $e \in E(H)$, H has a tree with disjoint branches whose root is e.*

Definition 3.11. A property ρ of hypergraphs is closed under an operation when for all hypergaph H and all H' obtained by such operation, it holds that $\rho(H) \Rightarrow \rho(H')$.

Thanks to the closure property, some facts are easy to derive. For instance, if a hypergraph is β-acyclic (respectively γ- acyclic), then so is every subset of it. Hence β-acyclicity and γ-acyclicity are closed under taking a subset. This is not generally true for α-acyclic hypergraphs. There are α-acyclic hypergraphs whose subsets are not all α-acyclic. Indeed, $H = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}, \{v_1, v_2, v_3\}\}$ is α-acyclic, however, $H' = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}\} \subseteq H$ is not, as portrayed in Figure 3.6.

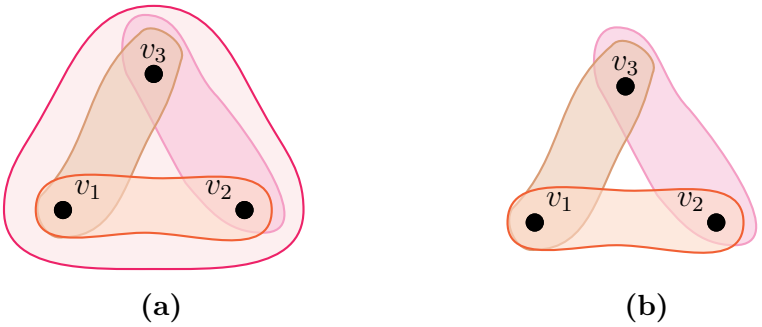


Figure 3.6: (**a**) α-acyclic hypergraph H. (**b**)α-cyclic hypergraph H'.

In [Ber73], Claude Berge gives a result of Berge-acyclicity for hypergraphs.

Theorem 3.6. *A hypergraph H is Berge-acyclic if and only if its incidence graph is acyclic.*

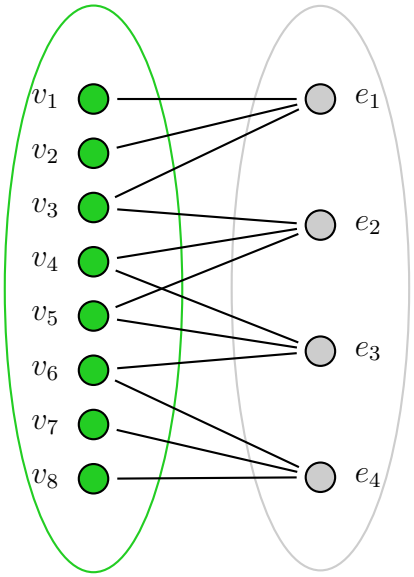
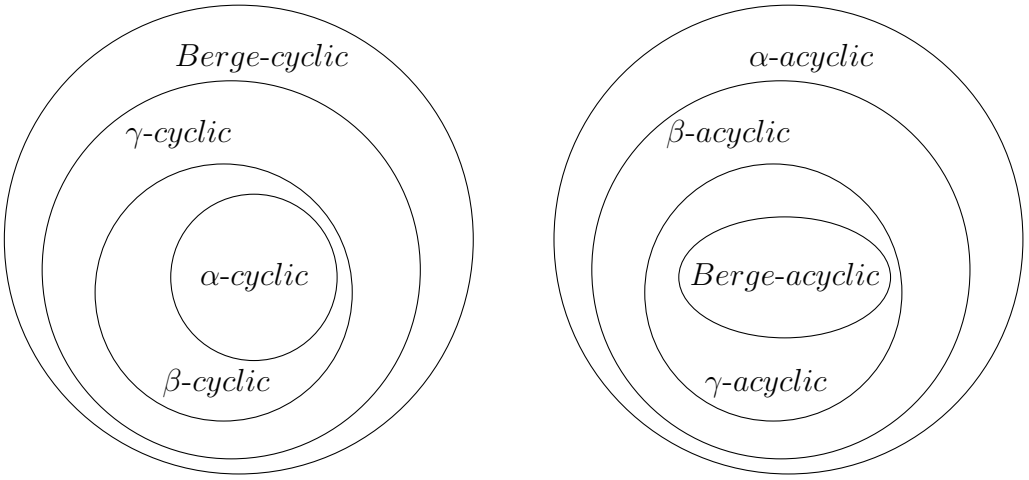


Figure 3.7: Incidence graph for hypergraph in figure 3.1 .

This equivalence, in fact matches with our definition of Berge acyclicity: if a hypergraph has some pair of distinct vertices u, v and some pair of distinct edges

e_1, e_2 such that both u and v belong to e_1 and e_2 simultaneously, then it is Berge-cyclic. For instance, the incidence graph for the Berge-cyclic hypergraph given in Figure 3.1 is clearly cyclic.

Berge acyclicity implies γ-*acyclicity*. Also, the notions of γ and β-acyclicity satisfy the property that, if a hypergraph is γ-acyclic then it is β-acyclic. Further still, β-acyclicity implies α-acyclicity. Yet, none of the reverse implications hold. A proof of these implications can be found in [Fag83]. The following figure represents these relationships.



(a) Cyclicity **(b)** Acyclicity

Figure 3.8: Cyclicity and acyclicity relationships on hypergraphs.

The following result is known regarding cycles in hypergraphs. Its proof can be found in [BB16]

Theorem 3.7. *α, β and γ acyclicity are polynomial time decidable.*

Remark α-acyclicity is even linear time decidable, as well as Berge-cyclicity since it can be tested by an exploration of the incidence graph.

3.3 Hypergraphs and Boolean Formulae in CNF

Since many important problems in computer science are intractable in general, it is a reasonable task to identify tractable subclasses of such problems which can be solved efficiently. One approach to do this is to restrict the structure of a problem represented as graph or hypergraph. For instance, the structure of Boolean conjunctive queries can be naturally encoded by hypergraphs.

Definition 3.12. Given a formula in CNF $\varphi(x_1, x_2, \ldots x_n)$, its *associated hypergraph* is a hypergraph H_φ such that $V(H_\varphi) = Lit(\varphi)$ and $E(H_\varphi) = \{Lit(c) \,|\, c$ is a clause of $\varphi\}$.

Notice that $V(H_\varphi) = \{x_1, x_2, \ldots, x_n\}$ and $E(H_\varphi) = \{Var(c) \,|\, c$ is a clause of $\varphi\}$ if φ is a monotone CNF formula.

Example 3.3. Consider the formula $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6 \vee x_7) \wedge (x_1 \vee x_7)$. Its associated hypergraph is shown in the following figure, where $V(H_\varphi) = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ and $E(H_\varphi) = \{\{x_1, x_2, x_3\}, \{x_3, x_4\}, \{x_5, x_6, x_7\}, \{x_6, x_7\}\}$.

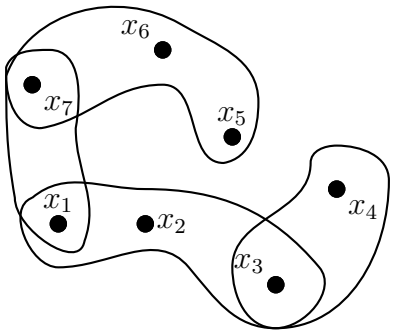


Figure 3.9: Associated hypergraph of the formula $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6 \vee x_7) \wedge (x_1 \vee x_7)$.

Consequently, if some graphic property on the formula φ is mentioned, it should be understood that such property refers to H_φ. For example, "the formula φ is connected" means that its associated hypergraph is connected.

Definition 3.13. The associated hypergraph of a formula φ in CNF represents a *simple chain* if its clauses can be ordered in such a way that two consecutive clauses match in one variable.

Figure 3.10 exemplifies a simple chain.

If consecutive clauses of a formula φ share one or more variables then φ can be written as $\varphi = c_1 c_2 \cdots c_m$ where $\mid c_i \cap c_{i+1} \mid \geq 1$, for $i \in \{1, 2, \ldots, m-1\}$ and $c_i \cap c_j = \emptyset$ when $\mid i - j \mid \geq 2$.

Definition 3.14. The *incidence graph of a CNF-formula* φ is the bipartite graph which has as vertices the set $Var(\varphi) \cup clauses(F)$ and $x \in Var(\varphi)$ and $c \in clauses(F)$ are connected by an edge if and only if x appears in c.

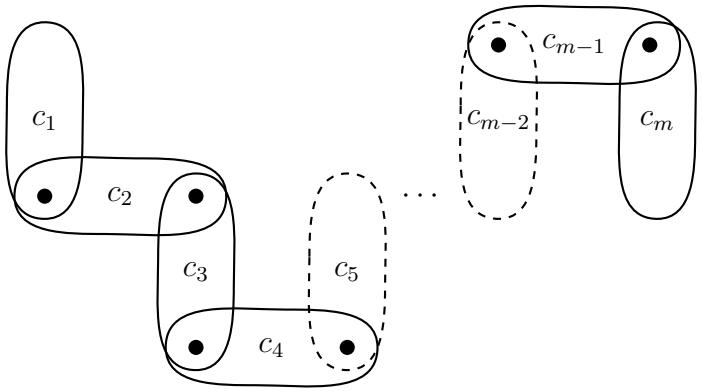


Figure 3.10: A simple chain φ.

The following result by Capelli et al. [CDM14] establishes the existence of a polynomial time algorithm which solves the #SAT problem for CNF formulas that allow a disjoint branches decomposition.

Theorem 3.8. *Given a CNF formula φ, and a disjoint branches decomposition of the hypergraph of φ, there is an algorithm that computes the number of models of φ in polynomial time.*

It is known known that #SAT for CNF formulas with α-acyclic hypergraphs is #P-hard and tractable for γ-acyclic hypergraphs. Unfortunately, #SAT for CNF formulas with β-acyclic hypergraphs is a problem whose complexity could so far not be determined despite considerable attempts by several authors. Even more, thanks to the last theorem, for CNF formulas whose hypergraphs have a disjoint branches decompositions #SAT can be solved in polynomial time. The next table shows the known complexity results for the restrictions of #SAT.

Class	Lower bound	Upper Bound
Primal treewidth		FPT
Incidence treewidth		FPT
Modular incidence treewidth		FPT
Signed incidence cliquewidth		FPT
Incidence cliquewidth	W[1]-hard	XP
γ-acyclic		FP
β-acyclic	?	?
α-acyclic	#P-hard	#P
Disjoint branches		FP

Figure 3.11: Known complexity results for structural restrictions of #SAT.

For more information on the table and the definitions of the appearing complexity classes see [CDM14].

Chapter 4

A Tractable Syntactic Subclass within $2\mu - e3MON$

Within the versions of #SAT whose complexity classification still remains undetermined, #2μ-3MON and #2μ-kSAT with $k > 2$, are found.

In the present thesis, using the idea of model counting by means of matrix operators, a tractable family of functions within this syntactic classification is proposed. Specifically, we deal with the $2\mu - e3MON$ syntactic class. The tractability of this family is determined by the topological structure of the hypergraph associated with the formula. Formulas belonging to this class are associated with 3-graphs. Most of the content of the chapter is based on [GLA13].

4.1 Charges

First, regarding Boolean formulas, the concepts of charge and conjoint charge of one and two variables, respectively, are introduced. Simple results involving the set of models of a formula, which will be useful, are established.

Consider a formula in CNF, for $t \in \{0, 1\}$ and $x \in Var(\varphi)$, $\varphi_{x=t}$ denotes the formula obtained after making $x = t$ in φ and simplifying the formula performing the following steps:

1. Eliminate every clause containing 1.

2. Remove each 0.

In particular, for $\varphi(x_1, x_2, x_3, x_4, x_5) = \{\{x_1, x_3, x_5\}, \{x_2, x_4\}, \{x_2, x_3, x_5\}, \{x_1, x_4\}\}$, $\varphi_{x_2=0} = \{\{x_1, x_3, x_5\}, \{x_4\}, \{x_3, x_5\}, \{x_1, x_4\}\}$ and $\varphi_{x_1=1,x_3=0} = \{\{x_2, x_4\}, \{x_2, x_5\}\}$.

If the formula is indexed, a vertical line "|" is used to separate, avoid confusions and prevent excessive use of parenthesis. For example, $\varphi_{j|x=t}$ is the same as $(\varphi_j)_{x=t}$.

Correspondingly, $M_{x=t}(\varphi)$ represents the set of models in $M(\varphi)$ which take the value t in the variable x. In general, for $t_j \in \{0, 1\}$, the set of models with the restriction r over $x_1, x_2, \ldots, x_k \in Var(\varphi)$ is symbolized as $M_{r(x_1,x_2,\ldots,x_k)}(\varphi)$, where $r(x_1, x_2, \ldots, x_k) = \{x_1 = t_1, x_2 = t_2, \ldots, x_k = t_k\}$. Observe that for any restriction r, $M_{r(x_1,x_2,\ldots,x_k)}(\varphi) \subseteq M(\varphi)$.

Let $S_{x=t}(\varphi)$ represent the set of variables from x eliminated when φ is simplified to $\varphi_{x=t}$, this is, $S_{x=t}(\varphi) = Var(\varphi) \setminus (\{x\} \cup Var(\varphi_{x=t}))$. For any two formulas φ_1 and φ_2 such that the intersection of sets $Var(\varphi_1)$ and $Var(\varphi_2)$ is equal to $\{x\}$ or $\emptyset$, the following equalities are true:

$$Var((\varphi_1 \cup \varphi_2)_{x=t}) = Var(\varphi_{1|x=t}) \cup Var(\varphi_{2|x=t}) \tag{4.1}$$

and

$$Var(\varphi_{1|x=t}) \cap Var(\varphi_{2|x=t}) = \emptyset. \tag{4.2}$$

Given a formula φ and $t \in \{0, 1\}$,

$$|M_{x=t}(\varphi)| = 2^{|S_{x=t}(\varphi)|}|M(\varphi_{x=t})|. \tag{4.3}$$

Definition 4.1. Given a formula φ and $x \in Var(\varphi)$, the *charge* of x relative to φ, is the ordered pair (ℓ, η) such that $\ell = |M_{x=1}(\varphi)|$ and $\eta = |M_{x=0}(\varphi)|$. This pair is denoted as $\#sat(\varphi, x)$.

Definition 4.2. Given a formula φ and $x, y \in Var(\varphi)$, the *conjoint charge* of x relative to φ is the matrix $(m_{i,j})$ such that $(m_{i,j}) = |M_{x=i,y=j}(\varphi)|$ where $i, j \in \{0, 1\}$. This matrix is denoted as $\#sat(\varphi, x, y)$.

Remark 1. Let A_i stand for the set of values in $\{0, 1\}^n$ where the restriction $r_i(x_1, x_2, \ldots, x_k)$ holds for $i \in \{0, 1\}$. If A_1 and A_2 are disjoint sets then $M_{r_1 \cup r_2}(\varphi) = M_{r_1}(\varphi) \cup M_{r_2}(\varphi)$, on that account

$$|M_{r_1 \cup r_2}(\varphi)| = |M_{r_1}(\varphi)| + |M_{r_2}(\varphi)|. \tag{4.4}$$

The following equalities, where x and y are variables of φ, are examples of the prior statement:

1. $|M(\varphi)| = |M_{x=1}(\varphi)| + |M_{x=0}(\varphi)|$
2. $|M(\varphi)| = |M_{x=1,y=1}(\varphi)| + |M_{x=1,y=0}(\varphi)| + |M_{x=0,y=1}(\varphi)| + |M_{x=0,y=0}(\varphi)|$
3. $|M_{x=0}(\varphi)| = |M_{x=0,y=1}(\varphi)| + |M_{x=0,y=0}(\varphi)|$
4. $|M_{x=1}(\varphi)| = |M_{x=1,y=1}(\varphi)| + |M_{x=1,y=0}(\varphi)|$.

Resultantly, the following matrix relation can be established

$$\#sat(\varphi, x) = \#sat(\varphi, x, y) \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \tag{4.5}$$

Therefore, the #SAT problem can be seen as the problem of finding the charge of a variable of an instance. Also note that the charges of the variables of a formula φ provide information which is more accurate than the one provided by the number $|M(\varphi)|$. In fact, not only the number of models of the formula is known, but also the number of models where a certain variable takes the value 1 and the number of models where this same variable takes the value 0. This information helps to obtain direct algorithms and more concise proofs on the results presented in this work.

The following definition presents a description of an operation that is directly related to model counting. This operation has the pecularity of relating charges of variables and allow certain reductions in a given formula.

Definition 4.3. Let $A = (a_{ij})$ and $B = (b_{ij})$ be $m \times n$ matrices with entries in $\mathbb{N}$. The *Hadamard product* of A and B is defined by

$$A \odot B = (a_{ij} b_{ij}) \text{ for all } 1 \leq i \leq m, 1 \leq j \leq n.$$

As it can be seen, the Hadamard product is simply entrywise multiplication. Because of this, it inherits the same benefits (and restrictions) of multiplication in the set of natural numbers.

A component of a formula φ is a maximal connected subformula of φ. If φ_1 and φ_2 are different components of φ, it is clear that

$$|M(\varphi) = |M(\varphi_1)||M(\varphi_2)|. \tag{4.6}$$

For simplicity, it can be assumed that every formula is connected. The complexity in time of a procedure for an arbitrary formula is not affected, since the procedures to determine the components can be carried out in linear time.

The next lemma states that if the charges of one variable with respect to two formulas that have only this variable in common is known, then the charge of the union of the formulas can be calculated by the Hadamard of product of the respective charges.

Lemma 4.1. *If* $Var(\varphi_1) \cap Var(\varphi_2) = \{x\}$ *then*

$$\#sat(\varphi_1 \cup \varphi_2, x) = \#sat(\varphi_1, x) \odot \#sat(\varphi_2, x).$$

Proof. By definition $\#sat(\varphi_1 \cup \varphi_2, x) = (|M_{x=0}(\varphi_1 \cup \varphi_2)|, |M_{x=1}(\varphi_1 \cup \varphi_2)|)$. Knowing that $Var(\varphi_{1|x=t}) \cap Var(\varphi_{2|x=t}) = \emptyset$ for $t \in \{0, 1\}$ and using equation (4.3) the following is acomplished

$$\begin{aligned}
|M_{x=t}(\varphi_1 \cup \varphi_2)| &= 2^{|S_{x=t}(\varphi_1 \cup \varphi_2)|}|M((\varphi_1 \cup \varphi_2)_{x=t})| \\
&= 2^{|S_{x=t}(\varphi_1)|+|S_{x=t}(\varphi_2)|}|M(\varphi_{1|x=t} \cup \varphi_{2|x=t})| \\
&= 2^{|S_{x=t}(\varphi_1)|}2^{|S_{x=t}(\varphi_2)|}|M(\varphi_{1|x=t})||M(\varphi_{2|x=t})| \\
&= 2^{|S_{x=t}(\varphi_1)|}|M(\varphi_{1|x=t})|2^{|S_{x=t}(\varphi_2)|}|M(\varphi_{2|x=t})| \\
&= |M_{x=t}(\varphi_1)||M_{x=t}(\varphi_2)|.
\end{aligned}$$

In that event

$$\begin{aligned}
\#sat(\varphi_1 \cup \varphi_2, x) &= (|M(\varphi_{1|x=1})||M(\varphi_{2|x=1})|, |M(\varphi_{1|x=0})||M(\varphi_{2|x=0})|) \\
&= (|M(\varphi_{1|x=1})||M(\varphi_{2|x=0})|) \odot (|M(\varphi_{1|x=1})||M(\varphi_{2|x=0})|) \\
&= \#sat(\varphi_1, x) \odot \#sat(\varphi_2, x).
\end{aligned}$$

□

There is an analogous result for the case of conjoint charges of two variables with respect to the union of two formulas that have only one variable in common.

Lemma 4.2. *If* $Var(\varphi_1) \cap Var(\varphi_2) = \{x\}, w \in Var(\varphi_1) \setminus Var(\varphi_2)$ *and* $z \in Var(\varphi_2) \setminus Var(\varphi_1)$ *then*

$$\#sat(\varphi_1 \cup \varphi_2, w, z) = \#sat(\varphi_1, w, x)\#sat(\varphi_2, x, z).$$

Proof. To make the notation easier to follow, let $a_{ij} = |M_{w=i,x=j}(\varphi_1)|$, $b_{ij} = |M_{x=i,z=j}(\varphi_2)|$, and $c_{ij} = |M_{w=i,z=j}(\varphi_1 \cup \varphi_2)|$, which means that the entries of $\#sat(\varphi_1, w, x)$, $\#sat(\varphi_2, x, z)$, and $\#sat(\varphi_1 \cup \varphi_2, w, z)$ are a_{ij}, b_{ij}, and c_{ij}, respectively. From remark 1 ,

$$M_{w=i,z=j}(\varphi_1 \cup \varphi_2) \;=\; M_{w=i,x=1,z=j}(\varphi_1 \cup \varphi_2) \;\cup\; M_{w=i,x=0,z=j}(\varphi_1 \cup \varphi_2).$$

Since $\varphi_{1|x=t}$ and $\varphi_{2|x=t}$ have no variables in commmon for each for $t \in \{0, 1\}$, by equation (4.6) the equality $|M_{w=i,x=t,z=j}(\varphi_1 \cup \varphi_2)| = a_{it}b_{tj}$ holds. As a result, it is clear that $c_{ij} = a_{i1}b_{i1} + a_{i0}b_{i0}$. □

4.2 Counting for chains

Using a truth table, a counting operator for formulas whose associated hypergraphs represent simple chains can be identified. This operator is proposed in the following definition.

Definition 4.4. Let $\mathbb{S} : \mathbb{N}^2 \longrightarrow \mathbb{N}^2$ be the operator defined by

$$\mathbb{S} = \begin{pmatrix} 2 & 2 \\ 2 & 1 \end{pmatrix},$$

$\mathbb{S}$ is called the *edge operator*.

4.2.1 Simple chains

If a clause having exactly three variables is added to a formula φ which shares one variable with the clause, it is possible to compute the charge of the resulting formula as stated by the following lemma.

Lemma 4.3. *Let* φ *be an arbitrary formula and* $c = \{x, y, z\}$ *a clause such that* $Var(\varphi) \cap c = \{x\}$, *then* $\#sat(\varphi \cup c, z) = \#sat(\varphi \cup c, y) = \mathbb{S}\#sat(\varphi, x)$, *where* $\mathbb{S}$ *is the edge operator.*

Proof. Because φ and c are both formulas satisfying the conditions of Lemma 4.1, then

$$\#sat(\varphi \cup c, x) = \#sat(\varphi, x) \odot \#sat(c, x).$$

It can be easily computed that $\#sat(c, x) = (4, 3)$. Also, let (ℓ, η) stand for $\#sat(\varphi, x)$, therefore

$$\#sat(\varphi \cup c, x) = (4, 3) \odot (\ell, \eta) = (4\ell, 3\eta).$$

This means that $|M_{x=1}(\varphi \cup c)| = 4\ell$ and $|M_{x=0}(\varphi \cup c)| = 3\eta$.

Each model $\sigma \in M_{x=1}(\varphi \cup c)$ takes two possible values in y (as well as in z), $\sigma(y) = 1$ or $\sigma(y) = 0$. Hence $|M_{x=1,y=1}(\varphi \cup c)| = 2\ell$ and $|M_{x=1,y=0}(\varphi \cup c)| = 2\eta$. Conjointly, for each model $\sigma \in M_{x=0}(\varphi \cup c)$ there are three possible values of σ in the variables y and z: $\sigma(y) = 0 \wedge \sigma(z) = 1$, $\sigma(y) = 1 \wedge \sigma(z) = 0$, and $\sigma(y) = 1 \wedge \sigma(z) = 1$. That is to say, two thirds of $M_{x=0}(\varphi \cup c)$ are models which take the value 1 in the variable y. By all means, the remaining models take the value 0 in this particular variable. It follows that $|M_{x=0,y=1}(\varphi \cup c)| = 2\eta$ and $|M_{x=0,y=0}(\varphi \cup c)| = \eta$.

So, the charge of y relative to φ is

$$\begin{aligned}\#sat(\varphi \cup c, y) &= (|M_{y=1}(\varphi \cup c, y)|, |M_{y=0}(\varphi \cup c, y)|) \\ &= (2\ell + 2\eta, 2\ell + \eta) \\ &= \mathbb{S}\#sat(\varphi, x).\end{aligned}$$

In the same manner, $\#sat(\varphi \cup c, z) = \mathbb{S}\#sat(\varphi, x)$. □

The following theorem establishes how to count the number of models of a formula whose associated hypergraph is a simple chain.

Theorem 4.1. *Let $\varphi_m = c_1 c_2 \cdots c_m$, be a simple chain in $2\mu - e3MON$. Suppose that $c_m = \{x, y, z\}$ and $x \in c_m \cap c_{m-1}$, then*

$$\#sat(\varphi_m, z) = \#sat(\varphi_m, y) = \mathbb{S}^m \begin{pmatrix} 1 \\ 1 \end{pmatrix};$$

and if $w \in \varphi_{m-1} \setminus c_m$, then

$$\#sat(\varphi_m, w, z) = \mathbb{S}^m,$$

where $\mathbb{S}$ *is the edge operator.*

Proof. By induction over m. It is trivially true in the base case $m = 1$. For the induction step, assume the theorem is valid for $m - 1$, that is

$$\#sat(\varphi_{m-1}, x) = \mathbb{S}^{m-1}\begin{pmatrix}1\\1\end{pmatrix} \quad \text{and} \quad \#sat(\varphi_m, w, x) = \mathbb{S}^{m-1}.$$

Consider the formula $\varphi_{m-1} = c_1 c_2 \cdots c_{m-1}$ and the clause c_m, they fulfill the conditions of Lemma 4.3, in consequence

$$\begin{aligned}\#sat(\varphi_m, z) = \#sat(\varphi_{m-1} \cup c_m, z) &= \mathbb{S}\#sat(\varphi_{m-1}, x)\\ &= \mathbb{S}\mathbb{S}^{m-1}\begin{pmatrix}1\\1\end{pmatrix} = \mathbb{S}^m\begin{pmatrix}1\\1\end{pmatrix}.\end{aligned}$$

At the same time, by Lemma 4.2

$$\begin{aligned}\#sat(\varphi_m, w, z) &= \#sat(\varphi_{m-1} \cup c_m, w, z)\\ &= \#sat(\varphi_{m-1,}w, x)\#sat(c_m, x, z)\\ &= \mathbb{S}^{m-1}\mathbb{S}.\end{aligned}$$

□

Given two hypergraphs with a single node in common, by Lemma 4.1, it is possible to calculate the charge of the node with respect to the union of the hypergraphs. This means that if the charge of the node relative to each of the substructures is known, then the charge of the node in the complete structure is obtainable. So, an important consideration when analyzing model counting based on substructures is to calculate the distribution of charges of the nodes in each substructure, or at least those nodes useful to relate a substructure with another. A case in point is the analysis of the distribution of the charges of the variables y and z of clause $c = \{x, y, z\}$ knowing the charge of the variable x with respect to a formula φ such that $Var(\varphi) \cap c = \{x\}$. This has been already stated in Lemma 4.3.

Example 4.1. Compute the distribution of charges of x_4 and x_5 with respect to the formula $\varphi = \{c_1, c_2, c_3\}$ where $c_1 = \{x_1, x_2, x_3\}$, $c_2 = \{x_1, x_4, x_5\}$ and $c_3 = \{x_1, x_6, x_7\}$.

Solution. From Lemma 4.1,

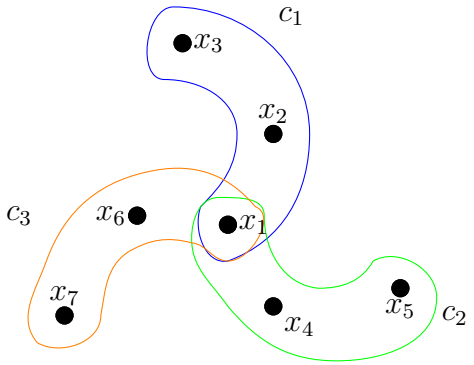


Figure 4.1: Hypergraph associated with $\varphi = \{c_1, c_2, c_3\}$.

$$\begin{aligned} \#sat(c_1 \cup c_2, x_1) &= \#sat(c_1, x_1) \odot \#sat(c_2, x_1) \\ &= \begin{pmatrix} 4 \\ 3 \end{pmatrix} \odot \begin{pmatrix} 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 16 \\ 9 \end{pmatrix} \end{aligned}$$

and, by Lemma 4.3

$$\begin{aligned} \#sat(c_1c_2 \cup c_3, x_4) &= \#sat(c_1c_2 \cup c_3, x_5) = \mathbb{S}\#sat(\varphi, x_1) \\ &= \begin{pmatrix} 2 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 16 \\ 9 \end{pmatrix} = \begin{pmatrix} 50 \\ 41 \end{pmatrix}. \end{aligned}$$

Therefore, there are 91 models that satisfy the formula φ. Observe that in this case φ is not a chain and does not belong to $2\mu - 3MON$.

4.2.2 Alternating chains

In this section we analyze model counting on $2\mu - e3MON$ formulas whose hypergraphs correspond to alternating chains. The following definition clarifies this concept.

Definition 4.5. (a) Two clauses c_1 and c_2 are said to be *simply linked* if and only if $|c_1 \cap c_2| = 1$.

(b) Clauses c_1 and c_2 are *doubly linked* if and only if $|c_1 \cap c_2| = 2$.

(c) A formula φ is an *alternating chain* if it can be writen as $\varphi = c_1c_2 \cdots c_m$ where c_i and c_{i+1} are simply or doubly linked, c_{i-1} and c_{i+1} can not be doubly linked to c_i simultaneously, and $c_i \cap c_j = \emptyset$ if $|i - j| \geq 2$.

Example 4.2. Let $c_1 = \{x_1, x_2, x_3\}$, $c_1' = \{x_2, x_3, x_4\}$, $c_2 = \{x_3, x_4, x_5)\}$, $c_3 = \{x_4, x_5, x_6\}$, $c_4 = \{x_6, x_7, x_8\}$ and $c_5 = \{x_8, x_9, x_{10}\}$, the formula $\varphi = \{c_1, c_2, c_3, c_4, c_5\}$ is an alternating chain as opposed to $\varphi' = \{c_1', c_2, c_3, c_4, c_5\}$.

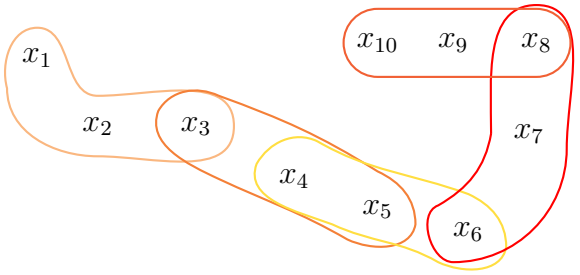


Figure 4.2: Alternating chain.

A model counting operator which acts on structures that correspond to doubly linked clauses can be determined from the analysis of the simplest structure of an alternating chain. In the following definition, this operator is detailed.

Definition 4.6. The operator $\mathbb{D} : \mathbb{N}^2 \longrightarrow \mathbb{N}^2$ defined by

$$\mathbb{D} = \begin{pmatrix} 1 & 1 \\ 3/2 & 0 \end{pmatrix}$$

is called the *double-edge operator*.

The operator $\mathbb{D}$ applied to the charge of any variable of an alternating chain always produces charges with integer entries. Indeed, as an alternating chain in $2\mu - e3MON$ cannot have two consecutive doubly linked clauses, when the operator $\mathbb{D}$ is applied to a charge (ℓ, η), there are two possibilities: either (ℓ, η) is an initial charge, meaning $(\ell, \eta) = (4, 3)$, or (ℓ, η) comes from previously applying the edge operator $\mathbb{S}$ to another charge (ℓ_1, η_1), that is $(\ell, \eta) = S(\ell_1, \eta_1) = (2\ell_1 + 2\eta_1, 2\ell_1 + \eta_1)$. In any case $\mathbb{D}(\ell, \eta)$ has integer entries.

The following lemma specifies how to calculate the charge of a variable with respect to the union of a formula φ and a clause c such that they match in two variables.

Lemma 4.4. *Given a formula φ and a clause $c = \{x, y, z\}$ such that $Var(\varphi \cup c) = \{x, y\}$, then*

$$\#sat(\varphi \cup c, z) = (|M(\varphi)|, |M(\varphi)| - a_{00})$$

and

$$\#sat(\varphi \cup c, x) = (\mathbb{S} \odot \#sat(\varphi, x, y)) \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Proof. By definition $\#sat(\varphi \cup c, z) = (|M_{z=1}(\varphi \cup c)|, |M_{z=0}(\varphi \cup c)|)$. Let $a_{ij} = |M_{x=i,y=j}(\varphi)|$ be the elements of the matrix $\#sat(\varphi, x, y)$. On one hand, if $z = 1$ under an extension of a model in $M(\varphi)$, clause c is valid regardless of the values of the variables x and y. Hence there are $a_{11} + a_{10} + a_{01} + a_{00} = |M(\varphi)|$ models in $M(\varphi \cup c)$ where z takes the value 1. On the other hand, if z takes the value 0 under another extension of a model in $M(\varphi)$, then under this same extension x or y must take the value 1 for clause c to be valid. As a result there are $a_{11} + a_{10} + a_{01} = |M(\varphi)| - a_{00}$ models in $M(\varphi \cup c)$ where z takes the value 0. Thus $\#sat(\varphi \cup c, z) = (|M(\varphi)|, |M(\varphi)| - a_{00})$.

For the latter part of the lemma, it follows from the above that in $M(\varphi \cup c)$ there are $2a_{11} + 2a_{10}$ models where the variable x takes the value 1 and $2a_{01} + a_{00}$ models where it takes the value 0. This is

$$\begin{aligned}
\#sat(\varphi \cup c, x) &= (|M_{x=1}(\varphi \cup c)|, |M_{x=0}(\varphi \cup c)|) \\
&= (2a_{11} + 2a_{10}, 2a_{01} + a_{00}) \\
&= \left(\begin{pmatrix} 2 & 2 \\ 2 & 1 \end{pmatrix} \odot \begin{pmatrix} a_{11} & a_{10} \\ a_{01} & a_{00} \end{pmatrix} \right) \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\
&= (\mathbb{S} \odot \#sat(\varphi, x, y)) \begin{pmatrix} 1 \\ 1 \end{pmatrix}.
\end{aligned}$$

□

The upcoming theorem determines the model counting for alternating chains.

Theorem 4.2. *Given an alternating chain in $2\mu - e3MON$, $\varphi = c_1 c_2, \cdots, c_m$ and $x \in c_m \setminus c_{m-1}$, $\#sat(\varphi_m, x)$ is obtained applying the recurrence equation*

$$\begin{cases} \mathbf{q_1} = (4, 3) \\ \mathbf{q_i} = \Delta \mathbf{q_{i-1}} \quad \textit{for} \quad i = 2, 3, \ldots m, \end{cases} \tag{4.7}$$

where $\mathbf{q_1} = \#sat(\varphi_1, x_1), x_1 \in c_1$, $\mathbf{q_i} = \#sat(\varphi_i, x_i)$, $x_i \in c_i \setminus c_{i-1}$, and Δ is the operator defined as

$$\Delta = \begin{cases} \mathbb{S} & \textit{if } c_i \textit{ and } c_{i-1} \textit{are simply linked} \\ \mathbb{D} & \textit{if } c_i \textit{ and } c_{i-1} \textit{are doubly linked.} \end{cases}$$

Proof. A proof of the theorem is about to be performed by mathematical induction over m. If $m = 1$, then there is nothing to prove. For $i = 1, 2, \ldots, 3$ there are two possible cases.

Case 1 c_{i-1} and c_i are simply linked. Let $c_i = \{x, y, z\}$ and assume $x \in c_i \setminus c_{i-1}$. By virtue of Lemma 4.3, it directly follows that $\mathbf{q_i} = \mathbb{S}\mathbf{q_{i-1}}$.

Case 2 c_{i-1} and c_i are double linked. In this case, it must happen that c_{i-2} and c_{i-1} are simply linked. Assume $c_i = \{x, y, z\}$ and $c_{i-1} = \{w, x, y\}$ and let (ℓ, η)

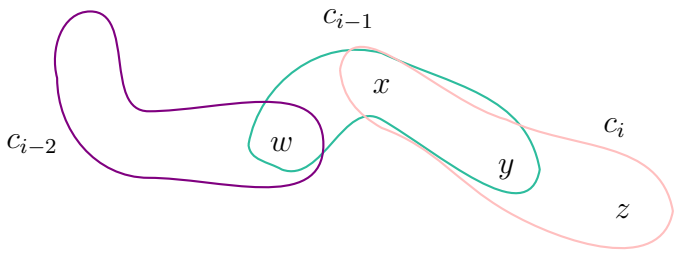


Figure 4.3: Alternating chain, case 2 of Theorem 4.2.

be the charge of w relative to the chain φ_{i-2}. Notice that $y \in c_{i-1} \setminus c_{i-2}$. On one hand, because of the first case, the charge of y relative to φ_{i-1} is $(2\ell + 2\eta, 2\ell + \eta)$, therefore $|M(\varphi_{i-1})| = 4\ell + 3\eta$. Now, the last lemma assures that

$$\mathbf{q_i} = \#sat(\varphi_{i-1} \cup c_i, z) = (|M(\varphi_{i-1})|, |M(\varphi_{i-1})| - a_{00}),$$

where a_{00} is the number of models of φ_{i-1} where x and y take the value 0. On the other hand, $|M(\varphi_{i-1})| = 4\ell + 3\eta$ and

$$\begin{aligned} |M(\varphi_{i-1})| - a_{00} &= |M_{z=0}(\varphi_{i-1} \cup c_i)| = |M_{x\neq 0 \vee y \neq 0}(\varphi_{i-1})| \\ &= 3|M(\varphi_{i-2})| = 3\ell + 3\eta. \end{aligned}$$

This way, the charge of z relative to $\varphi_{i-1} \cup c_i$ is

$$\mathbf{q_i} = (4\ell + 3\eta, 3\ell + 3\eta) = \begin{pmatrix} 1 & 1 \\ 3/2 & 0 \end{pmatrix} \begin{pmatrix} \ell + 2\eta \\ 2\ell + \eta \end{pmatrix} = \mathbb{D}\mathbf{q_{i-1}}.$$

□

An example to illustrate this theorem is hereupon given.

Example 4.3. Consider the alternating chain $\varphi = \{\{x_1, x_2, x_3\}, \{x_2, x_3, x_4\}, \{x_4, x_5, x_6\}, \{x_6, x_7, x_8\}, \{x_7, x_8, x_9\}\}$. Using the recurrence equation (4.7)

$$\mathbf{q_1} = \begin{pmatrix} 4 \\ 3 \end{pmatrix},$$

$$\mathbf{q_2} = \begin{pmatrix} 1 & 1 \\ 3/2 & 0 \end{pmatrix} \begin{pmatrix} 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6 \end{pmatrix},$$

$$\mathbf{q_3} = \begin{pmatrix} 2 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 7 \\ 6 \end{pmatrix} = \begin{pmatrix} 26 \\ 20 \end{pmatrix},$$

$$\mathbf{q_4} = \begin{pmatrix} 2 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 26 \\ 20 \end{pmatrix} = \begin{pmatrix} 92 \\ 72 \end{pmatrix}, \text{ and}$$

$$\mathbf{q_5} = \begin{pmatrix} 1 & 1 \\ 3/2 & 0 \end{pmatrix} \begin{pmatrix} 92 \\ 72 \end{pmatrix} = \begin{pmatrix} 164 \\ 138 \end{pmatrix},$$

where $\mathbf{q}_1$ is the charge of variables x_1, x_2, and x_3; $\mathbf{q}_2$ is the charge of variable x_4; $\mathbf{q}_3$ is the charge of variables x_5 and x_6; $\mathbf{q}_4$ is the charge of variables x_7 and x_8; and $\mathbf{q}_5$ is the charge of variable x_9. It is concluded that there are 302 models satisfying the formula φ.

4.3 Counting for cycles

In this section we distinguish two types of cycles in $2\mu - e3MON$, simple cycles and alternating cycles.

Definition 4.7. A hypergraph of the form $\varphi = c_1 c_2 \cdots c_m$ is a *simple cycle* if $c_1 c_2 \cdots c_m$ is a simple chain and $|c_1 \cap c_m| = 1$; and it is an *alternating cycle* if $c_1 c_2 \cdots c_m$ is an alternating chain and $1 \leq |c_1 \cap c_m| \leq 2$.

Thereupon, model counting is performed on formulas whose associated hypergraphs represent specific cases of α-cycles.

4.3.1 Simple Cycles

From the analysis of simple structures and with the help of Hadamard's product, the following theorem is reached.

Theorem 4.3. *Let $\varphi_m = c_1 c_2 \cdots c_{m-1} c_m$ be a simple cycle in $2\mu - e3MON$, then*

$$\#sat(\varphi_m, x) = (\mathbb{S} \odot \mathbb{S}^{m-1}) \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

where $x \in c_1 \cap c_m$ and $\mathbb{S}$ is the edge operator.

Proof. Consider the formula $\varphi_{m-1} = c_1 c_2 \dots c_{m-1}$ and clause $c_m = \{x, y, z\}$ such that $\varphi_{m-1} \cap c_m = \{x, z\}$. Lemma 4.4 guarantees that

$$\begin{aligned} \#sat(\varphi, x) &= \#sat(\varphi_{m-1} \cup c_m, x) \\ &= (\mathbb{S} \odot \#sat(\varphi_{m-1}, x, z)) \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \end{aligned}$$

The formula φ_{m-1} is evidently a simple chain in 2μ-$e3MON$. Also, $z \in c_{m-1} \setminus \varphi_{m-2}$ and $x \in \varphi_{m-2} \setminus c_{m-1}$. Therefore by Theorem 4.1

$$\#sat(\varphi_{m-1}, x, z) = \mathbb{S}^{m-1},$$

eventually $\#sat(\varphi, x) = (\mathbb{S} \odot \mathbb{S}^{m-1}) \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. □

Example 4.4. Compute the number of models of the formula whose associated hypergraph is the following.

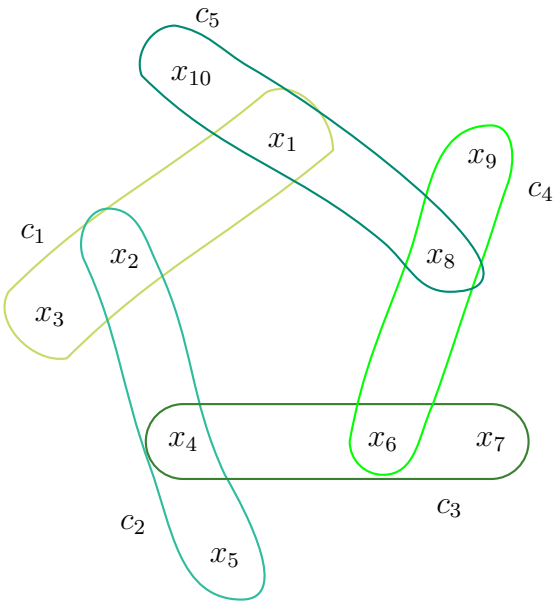


Figure 4.4: Simple cycle.

Solution.
Unmistakably, the given hypergraph represents a simple cycle whose formula is $\varphi = \{\{x_1, x_2, x_3\}, \{x_2, x_4, x_5\}, \{x_4, x_6, x_7\}, \{x_6, x_8, x_9\}, \{x_8, x_{10}, x_1\}\}$. According to the previous theorem, for $x_1 \in c_1 \cap c_5$

$$\begin{aligned}
\#sat(\varphi, x_1) &= (\mathbb{S} \odot \mathbb{S}^4) \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\
&= \left(\begin{pmatrix} 2 & 2 \\ 2 & 1 \end{pmatrix} \odot \begin{pmatrix} 100 & 78 \\ 78 & 61 \end{pmatrix} \right) \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 200 & 156 \\ 156 & 61 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 356 \\ 217 \end{pmatrix}.
\end{aligned}$$

In consequence there are 573 models for φ.

4.3.2 Alternating Cycles

Given the alternating chain $c_1 c_2 \cdots c_m$, the operator Δ_i is defined as $\Delta_1 = \mathbb{S}$ and for $i \in \{2, \ldots, m\}$,

$$\Delta_i = \begin{cases} \mathbb{S} & \text{if } c_i \text{ and } c_{i-1} \text{ are simply linked} \\ \mathbb{D} & \text{otherwise.} \end{cases}$$

For each $k \in \{1, 2, \ldots, m\}$, Ω_k represents the product $\prod_{i=1}^{k} \Delta_i$, and J_1, J_2 denote the matrices $\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$ and $\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$, respectively.

Theorem 4.4. *Let $\varphi_m = c_1 c_2 \cdots c_m$ be an alternating cycle in $2\mu - e3MON$ and $x \in c_1 \cap c_m$. If c_{m-1} and c_m are simply linked then*

$$\#sat(\varphi, x) = (\mathbb{S} \odot \Omega_{m-1}) \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

and

$$\#sat(\varphi, x) = (\Omega_{m-1} - J_1\Omega_{m-2}J_2)\begin{pmatrix}1\\1\end{pmatrix}$$

otherwise.

Proof. Let $\varphi_{m-2} = c_1c_2\ldots c_{m-2}$ and $\varphi_{m-1} = c_1c_2\ldots c_{m-1},$. Naturally, the alternating chain φ_m can be written in terms of the aforementioned chains as

$$\varphi_m = \varphi_{m-1} + c_m = \varphi_{m-2} + c_{m-1} + c_m.$$

By Theorem 4.2,

$$\#sat(\varphi_{m-1}, x) = \Omega_{m-1}\begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}a_{11} & a_{10}\\ a_{01} & a_{00}\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}$$

and

$$\#sat(\varphi_{m-2}, x) = \Omega_{m-2}\begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}b_{11} & b_{10}\\ b_{01} & b_{00}\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}.$$

From definition 4.2 and remark 1, it is known that

$$a_{ij} = |M_{x=i,z=j}(\varphi_{m-1})| \quad \text{and} \quad b_{ij} = |M_{x=i,w=j}(\varphi_{m-1})|.$$

The situation can be divided into two cases:
Case 1 c_{m-1} and c_m are simply linked. The proof is followed step by step from the proof of Theorem 4.3 by replacing $\mathbb{S}^{m-1}$ with Ω_{m-1}.
Case 2 c_{m-1} and c_m are double linked. This means that c_{m-1} and c_{m-2} must be simply linked. Assume $w \in c_{m-1} \cap c_{m-2}$ and let $c_{m-1} = \{w, y, z\}$, on that account $c_m = \{x, y, z\}$.
Since clause c_m is a restriction that must satisfy a model of φ_{m-1} which is also a model of φ_m, it happens that $M(\varphi_m) \subseteq M(\varphi_{m-1})$.

Because the assignments $x = 1 \wedge z = 1$, $x = 1 \wedge z = 0$, and $x = 0 \wedge z = 1$ make the clause c_m valid, the following equalities hold.

$$\begin{aligned}|M_{x=1,z=1}(\varphi_m)| &= a_{11}\\ |M_{x=1,z=0}(\varphi_m)| &= a_{10}\\ |M_{x=0,z=1}(\varphi_m)| &= a_{01}.\end{aligned}$$

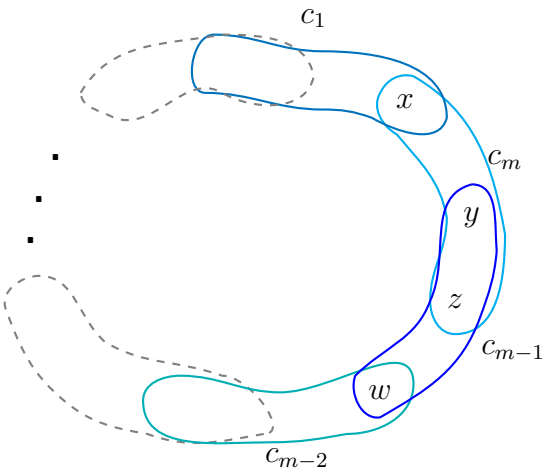


Figure 4.5: Alternating cycle, case 2 of Theorem 4.4.

It is also unquestionable that

$$\begin{aligned} M_{x=0,z=0}(\varphi_m) &= M_{x=0,y=1}(\varphi_{m-1}) \\ &= M_{x=0,z=0}(\varphi_{m-2}) \cup M_{x=0,z=1}(\varphi_{m-2}). \end{aligned}$$

As a consequence $|M_{x=0,z=0}(\varphi_m)| = b_{00} + b_{01}$.

Given that c_{m-1} and c_{m-2} are simply linked, by virtue of Theorem 4.2, the equality $\Omega_{m-1} = \mathbb{S}\Omega_{m-2}$ is true, thereupon $a_{00} = b_{00} + 2b_{01}$.
Under these circumstances,

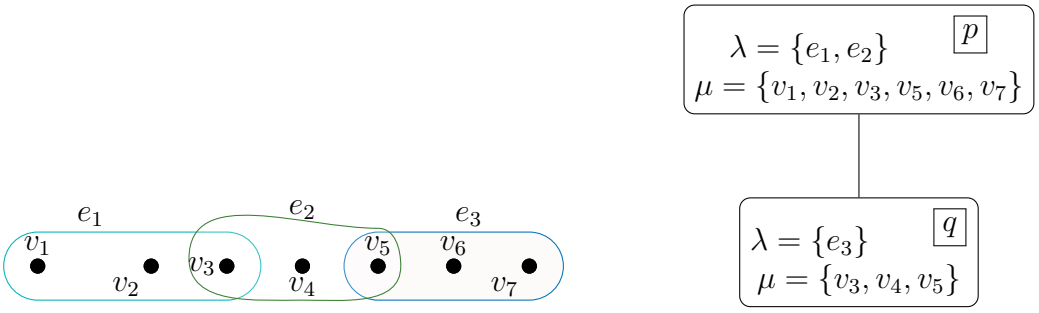
$$\begin{aligned} \#sat(\varphi_m, x) &= \begin{pmatrix} a_{11} & a_{10} \\ a_{01} & b_{00} + b_{01} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} a_{11} & a_{10} \\ a_{01} & a_{00} + b_{01} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ &= \left[\begin{pmatrix} a_{11} & a_{10} \\ a_{01} & a_{00} \end{pmatrix} - \begin{pmatrix} 0 & 0 \\ 0 & b_{01} \end{pmatrix} \right] \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \end{aligned}$$

Straightforward calculations show that

$$\begin{aligned} \begin{pmatrix} 0 & 0 \\ 0 & b_{01} \end{pmatrix} &= \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} b_{11} & b_{10} \\ b_{01} & b_{00} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \\ &= J_1 \Omega_{m-2} J_2. \end{aligned}$$

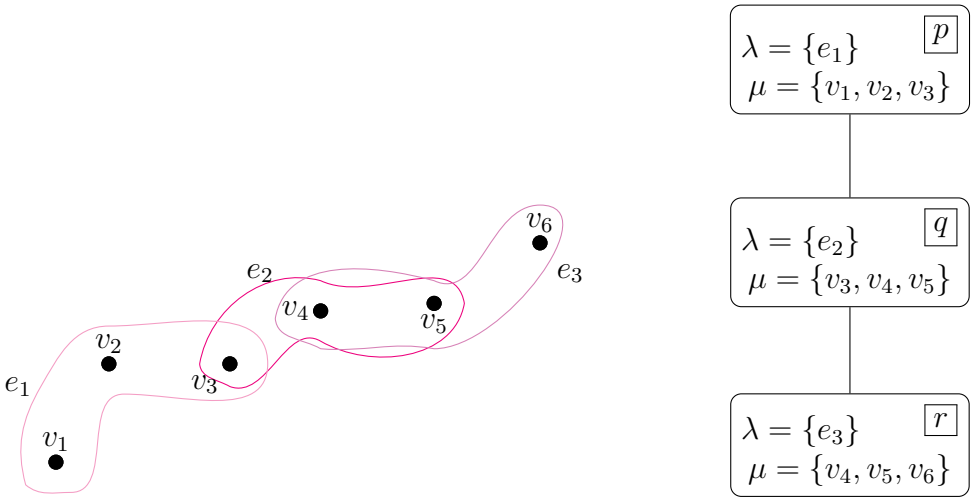
As a conclusion $\#sat(\varphi, x) = (\Omega_{m-1} - J_1\Omega_{m-2}J_2)\begin{pmatrix}1\\1\end{pmatrix}$. □

Theorem 4.3 is important as it ensures that as long as a formula in CNF allows a disjoint branches decomposition of its associated hypergraph then there exists an algorithm that computes the number of models of the formula in polynomial time. In fact, simple chains, alternating chains, simple cycle, and alternating chains admit disjoint branches decomposition. Examples of some of them are displayed below.



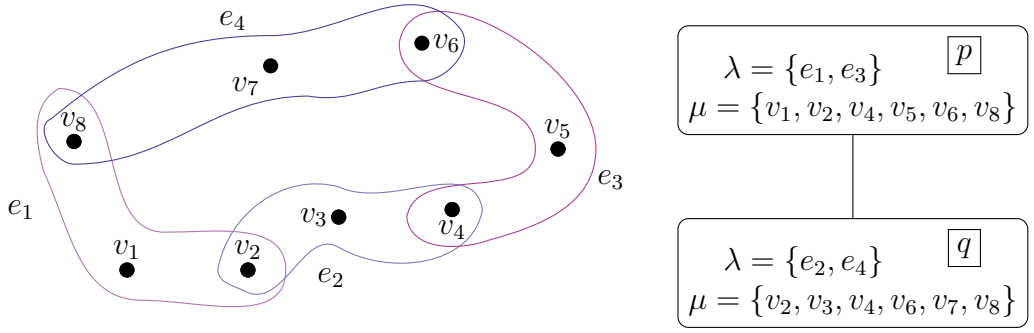
(a) Simple chain **(b)** (T, μ, λ)

Figure 4.6: A simple chain and its tree decomposition of width 2.



(a) Alternating chain **(b)** (T, μ, λ)

Figure 4.7: An alternating chain and its tree decomposition of width 1.



(a) Simple Cycle **(b)** (T, μ, λ)

Figure 4.8: A simple cycle and its tree decomposition of width 2.

Conclusions

A tractable subclass of the syntactic class $\#2\mu - e3MON$ is obtained by means of methods based on matrix counting operators that act on the structures of single chains, alternating chains, simple cycles and alternating cycles of the hypergraph associated with the formula. All the cases presented here lead us to treatable algorithms, given that identifying whether a CNF formula belongs to one of the studied classes can be done in quadratic time. In all the cases studied the calculation of the number of models is reduced to the multiplication of at most m matrices of size 2×2, where m is the number of clauses of the formula, thus the multiplication of matrices can be done in linear time with respect to m. Much remains to be done in this topic. As a future work, in the direction of the identification of tractable structures in hypergraphs, new structures can be recognized if the idea of matrix transfer methods applied to graphs is generalized to hypergraphs.

Bibliography

[BB16] Johann Brault-Baron. Hypergraph acyclicity revisited. *ACM Computing Surveys (CSUR)*, 49(3):54:1–54:26, December 2016. 28, 29, 33

[BBCM15] Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for β-acyclic CNF-formulas. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 143–156, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BC94] Daniel Pierre Bovet and Pierluigi Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall, 1994.

[BDG95] José L. Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity*. Springer, Verlag Berlin Heidelberg, 2nd edition, 1995.

[Ber73] Claude Berge. *Graphs and Hypergraphs*. North-Holland mathematical library, volume 6. American Elsevier Publishing Company; Revised edition, 2 edition, 1973. 32

[Ber89] Claude Berge. *Hypergraphs*. Elsevier Science, North-Holland, Amsterdam, 2nd edition, 1989.

[CDM14] Florent Capelli, Arnaud Durand, and Stefan Mengel. Hypergraph acyclicity and propositional model counting. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 399–414, Cham, 2014. Springer International Publishing. viii, 35, 36

[Die10] Reihard Diestel. *Graph Theory*. Springer, 4 edition, 2010. 1

[Dur12] David Duris. Some characterizations of gamma and beta-acyclicity of hypergraphs. *Information Processing Letters*, 112(16):617–620, 2012. 29, 31

[Fag83] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the Association for Computing Machinery*, 30(3):514–550, 1983. 28, 29, 33

[FMR08] E. Fischer, J.A. Makowsky, and E.V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511 – 529, 2008. Third Haifa Workshop on Interdisciplinary Applications of Graph Theory, Combinatorics & Algorithm.

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

[Gal08] Carlos Guillén Galván. *Diseño de Algoritmos Combinatorios para #SAT y su Aplicación al Razonamiento Proposicional*. PhD thesis, Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Pue., 2008.

[GLA13] Carlos Guillén, Rafael Lemuz, and Irene Ayaquica. Conteo de modelos en la clase sintáctica 2mu-3MON. *Computación y Sistemas*, 17(4):501–513, 2013. 37

[GLS02] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64:579–627, May 2002. 23

[GP04] Georg Gottlob and Reinhard Pichler. Hypergraphs in Model Checking: Acyclicity and Hypertree-Width versus Clique-Width. *SIAM Journal on Computing*, 33(2):351–378, 2004.

[GS08] Georg Gottlob and Marko Samer. A backtracking-based algorithm for computing hypertree-decompositions. *Journal of Experimental Algorithmics (JEA)*, 13(1):1:1–1:19, 2008.

[JN09] Philippe Jégou and Samba Ndojh Ndiaye. On the notion of cycles in hypergraphs. *Discrete Mathematics*, 309(23):6535 – 6543, 2009. 29

[Men15] Elliott Mendelson. *Introduction to Mathematical Logic*. CRC Press, 6 edition, 2015.

[Tru94] Richard J. Trudeau. *Introduction to Graph Theory*. Dover Books on Mathematics. Dover Publications, 2 edition, 1994.