

Complexity issues in counting, polynomial evaluation and zero finding

A dissertation submitted to

École Normale Supérieure de Lyon

and

City University of Hong Kong

in partial satisfaction of the requirements
for the Degree of

Doctor of Philosophy

in

Computer Science

by

Irénée BRIQUEL

The dissertation will be defended on 29. November 2011.

Examination Panel :

President

Qiang Zhang Professor, City University of Hong Kong

Reviewers

Arnaud Durand Professor, Université Denis Diderot, Paris

Jean-Pierre Dedieu Professor, Université Paul Sabatier, Toulouse

Examiners

Nadia Creignou Professor, Université de la Méditerranée, Marseille

Jonathan Wylie Professor, City University of Hong Kong

Advisors

Pascal Koiran Professor, École Normale Supérieure de Lyon

Felipe Cucker Professor, City University of Hong Kong

Many years ago I was invited to give a lecture on what is today called "computer science" at a large eastern university. I titled my lecture "Turing machines", because the most famous abstract model of a computer is the model presented by Alan Turing. Today biographies of Turing are reviewed in the *New York Times*, but in those early days of the computer Turing was virtually unheard of. Thus it wasn't surprising that someone at the university "corrected" what he assumed to be my typographical error, with the result that posters announcing that I would give a lecture on TOURING MACHINES went up all over the campus. (A few people left rather early in the lecture).

Representation and Reality
HILARY PUTNAM

Résumé

En informatique, le modèle de calcul généralement utilisé est la *machine de Turing*. Cette dernière est une modélisation théorique de l'ordinateur digital, et est pertinente pour plusieurs raisons.

D'abord, la thèse de Church, formulée dans les années 1940, et qui est communément admise dans la communauté des informaticiens, affirme que les fonctions qui sont *effectivement calculables* sont exactement les fonctions calculées par les machines de Turing. Cette thèse repose sur le fait que de nombreuses tentatives pour formaliser la notion de calcul a conduit à des modèles calculant ou exprimant exactement les mêmes fonctions que les machines de Turing.

Et ensuite car le fonctionnement des ordinateurs actuels est assez proche de la machine de Turing. Cette machine est donc un bon modèle pour étudier non seulement si certaines fonctions sont calculables, mais aussi pour étudier leur *complexité*, c'est-à-dire les ressources en temps et en mémoire dont un ordinateur aura besoin pour calculer ces fonctions. Ce modèle de calcul est ainsi à la base de l'étude de l'efficacité des algorithmes, et donc de l'étude de la difficulté des problèmes résolus par ces algorithmes. Il a permis l'essor du domaine de recherche de la complexité de calcul, qui s'intéresse à classer les problèmes en fonction de leur difficulté, en définissant des classes de problèmes de complexité comparable, et en étudiant les inclusions entre ces classes.

La machine de Turing fonctionne par définition sur l'alphabet $\{0, 1\}$, ou de manière équivalente sur un alphabet fini. Les problèmes définis sur des structures plus riches, comme l'ensemble des réels ou des complexes, ne sont donc pas à sa portée – plusieurs théories ont été proposées pour étudier le calcul continu sur la machine de Turing (discrète), mais aujourd'hui il n'y a pas de consensus sur l'approche à suivre. En particulier, les polynômes, qui sont les fonctions calculées avec les opérations $(+, -, \times)$, ne peuvent être évalués sur les nombres réels ou complexes par des machines de Turing. La complexité algébrique a donc été développée pour permettre l'étude des questions de complexité sur n'importe quel corps. On a donc introduit différents modèles de calcul permettant de construire des algorithmes utilisant les opérations arithmétiques usuelles sur un corps donné, et formalisant ainsi la notion intuitive de calcul en mathématiques classiques continues.

Malheureusement, contrairement au cas booléen, les différents modèles de calcul introduits calculent des classes de fonction différentes, et ne permettent donc pas de faire ressortir une notion claire de calculabilité. En outre, contrairement à la machine de Turing, ces modèles ne sont pas liés à des implémentations concrètes. S'il est donc intéressant d'étudier ces différents modèles de calcul algébriques, la machine de Turing reste le modèle de calcul de référence, et il est aussi essentiel d'étudier la traduction de ces calculs en algorithmes booléens.

Dans cette thèse, nous nous concentrons sur les polynômes, qui sont au cœur de la complexité algébrique. Nous considérons deux modèles de calcul algébriques, le modèle de Valiant et la machine de Blum, Shub et Smale (BSS).

Si les modèles de Valiant et de Blum, Shub et Smale ont respectivement déjà 33 et 22 ans et ont été abondamment étudiés, la complexité algébrique reste un champ de recherche bien plus petit que la complexité booléenne classique. Pour étudier la structure des classes de complexité algébriques, il est donc naturel de partir des résultats et des questions ouvertes dans le cas booléen, et de regarder ce qu'il en est dans le contexte algébrique. La comparaison

des résultats obtenus dans ces deux domaines permet ainsi d'enrichir notre compréhension des deux théories.

La première partie de cette thèse suit cette approche. En considérant un polynôme canoniquement associé à toute formule booléenne, nous obtenons un lien entre les questions de complexité booléenne sur la formule booléenne et les questions de complexité algébrique sur le polynôme. Nous avons étudié la complexité du calcul de ce polynôme dans le modèle de Valiant en fonction de la complexité de la formule booléenne. Nous avons obtenu dans le cas algébrique des résultats comparables à certains résultats booléens, tout en observant des différences notables. Nous avons aussi pu utiliser des méthodes algébriques pour améliorer certains résultats booléens, en particulier en obtenant de meilleures réductions entre problèmes de comptage.

Mais la motivation la plus naturelle aux modèles de calcul algébriques est d'offrir un niveau d'abstraction élégant pour définir et analyser des algorithmes algébriques. La seconde partie de ma thèse suit cette approche, et va dans la direction opposée, c'est-à-dire de la complexité algébrique à la complexité booléenne. Nous nous sommes intéressés à des algorithmes nouveaux pour un problème algébrique bien connu : la recherche de zéros (ou racines) approchés d'un système de n polynômes complexes à n inconnues, initiée par Steve Smale. Ce cas où le nombre de variables est égal au nombre de polynômes est intéressant puisque d'après le théorème de Bézout, l'on a alors presque sûrement un nombre fixé de zéros isolés. Ce problème a connu de grandes avancées dans la dernière décennie, et des algorithmes efficaces ont été proposés.

Jusqu'à présent il s'agissait d'algorithmes pour la machine BSS. Nous avons étudié l'implémentabilité de ces algorithmes sur un ordinateur booléen. Cela signifie représenter les nombres complexes par des approximations de précision finie, et prendre en compte les erreurs faites à chaque calcul sur ces approximations. Nous proposons un algorithme déterministe fonctionnant en précision finie pour ce problème, et dont la complexité est polynomiale en la taille de l'entrée lorsque les polynômes sont de degré borné.

Contents

1 Preliminaries	5
1.1 Boolean formulas	5
1.2 Graphs	6
1.3 Counting complexity	10
1.4 Algebraic models of computation	10
1.4.1 Valiant's model	10
1.4.2 The BSS model	13
1.5 Reductions and completeness	14
1.5.1 #P-completeness	14
1.5.2 VNP-completeness	17
I From boolean formulas to polynomials	19
2 A dichotomy result	21
2.1 Boolean formulas	22
2.1.1 Boolean constraint satisfaction problems	22
2.1.2 Boolean dichotomy results	23
2.2 In the algebraic setting	24
2.2.1 Overview of the proof of Theorem 2.2.2	26
2.3 Monotone 2-clauses	27
2.4 Implicative 2-clauses	29
2.5 Non affine constraints	32
2.6 Affine functions with width at least 3	34
2.7 #P-completeness proofs	37
2.8 The structure of the polynomials associated to S -formulas.	41
3 A lower bound from communication complexity	44
3.1 Previous results	45
3.2 Communication complexity	47
3.3 The lower bound	49
3.4 Further results	53

II	From complex numbers to booleans	55
4	Fast Computation of Zeros of Polynomial Systems with Bounded Degree under Finite-precision	56
4.1	Introduction	56
4.2	Preliminaries	58
4.2.1	Setting and Notation	58
4.2.2	Approximate Zeros, Complexity and Data Distribution	59
4.2.3	Condition Numbers	60
4.2.4	An Adaptive Homotopy Continuation	61
4.2.5	A Finite-precision setting	63
4.2.6	Roadmap	63
4.3	Error Bounds	63
4.3.1	Basic facts	64
4.3.2	Bounding errors for elementary computations	67
4.3.3	Bounding the error in the computation of $\mu_{\text{norm}}^{-1}(q, x)$	69
4.3.4	Bounding the error on the Newton step	73
4.3.5	Bounding the error for $\Delta\tau$	74
4.3.6	Bounding the distance between \tilde{q}_τ and q_τ	76
4.3.7	Computing u	79
4.4	Analysis of the Homotopy	81
4.4.1	Bounding errors in the homotopy	83
4.4.2	Proof of Theorem 4.4.3	88
4.5	Proof of Theorem 4.1.1	90
4.6	How to compute the initial precision in ALHF	92
5	Conclusion and perspectives	94

Introduction

In computer science, the classical model of computation is the *Turing machine*, which is a convenient model for digital computers. A central belief in computer science is Church's thesis, formulated in the 1940s, which states that *effectively computable* functions are exactly the functions computable by Turing machines. This thesis relies on the fact that many attempts to formalize the notion of computation led to models computing or expressing exactly the same class of functions as Turing machines.

And indeed, today's digital computers are quite close to the Turing machine. It is a good model, not only to study the computability of functions, but also to study the *complexity* of computable functions, that is, the quantity of time or memory it takes for a computer to compute them. This model of computation gives strong foundations to study the efficiency of algorithms, and thus, to study the hardness of the problems solved by those algorithms. This led to the research field of boolean computational complexity, which focuses on the question of classifying problems according to their computational hardness, by defining classes of problems of comparable complexity, and studying the inclusions between these classes.

A defining feature to the Turing machine is that it works over $\{0, 1\}$, or equivalently over any finite alphabet. Problems expressed over structures richer than $\{0, 1\}$, such as the real or complex numbers, are out of the reach —there have been attempts to build a theory of continuous computations based on the (discrete) Turing machine but, as of today, there is no consensus on the right approach for the study of this problems. In particular polynomials, which are the functions computed with the usual operations $(+, -, \times)$, cannot be evaluated on real or complex numbers by Turing machines. The algebraic complexity theory was developed in this context, in order to study complexity issues on any field. Various algebraic models of computation have been defined allowing to construct algorithms using basic arithmetic operations in a given field, which is the intuitive notion of computation in classical continuous mathematics.

Unfortunately, contrary to the boolean case, these various algebraic computational models compute different classes of functions. Furthermore, in opposition to the Turing machine, these theoretical models do not have concrete implementations. It may thus be interesting to study different algebraic models of computation, but also to study the translation of algebraic algorithms into boolean algorithms, since the Turing machine remains the model of reference.

In the present thesis, we will focus on polynomials, which are central in algebraic complexity. We will consider two algebraic models of computations, Valiant's model and

the Blum, Shub and Smale (BSS) machine.

Even though the structure of the complexity classes on these models has already been studied for a few decades, it remains a far smaller research field than structural boolean complexity. To explore the structure of algebraic complexity classes, it is therefore natural to consider results and open questions in the boolean case, and look at their translation into the algebraic context. The comparison of the results obtained in the two settings will then boost our understanding of both complexity theories. The first part of our thesis follows this framework. Considering polynomials associated to boolean formulas, we have a bridge between boolean complexity issues on the formulas and algebraic complexity issues on the polynomials. We studied the complexity of the polynomial in Valiant's model as a function of the complexity of the boolean formula. We were able to find algebraic counterparts to some boolean results, and observed several differences between both. Along the way, we also used our algebraic works to improve some boolean results, in particular by getting better reductions.

But the most natural motivation for the algebraic models of computation is to offer an elegant level of abstraction that helps define and analyze algebraic algorithms. The second part of our thesis follows this approach, and tackles the comparison between algebraic and boolean complexity in the opposite direction. We focused on new algorithms for a well-known algebraic problem: the search of approximate zeros of complex systems of n polynomials in n variables, initiated by Steve Smale. This special case is interesting since when the number of variables equals the number of polynomials, the system has almost surely isolated zeros. The problem has known great advances in the past decade, and efficient algorithms have been designed.

Up to now, those were BSS machine algorithms. We studied the implementation of these algorithms on digital computers. This means representing the complex numbers by finite precision approximations, and taking into account the errors made in finite precision computations. We developed a deterministic finite precision algorithm for this problem, that works in average polynomial time when the polynomials have a bounded degree.

Chapter 1

Preliminaries

1.1 Boolean formulas

In the present thesis, we will consider families of boolean formulas, and their link with the algebraic complexity. We introduce here some classical definitions on boolean formulas.

Definition 1.1.1 (Boolean circuit). A *boolean circuit* C is an acyclic directed graph with labelled vertices called nodes or gates of the following type:

- variables, with indegree 0, labelled with a variable name (eg. x_i),
- negation, with indegree 1,
- conjunction, with indegree 2,
- disjunction, with indegree 2.

A single gate has outdegree 0, and is called the output gate; the other gates have unbounded outdegree.

A boolean function on the variables appearing in the circuit is recursively associated to each gate : to each variable gate is associated the value of the variable, and to each operation is associated the result of the corresponding operation on the values of the parent gates. The value of the circuit is by definition the value of the output gate.

The formulas form a restricted class of circuits where a computation can be used only once.

Definition 1.1.2 (Boolean formula). In the case when all the gates of a circuit, except the output, have outdegree 1, the circuit is called *formula*. The negation of a formula ϕ will be denoted $\bar{\phi}$, the conjunction of ϕ_1 and ϕ_2 , $\phi_1 \wedge \phi_2$ and their disjunction, $\phi_1 \vee \phi_2$.

The possibility of sharing the result of a gate for several computations in a circuit allows to compute the same functions than formulas in a more compact way. But one can translate a circuit C with k gates and n variables (x_1, \dots, x_n) into a formula F with $6k$ gates and $n + k$ variables $(x_1, \dots, x_n, y_1, \dots, y_k)$ in the following way:

- To each gate g_i is associated a variable y_i , and each input edge coming from a gate g_j is replaced by an edge from a new gate labelled y_j .
- such gate g_i is placed in a sub-formula computing $y_j = g_j$. The formula $(y_j \wedge g_j) \vee (\bar{y}_j \wedge \bar{g}_j)$ suits.
- F is made of the conjunction of those sub-formulas.

Thus, for a given vector (x_1, \dots, x_n) such that $C(x_1, \dots, x_n) = 1$, a single assignment to the boolean variables (y_1, \dots, y_k) can satisfy F : each y_i must take the value of the gate g_i in the computation of $C(x_1, \dots, x_n)$. Conversely, no assignment of (y_1, \dots, y_k) can satisfy F for a vector (x_1, \dots, x_n) that do not satisfy C . Thus, problems such as deciding the satisfiability or counting the number of assignments are equivalent for boolean circuits and formulas.

An important subclass of boolean formulas are the *formulas in conjunctive boolean form* (CNF formulas for short).

Definition 1.1.3. A *literal* is a formula made of a single variable, or the negation of a single variable. A *clause* is a disjunction of literals; and a *CNF formula* is a conjunction of clauses.

Any boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ can be represented by a CNF formula of size at most $\mathcal{O}(2^n)$ in the following way. For each boolean vector of $\{0, 1\}^n$ a single clause over n variables (x_1, \dots, x_n) rejects exactly that vector. For instance the vector $(0, \dots, 0)$ is rejected by $x_1 \vee \dots \vee x_n$. The conjunction of all clauses rejecting each vector v such that $f(v) = 0$ forms a formula ϕ representing f .

A family of boolean functions will play a special role in this thesis.

Example 1. Let us consider the boolean function PERMUT_n in n^2 variables $e = (e_{ij})$ that accepts the $n \times n$ permutation matrices, that is, $\{0, 1\}$ -matrices that have exactly one 1 in each row, and one 1 in each column.

The family $(\text{PERMUT}_n)_{n \in \mathbb{N}}$ can be computed by a polynomial size family (f_n) of boolean formulas. One can for instance define f_n as

$$f_n(e) = \bigwedge_i \bigvee_j e_{ij} \wedge \bigwedge_{i,j,k:j \neq k} \bar{e}_{ij} \vee \bar{e}_{ik} \wedge \bigwedge_{i,j,k:i \neq k} \bar{e}_{ij} \vee \bar{e}_{kj}.$$

The first conjunction ensures that each line has at least one 1. The second and third conjunctions ensure that each line and each column has at most one 1. Finally, f_n accepts the matrices with exactly one 1 in each row and one 1 in each column, that is the permutation matrices.

1.2 Graphs

The various notions from graph theory that will be used at some point of the present thesis are introduced here. Since graphs are simply used as tools at different points of our presentation, those notions and definitions do certainly not form a comprehensive view of

the topic. One can consult [18], among the numerous books available on the subject, for a complete view.

To begin with, a graph $G = (V, E)$ is constituted of a finite set of vertices V and a set of edges $E \subseteq V \times V$.

We also consider *oriented graphs*, where the edges are oriented pairs, that is to say, the edges (u, v) and (v, u) are different.

A *path* is a sequence of vertices (v_1, \dots, v_k) such that for all $i = 1 \dots k - 1$, (v_i, v_{i+1}) is an edge of G . A path starting and finishing on the same vertex is a *cycle*. A graph is *connected* if a path links any pair of vertices of V .

The *degree* of a vertex v is the number of edges in the graph containing v . In an oriented graph, one distinguishes the outdegree, that is the number of edges of the form (v, w) , and the indegree: the number of edges of the form (w, v) .

A graph $G = (V, E)$ is *bipartite* with respect to the partition $V = V_1 \sqcup V_2$, and denoted (V_1, V_2, E) , if $E \subseteq V_1 \times V_2$ – that is, all edges go from a vertex in V_1 to a vertex in V_2 .

Two other important subclasses of graphs are the *paths* – graphs constituted of a single path made of distinct vertices – and the *trees* – connected graphs with no cycles. The reunions of disconnected trees are called *forests*.

The trees form a very restricted class of graphs, with strong algorithmic properties. Indeed, many problems that are hard for general graphs become easy or trivial for trees. It is thus tempting to define larger classes of graphs, less restrictive, but that inherit a part of the properties of the trees. This led to the definition of the tree-width of a graph. This parameter informs us of how much the graph looks, and thus behaves, like a tree. Figure 1.1 illustrates the definition.

Definition 1.2.1. Let $G = (V, E)$ be a graph. A k -tree-decomposition of G is a tree $T = (V_T, E_T)$ such that:

- (i) For each $t \in V_T$ there is an associated subset $X_t \subseteq V$ of size at most $k + 1$.
- (ii) For each edge $(u, v) \in E$ there is a $t \in V_T$ such that $\{u, v\} \subseteq X_t$.
- (iii) For each vertex $v \in V$ the set $\{t \in V_T \mid v \in X_t\}$ forms a (connected) subtree of T .

The tree-width $twd(G)$ of G is then the smallest k such that there exists a k -tree-decomposition for G .

If we require the decomposition trees to be paths, then we obtain the *path-width* of the given graph, denoted $pwd(G)$.

The adding of 1 in the definition (subsets of size $k + 1$) is ingeniously set, such that trees have a tree-width 1; thus, the tree-width is a kind of distance between a given graph and the set of trees.

The tree-width, among other graph parameters, seems to be a good indicator of the complexity of a graph. Indeed, many algorithms for hard problems have been designed that present a great complexity in terms of the tree-width, whereas they have a low complexity with respect to the size of the input: typically, the complexity is polynomial in the size of

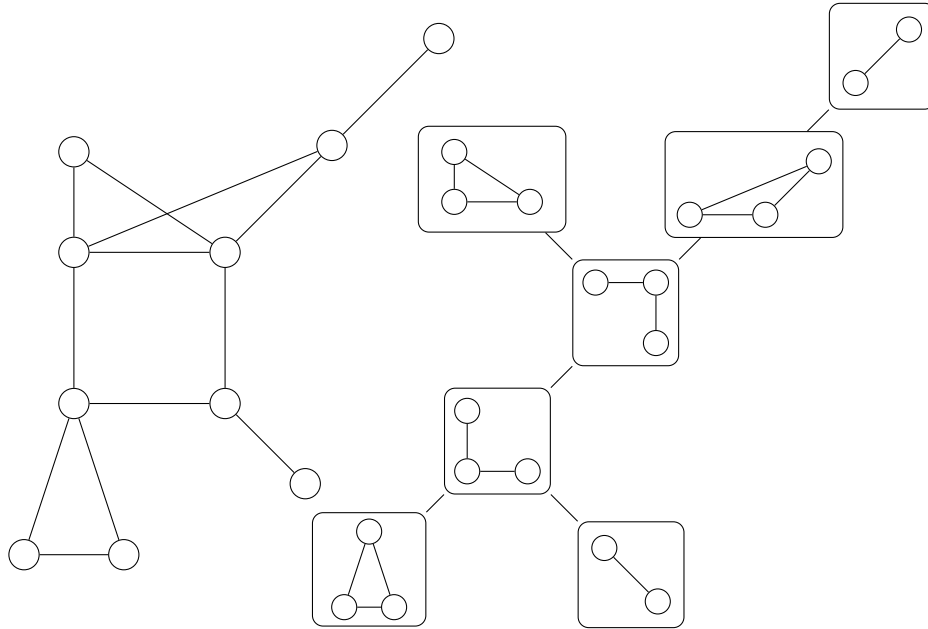


Figure 1.1: A graph of tree-width 2 and a corresponding tree decomposition

the graph, and exponential in the tree-width. For instance, finding a maximum independent set in a graph of tree-width t with n vertices can be done in time $\mathcal{O}(n \cdot 2^t)$.

This approach of expressing the complexity of an algorithm in terms of different parameters that capture the complexity of the problem, instead of expressing it exclusively as a function of the size of the instance, has led to the field of parameterized complexity which has known great developments in the recent years.

The path-width of a graph is obviously not greater than the tree-width. We recall now a useful reciprocal relation [28].

Lemma 1.2.2. [28] *For any forest F on n vertices, the pathwidth $\text{pwd}(F)$ satisfies $\text{pwd}(F) = \mathcal{O}(\log n)$.*

PROOF.

One can find a separating set of vertices in F (of cardinal either 0 or 1) such that its removal separates F into two disconnected parts, each one with no more than $2/3n$ vertices.

We will add this set to all subsets X_i in the path decomposition. We continue this process recursively in each one of the two forests separated. The path constituted of the junction of the path decompositions of each forest, with the separating set added to each subset X_i , constitutes a path decomposition of the forest F .

The width of the decomposition obtained verifies thus: $\text{pwd}(n) \leq 1 + \text{pwd}(2n/3)$, and thus, $\text{pwd}(F) = \mathcal{O}(\log n)$. \square

One can apply the previous lemma to any tree decomposition of a graph G , to prove

that $\text{pwd}(G) = \mathcal{O}(\text{tw}(G) \cdot \log n)$.

Proposition 1.2.3. [28] For any graph $G = (V, E)$ with n vertices, $\text{pwd}(G) = \mathcal{O}(\text{tw}(G) \cdot \log n)$.

PROOF. Let $T = (V_T, E_T)$ be a tree decomposition of G of width $\text{tw}(G)$, and $\{X_i : i \in V_T\}$ the collection of sets associated to each node of T .

From the previous lemma, there exists a path decomposition $P = (V_P, E_P)$ of T with associated sets $\{Y_i : i \in V_P\}$, of width $\mathcal{O}(\log n)$.

One checks easily that P with associated sets $Z_j = \bigcup_{i \in Y_j} X_i$ for all $j \in V_P$ forms a valid path decomposition of G , of width $\mathcal{O}(\text{tw}(G) \cdot \log n)$. \square

We now introduce two important properties on sets of vertices.

Definition 1.2.4. Let $G = (V, E)$ be a graph and $S \subseteq V$ a set of vertices.

- S is an *independent set* if for all pair of vertices $(v, w) \in S^2$, (v, w) do not belong to E .
- S is a *vertex cover* if for all $(u, v) \in E$, $u \in S$ or $v \in S$.

One can remark that a set S is an independent set if and only if $V \setminus S$ is a vertex cover. This defines a bijection between the independent sets and the vertex covers.

It is often interesting to consider graph structures induced by other kind of data, to apprehend them and estimate their complexity. We will follow this approach for the boolean formulas, and introduce the graphs canonically associated to a boolean formula.

Definition 1.2.5. Let φ be a boolean formula in conjunctive normal form with clauses C_1, \dots, C_m and Boolean variables x_1, \dots, x_n .

- a) The *signed clause graph* of φ is a bipartite graph with the x_i and the C_j as nodes. We call the former v-vertices and the latter c-vertices. Edges connect a variable x_i and a clause C_j if and only if x_i occurs in C_j . An edge is signed $+$ or $-$ if x_i occurs positively or negatively in C_j .
- b) The *incidence graph* of φ is the same as $SI(\varphi)$ except that we omit the signs $+$, $-$.
- c) The *primal graph* of φ has only the x_i 's as its nodes. An edge connects x_i and x_j iff both occur in one of the clauses.
- d) The tree-width of a CNF formula φ is defined to be the tree-width of the incidence graph.

From now on, when we want to speak about the tree-width of the primal graph, we will mention it explicitly.

1.3 Counting complexity

The counting complexity appears to be very close to the algebraic complexity. Indeed, many problems that are hard in the counting setting, such as the permanent, are also hard in the algebraic setting. We introduce some basic definitions from this field. The counting complexity focuses on the computation of integer functions: functions $f : \{0, 1\}^* \rightarrow \mathbb{N}$.

Definition 1.3.1 (FP). A function $f : \{0, 1\}^* \rightarrow \mathbb{N}$ is in FP if the function is computable in polynomial time by a Turing machine (writing the output in binary over its tape).

A counterpart of NP has been defined for the counting functions by Valiant [53], in terms of the number of accepting paths of a nondeterministic Turing machine.

Definition 1.3.2 (#P). A function $f : \{0, 1\}^* \rightarrow \mathbb{N}$ is in #P if there exists a nondeterministic Turing machine which runs in polynomial time, and which, on any input $x \in \{0, 1\}^*$, has exactly $f(x)$ accepting paths.

Thus, for any problem in NP, the problem of counting the number of solutions of a given instance is in #P. For example, the problem #SAT of counting the number of satisfying assignments of a boolean formula is in #P.

Another characterization of #P is the following:

Proposition 1.3.3. *A function $f : \{0, 1\}^* \rightarrow \mathbb{N}$ is in #P if and only if there exists a language B in P and a polynomial $p(n)$ such that for every word x ,*

$$f(x) = |\{y \in \{0, 1\}^{p(|x|)}, (x, y) \in B\}|.$$

1.4 Algebraic models of computation

To the study of algebraic algorithms, we first need to define the objects that will carry on the computation. We introduce two models of computation over the real or complex numbers; in fact, those models are defined over any field \mathbb{K} . First, the Valiant model deals with families of arithmetic circuits, and thus computes families of polynomials. The second model, the BSS machine, is a generalization of the Turing machine over any field \mathbb{K} .

1.4.1 Valiant's model

In Valiant's model one studies the computation of multivariate polynomials. More precisely, this model focuses on the computation of families of multivariate polynomials by non uniform families of arithmetic circuits.

This can be done over any field. In this subsection, we fix any field \mathbb{K} . All considered polynomials are over \mathbb{K} . Our definitions in this subsection follow [11].

Definition 1.4.1 (p -family). A p -family is a sequence $f = (f_n)$ of multivariate polynomials such that the number of variables and the degree are polynomially bounded functions of n .

A prominent example of a p -family is the permanent family $\text{PER} = (\text{PER}_n)$.

Definition 1.4.2 (Permanent). The *permanent* $\text{PER}_n(A)$ of a $n \times n$ matrix of indeterminates A is defined as

$$\text{PER}_n(A) = \sum_{\pi \in S_n} \prod_{i=1}^n A_{i\pi(i)},$$

where S_n is the group of the permutations from $[1, n]$ to $[1, n]$.

The computations will be carried on by arithmetic circuits.

Definition 1.4.3 (Arithmetic circuit). An arithmetic circuit C over \mathbb{K} will be an acyclic directed graph with labelled vertices called nodes or gates, the latter having either indegree 0 or 2.

- *Input gates* are labelled with variable names and have indegree 0.
- *Constant gates* are labelled with elements of k and have indegree 0.
- *Operation gates* have indegree 2 and are labelled with arithmetic operations from $\{+, -, \times\}$.

A single gate has outdegree 0, and is called the output gate; the other gates have unbounded outdegree.

In an arithmetic circuit C with l input gates, a function $\mathbb{K}^l \rightarrow \mathbb{K}$ is recursively associated to each gate: to each input or constant gate is associated the value of the label; to each operation gate is associated the result of the labelled arithmetic operation on the values of the parent gates. Since the circuit performs only arithmetic operations, all these functions are polynomial. And the polynomial associated to the output gate is said to be *computed* by the circuit C .

We also define the arithmetic formulas, which are restrictions of the arithmetic circuits where the result of a gate can be used only once.

Definition 1.4.4 (Arithmetic formula). An arithmetic formula is an arithmetic circuit for which all gates different from the output have outdegree 1.

The relative power of circuits and formulas remains an open problem in arithmetic circuit theory. No easy translation like the one presented Section 1.1 for boolean circuits is known.

We define the *complexity* of a polynomial f to be the minimum number $L(f)$ of nodes of an arithmetic circuit computing f .

Definition 1.4.5 (VP). A p -family (f_n) is p -computable if $L(f_n)$ is a polynomially bounded function of n . Those families constitute the complexity class VP.

In Valiant's model, VNP is the analogue of the class NP, or perhaps more accurately, of #P.

Definition 1.4.6 (VNP). A p -family (f_n) is called p -definable if there exists a polynomial $g : \mathbb{N} \rightarrow \mathbb{N}$ and a p -computable family $g = (g_n)$ such that

$$f_n(X_1, \dots, X_{p(n)}) = \sum_{\varepsilon \in \{0,1\}^{g(n)}} g_n(X_1, \dots, X_{p(n)}, \varepsilon_1, \dots, \varepsilon_{g(n)}).$$

The set of p -definable families forms the class VNP.

Clearly, any p -computable family $f = (f_n)$ is p -definable by taking $g = f$ and thus, VP is included in VNP.

The following criterion (a weak form of Valiant's criterion [52] proven as Proposition 2.20 in [11]) will establish the belonging to VNP of many polynomial families that we will encounter.

Proposition 1.4.7. *Suppose $(f_n) : \{0, 1\}^{p(n)} \rightarrow \{0, 1\}$ is a family of boolean formulas of polynomial size. Then, the family (P_n) of polynomials defined by*

$$P_n = \sum_{e \in \{0, 1\}^{p(n)}} f_n(e) X_1^{e_1} \cdots X_{p(n)}^{e_{p(n)}}$$

is p -definable.

PROOF. Using the usual transformation of a boolean formula into a conjunction of 3-clauses, one shows that there exists a boolean formula $\phi_n(e, \theta)$ formed of a conjunction of 3-clauses, and of polynomial size in n , such that

- $f_n(e) = 1 \Rightarrow \exists! \theta \phi_n(e, \theta) = 1$,
- $f_n(e) = 0 \Rightarrow \forall \theta \phi_n(e, \theta) = 0$.

Besides, for each 3-clause K of ϕ_n , there exists an integer polynomial Q_K in at most three of the variables X_i , and of degree at most three, such that

$$\forall x \in \{0, 1\}^3, Q_K(x) = \begin{cases} 1 & \text{if } K(x) \text{ is true} \\ 0 & \text{otherwise.} \end{cases}$$

For instance, for $K = u \vee v \vee w$, $Q_K := uvw + uv(1-w) + u(1-v)w + (1-u)vw + (1-u)(1-v)w + (1-u)v(1-w) + u(1-v)(1-w)$ suits. Let G_n be the product of the Q_K for all clauses K in ϕ_n . G_n is clearly p -computable, and $\phi_n(e, \theta) = G_n(e, \theta)$.

Then, by taking

$$H_n(E, \Theta, X) := G_n(E, \Theta) \prod_{i=1}^{p(n)} (E_i X_i + 1 - E_i),$$

the family (H_n) is p -computable, and

$$P_n(X) = \sum_{e \in \{0, 1\}^{p(n)}} \sum_{\theta} H_n(e, \theta, X).$$

Thus, (P_n) is p -definable. □

Corollary 1.4.8. *The family (PER_n) belongs to VNP.*

PROOF. Using the notations of the previous lemma, the polynomial PER_n can be expressed as the family P_n associated to a family of boolean formulas (f_n) , where f_n computes the function $PERMUT_n$. One can define such family f_n in a polynomial size (Example 1), and thus the conditions of the previous lemma are satisfied. The lemma ensures that (PER_n) is p -definable. □

1.4.2 The BSS model

The Blum, Shub and Smale (BSS) model is another algebraic model of computation, first defined over \mathbb{R} or \mathbb{C} in [7], and further extended to any field \mathbb{K} in the book [6].

The BSS machine can be seen as a generalization of the Turing machine, since the definition of the BSS machine over \mathbb{Z}_2 coincides (up to a simple adaptation) with the Turing machine. Similarly to the Turing machine, the BSS machine works on tapes, on which each cell contains an element of the field instead of a boolean.

Here we follow the definition of [6], but we restrict ourselves to the structure $(\mathbb{R}, +, \times, \div, \leq)$, sufficient for our needs.

Let us denote by $\mathbb{R}^\infty = \bigsqcup_{n \geq 0} \mathbb{R}^n$ the direct sum space over \mathbb{R} , and by \mathbb{R}_∞ the bi-infinite direct sum space over \mathbb{R} , that is elements of the form

$$x = (\dots, 0, x_{-k}, \dots, x_{-2}, x_{-1}, x_0, x_1, x_2, \dots, x_l, 0, \dots)$$

where the x_i belong to \mathbb{R} .

We define rational maps on \mathbb{R}_∞ as follows. Suppose $h : \mathbb{R}^m \rightarrow \mathbb{R}$ is a rational function, we obtain a *rational function* $\widehat{h} : \mathbb{R}_\infty \rightarrow \mathbb{R}$ on \mathbb{R}_∞ by taking $\widehat{h}(x) = h(x_1, \dots, x_m)$ for each $x \in \mathbb{R}_\infty$.

Then, given $g_i : \mathbb{R}^m \rightarrow \mathbb{R}, i = 1, \dots, m$ rational functions, one defines a *rational map* $\widehat{g} : \mathbb{R}_\infty \rightarrow \mathbb{R}_\infty$ by

- $(\widehat{g}(x))_i = \widehat{g}_i(x)$ for $i = 1, \dots, m$.
- $(\widehat{g}(x))_i = x_i$ otherwise.

A rational map is thus a rational function applied locally on a finite number of cells of \mathbb{R}_∞ .

Definition 1.4.9 (BSS machine). A BSS machine over \mathbb{R} consists of an input space $\mathcal{I} = \mathbb{R}^\infty$, an output space $\mathcal{O} = \mathbb{R}^\infty$, a bi-infinite tape $\mathcal{S} = \mathbb{R}_\infty$ and a finite connected directed graph, containing

- one input node, with no incoming edge and one outgoing edge, associated to a map $\mathcal{I} \rightarrow \mathcal{S}$ (that copies the content of \mathcal{I} on \mathcal{S}).
- one output node, with no outgoing edge, associated to a linear map $\mathcal{S} \rightarrow \mathcal{O}$.
- computation nodes, associated to rational maps $\mathcal{S} \rightarrow \mathcal{S}$, with one outgoing edge.
- branching nodes, with two outgoing edges. To a branching node n with two outgoing edges e_+ and e_- , is associated a polynomial $h_n : \mathcal{S} \rightarrow \mathbb{R}$, e_+ being associated to the condition $h_n(x) \geq 0$, and e_- to the condition $h_n(x) < 0$.
- shift nodes, with one outgoing edge, and associated to a map $\mathcal{S} \rightarrow \mathcal{S}$ which is either a left or a right shift.

Remark 1.4.10. The use of the finite number of rational maps allows in particular the machine to have access to a finite number of real constants and copy values from one cell to

another. Following [38], one alternatively suppress the maps in the definition and instead restrain the computation nodes to operations in $\{+, \times, -, \div\}$, or to the writing of a constant on the first cell, or the copy or exchange of the contents of the first two cells.

It is also possible to define the BSS model with the help of algebraic circuits, endowed with branching gates allowing comparisons. Thus, one should not see the use of circuits in Valiant's model and of machines in the BSS model as a major difference, and rather see the distinction in the possibility of making comparisons in the BSS model.

As for a Turing machine, a computation is naturally associated to the BSS machine with a given input, by considering the uniquely determined path starting on the input node and eventually stopping on the output node, obtained by applying all the maps recursively to the input.

The *complexity* of a computation is the length of that path.

The BSS model offers an elegant framework for writing algorithms over the real or complex numbers and analyzing their complexity. For our purposes, we do not need to introduce any complexity classes defined in the BSS model, and we refer to [6] for a view on the structural complexity associated.

1.5 Reductions and completeness

The notion of reduction is a central notion in structural complexity. In order to organize problems into a complexity hierarchy, it is necessary to establish that some problems are at least as difficult as others. Reductions formalize this idea. A reduction from a problem A to a problem B will ensure that an efficient algorithm for B leads to an efficient algorithm for A.

1.5.1 #P-completeness

Reductions for counting problems is a widely discussed issue. We introduce three different reductions: the parsimonious reduction, the Turing reduction, and the 1-Turing reduction. Three definitions of #P-completeness will follow.

Let us begin with the notion of parsimonious reduction for counting problems [59]:

Definition 1.5.1 (Parsimonious reduction). Let $f : \{0, 1\}^* \rightarrow \mathbb{N}$ and $g : \{0, 1\}^* \rightarrow \mathbb{N}$ be two counting problems. A *parsimonious reduction* from f to g consists of a polynomial-time computable function $\sigma : \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that for every $x \in \{0, 1\}^*$, the equality $f(x) = g(\sigma(x))$ holds.

We will denote $f \leq_p g$ when f reduces parsimoniously to g .

Definition 1.5.2 (Turing reduction). Let $f : \{0, 1\}^* \rightarrow \mathbb{N}$ and $g : \{0, 1\}^* \rightarrow \mathbb{N}$ be two counting problems. A *Turing reduction* from f to g is a Turing machine with oracle, which computes f in polynomial time when given an oracle that computes g .

Let us denote $f \leq_T g$ a Turing reduction from f to g . Between those two reductions, one can define the 1-Turing reduction, also often called *many-one reduction*.

Definition 1.5.3 (1-Turing reduction). Let $f : \{0, 1\}^* \rightarrow \mathbb{N}$ and $g : \{0, 1\}^* \rightarrow \mathbb{N}$ be two counting problems. A *1-Turing reduction* from f to g consists of a Turing reduction from f to g , where the machine makes a single call to the oracle.

One can equivalently define the 1-Turing reduction in a similar way to the parsimonious reduction, but with the help of two auxiliary functions, one translating the entry of f into an entry of g , the second one translating the result of g into the result of f – then, a parsimonious reduction is a 1-Turing reduction where that second function is the identity.

We denote $f \leq_1 g$ a 1-Turing reduction from f to g . By the two characterizations of the 1-Turing reduction, one sees easily that $f \leq_p g \Rightarrow f \leq_1 g \Rightarrow f \leq_T g$.

Definition 1.5.4 (#P-completeness). A counting problem f is #P-hard under parsimonious (resp. Turing, 1-Turing) reduction if every problem in #P admits a parsimonious (resp. Turing, 1-Turing) reduction to f . Such problem is #P-complete if in addition it belongs to #P.

It is believed that the notion of #P-completeness under parsimonious reductions do not coincide with the other two notions. Indeed, a parsimonious counting reduction has consequences on the associated decision problems.

Remark 1.5.5. A parsimonious reduction between two counting problems implies a similar reduction between the associated decision problems.

Thus, hard counting problems for parsimonious reductions correspond to hard decision problems. To the contrary, some problems whose associated decision problem is easy are proven to be #P-complete under Turing, and even 1-Turing reduction. For instance, #PERFECT MATCHING, which is the problem of counting the perfect matchings in a bipartite graph, is proven to be “easy to decide, but hard to count”: Valiant [53] proved that this problem is complete under 1-Turing reductions, whereas the decision problem is polynomial. Thus, #PERFECT MATCHING is not #P-complete for parsimonious reductions unless $P = NP$.

Another distinction between the parsimonious reduction on one hand, and 1-Turing and Turing reductions on the other hand is the question of the closure of #P. It is obvious that #P is closed under parsimonious reductions. On the other hand, Watanabe and Toda [51] proved that a problem – such as #PERFECT MATCHING – that is #P-hard under 1-Turing reductions is also “#PH-hard”: each problem in the polynomial hierarchy #PH admits a 1-Turing reduction to #PERFECT MATCHING. Since #PERFECT MATCHING is in #P, this result suggests that #P is not closed under 1-Turing and Turing reductions, unless the counting polynomial hierarchy collapses. It is an open problem, wether one can find an intermediate reduction between the parsimonious reduction and the 1-Turing reduction that could preserve the closure property of #P, and in the same time, exhibit a “easy to decide, but hard to count” behaviour. This question is investigated in [21], where a notion of subtractive reduction, which only allows subtractions after the oracle call, is introduced.

We noticed that as long as $P \neq NP$, parsimonious and 1-Turing reductions lead to different sets of #P-complete problems. To the contrary, the distinction between #P-completeness under Turing and 1-Turing reductions is an open problem, and does not follow from such widely admitted assumption as $P \neq NP$. However, many counting problems

are proven to be #P-complete only under Turing reductions. It appears that the Turing reduction is mostly used as a tool for performing polynomial interpolation – which requires several calls to the oracle. Indeed, as pointed out in [8], “interpolation features prominently in a majority of #P-completeness proofs”, and “it is not clear whether the phenomenon of #P-completeness would be as ubiquitous if [1-Turing] reducibility were to be used in place of Turing”.

Thus, the size of the gap between counting Turing and 1-Turing reductions remains an open question. One can observe that polynomial interpolation does not make use of the full power of the Turing reduction. Indeed, to interpolate a polynomial, one can decide before the first call to the oracle computing the polynomial, the list of all oracle calls that should be performed, whereas the Turing reduction allows us to adapt the next call to the result of the previous ones. We are not aware of a counting Turing reduction in which the different calls to the oracle depend on each other.

Toda [50] showed that one can transform any of such weak Turing reduction with oracle #SAT to a 1-Turing reduction.

Proposition 1.5.6. [50] *Any Turing reduction from a counting problem f to #SAT, in which the list of oracle calls can be decided before the first call, can be transformed into a 1-Turing reduction.*

PROOF. Let us show, how to transform two calls to the oracle into one single call; the general proof follows by induction.

Let us suppose that the reduction requires the call to two instances $\varphi_1(x_1, \dots, x_{k_1})$ and $\varphi_2(y_1, \dots, y_{k_2})$ of #SAT.

The numbers n_1 and n_2 of satisfying assignments of φ_1 and φ_2 are respectively smaller than or equal to 2^{k_1} and 2^{k_2} .

Let us consider the formula $\psi(x_1, \dots, x_{k_1}, y_1, \dots, y_{k_2}, z)$, where z is a new variable, defined by

$$(\varphi_1(x_1, \dots, x_{k_1}) \wedge z) \vee (\varphi_2(y_1, \dots, y_{k_2}) \wedge \bar{z} \wedge (x_1 \wedge x_2 \wedge \dots \wedge x_{k_1})).$$

One sees easily that the formula in the first parenthesis admits $2^{k_2} \times n_1$ satisfying assignments; that the second formula admits n_2 satisfying assignments; and that they do not share common satisfying assignments, since z is set to 1 in the first formula, and to 0 in the second one.

Thus, the disjunction ψ admits $n := 2^{k_2} \times n_1 + n_2$ satisfying assignments. One can recover n_1 and n_2 by taking the modulo and the remainder of the division of n by 2^{k_2} . And one can get n by a single call to the oracle #SAT on the entry ψ . \square

One can thus wonder if similarly, other “weak” Turing reductions may be transformed into 1-Turing reductions. We offer a contribution to this question in Section 2.7, where we establish that 1-Turing reductions are sufficient for the #P-completeness of monotone and implicative #2SAT, and thus that Creignou and Herman’s dichotomy theorem [13] is still valid for 1-Turing reductions.

1.5.2 VNP-completeness

To our knowledge, there exists two different notions of reduction in Valiant's model. We will use both in this thesis. We follow the definitions of [11]. First, the p -projection is a rather restrictive notion, similar to the parsimonious counting reduction.

Definition 1.5.7 (p -projection). A polynomial f with v arguments is said to be a projection of a polynomial g with u arguments, and we denote it $f \leq g$, if

$$f(X_1, \dots, X_v) = g(a_1, \dots, a_u)$$

where each a_i is a variable of f or a constant from K .

A p -family $f = (f_n)$ is a p -projection of $g = (g_n)$, and we denote it $f \leq_p g$, if there exists a function $t : \mathbb{N} \rightarrow \mathbb{N}$ polynomially bounded from above and below, such that

$$\exists n_0 \forall n \geq n_0, f_n \leq g_{t(n)}.$$

By *polynomially bounded from above and below*, we mean that there exists some $c > 0$ such that $n^{1/c} - c \leq t(n) \leq n^c + c$. The lower bound ensures that $t(n) \rightarrow \infty$ as $n \rightarrow \infty$, which is necessary to guarantee the transitivity of \leq_p .

The p -projection is the usual reduction in Valiant's setting. Thus, we will talk about VNP-completeness, without specifying the reduction, when p -projections are used.

Definition 1.5.8 (VNP-completeness). A p -family $g \in \text{VNP}$ is VNP-complete if every p -family $f \in \text{VNP}$ is a p -projection of g .

The VNP-completeness of the permanent under p -projections [52, 11] is a central result in Valiant's theory.

Proposition 1.5.9. *In a field \mathbb{K} of characteristic different from 2, the family (PER_n) is VNP-complete.*

In a field of characteristic 2, the permanent coincides with the determinant, and is thus in VP.

It seems that p -projections are too weak for some of our completeness results. Instead, we use the more general notion of c -reduction [10, 11]. First we recall the notion of oracle computation.

Definition 1.5.10. The *oracle complexity* $L^g(f)$ of a polynomial f with respect to the oracle polynomial g is the minimum number of arithmetic operations $(+, *)$ and evaluations of g over previously computed values that are sufficient to compute f from the indeterminates X_i and constants from \mathbb{K} .

Definition 1.5.11 (c -reduction). Let us consider two p -families $f = (f_n)$ and $g = (g_n)$. We have a polynomial oracle reduction, or c -reduction, from f to g (denoted $f \leq_c g$) if there exists a polynomially bounded function $t : \mathbb{N} \rightarrow \mathbb{N}$ such that the map $n \mapsto L^{g_{t(n)}}(f_n)$ is polynomially bounded.

We can define a more general notion of VNP-completeness based on c -reductions: A p -family f is VNP-hard if $g \leq_c f$ for every p -family $g \in \text{VNP}$. It is VNP-complete if,

in addition, f belongs to VNP. The new class of VNP-complete families contains all the classical VNP-complete families since every p -reduction is a c -reduction.

In our completeness proofs we need c -reductions to compute the homogeneous components of a polynomial. This can be achieved thanks to a well-known lemma (see e.g. [11]).

Lemma 1.5.12. *Let f be a polynomial in the variables X_1, \dots, X_n , in a field \mathbb{K} with at least $\deg f + 1$ elements. For any δ such that $\delta \leq \deg f$, let denote $f^{(\delta)}$ the homogeneous component of degree δ of f . Then, $L^f(f^{(\delta)})$ is polynomially bounded in the degree of f .*

PROOF. We remark that for all $\lambda \in \mathbb{K}$, we have

$$f(\lambda X) = \sum_{i=0}^{\deg f} \lambda^i f^{(i)}(X)$$

where λX denotes $(\lambda X_1, \dots, \lambda X_n)$. We can thus compute the homogenous components of f by an interpolation algorithm on the polynomial in λ for $\deg f + 1$ different values of λ . \square

Part I

From boolean formulas to polynomials

Algebraic complexity theory, which deals mainly with the computation of polynomial functions, is far less studied than the classical boolean complexity. It is thus natural to look for results on the boolean complexity, to see how they transpose in the algebraic setting.

An interesting bridge between those two domains is to consider, for a given multilinear polynomial with integer coefficients, what Malod [35] calls the *coefficient function* of the polynomial: the integer function, that to a degree pattern, associates the coefficient of the monomial. One can then compare the complexities of computing the polynomial and the coefficient function.

In this study, we follow the settings introduced by Koiran and Meer [27]. We associate to a boolean formula φ over n variables $\bar{e} = (e_1, \dots, e_n)$ the multilinear polynomial

$$P(\varphi)(\bar{X}) = \sum_{\bar{e} \in \{0,1\}^n} \varphi(\bar{e}) \cdot \bar{X}^{\bar{e}}. \quad (1.1)$$

We call this polynomial the *polynomial associated to φ* . It can be seen as a restriction to boolean coefficient functions. This restriction does not seem to be dramatic, since most of the natural multilinear polynomial families such as the determinant have small coefficients ($-1, 0$ or 1) and even often boolean coefficients like the permanent. Besides, the notion of p -projection between families of polynomials allows us to compare the complexity of other polynomial families with the ones that are expressible in the form $P(\phi)$.

We have been investigating the link between the complexities of a boolean formula and its associated polynomial. In particular, at which condition on the boolean formula does the polynomial becomes easy to compute?

If the problems SAT and #SAT of deciding the satisfiability and counting the number of satisfying assignments of a boolean formula are hard, those problems become tractable for various subclasses of boolean formulas.

A first approach to define such subclasses has been initiated by Schaefer [40], leading to the wide domain of the Constraint Satisfaction Problems (CSP). In this approach, one consider formulas made of conjunctions of constraints chosen among a finite set S . In Chapter 2, we will follow this setting and will show, that depending on the set S , either all the polynomial families of boolean formulas lead to associated families of polynomials in VP, or there exists a VNP-complete family of associated polynomials.

Another successful approach to define tractable subproblems is, instead of restricting the individual constraints, to give restrictions on the structure of the formulas built with these constraints. To do so, one can define such subclasses via bounding some significant problem parameters.

In [27], Koiran and Meer considered formulas whose incidence graph has a bounded tree-width. They showed, that families of polynomials associated to such formulas can be computed by arithmetic formulas of polynomial size. In Chapter 3, we will study how large this class of boolean formulas whose incidence graph has a bounded tree-width is. In particular, we will show that the boolean function accepting the $n \times n$ permutations cannot be expressed by such formula; thus, the permanent cannot be obtained as a family of polynomials associated to such formulas. The point is that this result is unconditional.

Chapter 2

A dichotomy result

A natural approach to the study of boolean formulas is the research of restricted classes of formulas, to see if difficult problems become easy over some restricted classes. One way of defining those subclasses is to consider the S -formulas, for a fixed set S of constraints. Here, an S -formula over a set of n variables is a conjunction of relations of S where the arguments of each relation are freely chosen among the n variables.

In a seminal paper, Schaefer [40] proved a dichotomy theorem for boolean constraint satisfaction problems: he showed that for any finite set S of constraints the satisfiability problem $\text{SAT}(S)$ for S -formulas is either in P, or NP-complete. Schaefer's result was subsequently extended in a number of directions. In particular, dichotomy theorems were obtained for counting problems, optimization problems and the decision problem of quantified boolean formulas. An account of this line of work can be found in the book by Creignou, Khanna and Sudan [14]. In a different direction, constraint satisfaction problems were also studied over non-boolean domains. This turned out to be a surprisingly difficult question, and it took a long time before a dichotomy theorem over domains of size 3 could be obtained [9].

Our first approach to link the complexity of the polynomial associated to a formula and what we know about this formula will be to study the evaluation of those polynomials from this dichotomic point of view. We work within Valiant's algebraic framework: the role of the complexity class NP in Schaefer's dichotomy theorem will be played by the class VNP of "easily definable" polynomial families, and the role of P will be played by the class VP of "easily computable" polynomial families (Section 1.4.1).

It turns out that a part of our study follows closely the similar dichotomy theorem obtained by Creignou and Hermann [13, 14] in the realm of counting problems: their hard cases correspond to hard cases in our setting. This is not surprising, since there is a well-known connection between counting problems and polynomial evaluation. For instance, as shown by Valiant the permanent is complete in both settings [53, 52]. However, the picture is different for their easy cases : we split them into easy and hard cases in our algebraic setting.

Along the way, we will introduce several new VNP-complete families, and we will use our algebraic work to reinvestigate the counting problems and show new #P-completeness

results. Valiant's model differs also from boolean complexity in the fact that families of polynomials are studied instead of problems. We will precise the differences induced for our constraint satisfaction problems, since for a set S of constraints, we do not have a single corresponding problem as in the boolean setting.

2.1 Boolean formulas

2.1.1 Boolean constraint satisfaction problems

Constraint satisfaction problems (or CSP for short) constitute a common formalism to define generalized classes of satisfiability problems over various domains. For the purpose of this study, we can restrict ourselves to the boolean case. We follow the notations of [14], and refer to this book for a much larger introduction to boolean CSPs.

The general idea of the constraint satisfaction problems is to define subclasses of problems by restricting the set of constraints allowed. We first need to define what a constraint is.

Definition 2.1.1 (Constraint). A *constraint* (or *relation*) is a function f from $\{0, 1\}^k$ to $\{0, 1\}$, for some integer k called the *arity* of the constraint. We say that f is *satisfied* by an assignment $s \in \{0, 1\}^k$ if $f(s) = 1$.

Let us fix a finite set $S = \{\phi_1, \dots, \phi_n\}$ of constraints. This allows us to define a restricted class of boolean constraints, the S -formulas.

Definition 2.1.2 (S -formula). An S -formula over n variables (x_1, \dots, x_n) is a conjunction of boolean formulas, each of the form $g_i(x_{j_i(1)}, \dots, x_{j_i(k_i)})$ where each g_i belongs to S and k_i is the arity of g_i .

In words, each element in the conjunction is obtained by applying a constraint from S to some variables chosen among the n variables.

Various problems have been studied over these S -formulas, among them their satisfiability, the problem of counting the number of satisfying assignments of an S -formula, the problem of deciding the correctness of a quantified S -formula, etc. We only introduce the first two problems.

Definition 2.1.3 ($\text{SAT}(S)$). [40] Given an S -formula ϕ , decide whether ϕ is satisfiable.

For instance, consider the 3 boolean relations OR_0 , OR_1 and OR_2 defined by $\text{OR}_0(x, y) = x \vee y$, $\text{OR}_1(x, y) = \bar{x} \vee y$ and $\text{OR}_2(x, y) = \bar{x} \vee \bar{y}$. When $S = \{\text{OR}_0, \text{OR}_1, \text{OR}_2\}$, the S -formulas are exactly all conjunctions of 2-clauses. Thus, the classical problem 2-SAT is $\text{SAT}(S)$ where $S = \{\text{OR}_0, \text{OR}_1, \text{OR}_2\}$.

The counting problem $\#\text{SAT}(S)$ was studied by Creignou and Hermann [13].

Definition 2.1.4 ($\#\text{SAT}(S)$). [13] Given an S -formula ϕ , count the number of satisfying assignments of ϕ .

2.1.2 Boolean dichotomy results

The first systematic study of a class of constraint satisfaction problems is due to Schaefer [40], who fully classified the complexity of the problems $\text{SAT}(S)$.

Let's introduce first some classical special cases of boolean formulas.

Definition 2.1.5. • A clause with a single negated literal is a *Horn clause*. A conjunction of Horn clauses is a *Horn formula*.

- A clause with a single unnegated literal is a *co-Horn clause*, a conjunction of co-Horn clauses is a *co-Horn formula*.
- A boolean function f is called *1-positive* if $f(1, \dots, 1) = 1$, and *0-positive* if $f(0, \dots, 0) = 1$.
- A boolean formula is called *affine* if it is expressible as a system of affine equations over $\mathbb{Z}/2\mathbb{Z}$; affine equations are of the form $x_1 + \dots + x_n = b$, where b is a boolean constant, where n is an integer called the *width* of the equation, and where the operation $x + y$ is made of $(x \wedge \bar{y}) \vee (\bar{x} \wedge y)$. The *width* of an affine formula is the maximum of the widths of its affine equations.

Theorem 2.1.6. [40] *Let S be a finite set of constraints. The satisfaction problem $\text{SAT}(S)$ is polynomial-time decidable if at least one of the following conditions holds.*

- *Every constraint in S is 0-positive.*
- *Every constraint in S is 1-positive.*
- *Every constraint in S is definable as a Horn formula.*
- *Every constraint in S is definable as a co-Horn formula.*
- *Every constraint in S is definable as a conjunction of 2-clauses.*
- *Every constraint in S is definable as an affine formula.*

Otherwise, $\text{SAT}(S)$ is NP-complete (for many-one reductions).

This result was surprising for several reasons. First, Schaefer was the first to establish the NP-completeness of an infinite collection of problems with a uniform proof, whereas previous NP-completeness proofs concerned one problem at a time. The second unexpected point was the fact that among an infinite class of problems, Schaefer could separate the complexity of the problems in only two equivalence classes (P and NP). This result balanced Ladner's theorem [31] who proved that if $P \neq NP$, there exists problems in NP that are neither in P, nor NP-complete.

In the realm of counting problems, a dichotomy theorem was obtained by Creignou and Hermann [13, 14].

Theorem 2.1.7. [13] *Let S be a finite set of constraints. If every relation in S is affine, then the counting problem $\#\text{SAT}(S)$ is in FP. Otherwise, $\#\text{SAT}(S)$ is $\#\text{P}$ -complete for Turing reductions.¹*

If the result is very similar to Schaefer's theorem, an interesting point is that the sets S of constraints leading to hard problems in their setting strictly contains the sets leading to hard decision problems in Schaefer's setting. It is not surprising that hard decision problems lead to hard counting problems: counting the number of assignments of a boolean formula allows to decide whether that number is nonzero. But the fact that some easy decision problems correspond to hard counting problems is interesting, and informs us on the increase of difficulty between finding a solution and counting the total number of solutions.

Since our proof follows closely Hermann and Creignou's, we briefly sketch their proof. One can remark that an instance of $\#\text{SAT}(S)$ where all constraints in S are affine can be seen as an affine system over $\mathbb{Z}/2\mathbb{Z}$. The number of solutions is thus fully determined by the dimension of the solution space, which can be computed in polynomial time by Gaussian elimination. All the difficulty lies thus in the completeness proof. This proof first relies on the following $\#\text{P}$ -completeness results.

Proposition 2.1.8. [54, 32] *$\#\text{SAT}(\text{OR}_0)$, $\#\text{SAT}(\text{OR}_1)$ and $\#\text{SAT}(\text{OR}_2)$ are $\#\text{P}$ -complete under Turing reductions.*

Then, Creignou and Hermann establish a reduction, for every set S that contains non affine constraints, from one of those three basic problems to $\#\text{SAT}(S)$.

2.2 In the algebraic setting

Let's consider the polynomials associated (in the sense of (1.1)) to a family of boolean formulas. That is, to a family (ϕ_n) we associate the multilinear polynomial family

$$P(\phi_n)(\bar{X}) = \sum_{\bar{\varepsilon}} \phi_n(\bar{\varepsilon}) \bar{X}^{\bar{\varepsilon}}, \quad (2.1)$$

where $\bar{X}^{\bar{\varepsilon}}$ is the monomial $X_1^{\varepsilon_1} \cdots X_{k(n)}^{\varepsilon_{k(n)}}$, and $k(n)$ is the number of variables of ϕ_n .

For instance, as remarked in Corollary 1.4.8, if the formula ϕ_n computes the boolean function PERMUT_n that accepts exactly the $n \times n$ permutation matrices (0/1 matrices with exactly one 1 in each row and each column), the family $(P(\phi_n))$ is the family of the $n \times n$ permanents.

Imagine that the ϕ_n are chosen among the S -formulas of a fixed finite set S of constraints. One would like to understand how the complexity of the polynomials $P(\phi_n)$ depends on S .

First, we define the S -families as sequences of S -formulas of polynomial size.

¹1-Turing reductions (Definition 1.5.3) are called *many-one counting reductions* in [13, 14]. It was already claimed in [13, 14] that Theorem 2.1.7 holds true for 1-Turing reductions. This was not fully justified since the proof of Theorem 2.1.7 is based on 1-Turing reductions from problems which were previously known to be $\#\text{P}$ -complete under Turing reductions only. The present chapter shows that this claim was indeed correct.

Definition 2.2.1. Given a set S , a family (ϕ_n) of boolean formulas is called a S -family if ϕ_n is an S -formula made of a conjunction of at most $p(n)$ relations from S , for some polynomial p . In particular, when S is finite, ϕ_n depends on polynomially many variables.

Our main theorem in this chapter is the following, exhibiting a dichotomy for the families of polynomials associated to S -formulas. The field \mathbb{K} is supposed to be of characteristic different from two and infinite; the VNP-completeness is in the sense of c -reductions.

Theorem 2.2.2 (Main Theorem). *Let S be a finite set of constraints. If S contains only contains only affine functions of width at most 2, then the families $(P(\phi_n))$ of polynomials associated to S -families (ϕ_n) are in VP. Otherwise, there exists an S -family (ϕ_n) such that the corresponding polynomial family $P(\phi_n)$ is VNP-complete.*

For every set S of boolean formulas, one can always exhibit associated polynomial families that are in VP. Thus, for the hard cases, we can only assure the existence of some associated VNP-complete families.

We can observe that the hard cases for counting problems are strictly included in our hard evaluation problems, exactly as the hard decision problems in Schaefer's theorem were strictly included in the hard counting problems.

In our algebraic framework, the evaluation of the polynomial associated to a given formula amounts to solving a “weighted counting” problem: each assignment $(\varepsilon_1, \dots, \varepsilon_k)$ of the variables of ϕ comes with a weight $X_1^{\varepsilon_1} \cdots X_k^{\varepsilon_k}$. In particular when the variables X_i are all set to 1, we obtain the counting problem $\#\text{SAT}(S)$. It is therefore natural that evaluation problems turn out to be harder than counting problems.

The remainder of this chapter is mostly devoted to the proof of Theorem 2.2.2.

Along the way, we obtain several results of independent interest. First, we obtain several new VNP-completeness results. The main ones are about:

- (i) the vertex cover polynomial $\text{VCP}(G)$ and the independent set polynomial $\text{IP}(G)$, associated to a vertex-weighted graph G . Most VNP-completeness results in the literature (and certainly all the results in Chapter 3 of [11]) are about edge-weighted graphs.
- (ii) the antichain polynomial $\text{AP}(X)$ and the ideal polynomial $\text{IPP}(X)$, associated to a weighted poset (X, \leq) .

Unlike in most VNP-completeness results, we need more general reductions to establish VNP-completeness results than Valiant's p -projection. In Section 2.6, we use the c -reductions (Definition 1.5.11) which were introduced by Bürgisser [10, 11] in his work on VNP families that are neither p -computable nor VNP-complete (this work is mentioned in Section 2.8). They are akin to the oracle (or Turing) reductions from discrete complexity theory. The c -reduction has not been used widely in VNP-completeness proofs. The only examples that we are aware of are:

- (i) A remark in [11] on probability generating functions.
- (ii) The VNP-completeness of the weighted Tutte polynomial in [33]. Even there, the power of c -reductions is used in a very restricted way since a single oracle call is performed in each reduction.

By contrast, the power of oracle reductions has been put to good use in #P-completeness theory (see Section 1.5.1). We argue that the importance of Turing reductions in #P-completeness should be revised downwards since, as a byproduct of our VNP-completeness results, we can replace Turing reductions by 1-Turing reductions in several #P-completeness results from the literature. In particular, we obtain a 1-Turing version of Creignou and Hermann's dichotomy theorem. We leave it as an open problem whether the 0/1 partial permanent is #P-complete under 1-Turing reductions (see Section 2.3 for a definition of the partial permanent, and [25] for a #P-completeness proof under oracle reductions).

2.2.1 Overview of the proof of Theorem 2.2.2

By Proposition 1.4.7, for any finite set S of constraints and any S -family (ϕ_n) , the polynomials $(P(\phi_n))$ form a VNP family. We will first briefly deal with the easy cases of Theorem 2.2.2.

Proposition 2.2.3. *For a set S of affine functions of width at most two, every p -family of polynomials associated to S -formulas is in VP.*

PROOF. The only four boolean affine equations with at most two variables are $(e = 0)$, $(e = 1)$, $(e_1 + e_2 = 1)$ and $(e_1 + e_2 = 0)$. The last two are equivalent to $(e_1 = e_2)$ and $(e_1 \neq e_2)$.

For a conjunction ϕ of such relations, two boolean variables e_1 and e_2 of ϕ are either independent or bounded by a relation $(e_1 = e_2)$ or $(e_1 \neq e_2)$ (if e_1 and e_2 are bounded by both relations, ϕ is null and so is the associated polynomial).

One can thus separate the variables of ϕ in disjoint sets $S_i = A_i \sqcup B_i, i = 1 \dots k$, such that variables in distinct sets S_i and S_j are independent, variables in a same set A_i or B_i are bounded by the equality relation and two variables in A_i and B_i are bounded by the relation $(e_1 \neq e_2)$.

Finally, the polynomial associated to a ϕ can be factorized as

$$P(\phi)(X) = \prod_{i=1}^k \left(\prod_{e_j \in A_i} X_j + \prod_{e_l \in B_i} X_l \right)$$

and thus be computed by a circuit of polynomial size. \square

All the work in the proof of Theorem 2.2.2 therefore goes into the hardness proof.

Let's begin the proof of the hard cases of Theorem 2.2.2 with the case of non affine constraints. For that case, the high-level structure of the proof is similar to Creignou and Hermann's proof of #P-completeness of the corresponding counting problems in [13]. The singletons $S = \{\text{OR}_2\}$, $S = \{\text{OR}_1\}$ and $S = \{\text{OR}_0\}$ play a special role in the proof. Here OR_2 denotes the negative two-clause $(x, y) \mapsto (\bar{x} \vee \bar{y})$; OR_0 denotes the positive two-clause $(x, y) \mapsto (x \vee y)$; and OR_1 denotes the implicative two-clause $(x, y) \mapsto (\bar{x} \vee y)$.

The corresponding VNP-completeness results for $S = \{\text{OR}_2\}$ and $S = \{\text{OR}_0\}$ are established in section 2.3; the case of $\{\text{OR}_1\}$ is treated in Section 2.4. These results are put

together with Creignou and Hermann's results in Section 2.5 to establish the existence of a VNP-complete family for any set S containing non affine constraints (Theorem 2.5.1).

Section 2.6 deals with the affine functions of width at least three (Theorem 2.6.2). This completes the proof of Theorem 2.2.2.

The result on non affine constraints is valid on any field \mathbb{K} of characteristic $\neq 2$. The result on affine constraints is valid on any infinite field. Finally, Theorem 2.2.2 is valid on any infinite field of characteristic different from 2.

2.3 Monotone 2-clauses

In this section we consider the sets $\{\text{OR}_2\} = \{(x, y) \mapsto (\bar{x} \vee \bar{y})\}$ and $\{\text{OR}_0\} = \{(x, y) \mapsto (x \vee y)\}$. For $S = \{\text{OR}_2\}$ and $S = \{\text{OR}_0\}$, we show that there exists a VNP-complete family of polynomials $(P(\phi_n))$ associated to a S -family (ϕ_n) .

The partial permanent $\text{PER}_n^*(A)$ of a $n \times n$ matrix $A = (A_{i,j})$ of indeterminates is defined by

$$\text{PER}_n^*(A) = \sum_{\pi} \prod_{i \in \text{def}(\pi)} A_{i\pi(i)},$$

where the sum runs over all injective partial maps from $[1, n]$ to $[1, n]$. It is shown in [11] that if the characteristic of \mathbb{K} is different from 2, the partial permanent is VNP-complete (the proof is attributed to Jerrum).

The partial permanent may be written as in (1.1), where ϕ_n is the boolean formula that recognizes the matrices of partial maps from $[1, n]$ to $[1, n]$. But ϕ_n can be defined as a polynomial size $\{\text{OR}_2\}$ -formula since

$$\phi_n(\varepsilon) := \bigwedge_{i,j,k:j \neq k} \bar{\varepsilon}_{ij} \vee \bar{\varepsilon}_{ik} \wedge \bigwedge_{i,j,k:i \neq k} \bar{\varepsilon}_{ij} \vee \bar{\varepsilon}_{kj}$$

suits. Here the first conjunction ensures that the matrix ε has no more than one 1 in each row; the second one ensures that ε has no more than one 1 in each column. Thus, ϕ_n accepts exactly the partial maps from $[1, n]$ to $[1, n]$. We have obtained the following result.

Theorem 2.3.1 ($\{\text{OR}_2\}$ -formulas). *The family (ϕ_n) is an $\{\text{OR}_2\}$ -family, and the polynomial family $(P(\phi_n))$ is VNP-complete under p -projections.*

The remainder of this section is devoted to the set $S = \{\text{OR}_0\} = \{(x, y) \mapsto x \vee y\}$. The role played by the partial permanent in the previous section will be played by vertex cover polynomials. There is more work to do because the corresponding VNP-completeness result is not available from the literature.

Consider a vertex-weighted graph $G = (V, E)$: to each vertex $v_i \in V$ is associated a weight X_i . The vertex cover polynomial of G is

$$\text{VCP}(G) = \sum_S \prod_{v_i \in S} X_i \tag{2.2}$$

where the sum runs over all vertex covers of G (vertex covers are defined in Definition 1.2.4). The univariate vertex cover polynomial defined in [19] is a specialization of ours; it is

obtained from $\text{VCP}(G)$ by applying the substitutions $X_i := X$ (for $i = 1, \dots, n$), where X is a new indeterminate.

Our main result regarding $\{\text{OR}_0\}$ -formulas is as follows.

Theorem 2.3.2 ($\{\text{OR}_0\}$ -formulas). *There exists a family G_n of polynomial size bipartite graphs such that:*

1. *The family $(\text{VCP}(G_n))$ is VNP-complete.*
2. *$\text{VCP}(G_n) = P(\phi_n)$ where (ϕ_n) is an $\{\text{OR}_0\}$ -family.*

Given a vertex-weighted graph G , let us associate to each $v_i \in V$ a boolean variable ε_i . The interpretation is that v_i is chosen in a vertex cover when ε_i is set to 1. We then have

$$\text{VCP}(G) = \sum_{\varepsilon \in \{0,1\}^{|V|}} \left[\bigwedge_{(v_i, v_j) \in E} \varepsilon_i \vee \varepsilon_j \right] \bar{X}^{\varepsilon},$$

and thus the second property in Theorem 2.3.2 will hold true for any family (G_n) of polynomial size graphs for which the first property holds.

To obtain the first property, we first establish a VNP-completeness result for the independent set polynomial $\text{IP}(G)$. This polynomial is defined like the vertex cover polynomial, except that the sum in (2.2) now runs over all independent sets S (Definition 1.2.4).

Theorem 2.3.3. *There exists a family (G'_n) of polynomial size graphs such that $\text{IP}(G'_n) = \text{PER}_n^*$ where PER_n^* is the $n \times n$ partial permanent. The family $\text{IP}(G'_n)$ is therefore VNP-complete.*

PROOF. The vertices of G'_n are the n^2 edges ij of the complete bipartite graph $K_{n,n}$, and the associated weight is the indeterminate X_{ij} . Two vertices of G'_n are connected by an edge if they share an endpoint in $K_{n,n}$. An independent set in G'_n is nothing but a partial matching in $K_{n,n}$, and the corresponding weights are the same.

Hence $\text{IP}(G'_n) = \text{PER}_n^*$ and the result follows from the VNP-completeness of the partial permanent. \square

Next we obtain a reduction from the independent set polynomial to the vertex cover polynomial. The connection between these two problems is not astonishing since vertex covers are exactly the complements of independent sets. But we deal here with weighted counting problems, so it needs a little more investigation. The connection between independent sets and vertex covers does imply a relation between the polynomials $\text{IP}(G)$ and $\text{VCP}(G)$. Namely,

$$\text{IP}(G)(X_1, \dots, X_n) = X_1 \cdots X_n \cdot \text{VCP}(G)(1/X_1, \dots, 1/X_n). \quad (2.3)$$

Indeed,

$$\text{IP}(G) = \sum_{S \text{ independent}} \frac{X_1 \cdots X_n}{\prod_{v_i \notin S} X_i} = X_1 \cdots X_n \sum_{S' \text{ vertex cover}} \frac{1}{\prod_{v_i \in S'} X_i}.$$

Recall that the incidence graph of a graph $G' = (V', E')$ is a bipartite graph $G = (V, E)$ where $V = V' \cup E'$. In the incidence graph there is an edge between $e' \in E'$ and $u' \in V'$

if u' is one of the two endpoints of e' in G . When G' is vertex weighted, we assign to each V' -vertex of G the same weight as in G' and we assign to each E' -vertex of G the constant weight -1 .

Lemma 2.3.4. *Let G' be a vertex weighted graph and G its vertex weighted incidence graph as defined above. Then, we have*

$$\text{VCP}(G) = (-1)^{e(G')} \text{IP}(G') \quad (2.4)$$

and

$$\text{IP}(G) = (-1)^{e(G')} \text{VCP}(G') \quad (2.5)$$

where $e(G')$ is the number of edges of G' .

PROOF. We begin with (2.4). To each independent set I' of G' we can injectively associate the vertex cover $C = I' \cup E'$. The weight of C is equal to $(-1)^{e(G')}$ times the weight of I' . Moreover, the weights of all other vertex covers of G add up to 0. Indeed, any vertex cover C which is not of this form must contain two vertices $u', v' \in V'$ such that $u'v' \in E'$. The symmetric difference $C \Delta \{u'v'\}$ remains a vertex cover of G , and its weight is opposite to the weight of C since it differs from C only by a vertex $u'v'$ of weight -1 .

It is possible to obtain (2.5) by a similar argument. Here, we will deduce this relation from (2.3) and (2.4):

$$\begin{aligned} \text{IP}(G)(X_1, \dots, X_n) &= X_1 \cdots X_n \cdot \text{VCP}(G)(1/X_1, \dots, 1/X_n) \\ &= (-1)^{e(G')} X_1 \cdots X_n \cdot \text{IP}(G')(1/X_1, \dots, X_1, \dots, 1/X_n) \\ &= (-1)^{e(G')} \text{VCP}(G'). \end{aligned}$$

The first and last equalities follow from (2.3), and the second equality follows from (2.4). \square

To complete the proof of Theorem 2.3.2 we apply Lemma 2.3.4 to the graph $G' = G'_n$ of Theorem 2.3.3. The resulting graph $G = G_n$ satisfies $\text{VCP}(G_n) = \text{IP}(G'_n) = \text{PER}_n^*$ since G'_n has an even number of edges: $e(G'_n) = n^2(n-1)$.

In a finite field of characteristic 2, Malod [34] proved that the partial permanent is in a subclass of VP, namely VP_{ws} , of polynomials computed by polynomial weakly-skew circuits (which are intermediates between formulas and circuits). The result can be derived from a result of Valiant on Pfaffian Sums [55]. Thus, the partial permanent is likely to be no more VNP-complete, and the results of this section no longer hold.

2.4 Implicative 2-clauses

Here we consider the set $S = \{\text{OR}_1\} = \{(x, y) \rightarrow (x \vee \bar{y})\}$. Those constraints are called implicative because $x \vee \bar{y}$ may be reformulated into $y \Rightarrow x$.

The #P-completeness of $\#\text{SAT}(S)$ was established by a chain of reductions in [39] and [32]. Here we will follow this chain of reductions to find a VNP-complete family associated to S -formulas. These two articles show consecutively that the problems of counting the independent sets, the independent sets in a bipartite graph, the antichains in partial

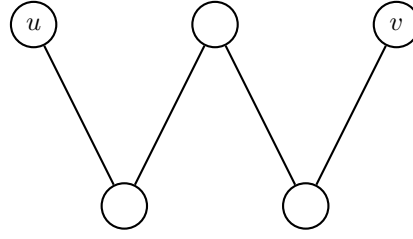


Figure 2.1: The transformation of Lemma 2.4.1

ordered sets (posets), the ideals in posets, and finally the number of satisfying assignments of conjunctions of implicative 2-clauses are #P-complete. We will start from the family (G'_n) such that $\text{IP}(G'_n) = \text{PER}_n^*$, whose existence is stated in Theorem 2.3.3, and follow the reductions for the counting problems.

We first transform the family (G'_n) into a family of bipartite graphs, without changing the independent set polynomials.

Lemma 2.4.1. *There exists a family of bipartite graphs (G''_n) such that $\text{IP}(G''_n) = \text{PER}_n^*$, the partial permanent of size $n \times n$.*

PROOF. By Lemma 2.3.4, we know how to transform a graph G_1 into a bipartite graph G_2 such that $\text{VCP}(G_2) = (-1)^{e(G_1)}\text{IP}(G_1)$ and $\text{IP}(G_2) = (-1)^{e(G_1)}\text{VCP}(G_1)$ where $e(G_1)$ is the number of edges of G_1 . Besides, the constructed graph G_2 has $e(G_2) = 2e(G_1)$ edges. By applying this transformation one more time to G_2 , we obtain a graph G_3 such that

$$\text{IP}(G_3) = (-1)^{e(G_2)}\text{VCP}(G_2) = (-1)^{e(G_1)}\text{IP}(G_1).$$

Thus, the transformation of G_1 into G_2 consists of the replacement of each edge (u, v) in G_1 by the subgraph represented in Figure 2.1.

In Theorem 2.3.3 we introduced a family (G'_n) such that G'_n has an even number of edges, and $\text{IP}(G'_n) = \text{PER}_n^*$. By applying the transformation above to (G'_n) , we obtain a bipartite family (G''_n) such that $\text{IP}(G''_n) = \text{PER}_n^*$. \square

In what follows we will not only use the statement of this lemma, but we will also use the structure of G''_n provided by our transformation of the family (G'_n) into bipartite graphs. More precisely, let us denote by V_1 and V_2 the partite sets of G''_n . We will use for instance the fact that in one of those two sets, say V_1 , all vertices have weight -1 .

It is pointed out in [39] that, given a bipartite graph, one can construct naturally a partially ordered set (poset for short). From the bipartite graph $G''_n = (V_1, V_2, E)$, we define the partially ordered set $(X_n, <)$ with $X_n = V_1 \cup V_2$, and given x and y in X_n , $x < y$ if and only if $x \in V_1$, $y \in V_2$ and $(x, y) \in E$. We see easily that $<$ is transitive and antisymmetric.

Let us recall the definition of an antichain.

Definition 2.4.2 (Antichain). An antichain A in a poset $(X, <)$ is a subset of X such that for all pairs (x, y) of distinct elements of A , x and y are incomparable.

We define the antichain polynomial of a (weighted) poset $(X, <)$ as the polynomial

$$\text{AP}(X) = \sum_A \prod_{x \in A} w(x)$$

where the sum runs over all antichains A of $(X, <)$.

Let us consider a bipartite graph G and its corresponding poset $(X, <)$. A set $S \subseteq X$ is an antichain in $(X, <)$ if and only if it is independent in G . We thus have: $\text{AP}(X) = \text{IP}(G)$. Thus, we can identify the families $(\text{AP}(X_n))$ and $(\text{IP}(G_n''))$.

We then define the notion of ideal in a poset.

Definition 2.4.3 (Ideal). An ideal I in a poset $(X, <)$ is a subset of X such that for all $x \in I$, all y such that $y < x$ belong to I .

We can also define the ideal polynomial $\text{IPP}(X)$ in a poset $(X, <)$

$$\text{IPP}(X) = \sum_I \prod_{x \in I} w(x)$$

where the sum runs over all ideals I of $(X, <)$.

Given an ideal I in a poset $(X, <)$, the maximal elements of I form an antichain A : since they are maximal in I , they cannot be compared. Conversely, given an antichain A , the set of elements x that are not greater than an element of A forms an ideal. One can verify easily that those transformations are bijective and inverse of each other. We thus have a bijection between the ideals and the antichains of a given poset. This fact is sufficient for the authors of [39], since the bijection shows that a poset has the same number of antichains and ideals; the counting problems are thus equivalent.

But for our weighted counting problems, since the ideals and the antichains do not correspond to the same sets of elements, their weights are not equal; we cannot identify simply $\text{AP}(X)$ and $\text{IPP}(X)$ for any poset X .

We do not know how to reduce a family of antichain polynomials into ideal polynomials in general, but in the case of the family $(\text{AP}(X_n))$, since the structure of the family (G_n'') is particular, the problem is easier. We claim the following.

Theorem 2.4.4. *For all integers n , we have $\text{AP}(X_n) = \text{IPP}(X_n)$.*

The proof will be given at the end of this section.

Corollary 2.4.5. *There exists a VNP-complete family of polynomials of the form $(\text{IPP}(X_n))$.*

To conclude, we note that the ideal polynomial in a poset $(X, <)$ may be expressed as a polynomial associated to a S -formula. Namely, we associate to each $x_i \in X$ a boolean variable ε_i with the intended meaning that x_i belongs to an ideal when ε_i is true. For every pair $(x_i, x_j) \in X$ such that $x_i < x_j$, the condition $x_j \in I \Rightarrow x_i \in I$ may be expressed by $(\varepsilon_j \Rightarrow \varepsilon_i)$, or $(\varepsilon_i \vee \bar{\varepsilon}_j)$. Thus, we have

$$\text{IPP}(X) = \sum_{\varepsilon \in \{0,1\}^{|X|}} \left[\bigwedge_{(i,j): x_i < x_j} \varepsilon_i \vee \bar{\varepsilon}_j \right] \bar{X}^{\bar{\varepsilon}}$$

Since the ideals X_n have a polynomial size, the family of S -formulas (ψ_n) coding the relations in the posets (X_n) is polynomially bounded. We thus obtain the following result.

Theorem 2.4.6. *There exists a VNP-complete family of polynomials associated to a polynomially bounded family of $\{\text{OR}_1\}$ -formulas.*

To complete this section, we now provide the proof of Theorem 2.4.4. Let us fix an integer n . We recall that in the bipartite graph $G''_n = (V_1, V_2, E)$ constructed in Lemma 2.4.1, each vertex of V_1 has weight -1 . We also know that $|V_1|$ is even, since the elements of V_1 are added two by two in the transformation from G'_n to G''_n .

Fortunately, by modifying the correspondence between antichains and ideals, we can preserve the weights: we will construct in Lemma 2.4.7 a bijection from the antichains to the ideals of X_n that preserves the weights so that we have:

$$\text{AP}(X_n) = \text{IPP}(X_n).$$

Lemma 2.4.7. *There exists a bijection (different from the natural one considered previously) from the antichains to the ideals of X_n , this one keeping the weights unchanged.*

PROOF. To an antichain A of X_n , we associate the set I such that

- A and I coincide on V_2 .
- $I \cap V_1$ is the complement of $A \cap V_1$ in V_1 .

The map $A \mapsto I$ is clearly injective, and one can verify that the image I is an ideal: given $x \in X_n$ and $y \in I$ such that $x < y$, we have that $x \in V_1$ and $y \in V_2$. Therefore, $y \in A$ and x cannot belong to A as the elements of A are incomparable. Thus, x belong to I . Our map is finally a bijection from the antichains to the ideals of X_n .

Since all the elements of V_1 have weight -1 and $|V_1|$ is even, the weights of I and A differ by a factor $(-1)^{|V_1|} = 1$. \square

2.5 Non affine constraints

In this section we consider the general case of a set S containing non affine constraints: The main result is the following.

Theorem 2.5.1. *For every set S containing a non affine relation, there exists a VNP-complete family (under p -projection) of polynomials associated to S -formulas.*

The proof of this result is analogous to the proof of the $\#\text{P}$ -completeness of the corresponding counting problems given in [14]. We will adapt this proof to our context. The authors use the notion of perfect and faithful implementation (definition 5.1 in [14]):

Definition 2.5.2. A conjunction of α boolean constraints $\{f_1, \dots, f_\alpha\}$ over a set of variables $x = \{x_1, \dots, x_n\}$ and $y = \{y_1, \dots, y_n\}$ is a *perfect and faithful implementation* of a boolean formula $f(x)$, if and only if

1. for any assignment of values to x such that $f(x)$ is true, there exists a unique assignment of values to y such that all the constraints $f_i(x, y)$ are satisfied.
2. for any assignment of values to x such that $f(x)$ is false, no assignment of values to y can satisfy more than $(\alpha - 1)$ constraints.

We refer to the vector x as the function variables and to the vector y as the auxiliary variables.

We say that a set S of constraints *implements perfectly and faithfully* a boolean formula $f(x)$ if there is a S -formula that implements $f(x)$ perfectly and faithfully. We also extend the definition to constraints: a set S of constraints implements perfectly and faithfully a constraint f if S implements perfectly and faithfully every application of f to a set of variables x .

Let us denote F the unary relation $F(x) = \bar{x} = 1 - x$. From [14], lemma 5.30, we have:

Lemma 2.5.3. *If a constraint f is not affine, then $\{f, F\}$ implements at least one of the three logical relations OR_0 , OR_1 or OR_2 perfectly and faithfully.*

The following lemma, analogue to lemma 5.15 from [14], shows that perfect and faithful implementation provides a mechanism to do projections between the polynomials associated to sets of constraints.

Lemma 2.5.4. *Let S and S' be two sets of constraints such that every relation of S can be perfectly and faithfully implemented by S' . Then every p -family of polynomials associated to an S -family is a projection of a p -family of polynomials associated to an S' -family.*

PROOF. Let (ϕ_n) be a S -family, and let us fix an integer n .

Let $\bar{x} = \{x_1, \dots, x_p\}$ be the set of variables of the formula ϕ_n . This formula ϕ_n is a conjunction of constraints $f_i \in S$ applied on variables from $\{x_1, \dots, x_p\}$. If we replace each of those relations f_i by a perfect and faithful implementation using constraints in S' , using for each f_i a new set of auxiliary variables, we obtain a conjunction ψ_n of constraints from S' applied on variable set $\bar{x} \cup \bar{y}$, where $\bar{y} = \{y_1, \dots, y_q\}$ is the union of the auxiliary variables sets added for each constraint f_i .

Since all implementations are perfect and faithful, every assignment to \bar{x} that satisfies all constraints of ϕ_n can be extended by a unique assignment to $\bar{x} \cup \bar{y}$ that satisfies all constraints of ψ_n . Conversely, for an assignment to \bar{x} that does not satisfy all constraints of ϕ_n , no assignment to $\bar{x} \cup \bar{y}$ can extend the previous one and satisfy every constraint of ψ_n .

Since ψ_n is a conjunction of constraints from S' applied on a set of variables $\bar{x} \cup \bar{y}$, ψ_n is a S' -formula. Furthermore, the number of constraints of ψ_n is bounded by the product of the number of constraints of ϕ_n and the maximum number of constraints from S' needed to implement a constraint from S – which does not depend on n . The size of ψ_n is therefore linear in the size of ϕ_n . We have:

$$\begin{aligned}
P(\phi_n)(X_1, \dots, X_p) &= \sum_{\bar{\varepsilon} \in \{0,1\}^p} \phi_n(\bar{\varepsilon}) \bar{X}^{\bar{\varepsilon}} \\
&= \sum_{\bar{\varepsilon} \in \{0,1\}^p, \bar{y} \in \{0,1\}^q} \psi_n(\bar{\varepsilon}, \bar{y}) \bar{X}^{\bar{\varepsilon}} \\
&= \sum_{\bar{\varepsilon} \in \{0,1\}^p, \bar{y} \in \{0,1\}^q} \psi_n(\bar{\varepsilon}, \bar{y}) \bar{X}^{\bar{\varepsilon}} 1^{y_1} \dots 1^{y_q} \\
&= P(\psi_n)(X_1, \dots, X_p, 1, \dots, 1)
\end{aligned}$$

Finally, the family $(P(\phi_n))$ is a projection of the family $(P(\psi_n))$, which is a p -family of polynomials associated to an S' -family. \square

From the two previous lemmas, and from the VNP-completeness of families of polynomials associated to $\{\text{OR}_0\}$ -, $\{\text{OR}_1\}$ - and $\{\text{OR}_2\}$ -families, we conclude that for every set of constraints S such that S contains non affine relations, there exists a VNP-complete family of polynomials associated to $S \cup \{\text{F}\}$ -families. To get rid of the constraint $\{\text{F}\}$, the authors of [14] need to re-investigate the expressiveness of a non affine relation, and distinguish various cases. For our polynomial problems, we can easily force a boolean variable to be set to false by giving to the associated polynomial variable the value 0. We can now give the proof of Theorem 2.5.1:

PROOF. Let (ϕ_n) be an $S \cup \{\text{F}\}$ -family of formulas such that $(P(\phi_n))$ is VNP-complete. The existence of such a family is ensured by lemmas 2.5.3 and 2.5.4.

Let us consider an integer n . $\phi_n(x_1, \dots, x_n)$ is a conjunction of constraints from S applied to variables from \bar{x} and constraints of the form $(x_i = 0)$. We notice that if $\phi_n(\bar{x})$ contains the constraint $(x_i = 0)$, then the variable X_i does not appear in the polynomial $P(\phi_n)(X_1, \dots, X_n)$: all the monomials containing the variable X_i have null coefficients. If we suppress from the conjunction the constraint $(x_i = 0)$, and instead replace the corresponding variable X_i by 0, we obtain exactly the same polynomial: the monomials such that X_i appears in it have null coefficients; the others correspond to assignments such that $x_i = 0$.

Let us denote ψ_n the formula obtained by suppressing from ϕ_n all the constraints of the form $(x_i = 0)$. Since $P(\phi_n)(X_1, \dots, X_n) = P(\psi_n)(y_1, \dots, y_n)$, where y_i is 0 if the constraint $(x_i = 0)$ was inserted in ϕ_n , and X_i otherwise, $(P(\phi_n))$ is a p -projection of $(P(\psi_n))$. Thus, the family $(P(\psi_n))$ is VNP-complete. \square

2.6 Affine functions with width at least 3

Here we consider the case of a set S containing large affine constraints. We first establish the existence of a VNP-complete family of polynomials associated to a polynomially bounded family of affine formulas, and then show how to reduce this family to each affine function of width at least three. In this section, our VNP-completeness results are in the sense of c -reduction.

Let us consider the $n \times n$ permanent $\text{PER}_n(M)$ of a matrix $M = (M_{i,j})$. It may be expressed as the polynomial associated to a formula ϕ_n accepting the $n \times n$ permutation matrices: $\text{PER}_n(M) = \sum_{\varepsilon} \phi_n(\varepsilon) \bar{X}^{\varepsilon}$.

Let us consider the formula φ_n defined by

$$\varphi_n(\varepsilon) = \bigwedge_{i=1}^n \varepsilon_{i1} \oplus \dots \oplus \varepsilon_{in} = 1 \wedge \bigwedge_{j=1}^n \varepsilon_{1j} \oplus \dots \oplus \varepsilon_{nj} = 1.$$

The formula φ_n expresses that each row and each column of ε contains an odd number of values 1. Thus, φ_n accepts the permutation matrices, and other assignments that contain more values 1. We therefore remark that the $n \times n$ permanent is exactly the homogeneous component of degree n of $P(\varphi_n)$. But from Lemma 1.5.12, this implies a c -reduction from the permanent family to the p -family ($P(\varphi_n)$). Thus:

Lemma 2.6.1. *The family ($P(\varphi_n)$) is VNP-complete with respect to c -reductions.*

Through c -reductions and p -projections, this suffices to establish the existence of VNP-complete families for affine formulas of at least three variables.

Theorem 2.6.2. *1. There exists a VNP-complete family of polynomials associated to a $\{x \oplus y \oplus z = 0\}$ -family.*

2. There exists a VNP-complete family of polynomials associated to a $\{x \oplus y \oplus z = 1\}$ -family.

3. For every set S containing an affine function with width at least three, there exists an S -family such that the associated family of polynomials is VNP-complete.

PROOF.

1. Let us consider the formula φ_n . This formula is a conjunction of affine relations with constant term 1: $x_1 + \dots + x_k = 1$. Let φ'_n be the formula obtained from φ_n by adding a variable a and replacing such clauses by $x_1 + \dots + x_k + a = 0$. In the polynomial associated to φ'_n , the term of degree 1 in the variable associated to a is exactly the polynomial $P(\varphi_n)$: when a is assigned to 1, the satisfying assignments of φ'_n are equal to the satisfying assignments of φ_n . Since this term of degree 1 can be recovered by polynomial interpolation of $P(\varphi'_n)$, the family ($P(\varphi_n)$) c -reduces to ($P(\varphi'_n)$).

φ'_n is a conjunction of affine relations with constant term 0. The polynomial $P(\varphi'_n)$ is the projection of the polynomial $P(\psi_n)$, where the formula ψ_n is obtained from φ'_n by replacing each affine relation of the type $x_1 \oplus \dots \oplus x_k = 0$ by the conjunction of relations

$$(x_1 \oplus x_2 \oplus a_1 = 0) \wedge (a_1 \oplus x_3 \oplus a_2 = 0) \wedge \dots \wedge (a_{k-2} \oplus x_{k-1} \oplus x_k = 0)$$

where the a_i are new variables. In fact, one notices easily that for a given assignment of the x_i satisfying φ'_n , a single assignment of the a_i gives a satisfying assignment of ψ_n ; and that if the x_i do not satisfy φ'_n , no assignment of the a_i fits. The polynomial $P(\varphi'_n)$ is thus the polynomial obtained by replacing the variables associated to a_i by the value 1 in $P(\psi_n)$; the family ($P(\varphi'_n)$) is a p -projection of ($P(\psi_n)$).

2. The formula ψ_n constructed above is a conjunction of relations of the type $x \oplus y \oplus z = 0$. Let us construct a new formula ψ'_n by introducing two new variables a and b and replacing each of such relations by the conjunction $(x \oplus y \oplus a = 1) \wedge (a \oplus z \oplus b = 1)$. One can easily figure out that $P(\psi_n)$ is the projection of $P(\psi'_n)$ obtained by setting the variables associated to a and b to 1 and 0 respectively.
3. We consider an affine function f of arity p and of width at least three, and show the existence of a VNP-complete family associated to $\{f\}$ -formulas. To do so, we use an implementation result from Creignou, Khanna and Sudan [14].

It is shown in the proof of [14, Lemma 5.34] that a function of the form $(\varepsilon_1 \oplus \dots \oplus \varepsilon_l = a)$, where $l \geq 3$ and where $a \in \{0, 1\}$, can be written as

$$\exists \varepsilon_{l+1}, \dots, \varepsilon_p f(\varepsilon_1, \dots, \varepsilon_p).$$

For $l \leq k \leq p$, let us denote by f_k the function

$$f_k(\varepsilon_1, \dots, \varepsilon_k) = \exists \varepsilon_{k+1}, \dots, \varepsilon_p f(\varepsilon_1, \dots, \varepsilon_p) = \exists \varepsilon_{k+1} f_{k+1}(\varepsilon_1, \dots, \varepsilon_{k+1}).$$

Remark that f_l is $(\varepsilon_1 \oplus \dots \oplus \varepsilon_l = a)$ and that $f_p = f$.

From [14, Lemma 4.5] those functions are all affine, and thus the number of satisfying assignments are powers of two. If 2^i denotes the number of satisfying assignments of f_k , one remarks that f_{k+1} has either 2^i or 2^{i+1} satisfying assignments. Indeed, if a vector $(\varepsilon_1, \dots, \varepsilon_k)$ do not satisfy f_k , then by definition $f_{k+1}(\varepsilon_1, \dots, \varepsilon_{k+1}) = 0$ regardless of the value of ε_{k+1} ; and conversely, if $f_k(\varepsilon_1, \dots, \varepsilon_k)$ holds, either $f_{k+1}(\varepsilon_1, \dots, \varepsilon_k, 0)$ or $f_{k+1}(\varepsilon_1, \dots, \varepsilon_k, 1)$ hold.

If f_{k+1} has 2^i satisfying assignments, this means that each satisfying assignment of f_k is completed in exactly one satisfying assignment of f_{k+1} . In other words, f_{k+1} implements perfectly and faithfully f_k . From Lemma 2.5.4, any p -family of polynomials associated to a $\{f_k\}$ -family can be written as a projection of a p -family of polynomials associated to a $\{f_{k+1}\}$ -family (the projection being on the constant 1).

If f_{k+1} has 2^{i+1} satisfying assignments, each satisfying assignment of f_k is always completed in two satisfying assignments of f_{k+1} . Thus, $f_k(\varepsilon_1, \dots, \varepsilon_k) = f_{k+1}(\varepsilon_1, \dots, \varepsilon_k, 0)$. Thus, any polynomial P associated to a $\{f_k\}$ -formula ϕ is the projection of the polynomial associated to the $\{f_{k+1}\}$ -formula obtained by replacing each constraint $f_k(x_1, \dots, x_k)$ from ϕ by $f_{k+1}(x_1, \dots, x_k, y)$, where y is a new variable, and by projecting the polynomial variable associated to y on the value 0 —forcing y to be 0.

Furthermore, since $f_l = (\varepsilon_1 \oplus \dots \oplus \varepsilon_l = a)$, $(\varepsilon_1 \oplus \varepsilon_2 \oplus \varepsilon_3 = a)$ can be expressed as $f_l(\varepsilon_1, \varepsilon_2, \varepsilon_3, 0, \dots, 0)$, and this ensures a p -projection of a family associated to a $\{f_l\}$ -family on any p -family associated to a $\{(\varepsilon_1 \oplus \varepsilon_2 \oplus \varepsilon_3 = a)\}$ -family.

Finally, by an immediate recursion, the VNP-complete family associated to a $\{(\varepsilon_1 \oplus \varepsilon_2 \oplus \varepsilon_3 = a)\}$ -family whose existence is ensured by the first two items is a p -projection of a p -family associated to $\{f\}$ -formulas.

□

It remains an open question, whether those completeness results also hold for p -projections.

Problem 1. Given an affine formula ϕ of width at least three, does there exist a $\{\phi\}$ -family such that the associated family of polynomials is VNP-complete under p -projections?

If we have no answer to this question, we can show unconditionally that the permanent and the partial permanent, for $n \geq 3$, cannot be expressed as polynomials associated to affine constraints. Of course, this does not discard the possibility of a p -projection.

The permanent is the polynomial associated to the function PERMUT_n . We can observe that we can use PERMUT_n to express via projections the boolean function $\mathbf{1-in-3}(x, y, z)$ accepting (x, y, z) if and only if exactly one of the three variables is true, whereas it is impossible with a conjunction of affine constraints. We use three new boolean variables x, y and z , and denote $f(x, y, z)$ the function obtained by applying PERMUT_n to the boolean matrix

$$\begin{pmatrix} x & y & z & 0 & \dots & 0 \\ 0 & & & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & & & & & z \\ z & 0 & & & & y \\ y & z & 0 & \dots & 0 & x \end{pmatrix}.$$

Since a permutation matrix has exactly one 1 in each row and each column, f accepts exactly the entries $(1, 0, 0)$, $(0, 1, 0)$ or $(0, 0, 1)$ and thus is equal to the function $\mathbf{1-in-3}$.

On the other hand, a formula ψ made of a conjunction affine constraints can be seen as an affine system over the field F_2 . Any formula $\psi'(x, y, z)$ obtained from ψ by projection on x, y, z or constants 1 or 0 is still a conjunction on affine constraints. Thus the set of the solutions of ψ' is a affine space over F_2^3 , and its cardinality must be a power of two. ψ' cannot accept the set $\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$. Thus PERMUT_n , for $n \geq 3$, cannot be expressed as a conjunction of affine constraints. A similar reasoning can be done for the function accepting the matrices of injective partial maps from $[1, n]$ to $[1, n]$: the function accepting the matrices of injective partial maps applied to the previously considered matrix, where z is set to 0, defines a function $g : F_2^2 \mapsto F_2$ accepting exactly the set of entries $\{(1, 0), (0, 0), (0, 1)\}$. For the same cardinality reason, this boolean function cannot be expressed as a conjunction of affine formulas.

Thus, neither (PER_n) nor (PER_n^*) can be obtained as families associated to affine formulas. In particular, they cannot be expressed as polynomials associated to a conjunction of affine formulas of width at most two and computed polynomially via the algorithm proposed in Proposition 2.2.3.

2.7 #P-completeness proofs

Up to now, we have studied vertex weighted graphs mostly from the point of view of algebraic complexity theory. Putting weights on edges, or on vertices, can also be useful as an intermediate step in #P-completeness proofs [53, 25]. Here we follow this method to obtain new #P-completeness results. Namely, we prove #P-completeness under 1-Turing reductions for several problems which were only known to be #P-complete under Turing reductions.

Theorem 2.7.1. *The following problems are #P-complete for 1-Turing reductions.*

1. *Vertex Cover: counting the number of vertex covers of a given a graph.*
2. *Independent Set: counting the number of independent sets of a given graph.*
3. *Bipartite Vertex Cover: the restriction of vertex cover to bipartite graphs.*
4. *Bipartite Independent Set: the restriction of independent set to bipartite graphs.*
5. *Antichain: counting the number of antichains of a given poset.*
6. *Ideal: counting the number of ideals of a given poset.*
7. *Implicative 2-SAT: counting the number of satisfying assignments of a conjunction of implicative 2-clauses.*
8. *Positive 2-SAT: counting the number of satisfying assignments of a conjunction of positive 2-clauses.*
9. *Negative 2-SAT: counting the number of satisfying assignments of a conjunction of negative 2-clauses.*

Remark 2.7.2. #P-completeness under Turing reductions is established in [39] for the first six problems, in [32] for the 7th problem and in [54] for the last two. In Section 2.1.2, the last three problems are denoted $\#\text{SAT}(S)$ where S is respectively equal to $\{\text{OR}_1\}$, $\{\text{OR}_0\}$ and $\{\text{OR}_2\}$.

PROOF. Provan and Ball establish in [39] the equivalence of Problems 1 and 2, of Problems 3 and 4, and of Problems 5 and 6; they produce 1-Turing reductions from 1 to 8 and from 4 to 5, and Linial gives in [32] a 1-Turing reduction from 6 to 7. Problems 8 and 9 are clearly equivalent. Therefore, to obtain #P-completeness under 1-Turing reductions for all those problems, we just need to show the #P-completeness of Problem 1 and to produce a 1-Turing reduction from Problem 1 to Problem 3 (replacing the Turing reduction from [39]).

In order to prove the #P-completeness of Problem 1, we first establish a 1-Turing reduction from the #P-complete problem of computing the permanent of $\{0, 1\}$ -matrices (which is known to be #P-complete under 1-Turing reductions [59]) to the problem of computing the vertex cover polynomial of a weighted graph with weights in $\{0, 1, -1\}$.

In [11], Bürgisser attributes to Jerrum a projection of the partial permanent on the permanent, with the use of the constant -1 (see Section 2.3). Applied to a $\{0, 1\}$ -matrix, this gives a 1-Turing reduction from the permanent on $\{0, 1\}$ -matrices to the partial permanent on $\{0, 1, -1\}$ -matrices. By Theorem 2.3.3, the $n \times n$ partial permanent is equal to the independent set polynomial of the graph G'_n ; the reduction is obviously polynomial. Moreover, by Lemma 2.3.4 this polynomial is the projection of the vertex cover polynomial of G_n , with the use of the constant -1 . The partial permanent on entries in $\{0, 1, -1\}$ therefore reduces to the vertex cover polynomial on graphs with weights in $\{0, 1, -1\}$.

Let G be such a vertex weighted graph, with weights in $\{0, 1, -1\}$. A vertex cover of nonzero weight does not contain any vertex v of weight 0, and in order to cover the edges that are incident to v , it must contain all its neighbours. One can therefore remove v , and

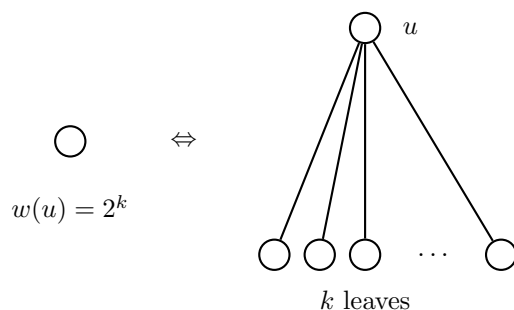


Figure 2.2: Subgraph of H simulating a weight 2^k in G' .

replace each edge from v to another vertex u by a self-loop (an edge from u to u). Thus, we obtain a graph G' with weights in $\{1, -1\}$ such that $\text{VCP}(G) = \text{VCP}(G')$.

To deal with the weights -1 , we use a method similar to [53]. Since $\text{VCP}(G')$ is the value of a permanent on a $\{0, 1\}$ -matrix, it is positive. We will construct an integer N and a graph H such that the number of vertex covers of H modulo N is equal to $\text{VCP}(G')$. Since the computation of the residue modulo N is easy to carry on, this will establish a reduction from the boolean permanent to counting vertex covers in a graph.

We choose N larger than the maximum value of the number of vertex covers of G' : $N = 2^{v(G')} + 1$, where $v(G')$ is the number of vertices of G' , will suit our purposes. Now that we compute the number of vertex covers modulo N , we can replace each -1 weight in G' by the weight $N - 1 = 2^{v(G')}$. But one can simulate such a weight on a vertex by adding to H $v(G')$ leaves linked to v .

Finally, we construct a 1-Turing reduction from vertex cover to bipartite vertex cover. From Lemma 2.4.1, we have a projection of the vertex cover polynomial of a bipartite graph on a vertex cover polynomial of any graph, with the use of -1 weights (and up to a factor $(-1)^e$, where e is the number of edges of the original graph; that detail can be fixed by distinguishing both cases in the reduction).

To eliminate the weights -1 , we can follow the method used in our above proof of the $\#P$ -completeness of Problem 1. Indeed, since the leaves added to the graph preserve bipartiteness, we obtain a reduction from counting vertex covers in a general graph to counting vertex covers in a bipartite graph. \square

The proof of Creignou and Hermann's dichotomy theorem [13, 14] is based on 1-Turing reductions from the last 3 problems of Theorem 2.7.1. We have just shown that these 3 problems are $\#P$ -complete under 1-Turing reductions. As a result, we have the following corollary to Theorem 2.7.1.

Corollary 2.7.3. *Creignou and Hermann's dichotomy theorem (Theorem 2.1.7) still holds for $\#P$ -completeness under 1-Turing reduction.*

Theorem 2.7.1 is based on the transformation of p -projections between polynomials into 1-Turing reductions between counting problems. The situation is analogous to the one of Remark 1.5.5. However, a parsimonious counting reduction is directly a reduction between the decision problems, whereas there is some work to do to transform the p -projections into

many-one counting reductions. In particular, it depends on the final problem, since it is necessary to simulate the constants used in the projection.

In the case of the parsimonious counting reductions, the implications on the decision problems allow us to discard the possibility of some reductions. For instance, unless $P = NP$, there is no parsimonious reduction from $\#SAT$ to $\#-2-SAT$. One could wonder if a similar impossibility result could be obtained to answer negatively Problem 1. But in the case of the p -projections, such result seems unlikely to obtain. Indeed, to show the impossibility of a p -projection between the permanent and a polynomial associated to affine formulas, one would have to simulate *any* constant in the field. In the case of affine formulas, we do not know how to simulate constants different from 0 and 1. In the case of affine constraints, if counting the number of elements in an affine space is easy, it seems that the weighted problem with integer weights is hard. Indeed, this question is linked to the notion of linear codes.

A *linear code* over a field F_q is a subspace of F_q^n for some n , of a given dimension k , defined by the equation $H \cdot c = 0$ for some $n \times (n - k)$ matrix H . A central question on linear codes is the *minimal Hamming distance* between two words, that is, the minimal number of bit-to-bit differences between two words in C : codes are used to correct errors in a bit transmission, and thus this minimal distance informs us of how many errors can be detected correctly. Since the code is linear, this distance is equal to the minimal distance between a word in C and the word $(0, \dots, 0)$. Let us call the *weight* of a word $c \in C$ its number of nonzero coefficients. Given a linear code through the equations $H \cdot c = 0$, computing the minimal distance is a NP-hard problem, and the problem of deciding whether there exists a word of a given weight w is NP-complete [56]. It has also been shown, that even approximating the minimal distance within a constant factor is hard [20].

We can adapt our proof of the VNP-completeness of a family associated to affine formulas to show that the associated counting problem is $\#P$ -hard. This is not surprising since the NP-complete problems have usually $\#P$ -complete counting counterparts, whereas the opposite is not true (see Section 1.5.1). Yet, our proof is very simple, compared with the NP-complete proof.

Proposition 2.7.4. *The problem of counting the number of words of a given weight w in a binary linear code C is $\#P$ -complete under 1-Turing reductions.*

PROOF. The computation of the permanent on $\{0, 1\}$ -matrices is $\#P$ -hard for 1-Turing reductions. Let us reduce parsimoniously the computation of such permanent to the problem of counting the number of words of weight w in a binary affine code. We then reduce it to the same problem on linear codes.

Let us consider an $n \times n$ boolean matrix $A = (a_{ij})$. The permanent $\text{PER}(A)$ equals

$$\text{PER}(A) = \sum_{\varepsilon \in \{0,1\}^{n^2}} \text{PERMUT}_n(\varepsilon) A^\varepsilon.$$

Let us consider the affine space consisting of the $n \times n$ boolean matrices $X = (x_{ij})$ satisfying the equations

$$\forall i, \sum_j a_{ij} x_{ij} = 1 \quad \text{and} \quad \forall j, \sum_i a_{ij} x_{ij} = 1.$$

Such matrices have obviously at least one 1 in each row and one 1 in each column. Thus, the matrices that belong to the code and have a weight n are permutation matrices. Conversely, a permutation matrix $\varepsilon = (\varepsilon_{ij})$ corresponding to a permutation π belongs to the code if and only if $A^\varepsilon = \prod_i A_{i\pi(i)} = 1$.

Thus, the number of words of weight n in the code is exactly the number of permutation matrices ε such that $A^\varepsilon = 1$, and thus is equal to $\text{PER}(A)$. This establishes the reduction from computing $\text{PER}(A)$ to counting the number of words of weight w in an affine code.

We now reduce the problem of counting the number of words of weight k in a binary affine code to the same problem on binary linear codes. Let C be an affine code on \mathbb{F}_2^n defined by the equations $Hx = a$, where H is a $m \times n$ matrix and $a \in \{0, 1\}^m$ a constant boolean vector. Let us add a new variable y_1 and replace each equation $\sum_j h_{ij}x_j = 1$ by $\sum_j h_{ij}x_j + y_1 = 0$.

The equations are now linear, and the solutions such that $y_1 = 1$ coincide with the solutions of the affine problem. To force y_1 to be equal to 1, we add n new variables y_2, \dots, y_{n+1} and n equations $y_i + y_{i+1} = 0$, $i = 1 \dots n$. Thus, a word of the new obtained code C' satisfies $y_1 = y_2 = \dots = y_{n+1}$. A word of the new code of weight $n + 1 + k$ must satisfy $y = (y_1, \dots, y_{n+1}) = (1, \dots, 1)$, otherwise its weight would be less than $n + 1$. Thus, the number of words of the linear code of weight $n + 1 + k$ is equal to the number of words of weight k in the affine code, and this establishes a parsimonious reduction. \square

The numbers A_w of words of weight w in a code C of length n are the coefficients of the *weight enumerator polynomial* of the code, defined as

$$W_C(x, y) = \sum_{w=0}^n A_w x^w.$$

Since computing the coefficients is #P-complete, evaluating the polynomial on integer values is #P-hard. But this polynomial is the projection on a single variable of the polynomial associated to the affine formulas defining the linear space. Thus, computing the polynomial associated to a conjunction of affine formulas on integer values is #P-hard.

2.8 The structure of the polynomials associated to S -formulas.

In Theorem 2.2.2, we have to cope with a major difference between the Valiant model and the classical decision and counting complexities. In Valiant's model, one focuses on the computation of families of polynomials, which are much "smaller" than the problems of classical complexity. Indeed, a polynomial family consists of a single polynomial of index n whereas a problem can have an exponential number of instances of size n . Thus, from a given set S , instead of getting one single problem – the evaluation of the polynomials associated to S -formulas – we have a collection of families of associated polynomials.

For any set S , some of those families are in VP: one can construct families of S -formulas that do not take advantage of the complexity of the set S . For instance, for a constant family (ϕ_n) , that is, $\forall n, \phi_n = \phi$ for a certain formula ϕ , $(P(\phi_n))$ is obviously in VP. Therefore, in

Theorem 2.2.2, we cannot expect a stronger statement than the existence of VNP-complete families in the hard cases.

As we remarked in Section 2.1.2, Schaefer defined a large class of problems in NP for which all problems are either in P, or NP-complete, whereas Ladner [31] ruled out this possibility for all problems in NP if $P \neq NP$. But in our settings, from the previous remarks, we cannot directly conclude that all families $(P(\phi_n))$, where (ϕ_n) is an S -family, are either in VP or VNP-complete.

We can even prove the opposite. Bürgisser has proved [10, 11] an equivalent of Ladner's theorem in Valiant's setting : if $VP \neq VNP$, then there exists a family of polynomials in VNP which is neither in VP, nor VNP-complete. We can use Bürgisser's very general proof to conclude that for any set S such that there exists associated VNP-complete families of polynomials, and provided that $VP \neq VNP$, some associated p -families are neither in VP nor VNP-complete.

Let us consider a set Ω and a quasi-order (a reflexive and transitive binary relation) \leq on Ω . One can define a quasi-order \leq_p on the families $\Omega^{\mathbb{N}}$ obtained similarly as in Definition 1.5.7 : we have $f = (f_n) \leq_p g = (g_n)$ if we have a term by term comparison $f_n \leq_{g_t(n)}$ from a certain index, and where $t(n)$ is polynomially bounded from above and below.

Bürgisser constructs almost all of his proof for any of such quasi-ordered set (Ω, \leq) , and then applies it to the set of polynomials $\mathbb{K}[X_1, X_2, \dots]$. Thanks to this general setting, we will be able to apply those results to the set of polynomials $P(\phi)$ for all S -formulas ϕ .

The proof is based on the use of certain subsets of $\Omega^{\mathbb{N}}$, the σ -limit sets. It also allows to define compatible orders.

Definition 2.8.1. A *cylinder* in $\Omega^{\mathbb{N}}$ is a subset of the form $F \times \Omega^{\mathbb{N}}$, for some set $F \subseteq \Omega^n$, where $n \in \mathbb{N}$. A *σ -limit set* in $\Omega^{\mathbb{N}}$ is a countable union of a countable intersection of cylinders.

A quasi-order \leq_c is said to be *compatible* with $(\Omega^{\mathbb{N}}, \leq_p)$ if the three following conditions hold.

- $\forall f, g : f \leq_p g \Rightarrow f \leq_c g$.
- $\forall f, g, h : (f \leq_c h) \wedge (g \leq_c h) \Rightarrow f \cup g \leq_c h$, where $f \cup g$ stands for the family $(f_1, g_1, f_2, g_2, \dots)$.
- $\{h | h \leq_c g\}$ and $\{h | f \leq_c h\}$ are σ -limit sets for all $f, g \in \Omega^{\mathbb{N}}$.

Let \leq_c be a fixed compatible quasi-order on $\Omega^{\mathbb{N}}$. Let us denote $f <_c g$ when $f \leq_c g$ and not $g \leq_c f$. The following theorem (Theorem 5.10 in [11]) is a restriction of Bürgisser's general result.

Theorem 2.8.2 (Bürgisser). *For $f, g \in \Omega^{\mathbb{N}}$ satisfying $f <_c g$ there exists $h \in \Omega^{\mathbb{N}}$ such that $f <_c h <_c g$. If also $f \leq_p g$, then we may additionally achieve that $f \leq_p h \leq_p g$.*

By a simple observation on the proof, we can also add the following.

Remark 2.8.3. Such family $h = (h_i)$ can be constructed such that for all i , h_i is either a term of the form f_k or a term of the form g_k , with k polynomially bounded in i .

PROOF. Bürgisser builds up a family h by constructing a family $h' = (h'_n)$ of the form $h'_n = f_n$ or $h'_n = g_n$, depending on the index n , and then, by taking $h = f \cup h = (f_0, h'_0, f_1, h'_1, f_2, h'_2 \dots)$. The constructed family is thus made of terms from f and g of polynomially bounded indices. \square

We now fix a finite set S of constraints among the hard cases of Theorem 2.2.2 (S contains either non affine constraints or affine constraints of width at least 3). We denote by Ω the set of polynomials associated to S -formulas. We consider the quasi-ordered set (Ω, \leq) , where \leq is the projection between polynomials, and the quasi-ordered set of families $(\Omega^{\mathbb{N}}, \leq_p)$, where \leq_p is the p -projection. The c -reduction \leq_c defines also a partial order on $\Omega^{\mathbb{N}}$, which is compatible with \leq_p (Lemma 5.15 in [11]). We can thus apply Theorem 2.8.2 to this setting.

Corollary 2.8.4. *If $VP \neq VNP$, then there exists a p -definable family of polynomials associated to an S -family which is neither p -computable, nor VNP-complete with respect to c -reduction.*

PROOF. Let g be a VNP-complete family of polynomials associated to an S -family, and let f be a family of polynomials in VP, also associated to an S -family. Then, from the VNP-completeness of g , $f \leq_c g$, and since $VP \neq VNP$, we have $f <_c g$. Thus, by Theorem 2.8.2, there exists a $h \in \Omega^{\mathbb{N}}$ such that $f <_c h <_c g$.

Since $f <_c h$, no circuit of polynomial size can compute h (even with the help of the oracle f), and thus h does not belong to VP. h is neither VNP-complete with respect to c -reductions since g does not c -reduce to h . And furthermore, h is associated to an S -family since by Remark 2.8.3, h is made of terms from f and g of polynomially bounded indices. \square

Chapter 3

A lower bound from communication complexity

An active field of research in complexity is devoted to the design of efficient algorithms for subclasses of problems which in full generality likely are hard to solve. It is common in this area to define such subclasses via bounding some significant problem parameters. Typical such parameters are the tree- and clique-width if a graph structure is involved in the problem's description.

In the case of boolean formulas, one can define different graphs representing the structure of a given formula, among them the incidence graph and the primal graph. One can then study formulas whose associated graph has a bounded tree- or clique-width. Unlike the approach of Chapter 2, the restriction consists thus in constraining the structure of a boolean formula, instead of limiting the basic constraints.

Still by considering the polynomials associated to S -formulas (Equation (1.1)), it offers an other way to define restricted classes of tractable families of polynomials. This path has been successfully followed by Koiran and Meer [27, 8], who investigated the expressive power of p -families of polynomials associated to formulas of bounded tree-width. They established that those polynomials could express (via p -projections) exactly the class of functions representable by arithmetic formulas of polynomial size.

In particular, those families of polynomials are easy to compute, since they are computable by families of arithmetic formulas of polynomial size. It is thus interesting to wonder how far this approach leads for computing efficiently the permanent family, which is the basic VNP-complete family. One does not expect this approach to succeed, since it would imply that $VP = VNP$. But Koiran and Meer [27] proved the failure of this approach *unconditionally*, without relying on such complexity conjecture. This result restricts thus the approaches for computing the permanent family, and reinforces our impression that this family of polynomials is hard to compute.

We reinvestigate the above mentioned result from Koiran and Meer, and exhibit a link between the tree-width of a boolean formula and its communication complexity. This link gives us a new proof of Koiran and Meer's result, and allows us to drop some assumptions. The relation between tree-width and communication complexity is interesting by itself, since

it offers a framework to show lower bounds on the tree-width of various boolean formulas whose communication complexity is known.

We will first briefly present Koiran and Meer's result in Section 3.1. Then, we will introduce some definitions from communication complexity (Section 3.2), before exhibiting the link between tree-width and communication complexity in Section 3.3.

3.1 Previous results

Recall that to a boolean formula are canonically associated the primal graph, where variables are linked when they belong to a same clause, and the incidence graph, where clauses are linked to the variables they contain (Definition 1.2.5).

The tree-width of the incidence graph is a relevant parameter to estimate the complexity of a given boolean formula. Indeed, Fischer, Makowsky and Ravve [22] proposed an algorithm for counting the number of satisfying assignments of a formula, which is polynomial in the size of the formula, and exponential in its tree-width.

Theorem 3.1.1. [22] *Given φ and a tree decomposition of its incidence graph of width k , one can compute the number of satisfying assignments of φ using $4^k n$ arithmetic operations.*

Since the problems of counting the number of satisfying assignments of a formula and of evaluating its associated polynomial are closely linked, as remarked in Section 2.2, one can naturally wonder if a small tree-width also ensures an efficient way to compute the associated polynomial.

Koiran and Meer not only established that, but also fully characterized the polynomials obtained by p -projections of families associated to boolean formulas of bounded tree-width (Theorem 5 of [27]).

Theorem 3.1.2. [27] *Let $(f_n)_{n \in \mathbb{N}}$ be a family of polynomials with coefficients in a field \mathbb{K} . The following properties are equivalent:*

- (i) $(f_n)_{n \in \mathbb{N}}$ can be represented by a family of polynomial size arithmetic formulas.
- (ii) There exists a family $(\varphi_n)_{n \in \mathbb{N}}$ of CNF formulas of size polynomial in n and of bounded tree-width such that $f_n(x)$ can be expressed as a p -projection of the family $(P(\varphi_n))$.

It is thus unlikely that hard families such as the permanent could be expressed as polynomials associated to p -formulas of bounded tree-width, since it would imply, among other unexpected results, that $\text{VP} = \text{VNP}$.

However, that last hypothesis is an open question, and very little is known on lower bounds for the permanent.

Fortunately, Koiran and Meer could show unconditionally that fact, that the permanent family is not associated to boolean families of bounded tree-width. We cannot discard the hypothesis that the permanent could be expressed as a p -projection of such polynomials (showing this impossibility would have strong consequences, since it would imply that arithmetic formulas of polynomial size cannot express all VNP families). But they prove

that even a limited kind of projection (limited to the projection on the constant 1) is not sufficient to express the permanent.

More precisely, consider the boolean function PERMUT_n defined in Example 1. Note that the permanent of an $(n \times n)$ -matrix $M = (m_{i,j})$ is given by $\sum_{e \in \{0,1\}^{n^2}} \text{PERMUT}_n(e) \cdot m^e$. Koiran and Meer have shown that the function PERMUT_n cannot be computed by a polynomial size formula of bounded tree-width.

Theorem 3.1.3. [27] *There does not exist a family $\{\varphi_n\}_n$ of CNF formulas $\varphi_n(e, \theta)$ such that the incidence graph is of bounded tree-width, the size of φ_n is polynomially bounded in n , and for all $e \in \{0,1\}^{n \times n}$,*

- $\text{PERMUT}_n(e) = 1 \Rightarrow \exists \theta \varphi_n(e, \theta) = 1,$
- $\text{PERMUT}_n(e) = 0 \Rightarrow \forall \theta \varphi_n(e, \theta) = 0.$

Thus, the permanent family cannot be computed as polynomials associated to formulas of bounded tree-width – or even, via the θ variables, as projections of such polynomials on the constant 1.

Koiran and Meer’s proof uses a known lower bound on the size of an OBDD (ordered binary decision diagram) representing the function PERMUT_n [29, 57]. We propose a different path for establishing this result, taking as an intermediate the nondeterministic communication complexity. We show that a formula which has a low tree-width must have a low communication complexity (Lemma 3.3.1). Besides, we show a lower bound on the communication complexity of the function PERMUT_n . Finally, we get a lower bound on the tree-width of a formula computing the function PERMUT_n (Theorem 3.3.3). This result gives more precision to Koiran and Meer’s statement.

The lower bound on the communication complexity is on the primal graph of a boolean formula. We first have to justify the replacement of a formula’s incidence graph by its primal graph.

Proposition 3.1.4. [27] *Let $\varphi = C_1 \wedge \dots \wedge C_m$ be a CNF formula with n variables x_1, \dots, x_n such that its incidence clause graph $I(\varphi)$ has tree-width k . Then there is a CNF formula $\tilde{\varphi}(x, y)$ such that the following conditions are satisfied:*

- each clause of $\tilde{\varphi}$ has at most $k + 3$ literals;
- the primal graph $P(\tilde{\varphi})$ has tree-width at most $4(k + 1)$. A tree-decomposition can be constructed in linear time from one of $I(\varphi)$;
- the number of variables and clauses in $\tilde{\varphi}$ is linear in n ;
- for all $x^* \in \{0,1\}^n$, we have $\varphi(x^*) = 1$ if and only if there exists a y^* such that $\tilde{\varphi}(x^*, y^*) = 1$. Such a y^* is, moreover, unique.

PROOF. [of Proposition 3.1.4, taken from [27].] Let $(T, \{X_t\}_t)$ be a (binary) tree-decomposition of $I(\varphi)$. Let C be a clause of φ and T_C the subtree of T induced by C . We replace C bottom up in T_C by introducing $O(n)$ many new variables and clauses. More precisely, start with a leaf box X_t of T_C . Suppose it contains \mathbb{K} variables that occur in

literals of C , without loss of generality say $x_1 \vee \dots \vee x_k$. Note that since C itself is contained in X_t , there are at most \mathbb{K} many variables included. Introduce a new variable y_t together with $k + 1$ many clauses expressing the equivalence $y_t \Leftrightarrow x_1 \vee \dots \vee x_k$. Each of the new clauses has at most $k + 1$ many literals.

Next, consider an inner node t of T_C having two sons t_1, t_2 . Suppose x'_1, \dots, x'_k to be those variables in X_t that occur as literals in C , again without loss of generality in the form $x'_1 \vee \dots \vee x'_k$. If y_{t_1} and y_{t_2} denote the new variables related to C that have been introduced for X_{t_1} and X_{t_2} , for X_t define a new variable y_t together with clauses expressing $y_t \Leftrightarrow y_{t_1} \vee y_{t_2} \vee x'_1 \vee \dots \vee x'_k$. Again, there are at most $k + 3$ new clauses containing at most $k + 3$ literals each. Finally, if t is the root of T_C we define y_t as before and add the clause y_t . Thus, we add for each node X_t at most $k + 4$ new clauses as well as one new variable.

Do the same for all clauses of φ . This results in a CNF formula $\tilde{\varphi}$ which depends on $O(m \cdot n)$ additional variables y and contains $O(m \cdot n \cdot k)$ many clauses. The construction guarantees that $\varphi(x)$ holds if and only if there exists a y such that $\tilde{\varphi}(x, y)$, and in that case y is unique.

A tree-decomposition of the primal graph $P(\tilde{\varphi})$ is obtained as follows. For each occurrence of a clause C in X_t of T replace the c-vertex by the newly introduced variables of the tuple y related to the clause and the box X_t . In addition, for boxes X_t, X_{t_1}, X_{t_2} such that t_1, t_2 are sons of t include the variables y_{t_1}, y_{t_2} also in the upper box X_t . The x_i variables that previously occurred are maintained. Since for a single box X_t at most three y_j are included for each clause, and since there are at most $k + 1$ c-vertices in an original box, the tree-width of $P(\tilde{\varphi})$ is $\leq 4(k + 1)$. The decomposition satisfies the requirements of a tree-decomposition since we did not change occurrences of the x_i 's and the only y_t -variables that occur in several boxes occur in two consecutive ones. \square

3.2 Communication complexity

Our proofs below rely on the notion of communication complexity. The model generally considered in communication complexity was introduced by Yao [58]. In this model, a boolean input is divided between two parties, that we call processors. Those processors must compute a given function of this input. To do so, since each processor has only a partial input, they need to share information: they will send bits to each other until one processor, say the second one, returns the value of the function on the given input (cf. Figure 3.1).

We then say that the processors have computed the function in common. We briefly recall some definitions. For more on this, see [30].

Definition 3.2.1. Let $f : \{0, 1\}^n \mapsto \{0, 1\}$ be a Boolean function.

- a) Consider a partition of the n variables of f into two disjoint sets $x = \{x_1, \dots, x_{n_1}\}, y = \{y_1, \dots, y_{n_2}\}, n_1 + n_2 = n$. The communication complexity of f with respect to (x, y) is the lowest amount of bits that two processors, the first working on the variables x and the second on the variables y , need to exchange in order to compute f in common.
- b) The one-way communication complexity of f with respect to (x, y) is the lowest amount of exchanged bits needed to compute f if only one processor is allowed to send bits

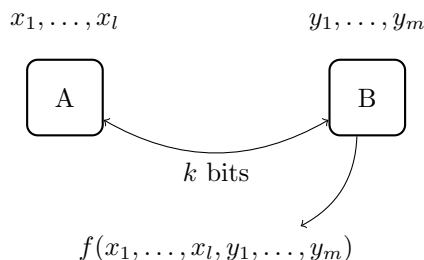


Figure 3.1: Illustration of the communication complexity between two processors A and B.

to the other.

- c) If above we only allow partitions of the variables of same cardinality, i.e., n is even and $|x| = |y|$, and minimize over all of them, we obtain the best-case and best-case one-way communication complexity, respectively.
- d) The non-deterministic communication complexity of f with respect to (x, y) is the lowest amount of bits that two processors, the first working on the variables x , the second on the variables y , and each having access to a source of non deterministic bits, need to exchange in order to compute in common the function f in the following sense :
 - If $f(x) = 1$, at least one of the possible non-deterministic computations must be accepting
 - If $f(x) = 0$, all the non-deterministic computations must be non-accepting.

Remark that in the non-deterministic model, one-way and general communication complexities are equivalent, since each processor can guess the bits that will be exchanged, and then, if the first processor sends its guesses, the other one can verify, that its own guesses were consistent with those of its partner.

A useful approach in communication complexity consists in considering for a given function $f(u, v)$ the matrix associated to it.

Definition 3.2.2. Let $f : U \times V \rightarrow \{0, 1\}$ be a boolean function.

- a) We call the matrix of f the matrix $(f(u, v))$, where the different assignments of u denote the rows and those of v denote the columns. Note that the matrix is a $|U| \times |V|$ matrix; that is, of size exponential in the length of the input vector.
- b) A rectangle of the matrix $(f(u, v))$ is a set of entries composed of the intersection of a certain set of rows and a certain set of columns. That is, a set of entries R is a rectangle if and only if the following is true : $\exists \tilde{U} \subseteq U, \tilde{V} \subseteq V$ such that $R = \tilde{U} \times \tilde{V}$. Equivalently, a set of entries R is a rectangle if and only if the following is true :

$$\forall (u_1, u_2, v_1, v_2) \in U^2 \times V^2, (u_1, v_1) \in R \wedge (u_2, v_2) \in R \Rightarrow (u_1, v_2) \in R.$$

	000	001	010	011	100	101	110	111
000	0	1	1	0	1	0	0	0
001	1	0	0	0	0	0	0	1
010	1	0	0	0	1	0	0	0
011	0	0	1	0	0	0	0	1
100	1	0	0	0	1	0	0	1
101	1	1	1	0	0	0	1	1
110	0	0	0	0	1	0	0	0
111	0	1	1	0	1	1	0	1

Figure 3.2: A 0-monochromatic rectangle

- c) A rectangle of the matrix $(f(u, v))$ is called monochromatic if f has the same value on each entry of the rectangle (cf. Figure 3.2).

The following two results are classical in communication complexity [30, 58]. They link the various communication complexities of a boolean function f with respect to a partition (U, V) of the variables to different parameters of the matrix of f .

Theorem 3.2.3. *Let $f(x, y)$ be a function over two boolean vectors x and y .*

- (i) *The one-way communication complexity of f equals the logarithm of the number of different rows in the matrix $(f(u, v))$.*
- (ii) *The non-deterministic communication complexity of f equals the logarithm of the minimal number of monochromatic rectangles of the matrix $(f(u, v))$ needed to cover all values 1 in the matrix.*

3.3 The lower bound

For the lower bound proof, the non-deterministic communication complexity with respect to certain partitions is the crucial notion. The following lemma relates it to the path-width of primal graphs. Recall that the path-width of a graph with n nodes is bounded from above by $O(t \cdot \log n)$, where t denotes its tree-width (Proposition 1.2.3).

Lemma 3.3.1. *Let $\phi(e, \theta)$ be a CNF formula depending on $n+s$ variables and $f : \{0, 1\}^n \mapsto \{0, 1\}$ a Boolean function such that :*

- *if $\phi(e, \theta) = 1$, then $f(e) = 1$*
- *if $f(e) = 1$, then there exists a θ such that $\phi(e, \theta) = 1$.*

Consider an arbitrary path-decomposition (X_1, \dots, X_p) of the primal graph $P(\phi)$ of width k . Choose a node X_i of the decomposition and a partition x, y of the variables e such that all variables of type e that have already occurred among those in X_1, \dots, X_{i-1} are distributed to x , and all the ones that never occur in X_1, \dots, X_i , to y . Then the non-deterministic communication complexity of f with respect to (x, y) is at most $k + 2$.

PROOF. We split ϕ as follows into two CNF formulas ϕ_1 and ϕ_2 such that $\phi = \phi_1 \wedge \phi_2$ and ϕ_1 and ϕ_2 have at most $k + 1$ variables in common. Formula ϕ_1 is made of all clauses in ϕ that only contain variables that appear in X_1, \dots, X_{i-1} . The remaining clauses are collected in ϕ_2 . Due to the path-width conditions, only variables in X_i can be common variables of ϕ_1 and ϕ_2 .

Note that all variables in x that appear in ϕ_2 must belong to X_i , and that no variables in y appear in ϕ_1 .

Now, given an assignment of the variables (x, y) , let the first processor complete its assignment x by guessing non-deterministically the values of the remaining variables needed to compute ϕ_1 – that is, variables of θ since no variables in y appear in ϕ_1 . Similarly, the second processor completes its assignment of y by guessing the values of the remaining variables appearing in ϕ_2 – variables of θ , and variables of x appearing in X_i as remarked previously.

Let the first processor send to the second processor the result of its computation of ϕ_1 along with the values of the variables in its assignment that ϕ_2 also uses. Those are variables in x appearing in ϕ_2 , and variables from θ that are common to ϕ_1 and ϕ_2 . Thus they all appear in X_i . As a result, the first processor sends at most $|X_i| + 1 \leq k + 2$ bits.

With those values, the second processor can check if the values of its guesses are consistent with the values the first processor had, and if both the computations of ϕ_1 and ϕ_2 are accepting.

Thus, if $e = (x, y)$ does not satisfy f , no guesses of the variables θ could complete e in an assignment that satisfies both ϕ_1 and ϕ_2 and the protocol will never be accepting; and if $f(e) = 1$, then if the two processors guess the proper values to compute ϕ_1 and ϕ_2 on the existing assignment (e, θ) that satisfies ϕ , both ϕ_1 and ϕ_2 will be satisfied, and the protocol will be accepting. \square

Remark 3.3.2. At the end of this section we obtain a similar lemma in order to obtain some results of independent interest relating best-case deterministic communication complexity and path-width.

An outline of the lower bound proof is as follows: Given a CNF formula for the function PERMUT_n and a partition of the variables as above we next define certain permutations called balanced. The number of balanced permutations can be upper bounded in terms of the non-deterministic communication complexity, by Lemma 3.3.7. Then in Lemma 3.3.8 we show that a CNF formula for the permanent function gives rise to a partition of the variables with sufficiently many balanced permutations. Combining this with Lemma 3.3.1 above and the well known relation between path- and tree-width gives the following lower bound result:

Theorem 3.3.3 (lower bound for the permanent). *Let $(\phi_n)_{n \in \mathbb{N}}$ be a family of CNF formulas $\phi_n(e, \theta)$ in n^2 variables $e = (e_{ij})$ and s_n auxiliary variables θ such that :*

- if $\phi_n(e, \theta) = 1$, then the matrix $e \in \{0, 1\}^{n \times n}$ is a permutation matrix
- if $e \in \{0, 1\}^{n \times n}$ is a permutation matrix, then there exists θ such that $\phi_n(e, \theta) = 1$.

Then the path-width $p(n)$ of the primal graphs $P(\phi_n)$ verifies $p(n) = \Omega(n)$, and the tree-width $t(n)$ verifies $t(n) = \Omega(n / \log(n + s_n))$.

As a result, the general permanent function cannot be expressed by a family of CNF formulas with a polynomial number of auxiliary variables and an incidence graph of bounded tree-width.

Remark 3.3.4. Unlike in the statement of Theorem 3.1.3, the above lower bounds are independent of the size of the CNF formulas.

Remark 3.3.5. It seems possible to improve the $t(n) = \Omega(n/\log(n + s_n))$ lower bound by working directly with tree decompositions instead of path decompositions. The proofs would get more cumbersome but do not seem to require new ideas. We will therefore stick to path decompositions in the remainder of this section.

We proceed as outlined above with:

Definition 3.3.6. Let $e = (e_{ij})_{1 \leq i, j \leq n}$ be a matrix of boolean variables and (x, y) a partition of the variables e into two disjoint blocks x and y . A permutation $\pi : \{1, \dots, n\} \mapsto \{1, \dots, n\}$ is called *balanced with respect to the partition (x, y)* if among the n variables $e_{i, \pi(i)}$, $1 \leq i \leq n$ precisely $\lceil \frac{n}{2} \rceil$ belong to x and $\lfloor \frac{n}{2} \rfloor$ belong to y .

Thus, if (e_{ij}) represents the matrix of a permutation π and if π is balanced with respect to (x, y) , then (almost) half of those e_{ij} with value 1 belong to x and the other half to y .

Lemma 3.3.7. Consider a $n \times n$ matrix of boolean indeterminates $e = (e_{ij})$ and let x, y be a partition of e . If there are m balanced permutations with respect to (x, y) , then the non-deterministic communication complexity c of $\text{PERMUT}_n(e)$ with respect to (x, y) satisfies

$$m \leq 2^c \cdot (\lceil n/2 \rceil!)^2 .$$

PROOF. Consider the matrix $(\text{PERMUT}_n(x, y))$ as defined in Theorem 3.2.3, where rows and columns are marked by the possible assignments for x and y , respectively. If π is a permutation which is balanced with respect to (x, y) , we denote by $(x(\pi), y(\pi))$ the corresponding assignments for the (e_{ij}) and we denote by $R(\pi)$ the row of index $x(\pi)$ in the communication matrix $(\text{PERMUT}_n(x, y))$.

We wish to compute an upper bound K such that any monochromatic rectangle covers at most K balanced permutations. The point then is that the communication matrix will have at least m/K distinct rectangles since there are m balanced permutations. We can then conclude that $m \leq 2^c \cdot K$ by Theorem 3.2.3.

Towards this aim let A be a monochromatic rectangle covering the value 1 corresponding to π in the matrix. This rectangle is the intersection of a certain set of rows and a certain set of columns. Since π is covered by A , $R(\pi)$ belongs to that set of rows. Let C be one of the columns.

The intersection of $R(\pi)$ and C belongs also to A , and thus contains a 1. Therefore, the assignment y_c indexing C completes $x(\pi)$ in a satisfying assignment of f_n . Since π is balanced, there are $\lceil n/2 \rceil$ variables set to 1 in $x(\pi)$. If $x(\pi), y_c$ are to form a permutation matrix, y_c must have exactly $\lfloor n/2 \rfloor$ variables set to 1, distributed in the intersection of the $\lceil n/2 \rceil$ rows and columns of e without any 1 in the assignment $x(\pi)$.

Consequently, there are at most $\lfloor n/2 \rfloor!$ possible values for y_c , and thus at most $\lfloor n/2 \rfloor!$ possible columns in A . Symmetrically, there are at most $\lceil n/2 \rceil!$ possible rows in A . Finally

one can take $K = \lceil n/2 \rceil! \cdot \lfloor n/2 \rfloor!$, and the conclusion of the lemma follows from the inequality $m \leq 2^c \cdot K$. \square

The final ingredient for the lower bound proof is

Lemma 3.3.8. *Let $\phi_n(e, \theta)$ be a CNF formula in n^2 variables $e = (e_{ij})$ and s_n auxiliary variables θ such that :*

- if $\phi_n(e, \theta) = 1$, then the matrix $e \in \{0, 1\}^{n \times n}$ satisfies $\text{PERMUT}_n(e) = 1$,
- if $\text{PERMUT}_n(e) = 1$, $e \in \{0, 1\}^{n \times n}$, then there exists θ such that $\phi_n(e, \theta) = 1$.

Then there exists a partition of e into two sets of variables x, y such that this partition is as in the statement of Lemma 3.3.1 for $f = \text{PERMUT}_n$ and such that there are at least $n! \cdot n^{-2}$ many balanced permutations with respect to (x, y) .

PROOF. Let (X_1, X_2, \dots, X_p) be the nodes of a path-decomposition of $P(\phi_n)$ (in that order). We define an ordering on the e_{ij} 's as follows: for an e_{ij} , let $\underline{X}(e_{ij})$ be the first node in the path-decomposition containing e_{ij} . We set $e_{ij} < e_{kl}$ if $\underline{X}(e_{ij}) < \underline{X}(e_{kl})$. If both values are equal for e_{ij} and e_{kl} we order them arbitrarily but in a consistent way to achieve transitivity.

Consider a permutation π . There are precisely n variables of the form $e_{i\pi(i)}$. We pick, according to the above order, the $\lceil \frac{n}{2} \rceil$ -th among those and denote it by e_π . Thus, among the $e_{i\pi(i)}$ exactly $\lfloor \frac{n}{2} \rfloor$ are greater than e_π and $\lceil \frac{n}{2} \rceil$ are less than or equal to e_π with respect to the defined order. By the pigeonhole principle there is at least one variable e_ℓ among the n^2 many e_{ij} 's such that for at least $\frac{n!}{n^2}$ many permutations of $\{1, \dots, n\}$ we get that same e_ℓ by the above procedure, i.e., $e_\pi = e_\ell$ for all those π . We choose a partition (x, y) of the e_{ij} as follows. The part x consists of all the variables e_{ij} that are less than or equal to e_ℓ , and the part y of the variables that are greater than e_ℓ . The partition (x, y) is as stated in Lemma 3.3.1, where the node $\underline{X}(e_\ell)$ plays the role of the X_i in the Lemma. The above arguments imply that at least $\frac{n!}{n^2}$ many permutations are balanced with respect to this partition. \square

We can now prove our lower bound on the tree-width of PERMUT_n .

PROOF. [of Theorem 3.3.3] Let ϕ_n be as in the theorem's statement. According to Lemma 3.3.8 there is a partition of the variables with at least $\frac{n!}{n^2}$ many balanced permutations. According to Lemmas 3.3.7 and 3.3.1 the path-width k of $P(\phi_n)$ satisfies

$$\frac{n!}{n^2} \leq 2^{k+2} \times (\lceil n/2 \rceil!)^2 .$$

Using Stirling's formula, we deduce that $k = \Omega(n)$. Now the tree-width t of ϕ_n satisfies $t \in \Omega(k/\log(n + s_n))$, which results in $t \in \Omega(n/\log(n + s_n))$. Finally, the statement about the tree-width of ϕ_n 's incidence graph follows from Proposition 3.1.4. \square

Remark 3.3.9. The lower bound obtained above seems not derivable from the known lower bounds on computing the permanent with monotone arithmetic circuits, see, e.g., [26]. The tree-width based algorithms for polynomial evaluation like the one in [22] are not monotone since they rely on the principle of inclusion and exclusion.

3.4 Further results

We can observe that the proof above can be simply adapted for boolean formulas accepting $n \times n$ matrices of injective partial maps from $[1, n]$ to $[1, n]$ (cf. Section 2.3). Indeed, the permutations are also injective partial maps, and one can still consider the number of permutations balanced by a partition of the variables in the case of the function accepting matrices of partial maps. Thus, the same lower bound holds for the partial permanent PER_n^* .

We remarked that, thanks to the θ variables in Theorem 3.3.3, it is established that the permanent family could not be obtained as a projection of polynomials associated to formulas of bounded tree-width, where some variables are projected on the constant 1. We can further notice that a projection on the constant 0 does not increase the tree-width.

Lemma 3.4.1. *Let $Q(Y_1, \dots, Y_n)$ be a polynomial associated to a boolean formula $\phi(\varepsilon_1, \dots, \varepsilon_n)$, and let t denote the tree-width of ϕ . If a polynomial P with k variables such that $k \leq n$ can be expressed as a projection $P(X_1, \dots, X_k) = Q(X_1, \dots, X_k, 0, \dots, 0)$, then P is associated to a formula of size smaller than ϕ and of tree-width at most t .*

PROOF. P is the polynomial associated to the formula ψ obtained from ϕ by removing all clauses containing a negated literal ε_i , $i > k$, and removing from each clause all non-negated literals $\bar{\varepsilon}_i$, $i > k$. Indeed, since the variables X_i for $i > k$ are set to 0, the nonzero monomials in P are exactly those for which all corresponding variables ε_i are set to 0.

A tree decomposition of ϕ is obviously also a tree decomposition of ψ , and the size of ψ is clearly smaller than the size of ϕ . \square

One remarks that even if the permanent cannot be obtained from polynomials associated to formulas of bounded tree-width via projections on 0 or 1, we cannot conclude that on the field \mathbb{F}_2 , the permanent cannot be obtained via any p -projection. Indeed, we cannot even prove that the projection of a variable on another (for instance, $P(X) = Q(X, X)$) do not increase the tree-width.

We close this subsection by strengthening slightly Lemma 3.3.1 in order to apply it also to the best-case one way communication complexity (Definition 3.2.1) and obtain some lower bound results of independent interest.

Lemma 3.4.2. *Let ϕ be a CNF formula depending on $2n$ variables. Assume that the primal graph $P(\phi)$ has path-width $k - 1$. Then, ϕ can be expressed as $\phi_1 \wedge \phi_2$ for CNF formulas ϕ_1, ϕ_2 such that both have at most k variables in common, and ϕ_2 depends on at least n variables and ϕ_1 on at least $n - k$ variables that do not occur in the other formula.*

PROOF. We briefly sketch how the splitting of ϕ done in Lemma 3.3.1 can be performed more carefully such that both formulas ϕ_1 and ϕ_2 depend at least on a certain number of variables. Let (X_1, X_2, \dots, X_p) be a path-decomposition of $P(\phi)$; order the variables once again as done in the proof of Lemma 3.3.8. Denote the ordered sequence by $v_1 < \dots < v_{2n}$. Choose $X_\ell := \underline{X}(v_n)$. Define ϕ_1 as conjunction of those clauses in ϕ containing only variables among the v_1, \dots, v_n and ϕ_2 as conjunction of all remaining clauses. Notice that the n variables v_{n+1}, \dots, v_{2n} do not occur in ϕ_1 . Due to the path-width conditions, the common variables in ϕ_1 and ϕ_2 must be variables in X_ℓ . Thus, there are at most k many. Moreover, X_ℓ contains at most k among the n many variables x_1, \dots, x_n . Therefore at least $n - k$ of these occur for the last time in some $X_{\ell'}$, where $\ell' < \ell$ and ϕ_2 cannot depend on them. \square

As consequence, Lemma 3.3.1 now also holds with respect to the best-case one-way communication complexity (Definition 3.2.1) of the function represented by ϕ .

Corollary 3.4.3. *The best-case one-way communication complexity of a function $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$ is lower than $k + 1$, where $k - 1$ is the path-width of the primal graph of any CNF formula computing f .*

PROOF. Let ϕ be a formula computing f , and $k - 1$ be its path-width. By Lemma 3.4.2, one can write ϕ as a conjunction $\phi_1 \wedge \phi_2$, where ϕ_1 has at least $n - k$ variables not shared with the other formula and ϕ_2 n of such variables. Let us consider the partition (x, y) , where y contains n variables, that belong to ϕ_2 exclusively, and x the n remaining variables. We do have $|x| = |y| = n$.

With this partition, the one way communication complexity of $f(x, y)$ is lower than $k + 1$. Indeed, the processor that has access to the variables x can compute ϕ_1 since the variables in y appear only in ϕ_2 . It can send the result of the computation of ϕ_1 – which is a single bit – along with the values of the at most k common variables to the second processor, which then can compute ϕ_2 and return the value of f .

Thus, the best case one way communication complexity is lower than $k + 1$. \square

If for a function f the best-case one way, or the best case communication complexity is known, then we can use the corollary to deduce lower bounds for the path- and tree-width of CNF formulas representing f .

Example 2. For $x, y \in \{0, 1\}^n$, $1 \leq i \leq n$ consider the boolean function $SEQ(x, y, i)$ which gives result 1 if and only if the string $x = x_0x_1 \dots x_n$ equals the string y shifted circularly by i bits to the right, that is to $y_iy_{i+1} \dots y_{n-1}y_0 \dots y_{i-1}$. It is known [30] that SEQ has a best case communication complexity which is at least linear in the size of the input. Thus, the path-width of the primal graph of any CNF formula computing SEQ is at least linear in the input.

The same argument holds as well for the function $PROD(a, b, i)$, which computes the i -th bit of the product $a \cdot b$, for the function $MATCH$ which on a $3m$ -string x and a m -string y returns 1 iff y is a substring of x , and for the function $USTCON$ which on a graph with ℓ vertices and two given vertices s and t outputs 1 if there exists a path from s to t . As noted in [30], the best-case communication complexity of those function is, respectively, linear, $\Omega(m/\log(m))$ and $\Omega(\sqrt{n})$. Consequently, they do not admit CNF formulas of path-width, respectively, linear, $\Omega(m/\log(m))$ and $\Omega(\sqrt{n})$.

Since the path-width p and the tree-width t are related via $p = O(t \cdot \log n)$, all above mentioned examples do not admit CNF formulas with a primal graph of bounded or even logarithmic tree-width.

Part II

**From complex numbers to
booleans**

Chapter 4

Fast Computation of Zeros of Polynomial Systems with Bounded Degree under Finite-precision

4.1 Introduction

The 17th of the problems for the 21st century posed by Steve Smale [49] asks for an algorithm computing an approximate zero of a polynomial system in average polynomial time.

The problem had occupied Smale during the 1990's and led him, together with Mike Shub, to a series of papers [43, 44, 45, 47, 46, 42] – known as *the Bézout series* – where a number of ideas and results approaching a solution for the 17th problem were proposed. These ideas are at the core of all further research done on Smale's problem.

Paramount within this research is the work of Carlos Beltrán and Luis Miguel Pardo [3, 4, 5] who provided a randomized algorithm computing the desired approximate zero in average expected polynomial time. Here the word “average” refers to expectation over the input data and the word “expected” to expectation over the random choices made by the algorithm¹. One can say that they gave a probabilistic solution to Smale's 17th problem. Further results, including a deterministic algorithm working in average time $N^{\mathcal{O}(\log \log N)}$ – referred to as “nearly polynomial” – are given in [12]. This deterministic algorithm, when restricted to systems with bounded (or even moderately growing) degree, becomes an average polynomial-time algorithm, referred to in [12] as MD.

All the mentioned work (as well as all the work on Smale's 17th problem not mentioned

¹Although technically similar, there is a remarkable difference between the probability distributions considered. The one for the input data is, explicitly or otherwise, claiming some closeness to the distribution of data “in practice.” The only requirement for the distribution for the random choices of the algorithm is, in contrast, that it will be efficiently computable. An undisputed merit of the work of Beltrán and Pardo is to come up with one distribution which is so and, at the same time, allows one to derive complexity bounds.

above) assume infinite precision. As Beltrán and Pardo put it in [5, p. 6]

With the assumption of exact arithmetic [...] the homotopy method [...] is guaranteed to produce an approximate zero of f .

This statement begs the question, what can one do if (as it happens with digital computers) only finite precision is available²? The goal of the present section is to give an answer for systems of moderate degree. To do so, we describe and analyze a finite-precision version MDF of algorithm MD. This version uses variable precision – that is, the algorithm adaptively adjusts its precision – and we give bounds for the largest required precision. Such bounds amount to a bound on the number of bits (or digits) required to store floating point approximations of the complex numbers occurring during the computation and, in this sense, is related to the bit-cost of performing the computation. In fact, if u_* denotes the smallest value of the round-off unit u used during a computation, then the maximum number of bits we will need to approximate the complex numbers occurring during this computation is essentially $|\log u_*|$.

In a related work [2], Leykin and Beltrán implement a digital algorithm for systems with rational coefficients; they manage to perform only rational computations, avoiding thus the finite precision assumptions; furthermore, they control the size of the rational involved. In contrast, we will work with any system with complex coefficients, by considering finite precision approximations of it. Thus, we are able to use the known expectation results on distributions of systems with complex coefficients.

Our main result deals both with complexity and accuracy. Firstly, we show that the complexity (understood as number of arithmetic operations performed) of MDF remains essentially the same as that of MD. Secondly, we exhibit polynomial bounds for the expected (over random input systems f) number of bits necessary to carry out the computation.

Our main result is the following (precise definitions for the intervening notions will be given in the next section).

Theorem 4.1.1. *Let $N(0, \text{Id})$ denote the standard Gaussian in the space $\mathcal{H}_{\mathbf{d}}$ of polynomial systems $f = (f_1, \dots, f_n)$ with f_i homogeneous of degree d_i in $n + 1$ variables. When f is randomly chosen from $N(0, \text{Id})$, algorithm MDF on input f stops almost surely, and when it does so, returns an approximate zero of f . The number of arithmetic operations $\text{cost}_{\text{MDF}}(f)$ of MDF on input f is bounded on the average as*

$$\mathbb{E}_{f \sim N(0, \text{Id})} \text{cost}_{\text{MDF}}(f) = \mathcal{O}(D^3 N^2 (n + 1)^{D+1}).$$

Here $N = \dim_{\mathbb{C}} \mathcal{H}_{\mathbf{d}}$ denotes the size of the input systems and $D := \max\{d_1, \dots, d_n\}$. Furthermore, the finest precision $u_*(f)$ used by MDF on input f is bounded on the average as

$$\mathbb{E}_{f \sim N(0, \text{Id})} \log |u_*(f)| = \mathcal{O}(D^3 N (n + 1)^{D+1})$$

and as a consequence, when D is bounded, the bit-cost of MDF is, on the average, polynomial in the size of the input.

²Incidentally, finite precision analysis for algorithms dealing with multivariate polynomial systems was pioneered by Steve Smale, and Felipe Cucker in [16].

4.2 Preliminaries

4.2.1 Setting and Notation

For $d \in \mathbb{N}$ we denote by \mathcal{H}_d the subspace of $\mathbb{C}[X_0, \dots, X_n]$ of homogeneous polynomials of degree d . For $f \in \mathcal{H}_d$ we write

$$f(X) = \sum_{\alpha} \binom{d}{\alpha}^{1/2} a_{\alpha} X^{\alpha}$$

where $\alpha = (\alpha_0, \dots, \alpha_n)$ is assumed to range over all multi-indices such that $|\alpha| = \sum_{k=0}^n \alpha_k = d$, $\binom{d}{\alpha}$ denotes the multinomial coefficient, and $X^{\alpha} := X_0^{\alpha_0} X_1^{\alpha_1} \dots X_n^{\alpha_n}$. That is, we take for basis of the linear space \mathcal{H}_d the *Bombieri-Weyl* basis consisting of the monomials $\binom{d}{\alpha}^{1/2} X^{\alpha}$. A reason to do so is that the Hermitian inner product associated to this basis is unitarily invariant. That is, if $g \in \mathcal{H}_d$ is given by $g(x) = \sum_{\alpha} \binom{d}{\alpha}^{1/2} b_{\alpha} X^{\alpha}$, then the canonical Hermitian inner product

$$\langle f, g \rangle = \sum_{|\alpha|=d} a_{\alpha} \bar{b}_{\alpha}$$

satisfies, for all elements ν in the unitary group $\mathcal{U}(n+1)$, that

$$\langle f, g \rangle = \langle f \circ \nu, g \circ \nu \rangle.$$

Fix $d_1, \dots, d_n \in \mathbb{N} \setminus \{0\}$ and let $\mathcal{H}_{\mathbf{d}} = \mathcal{H}_{d_1} \times \dots \times \mathcal{H}_{d_n}$ be the vector space of polynomial systems $f = (f_1, \dots, f_n)$ with $f_i \in \mathbb{C}[X_0, \dots, X_n]$ homogeneous of degree d_i . The space $\mathcal{H}_{\mathbf{d}}$ is naturally endowed with a Hermitian inner product $\langle f, g \rangle = \sum_{i=1}^n \langle f_i, g_i \rangle$. We denote by $\|f\|$ the corresponding norm of $f \in \mathcal{H}_{\mathbf{d}}$.

We let $N := \dim_{\mathbb{C}} \mathcal{H}_{\mathbf{d}}$, $D := \max_i d_i$, and $\mathcal{D} := \prod_i d_i$. Also, in the rest of this section, we assume $d_i \geq 2$ for all $i \leq n$ (linear equations can be easily eliminated). In particular, $D \geq 2$.

Let $\mathbb{P}^n := \mathbb{P}(\mathbb{C}^{n+1})$ denote the complex projective space associated to \mathbb{C}^{n+1} and $S(\mathcal{H}_{\mathbf{d}})$ the unit sphere of $\mathcal{H}_{\mathbf{d}}$. These are smooth manifolds that naturally carry the structure of a Riemannian manifold (for \mathbb{P}^n the metric is called Fubini-Study metric). We will denote by $d_{\mathbb{P}}$ and $d_{\mathbb{S}}$ their Riemannian distances which, in both cases, amount to the angle between the arguments. Specifically, for $x, y \in \mathbb{P}^n$ one has

$$\cos d_{\mathbb{P}}(x, y) = \frac{|\langle x, y \rangle|}{\|x\| \|y\|}. \quad (4.1)$$

Occasionally, for $f, g \in \mathcal{H}_{\mathbf{d}} \setminus \{0\}$, we will abuse language and write $d_{\mathbb{S}}(f, g)$ to denote this angle, that is, the distance $d_{\mathbb{S}}\left(\frac{f}{\|f\|}, \frac{g}{\|g\|}\right) = d_{\mathbb{S}}(f, g)$. We define the *solution variety* to be

$$V_{\mathbb{P}} := \{(f, \zeta) \in \mathcal{H}_{\mathbf{d}} \times \mathbb{P}^n \mid f \neq 0 \text{ and } f(\zeta) = 0\}.$$

This is a smooth submanifold of $\mathcal{H}_{\mathbf{d}} \times \mathbb{P}^n$ and hence also carries a Riemannian structure. We denote by $V_{\mathbb{P}}(f)$ the zero set of $f \in \mathcal{H}_{\mathbf{d}}$ in \mathbb{P}^n .

By Bézout's Theorem, $V_{\mathbb{P}}(f)$ contains \mathcal{D} points for all f except in a subvariety. Let $Df(\zeta)|_{T_{\zeta}}$ denote the restriction of the derivative of $f: \mathbb{C}^{n+1} \rightarrow \mathbb{C}^n$ at ζ to the tangent space

$T_\zeta := \{v \in \mathbb{C}^{n+1} \mid \langle v, \zeta \rangle = 0\}$ of \mathbb{P}^n at ζ . The *subvariety of ill-posed pairs* is defined as

$$\Sigma'_\mathbb{P} := \{(f, \zeta) \in V_\mathbb{P} \mid \text{rank } Df(\zeta)|_{T_\zeta} < n\}.$$

Note that $(f, \zeta) \notin \Sigma'_\mathbb{P}$ means that ζ is a simple zero of f . In this case, by the implicit function theorem, the projection $V_\mathbb{P} \rightarrow \mathcal{H}_\mathbf{d}$, $(g, x) \mapsto g$ can be locally inverted around (f, ζ) . The image Σ of $\Sigma'_\mathbb{P}$ under the projection $V_\mathbb{P} \rightarrow \mathcal{H}_\mathbf{d}$ is called the *discriminant variety*.

We say that a property holds *generically* when it is true excepted for elements in a subvariety of the considered set. For instance, a pair (f, ζ) in $V_\mathbb{P}$ is generically well posed. In particular, such property is almost sure for the gaussian densities we consider.

4.2.2 Approximate Zeros, Complexity and Data Distribution

In [41], Mike Shub introduced the following projective version of Newton's method. We associate to $f \in \mathcal{H}_\mathbf{d}$ (with $Df(x)$ of rank n for some x) a map $N_f : \mathbb{C}^{n+1} \setminus \{0\} \rightarrow \mathbb{C}^{n+1} \setminus \{0\}$ defined (generically) by

$$N_f(x) = x - Df(x)|_{T_x}^{-1} f(x).$$

Note that $N_f(x)$ is homogeneous of degree 0 in f and of degree 1 in x so that N_f induces a rational map from \mathbb{P}^n to \mathbb{P}^n (which we will still denote by N_f) and this map is invariant under multiplication of f by constants.

We note that $N_f(x)$ can be computed from f and x very efficiently: since the Jacobian $Df(x)$ can be evaluated with $\mathcal{O}(N)$ arithmetic operations [1], one can do with a total of $\mathcal{O}(N + n^3) = \mathcal{O}(N)$ arithmetic operations, the equality since $d_i \geq 2$ implies $N = \Omega(n^3)$.

It is well-known that when x is sufficiently close to a simple zero ζ of f , the sequence of Newton iterates beginning at x will converge quadratically fast to ζ . This property led Steve Smale to define the following intrinsic notion of approximate zero.

Definition 4.2.1. By an *approximate zero* of $f \in \mathcal{H}_\mathbf{d}$ associated with a zero $\zeta \in \mathbb{P}^n$ of f we understand a point $x \in \mathbb{P}^n$ such that the sequence of Newton iterates (adapted to projective space)

$$x_{i+1} := N_f(x_i)$$

with initial point $x_0 := x$ converges immediately quadratically to ζ , i.e.,

$$d_\mathbb{P}(x_i, \zeta) \leq \left(\frac{1}{2}\right)^{2^i - 1} d_\mathbb{P}(x_0, \zeta)$$

for all $i \in \mathbb{N}$.

It is this notion of approximation that is the one referred to in the statement of Smale's 17th problem.

The last notion necessary to formally state Smale's problem is that of 'average cost'. For the cost of a computation Smale proposes the number of arithmetic operations (this includes comparisons and possibly square roots) performed during the computation. In the case of a finite-precision algorithm one needs to multiply this number by the largest number of bits (or digits) necessary to approximate the complex numbers occurring during the computation.

The word ‘average’ refers to the standard normal distribution for the data (input) system $f \in \mathcal{H}_{\mathbf{d}}$. Recall, we express an element $f \in \mathcal{H}_{\mathbf{d}}$ as a linear combination of the monomials in the Bombieri-Weyl basis. The standard normal distribution corresponds to choosing the coefficients in this combination independently and identically distributed from the centered Gaussian distribution on \mathbb{C} (which in turn amounts to draw real and imaginary parts independently from the centered Gaussian distribution on \mathbb{R}). We denote this distribution on $\mathcal{H}_{\mathbf{d}}$ by $N(0, \text{Id})$.

Hence, if $\text{cost}(f)$ denotes the cost of computing an approximate zero for f with a given algorithm then the average cost of this algorithm, for inputs in $\mathcal{H}_{\mathbf{d}}$, is given by the expected value

$$\mathbb{E}_{f \sim N(0, \text{Id})} \text{cost}(f).$$

We remark that if the cost is homogeneous of degree zero, that is, if $\text{cost}(f) = \text{cost}(\lambda f)$ for all $\lambda \neq 0$, then the expectation above is the same as the expectation with f drawn from the uniform distribution on the unit sphere $S(\mathcal{H}_{\mathbf{d}})$.

Smale’s 17th problem asks for an algorithm computing an approximate zero (in the sense of Definition 4.2.1) with average cost (for the cost and data distribution described above) bounded by $N^{\mathcal{O}(1)}$.

4.2.3 Condition Numbers

How close need x to be from ζ to be an approximate zero? This depends on how well conditioned the zero ζ is.

For $f \in \mathcal{H}_{\mathbf{d}}$ and $x \in \mathbb{C}^{n+1} \setminus \{0\}$ we define the (*normalized*) *condition number* $\mu_{\text{norm}}(f, x)$ by

$$\mu_{\text{norm}}(f, x) := \|f\| \left\| (Df(x)|_{T_x})^{-1} \mathbf{diag} \left(\sqrt{d_1} \|x\|^{d_1-1}, \dots, \sqrt{d_n} \|x\|^{d_n-1} \right) \right\|,$$

where the right-hand side norm denotes the spectral norm and $\mathbf{diag}(a_i)$ denotes the diagonal matrix with entries a_i . Note that $\mu_{\text{norm}}(f, x)$ is homogeneous of degree 0 in both arguments, hence it is well defined for $(f, x) \in S(\mathcal{H}_{\mathbf{d}}) \times \mathbb{P}^n$. Also, it is well known (see [6, Ch. 12, Corollary 3]) that $\mu_{\text{norm}}(f, x) \geq 1$.

The following result (essentially, a γ -Theorem in Smale’s theory of estimates for Newton’s method [48]) quantifies our claim above (see [12] for its proof).

Theorem 4.2.2. *Assume $f(\zeta) = 0$ and $d_{\mathbb{P}}(x, \zeta) \leq \frac{\nu_0}{\mathcal{D}^{3/2} \mu_{\text{norm}}(f, \zeta)}$ where $\nu_0 := 3 - \sqrt{7} \approx 0.3542$. Then x is an approximate zero of f associated with ζ . \square*

The next result, Proposition 4.1 from [12], gives bounds on the variation of the condition number $\mu_{\text{norm}}(f, x)$ when f and x vary.

Proposition 4.2.3. *Assume $\mathcal{D} \geq 2$. Let $0 < \varepsilon \leq 0.13$ be arbitrary and $C \leq \frac{\varepsilon}{5.2}$. For all $f, g \in S(\mathcal{H}_{\mathbf{d}})$ and all $x, y \in \mathbb{C}^{n+1}$, if $d_{\mathbb{S}}(f, g) \leq \frac{C}{\mathcal{D}^{1/2} \mu_{\text{norm}}(f, x)}$ and $d_{\mathbb{P}}(x, y) \leq \frac{C}{\mathcal{D}^{3/2} \mu_{\text{norm}}(f, x)}$, then*

$$\frac{1}{1 + \varepsilon} \mu_{\text{norm}}(g, y) \leq \mu_{\text{norm}}(f, x) \leq (1 + \varepsilon) \mu_{\text{norm}}(g, y). \quad \square$$

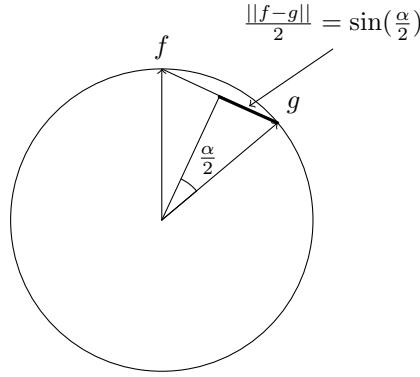


Figure 4.1: Computation of alpha

In what follows, we will fix the constants $\varepsilon := 0.13$ and $C := \frac{\varepsilon}{5.2} = 0.025$.

We also introduce the *mean square condition number* of q given by

$$\mu_2^2(q) := \frac{1}{\mathcal{D}} \sum_{\zeta:q(\zeta)=0} \mu_{\text{norm}}^2(q, \zeta). \quad (4.2)$$

4.2.4 An Adaptive Homotopy Continuation

Suppose that we are given an input system $f \in S(\mathcal{H}_{\mathbf{d}})$ and a pair $(g, \zeta) \in V_{\mathbb{P}}$, where g is also in the unit sphere and such that f and g are \mathbb{R} -linearly independent. Let $\alpha = d_{\mathbb{S}}(f, g)$.

As illustrated in Figure 4.1, one can compute α as

$$\alpha = 2 \arcsin \left(\frac{\|f - g\|}{2} \right). \quad (4.3)$$

Consider the line segment $E_{g,f}$ in $\mathcal{H}_{\mathbf{d}}$ with endpoints g and f . We parameterize this segment by writing

$$E_{g,f} = \{q_{\tau} \in \mathcal{H}_{\mathbf{d}} \mid \tau \in [0, 1]\}$$

with q_{τ} being the only point in $E_{g,f}$ such that $d_{\mathbb{S}}(g, q_{\tau}) = \tau\alpha$. Explicitly, as remarked in [12], we have $q_{\tau} = tf + (1-t)g$, where $t = t(\tau)$ is given by

$$t(\tau) = \frac{1}{\sin \alpha \cot(\tau\alpha) - \cos \alpha + 1}. \quad (4.4)$$

If $E_{g,f} \cap \Sigma = \emptyset$, and hence generically, this segment can be lifted to a path given by a continuous function $[0, 1] \rightarrow V_{\mathbb{P}}$ mapping $\tau \mapsto (q_{\tau}, \zeta_{\tau})$.

In order to find an approximation of the zero ζ_1 of $f = q_1$ we may start with the zero $\zeta = \zeta_0$ of $g = q_0$ and numerically follow the path (q_{τ}, ζ_{τ}) by subdividing $[0, 1]$ into points

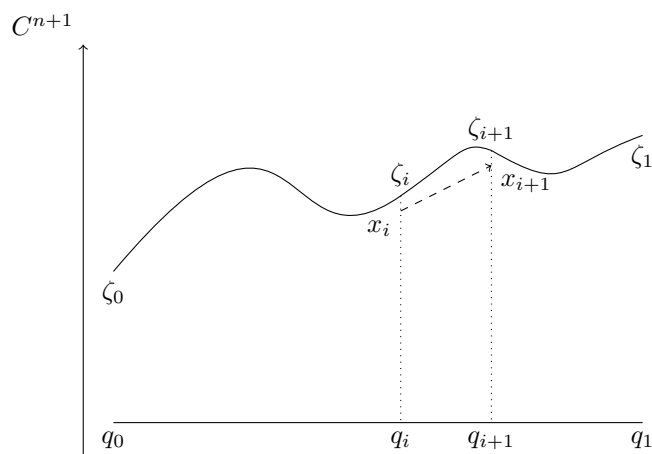


Figure 4.2: The step from q_i to q_{i+1}

$0 = \tau_0 < \tau_1 < \dots < \tau_k = 1$ and by successively computing approximations x_i of ζ_{τ_i} by Newton's method, as illustrated in Figure 4.2

This course of action is the one proposed in the Bézout series and further adopted in [3, 4, 5, 12]. The (infinite precision) continuation procedure is the following (here $\lambda = \frac{C(1-\varepsilon)}{2(1+\varepsilon)^4} \approx 6.67 \cdot 10^{-3}$, see [12]).

Algorithm ALH

input f, g, ζ

$(g, \zeta) \in V, f \neq g$

$\alpha := d_{\mathbb{S}}(f, g), \tau := 0, q_{\tau} := g$

repeat

$\Delta\tau := \frac{\lambda}{\alpha D^{3/2} \mu_{\text{norm}}^2(q_{\tau}, x)}$

$\tau := \min\{1, \tau + \Delta\tau\}$

$q_{\tau} := t(\tau)f + (1 - t(\tau))g$

$x := N_{q_{\tau}}(x)$

$x := x/\|x\|$

until $\tau = 1$

RETURN x

Note that the step-length $\Delta\tau$ depends on $\mu_{\text{norm}}(q_{\tau}, x)$. Hence, the adaptiveness.

The algorithm MD (Moderate Degree) from [12] is a direct application of ALH having as initial pair (g, ζ) the pair (\bar{U}, \mathbf{z}_1) , where $\bar{U} = (\bar{U}_1, \dots, \bar{U}_n) \in S(\mathcal{H}_{\mathbf{d}})$ with $\bar{U}_i = \frac{1}{\sqrt{2n}}(X_0^{d_i} - X_i^{d_i})$ and $\mathbf{z}_1 = \frac{1}{\sqrt{n+1}}(1, \dots, 1)$.

Algorithm MD
input $f \in \mathcal{H}_d$
run ALH on input (f, \bar{U}, z_1)

4.2.5 A Finite-precision setting

A distinctive feature in the present study is that computations are performed with finite-precision. Therefore, a goal in sight is to design a version ALHF of ALH which takes into account the errors proper to this mode of computation, and a finite precision version MDF of MD (we will describe in detail ALHF and MDF in Section 4.4).

Roughly speaking (a more detailed account on finite-precision computations is in §4.3.1), the precision of a computation is governed by a real number $u \in [0, 1]$, called *round-off unit*, with the property that all occurring numbers x in the computation are replaced by a number \tilde{x} such that $|x - \tilde{x}| \leq u|x|$. Algorithms where u remains fixed through the computation are said to have *fixed precision*. Otherwise, they have *variable precision*. The one we propose in this study is of the latter kind.

Key Remark The systems f and g play a very different role in ALHF. The system f is the one we want to solve, and in order to have a general algorithm that works for any system with complex coefficients, we cannot assume that we work with the exact input f . We thus have to work with approximations but, in accordance to our use of variable precision, we want to be able to adjust the precision of these approximations. For the input system f we will therefore assume a routine `read_input` such that `read_input()` returns an approximation of f with the current round-off unit u . Such an assumption is sine qua non if we want our algorithm to be correct.

The system g , in contrast, is used as an auxiliary in the resolution of f , and in our application of ALHF (i.e., in the algorithm MDF) it will be a fixed system (namely, \bar{U}).

4.2.6 Roadmap

We next summarize the main steps of our finite precision analysis and how they are related to the exact precision counterparts in [12]. Algorithms ALHF and MDF, described in detail in Section 4.4, are the finite precision versions of Algorithms ALH and MD in [12]. Theorem 4.4.3 is the finite precision version of [12, Theorem 3.1] and states the main properties of algorithm ALHF. Proposition 4.4.5 provides the backbone for the proof of Theorem 4.4.3. It is a finite precision version of the inductive proof of [12, Theorem 3.1]. The proof of Proposition 4.4.5 requires a number of technical finite precision estimates that we develop in Section 4.3.

4.3 Error Bounds

In this section we show bounds for the basic computations occurring in ALHF. We will use these bounds in subsequent sections to show our main result.

4.3.1 Basic facts

We recall the basics of a floating-point arithmetic which idealizes the usual IEEE standard arithmetic. In contrast to the standard model (as in [24]) we adapt our exposition to complex arithmetic. This system is defined by a set $F \subset \mathbb{C}$ containing 0 (the *floating-point complex numbers*), a transformation $r : \mathbb{C} \rightarrow F$ (the *rounding map*), and a constant $u \in \mathbb{R}$ (the *round-off unit*) satisfying $0 < u < 1$. The properties we require for such a system are the following:

- (i) For any $x \in F$, $r(x) = x$. In particular, $r(0) = 0$.
- (ii) For any $x \in \mathbb{C}$, $r(x) = x(1 + \delta)$ with $|\delta| \leq u$.

We also define on F arithmetic operations following the classical scheme

$$x \tilde{\circ} y = r(x \circ y)$$

for any $x, y \in F$ and $\circ \in \{+, -, \times, /\}$, so that

$$\tilde{\circ} : F \times F \rightarrow F.$$

The following is an immediate consequence of property (ii) above.

Proposition 4.3.1. *For any $x, y \in F$ we have*

$$x \tilde{\circ} y = (x \circ y)(1 + \delta), \quad |\delta| \leq u. \quad \square$$

The concrete implementation of the basic finite precision functions is a major research field, and we will not enter into this discussion. A comprehensive view of the subject can be found in the Arenal team's Handbook of Floating-Point Arithmetic [37].

When combining many operations in floating-point arithmetic, quantities such as $\prod_{i=1}^n (1 + \delta_i)^{\rho_i}$ naturally appear. Our round-off analysis uses the notations and ideas in Chapter 3 of [24], from where we quote the following results:

Proposition 4.3.2. *If $|\delta_i| \leq u$, $\rho_i \in \{-1, 1\}$, and $nu < 1$, then*

$$\prod_{i=1}^n (1 + \delta_i)^{\rho_i} = 1 + \theta_n,$$

where

$$|\theta_n| \leq \gamma_n = \frac{nu}{1 - nu}. \quad \square$$

Proposition 4.3.3. *For any positive integer k such that $ku < 1$, let θ_k, θ_j be any quantities satisfying*

$$|\theta_k| \leq \gamma_k = \frac{ku}{1 - ku} \quad |\theta_j| \leq \gamma_j = \frac{ju}{1 - ju}.$$

The following relations hold.

- 1. $(1 + \theta_k)(1 + \theta_j) = 1 + \theta_{k+j}$ for some $|\theta_{k+j}| \leq \gamma_{k+j}$.

2.

$$\frac{1 + \theta_k}{1 + \theta_j} = \begin{cases} 1 + \theta_{k+j} & \text{if } j \leq k, \\ 1 + \theta_{k+2j} & \text{if } j > k. \end{cases}$$

for some $|\theta_{k+j}| \leq \gamma_{k+j}$ or some $|\theta_{k+2j}| \leq \gamma_{k+2j}$.

3. If $ku, ju \leq 1/2$, then $\gamma_k \gamma_j \leq \gamma_{\min\{k,j\}}$.

4. $i\gamma_k \leq \gamma_{ik}$.

5. $\gamma_k + u \leq \gamma_{k+1}$.

6. $\gamma_k + \gamma_j + \gamma_k \gamma_j \leq \gamma_{k+j}$. □

From now on, whenever we write an expression containing θ_k we mean that the same expression is true for some θ_k , with $|\theta_k| \leq \gamma_k$.

When computing an arithmetic expression q with a round-off algorithm, errors will accumulate and we will obtain another quantity which we will denote by $\mathbf{fl}(q)$. For a complex number, we write $\mathbf{Error}(q) = |q - \mathbf{fl}(q)|$; for vectors or matrices, $\mathbf{Error}(q)$ will denote the vector or matrix of coordinates $|q_\alpha - \mathbf{fl}(q_\alpha)|$, allowing us to choose various norms to estimate this error.

An example of round-off analysis which will be useful in what follows is given in the next proposition, the proof of which follows the lines of the proof of the real version of this result that can be found in Section 3.1 of [24].

Proposition 4.3.4. *There is a finite-precision algorithm which, with input $x, y \in \mathbb{C}^n$, computes the inner product of x and y . The computed value $\mathbf{fl}(\langle x, y \rangle)$ satisfies*

$$\mathbf{fl}(\langle x, y \rangle) = \langle x, y \rangle + \theta_{\lceil \log_2 n \rceil + 1} \sum_{i=1}^n |x_i \bar{y}_i|.$$

In particular, if $x = y$, the algorithm computes $\mathbf{fl}(\|x\|^2)$ satisfying

$$\mathbf{fl}(\|x\|^2) = \|x\|^2(1 + \theta_{\lceil \log_2 n \rceil + 1}). \quad \square$$

We assume that, besides the four basic operations, we are allowed to compute basic trigonometric functions (such as \sin and \cos) and the square root with finite precision. That is, if \mathbf{op} denotes any of these two operators, we compute $\widetilde{\mathbf{op}}$ such that

$$\widetilde{\mathbf{op}}(x) = \mathbf{op}(x)(1 + \delta), \quad |\delta| < u.$$

The following sensitivity results will help us to deal with errors in computing trigonometric functions.

Lemma 4.3.5. (i) *Let $t, \theta \in \mathbb{R}$. Then*

$$|\cos(t + \theta) - \cos t| \leq |\theta|;$$

$$|\sin(t + \theta) - \sin t| \leq |\theta|;$$

(ii) Given two reals a and e such that both a and $a + e$ are in the interval $[0, 0.8]$, one has

$$|\arcsin(a + e) - \arcsin(a)| \leq 2|e|, \text{ with } |v| \leq |e|.$$

PROOF.

(i) Observe that

$$|\cos(t + \theta) - \cos t| = 2 \left| \sin \left(t + \frac{\theta}{2} \right) \right| \left| \sin \frac{\theta}{2} \right| \leq 2 \left| \sin \frac{\theta}{2} \right| \leq |\theta|,$$

and analogously

$$|\sin(t + \theta) - \sin t| = 2 \left| \cos \left(t + \frac{\theta}{2} \right) \right| \left| \sin \frac{\theta}{2} \right| \leq |\theta|. \quad \square$$

(ii) Without loss of generality, let us suppose that $e > 0$.

From the intermediate value theorem, there exists a ξ in $[a, a + e]$ such that $\arcsin(a + e) = \arcsin(a) + e \arcsin'(\xi) = \arcsin(a) + e \frac{1}{\sqrt{1-\xi^2}}$.

Since $\xi \in [a, a + e]$, $|\xi| \leq 0.8$ and thus $|\arcsin'(\xi)| \leq \frac{1}{\sqrt{1-0.8^2}} < 2$. \square

We now introduce a further notation that will considerably simplify our exposition. For an expression g , $\llbracket g \rrbracket$ will denote any expression h such that the following is true :

$$\llbracket gu \leq 1/2 \rrbracket \Rightarrow h = \mathcal{O}(ug).$$

Here the ‘big Oh’ notation is with respect to u .

We thus avoid burdening ourselves with the consideration of multiplicative constants and terms of degree higher than 1 in u .

Remark 4.3.6. This definition of $\llbracket g \rrbracket$ differs from the one used in [15], where it simply designates $\mathcal{O}(ug)$. This last definition, easier to manipulate, is however tricky when arbitrary and possibly big values of u are considered, which happens in our context when our algorithm first computes the precision u needed.

The next result shows some properties of this notation.

Proposition 4.3.7. *Let f and g be two expressions. The following relations hold:*

1. $\gamma_f = \llbracket f \rrbracket$ whenever γ_f is well defined.
2. $\llbracket f \rrbracket + \llbracket g \rrbracket = \llbracket \max(f, g) \rrbracket$.
3. $\llbracket f \rrbracket \llbracket g \rrbracket = \llbracket \max(f, g) \rrbracket$.
4. If $f \geq 1$, $f \llbracket g \rrbracket = \llbracket fg \rrbracket$.

PROOF.

1. Let us suppose, that $fu \leq 1/2$. In that case, $\gamma_f = \frac{fu}{1-fu} \leq 2fu = \mathcal{O}(fu)$. Thus, $\gamma_f = \llbracket f \rrbracket$.

2, 3. Let us suppose, that $\max(f, g)u \leq 1/2$. Then, both fu and gu are not greater than $1/2$. Thus, $\llbracket f \rrbracket = \mathcal{O}(fu) = \mathcal{O}(\max(f, g)u)$, and similarly $\llbracket g \rrbracket = \mathcal{O}(\max(f, g)u)$.

Thus, $\llbracket f \rrbracket + \llbracket g \rrbracket = \mathcal{O}(\max(f, g)u)$, which proves 2.

Furthermore, since $\llbracket f \rrbracket = \mathcal{O}(fu)$ and since $fu \leq 1/2$, $\llbracket f \rrbracket$ is bounded by a constant, and $\llbracket f \rrbracket \llbracket g \rrbracket = \mathcal{O}(\max(f, g)u)$, which proves 3.

4. Let us suppose, that $fgu \leq 1/2$. Since $f \geq 1$, this ensures that $gu \leq 1/2$.

Thus, $f\llbracket g \rrbracket = f\mathcal{O}(gu) = \mathcal{O}(fgu)$.

This proves, that $f\llbracket g \rrbracket = \llbracket fg \rrbracket$. □

4.3.2 Bounding errors for elementary computations

We now begin showing bounds for the errors in the crucial steps of our algorithm. To avoid burdening the exposition we will do so only for the steps dominating the accumulation of errors and simply warn the reader of the minor steps we consider as exact.

We begin with the evaluation of the errors in computing α . Remark that we suppose $\alpha \leq \pi/2$ in the following lemma. This will be ensured by the computation of α at the beginning of ALHF. If this quantity is more than $\pi/2$, we set $f = -f$, ensuring that $\alpha \leq \pi/2$. We neglect the errors in this operation, and thus suppose in the remainder that $\alpha \leq \pi/2$.

Lemma 4.3.8. *Given f and g in $S(\mathcal{H}_d)$ such that $d_{\mathbb{S}}(f, g) \leq \pi/2$, one can compute $\alpha = d_{\mathbb{S}}(f, g)$ with finite precision such that*

$$\mathbf{fl}(\alpha) = \alpha(1 + \theta_{\mathcal{O}(\log N)}).$$

PROOF. As remarked in (4.3), one can compute $\alpha = d_{\mathbb{S}}(f, g)$ as $\alpha = 2 \arcsin(\frac{\|f-g\|}{2})$.

We can compute the norm $\|f-g\|$ similarly as the vector norm in Proposition 4.3.4. In the case of polynomials in \mathcal{H}_d , the sum is over N coefficients, and thus one proves similarly that $\mathbf{fl}(\|f-g\|^2) = \|f-g\|^2(1 + \theta_{\lceil \log N \rceil + 1})$. Supposing, that we can compute square root with finite precision, one gets

$$\mathbf{fl}(\|f-g\|) = \|f-g\|(1 + \theta_{\lceil \log N \rceil + 2}).$$

Remark that since we supposed $d_{\mathbb{S}}(f, g) \leq \pi/2$, $\|f-g\|/2 \leq \arcsin(\pi/4) = 1/\sqrt{2} < 0.71$. We can suppose that u is small enough such that the term $\theta_{\lceil \log N \rceil + 2}$ is smaller than $0.8 - 0.71$, and thus such that $\mathbf{fl}(\|f-g\|/2)$ is also in $[0, 0.8]$. We can thus apply Lemma 4.3.5, and by supposing, that we are able to compute the function \arcsin with finite precision, we conclude that we can compute $\alpha = 2 \arcsin(\frac{\|f-g\|}{2})$ such that

$$\begin{aligned} \mathbf{fl}(\alpha) &= \left(2 \arcsin\left(\frac{\|f-g\|}{2}\right) + 2 \frac{\|f-g\|}{2} \theta_{\mathcal{O}(\log N)} \right) (1 + \theta_{\mathcal{O}(1)}) \\ &= 2 \arcsin\left(\frac{\|f-g\|}{2}\right) (1 + \theta_{\mathcal{O}(\log N)})(1 + \theta_{\mathcal{O}(1)}), \end{aligned}$$

the last line since $|\frac{\|f-g\|}{2}| \leq |\arcsin(\frac{\|f-g\|}{2})|$. □

Proposition 4.3.9. *Given $\tau \in \mathbb{R}_+$, f and g in $S(\mathcal{H}_d)$ such that $d_{\mathbb{S}}(f, g) \leq \pi/2$, we can calculate $t(\tau)$ with finite precision such that*

$$\mathbf{fl}(t) = t(1 + \theta_{\mathcal{O}(\log N)}).$$

PROOF. First of all, observe that

$$\begin{aligned} t(\tau) &= \frac{1}{\sin \alpha \cot(\alpha\tau) - \cos(\alpha) + 1} = \frac{\sin(\tau\alpha)}{\sin \alpha \cos(\tau\alpha) - \cos \alpha \sin(\tau\alpha) + \sin(\tau\alpha)} \\ &= \frac{\sin(\tau\alpha)}{\sin(\alpha - \tau\alpha) + \sin(\tau\alpha)} \\ &= \frac{\sin(\tau\alpha)}{2 \sin\left(\frac{(1-\tau)\alpha + \tau\alpha}{2}\right) \cos\left(\frac{(1-\tau)\alpha - \tau\alpha}{2}\right)} \\ &= \frac{\sin(\tau\alpha)}{2 \sin \frac{\alpha}{2} \cos\left(\left(\frac{1}{2} - \tau\right)\alpha\right)}. \end{aligned}$$

We compute $t(\tau)$ via the last equality. First, we compute α following Lemma 4.3.8. Then, one shows easily using Lemma 4.3.5 that each term in the fraction can be computed with finite precision up to a multiplicative factor $(1 + \theta_{\mathcal{O}(\log N)})$. We conclude using Proposition 4.3.3. \square

The following lemma bounds by $\|q\|$ the value of a polynomial q at any point on the unit sphere.

Lemma 4.3.10. *Given $d \in \mathbb{N}$, $q \in \mathbb{C}[X_0, \dots, X_n]$ homogeneous of degree d and $x \in S(\mathbb{C}^{n+1})$, we have $|q(x)| \leq \|q\|$.*

PROOF. Since our norm $\|\cdot\|$ on $\mathbb{C}[X_0, \dots, X_n]$ is unitarily invariant, for each element $\phi \in \mathcal{U}(n+1)$, one has $\|f \circ \phi\| = \|f\|$.

Taking ϕ such that $\phi(e_1) = x$, one has :

$$|q(x)| = |(q \circ \phi \circ \phi^{-1})(x)| = |(q \circ \phi)(e_1)|.$$

But $|q \circ \phi(e_1)|$ is exactly the coefficient $\binom{d}{d}^{1/2} (q \circ \phi)_{(d \cdot e_1)}$ in the Bombieri-Weyl basis of $\mathbb{C}[X_0, \dots, X_n]$, and thus $|q \circ \phi(e_1)| \leq \|q \circ \phi\| = \|q\|$. \square

Proposition 4.3.11. *Given $q \in S(\mathcal{H}_d)$ and $x \in S(\mathbb{C}^{n+1})$, we can compute $q(x)$ with finite precision u such that*

$$\|\mathbf{Error}(q(x))\| = \llbracket \log N + D \rrbracket.$$

PROOF. For $i \leq n$, write $q_i = \sum c_J x^J$. To compute $q_i(x)$ we compute each monomial $c_J x^J$ first, and then evaluate the sum. We have

$$\mathbf{fl}(c_J x^J) = c_J x^J (1 + \theta_{d_i+1}),$$

and thus $\mathbf{Error}(c_J x^J) \leq |c_J| |x|^J \gamma_{d_i+1}$.

As

$$\mathbf{fl}(q_i(x)) = \mathbf{fl}\left(\sum c_J x^J\right),$$

using pairwise summation (see section 4.2 in [24]) we have

$$\begin{aligned} \mathbf{Error}(q_i(x)) &= \left| \sum \mathbf{fl}(c_J x^J) - \sum (c_J x^J) \right. \\ &\quad \left. + \sum \mathbf{fl}(c_J x^J) \theta_{\lceil \log_2 N \rceil} \right| \\ &\leq \sum \mathbf{Error}(c_J x^J) + \sum |c_J x^J| \gamma_{\lceil \log_2 N \rceil} \\ &\quad + \sum \mathbf{Error}(c_J x^J) \gamma_{\lceil \log_2 N \rceil} \\ &\leq \sum |c_J| |x|^J (\gamma_{D+1} + \gamma_{\lceil \log_2 N \rceil} + \gamma_{D+1} \gamma_{\lceil \log_2 N \rceil}) \\ &\leq \sum |c_J| |x|^J \gamma_{\lceil \log_2 N \rceil + D + 1}. \quad (\text{by Proposition 4.3.3 6.}) \end{aligned}$$

Note that $\sum |c_J| |x|^J \leq \|q_i\|$, by applying Lemma 4.3.10 to the polynomial of coefficients $|c_J|$, which has the same norm as q_i , at the point $|x| \in S(\mathbb{C}^{n+1})$. Hence,

$$\mathbf{Error}(q_i(x)) \leq \|q_i\| \gamma_{\lceil \log_2 N \rceil + D + 1},$$

and therefore,

$$\|\mathbf{Error}(q(x))\|^2 \leq \gamma_{\lceil \log_2 N \rceil + D + 1}^2 \sum_i \|q_i\|^2 = \gamma_{\lceil \log_2 N \rceil + D + 1}^2 \|q\|^2 = \gamma_{\lceil \log_2 N \rceil + D + 1}^2.$$

Finally, by Proposition 4.3.7, we have

$$\|\mathbf{Error}(q(x))\| = \llbracket \log N + D \rrbracket. \quad \square$$

4.3.3 Bounding the error in the computation of $\mu_{\text{norm}}^{-1}(q, x)$

The bounds in $\mathbf{Error}(\mu_{\text{norm}}^{-1}(q, x))$ scale well with q . Hence, to simplify notation, in all what follows we assume $\|q\| = 1$.

The main result in this subsection is the following.

Proposition 4.3.12. *Given $q \in S(\mathcal{H}_d)$ and $x \in S(\mathbb{C}^{n+1})$ we can compute $\mu_{\text{norm}}^{-1}(q, x)$ satisfying*

$$\mathbf{Error}(\mu_{\text{norm}}^{-1}(q, x)) = \llbracket n(\log N + D + n) \rrbracket.$$

Note that under the assumption $\|q\| = 1$ our condition number becomes

$$\mu_{\text{norm}}(q, x) := \left\| (Dq(x)|_{T_x})^{-1} \mathbf{diag}(\sqrt{d_1}, \dots, \sqrt{d_n}) \right\|.$$

Given $q \in S(\mathcal{H}_d)$ and $x \in S(\mathbb{C}^{n+1})$, let $M_q \in \mathbb{C}^{n \times n}$ be a matrix representing the linear operator

$$\begin{bmatrix} \frac{1}{\sqrt{d_1}} & & & \\ & \frac{1}{\sqrt{d_2}} & & \\ & & \ddots & \\ & & & \frac{1}{\sqrt{d_n}} \end{bmatrix} Dq(x)|_{T_x} \quad (4.5)$$

in some orthonormal basis of T_x (note that M_q depends also on x ; that point x will always be clear from the context). We then have $\mu_{\text{norm}}^{-1}(q, x) = \|M_q^{-1}\|^{-1} = \sigma_{\min}(M_q)$ where σ_{\min} denotes smallest singular value. We will compute $\mu_{\text{norm}}^{-1}(q, x)$ by computing M_q and then $\sigma_{\min}(M_q)$.

The following proposition contains several technical ideas that will help us to deal with the matrices $Dq(x)|_{T_x}$ and M_q . We use ideas from the proof of [15] modifying them to the complex case.

Proposition 4.3.13. *Let $q \in \mathcal{H}_{\mathbf{d}}$ and $x \in S(\mathbb{C}^{n+1})$. Then the following statements are true:*

(i) *The restriction of the derivative of q to the tangent space T_x can be calculated as follows:*

$$Dq(x)|_{T_x} = Dq(x)H,$$

where $H \in \mathbb{C}^{(n+1) \times n}$ is the matrix made with the first n columns of the Householder symmetry matrix H_x defined by

$$H_x = I_{n+1} - 2yy^*, \quad y = \frac{x - e_{n+1}}{\|x - e_{n+1}\|}.$$

(ii)

$$\left\| \text{diag} \left(\frac{1}{\sqrt{d_1}}, \dots, \frac{1}{\sqrt{d_n}} \right) Dq(x)|_{T_x} \right\| \leq \|q\|.$$

(iii)

$$\|Dq(x)|_{T_x}\| \leq \sqrt{D}\|q\|.$$

PROOF. (i) Observe that the first n columns of H_x are an orthonormal basis of T_x , as H_x swaps vectors x and e_{n+1} , and fixes the subspace y^\perp .

(ii) Let $g = q \circ H_x$. Then, differentiating the equality $g_i(H_x^*x) = q_i(x)$ and multiplying both sides by H on the right, we have

$$Dg_i(e_{n+1})H_x^*H = Dq_i(x)H = Dq_i(x)|_{T_x}, \quad (4.6)$$

where the last equality is by (i). Observe that $H_x^*H = [e_1, \dots, e_n]$, hence,

$$Dg_i(e_{n+1})H_x^*H = Dg_i(e_{n+1})|_{T_{e_{n+1}}} = \left[\frac{\partial g_i}{\partial X_1}(e_{n+1}), \dots, \frac{\partial g_i}{\partial X_n}(e_{n+1}) \right]. \quad (4.7)$$

If we denote $g_i(X) = \sum_{\alpha} \binom{d}{\alpha}^{1/2} g_{i\alpha} X^\alpha$, it is straightforward that

$$\frac{\partial g_i}{\partial X_j}(e_{n+1}) = \binom{d}{d_i - 1}^{1/2} g_{i(e_j + (d_i - 1)e_{n+1})} = \sqrt{d_i} \cdot g_{i(e_j + (d_i - 1)e_{n+1})}.$$

Therefore, from (4.7),

$$\left\| \frac{1}{\sqrt{d_i}} Dg_i(e_{n+1})|_{T_{e_{n+1}}} \right\|_F^2 = \sum_j g_{i(e_j + (d_i - 1)e_{n+1})}^2 \leq \|g_i\|^2, \quad (4.8)$$

and hence by (4.8) we have

$$\begin{aligned} \left\| \mathbf{diag} \left(\frac{1}{\sqrt{d_1}}, \dots, \frac{1}{\sqrt{d_n}} \right) Dg(e_{n+1})_{T_{e_{n+1}}} \right\|_F^2 &\leq \sum_{i=1}^n \left\| \frac{1}{\sqrt{d_i}} Dg_i(e_{n+1})_{T_{e_{n+1}}} \right\|^2 \\ &\leq \sum \|g_i\|^2 = \|g\|^2. \end{aligned} \quad (4.9)$$

Since the Hermitian inner product associated with the Bombieri-Weyl basis is unitarily invariant, we have

$$\|g\|^2 = \langle g, g \rangle = \langle q \circ H_x, q \circ H_x \rangle = \langle q, q \rangle = \|q\|^2,$$

which by (4.6),(4.7) and (4.9), and since the spectral norm of a matrix is not greater than its Frobenius norm, yields

$$\left\| \mathbf{diag} \left(\frac{1}{\sqrt{d_1}}, \dots, \frac{1}{\sqrt{d_n}} \right) Dq(x)_{T_x} \right\| = \left\| \mathbf{diag} \left(\frac{1}{\sqrt{d_1}}, \dots, \frac{1}{\sqrt{d_n}} \right) Dg(e_{n+1})_{T_{e_{n+1}}} \right\| \leq \|g\| = \|q\|.$$

The relation (iii) can be shown similarly. \square

The following two statements are similar to those proved in [15] in the real case and similar ideas are used in the proofs.

Proposition 4.3.14. *Given $q \in S(\mathcal{H}_d)$ and $x \in S(\mathbb{C}^{n+1})$, we have $\|Dq(x)_{T_x}\| \leq \sqrt{D}$, and we can compute $Dq(x)_{T_x}$ with finite precision such that*

$$\|\mathbf{Error}(Dq(x)_{T_x})\|_F \leq \llbracket n\sqrt{D}(\log N + D + \log n) \rrbracket.$$

PROOF. The inequality $\|Dq(x)_{T_x}\| \leq \sqrt{D}$ follows from $\|q\| = 1$ and Proposition 4.3.13(iii).

We compute $Dq(x)_{T_x}$ as in Proposition 4.3.13(i). Hence each entry (i, j) of the matrix $Dq(x)_{T_x}$ is calculated as the product of $Dq_i(x)$ and the j th column $H_j = (h_{kj})_{1 \leq k \leq n+1}$ of H . Proceeding as in the proof of Proposition 4.3.11 we can compute $\frac{\partial q_i}{\partial X_k}(x)$ with

$$\mathbf{Error} \left(\frac{\partial q_i}{\partial X_k}(x) \right) = \llbracket (\log N + d_i)\sqrt{D} \rrbracket,$$

the factor \sqrt{D} being due to the fact that $\|Dq(x)_{T_x}\| \leq \sqrt{D}$. Observe that to compute $x - e_{n+1}$, we need to perform only one arithmetic operation. Also,

$$(yy^*)_{ij} = \frac{1}{\|x - e_{n+1}\|^2} ((x - e_{n+1})(x - e_{n+1})^*)_{ij}$$

and we therefore have

$$\begin{aligned} \mathbf{fl}((x - e_{n+1})(x - e_{n+1})^*)_{ij} &= \mathbf{fl} \left((x - e_{n+1})_i \overline{(x - e_{n+1})_j} \right) \\ &= ((x - e_{n+1})(x - e_{n+1})^*)_{ij} (1 + \theta_3) \end{aligned}$$

and therefore

$$\begin{aligned} \mathbf{fl}(\|x - e_{n+1}\|^2) &= \mathbf{fl}\left(\sum_{i=1}^n x_i \bar{x}_i + (x_1 - 1)(\bar{x}_1 - 1)\right) \\ &= \|x - e_{n+1}\|^2 (1 + \theta_{\lceil \log_2(n+1) \rceil + 3}). \end{aligned}$$

Here we used pairwise summation bounds again. Finally, by Proposition 4.3.3(2),

$$\mathbf{fl}(yy^*)_{ij} = (yy^*)_{ij} (1 + \theta_{2\lceil \log_2(n+1) \rceil + 6}).$$

Taking into account one more addition and one more multiplication, we get

$$\mathbf{Error}(h_{ij}) = \llbracket 2\lceil \log_2(n+1) \rceil + 8 \rrbracket = \llbracket \log n \rrbracket.$$

Applying Proposition 4.3.4, we conclude

$$\begin{aligned} \mathbf{Error}([Dq(x)|_{T_x}]_{ij}) &= |\mathbf{fl}(\langle Dq_i(x), H_j \rangle) - \langle Dq_i(x), H_j \rangle| \\ &= |\langle \mathbf{fl}(Dq_i(x)), \mathbf{fl}(H_j) \rangle| \\ &\quad + \theta_{\lceil \log_2 n \rceil + 1} \sum_k |Dq_i(x)_k \overline{H_{kj}}| - \langle Dq_i(x), H_j \rangle| \\ &\leq |\langle \mathbf{Error}(Dq_i(x)), H_j \rangle| + |\langle Dq_i(x), \mathbf{Error}(H_j) \rangle| \\ &\quad + |\langle \mathbf{Error}(Dq_i(x)), \mathbf{Error}(H_j) \rangle| + \gamma_{\lceil \log_2 n \rceil + 1} |Dq_i(x)| |H_j| \\ &= \llbracket \sqrt{D} \sqrt{n} \log n \rrbracket + \llbracket \sqrt{n} \sqrt{D} (\log N + D) \rrbracket \\ &\quad + \llbracket \sqrt{D} (\log N + D) \rrbracket \llbracket \sqrt{n} \log n \rrbracket + \llbracket \sqrt{D} \log n \rrbracket \\ &= \llbracket \sqrt{D} \sqrt{n} (\log n + \log N + D) \rrbracket. \end{aligned}$$

This implies

$$\|\mathbf{Error}(Dq(x)|_{T_x})\|_F \leq \llbracket n \sqrt{D} (\log n + D + \log N) \rrbracket. \quad \square$$

Proposition 4.3.15. *Given $q \in S(\mathcal{H}_d)$, $x \in S(\mathbb{C}^{n+1})$ and M_q defined by (4.5), we have $\|M_q\| \leq 1$. In addition, we can compute such a matrix M_q with finite precision u such that*

$$\|\mathbf{Error}(M_q)\|_F = \llbracket n(\log N + D + \log n) \rrbracket.$$

PROOF. The inequality $\|M_q\| \leq 1$ follows directly from Proposition 4.3.13(ii), as $\|q\| = 1$. Floating-point errors can be evaluated exactly as in Proposition 4.3.14; however, one gets rid of the factors \sqrt{D} since the bound on $\|M_q\|$ is better than the bound on $Dq(x)|_{T_x}$. As a counterpart, one has to take into account one more division by $\sqrt{d_i}$ of each entry of the matrix, which slightly changes the constants, but leaves the order in N, D and n unchanged. \square

PROOF OF PROPOSITION 4.3.12. We use ideas from the proof of an analogous proposition in [15]. Let $x \in S(\mathbb{C}^{n+1})$, $q \in S(\mathcal{H}_d)$ and M_q be as in Proposition 4.3.15. Then $\mu_{\text{norm}}^{-1}(q, x) = \sigma_{\min}(M_q) = \|M_q^{-1}\|^{-1}$ and we can compute the first expression by computing the last.

Let $E' = M_q - \mathbf{fl}(M_q)$. By Proposition 4.3.15,

$$\|E'\| \leq \|E'\|_F \leq \llbracket n(\log N + D + \log n) \rrbracket.$$

Let $\mathcal{M}_q = \mathbf{fl}(M_q)$. We compute $\sigma_{\min}(\mathcal{M}_q) = \|M_q^{-1}\|^{-1}$ using a backward stable algorithm (e.g., QR factorization). Then the computed $\mathbf{fl}(\sigma_{\min}(\mathcal{M}_q))$ is the exact $\sigma_{\min}(\mathcal{M}_q + E'')$ for a matrix E'' with

$$\|E''\| \leq cn^2u\|\mathcal{M}_q\|$$

for some universal constant c (see, e.g., [23, 24]). Thus,

$$\mathbf{fl}(\sigma_{\min}(M_q)) = \mathbf{fl}(\sigma_{\min}(\mathcal{M}_q)) = \sigma_{\min}(\mathcal{M}_q + E'') = \sigma_{\min}(M_q + E' + E'').$$

Write $E = E' + E''$. Then, using $\|M_q\| \leq 1$ (by Proposition 4.3.15),

$$\begin{aligned} \|E\| &\leq \|E'\| + \|E''\| \leq \|E'\| + cn^2u\|\mathcal{M}_q\| \leq \|E'\| + cn^2u(\|M_q\| + \|E'\|) \\ &= \llbracket n(\log N + D + \log n) \rrbracket + \llbracket n^2 \rrbracket (1 + \llbracket n(\log N + D + \log n) \rrbracket) \\ &= \llbracket n(\log N + D + \log n) \rrbracket + \llbracket n(\log N + D + n) \rrbracket \\ &= \llbracket n(\log N + D + n) \rrbracket, \end{aligned}$$

using Proposition 4.3.7 in the penultimate row.

Therefore, $\mathbf{fl}(\sigma_{\min}(M_q)) = \sigma_{\min}(M_q + E)$ which implies by [23, Corollary 8.3.2]:

$$\mathbf{Error}(\sigma_{\min}(M_q)) \leq \|E\| < \llbracket n(\log N + D + n) \rrbracket. \quad \square$$

4.3.4 Bounding the error on the Newton step

We next evaluate the error in the computation of a Newton step. Our result is the following.

Proposition 4.3.16. *There exists a universal constant $\mathbf{e} > 0$ such that given a system $q \in S(\mathcal{H}_d)$ and a point $x \in S(\mathbb{C}^{n+1})$, if the precision u satisfies*

$$u \leq \frac{\mathbf{e}}{D^2 \mu_{\text{norm}}^2(q, x) n(D + \log N + \log n)},$$

then the error $\mathbf{Error}(N_q(x))$ satisfies

$$\frac{\|\mathbf{Error}(N_q(x))\|}{\|N_q(x)\|} \leq \frac{C(1 - \varepsilon)}{2\pi(1 + \varepsilon)^2 D^{3/2} \mu_{\text{norm}}(q, x)},$$

where C and ε are the constants introduced in Proposition 4.2.3.

Recall the following result from [24, Chapter 7] (in fact, Theorem 7.2 therein applied to $f = b/\|b\|$ and $E = A/\|A\|$).

Lemma 4.3.17. *Given a linear system $Ax = b$, approximations A' of A and b' of b such that $\|A - A'\| \leq \epsilon$, $\|b - b'\| \leq \epsilon$ and $\epsilon\|A^{-1}\| < 1$, the solution x' of the perturbed system $A'x' = b'$ verifies :*

$$\|x' - x\| \leq \frac{\epsilon\|A^{-1}\|}{1 - \epsilon\|A^{-1}\|} (1 + \|x\|). \quad \square$$

From Propositions 4.3.11 and 4.3.14 we know that given $q \in S(\mathcal{H}_d)$ and $x \in S(\mathbb{C}^{n+1})$, we can compute $q(x)$ with finite precision u such that $\|\mathbf{Error}(q(x))\| = \llbracket D + \log N \rrbracket$, as well as $Dq(x)|_{T_x S(\mathbb{C}^{n+1})}$ such that $\|\mathbf{Error}(Dq(x)|_{T_x S(\mathbb{C}^{n+1})})\|_F = \llbracket n\sqrt{D}(\log N + D + \log n) \rrbracket$.

It follows that there exists a constant K such that for all $q \in S(\mathcal{H}_d)$ and $x \in S(\mathbb{C}^{n+1})$, if u is less than or equal to $1/2n\sqrt{D}(\log N + D + \log n)$, then both the errors on the computation of $q(x)$ and $Dq(x)|_{T_x S(\mathbb{C}^{n+1})}$ are less than or equal to $K \cdot u \cdot n\sqrt{D}(\log N + D + \log n)$. Without loss of generality, one can suppose, that $K \geq 2$.

Lemma 4.3.18. *Let $q \in S(\mathcal{H}_d)$ and $x \in S(\mathbb{C}^{n+1})$. If the precision u satisfies*

$$u \leq \frac{1}{2K\mu_{\text{norm}}(q, x)n\sqrt{D}(\log N + D + \log n)}$$

then we can compute $N_q(x)$ such that

$$\frac{\|\mathbf{Error}(N_q(x))\|}{\|N_q(x)\|} = \llbracket \mu_{\text{norm}}(q, x) \cdot n\sqrt{D}(D + \log N + \log n) \rrbracket.$$

PROOF. We compute $N_q(x) - x$ as the solution of the linear system $Dq(x)|_{T_x} y = q(x)$.

Let u satisfy the bound in the hypothesis. Then, one can compute both $q(x)$ and $Dq(x)|_{T_x S(\mathbb{C}^{n+1})}$ with error at most $K \cdot u \cdot n\sqrt{D}(\log N + D + \log n)$, and thus one can apply Lemma 4.3.17 with $\epsilon = K \cdot u \cdot n\sqrt{D}(\log N + D + \log n)$ to compute $N_q(x) - x$.

Indeed, since $\|Dq(x)|_{T_x}^{-1}\| \leq \mu_{\text{norm}}(q, x)$, we have $\epsilon \|Dq(x)|_{T_x}^{-1}\| \leq \frac{1}{2}$.

The error on $N_q(x) - x$ is bounded by

$$\frac{\epsilon \|Dq(x)|_{T_x}^{-1}\|}{1 - \epsilon \|Dq(x)|_{T_x}^{-1}\|} (1 + \|N_q(x) - x\|).$$

Observe that $N_q(x)$ belongs to the tangent space to the unit sphere T_x ; thus, $\|N_q(x)\| \geq 1$, and

$$1 + \|N_q(x) - x\| \leq 1 + \|N_q(x)\| + \|x\| \leq 3\|N_q(x)\|.$$

Hence,

$$\|\mathbf{Error}(N_q(x) - x)\| \leq 6\epsilon \|Dq(x)|_{T_x}^{-1}\| \|N_q(x)\| = \|N_q(x)\| \llbracket \mu_{\text{norm}}(q, x)n\sqrt{D}(D + \log N + \log n) \rrbracket,$$

using again $\|Dq(x)|_{T_x}^{-1}\| \leq \mu_{\text{norm}}(q, x)$. Then, the computation of $N_q(x)$ from $N_q(x) - x$ is a simple addition and does not change the order of the errors. \square

The proof of Proposition 4.3.16 is now immediate.

4.3.5 Bounding the error for $\Delta\tau$

We evaluate here the errors in the computation of the quantity $\Delta\tau$, that is, the size of the current step in the homotopy.

Proposition 4.3.19. *For $x \in S(\mathbb{C}^{n+1})$, and $f, g, q \in S(\mathcal{H}_d)$ such that $d_{\mathbb{S}}(f, g) \leq \pi/2$ define the quantity*

$$\Delta\tau := \frac{\lambda}{d_{\mathbb{S}}(f, g)D^{3/2}\mu_{\text{norm}}^2(q, x)}.$$

There exists a universal constant $\mathbf{f} > 0$ such that

$$u \leq \frac{\mathbf{f}}{n(\log N + D + n)\mu_{\text{norm}}^2(q, x)} \quad (4.10)$$

implies

$$\mathbf{Error}(\Delta\tau) \leq \frac{1}{4}\Delta\tau.$$

To prove this proposition we rely on the following lemma.

Lemma 4.3.20. *Given $x \in S(\mathbb{C}^{n+1})$ and $q \in S(\mathcal{H}_{\mathbf{d}})$, one can compute $\sigma_{\min}^2(M_q)$ with finite precision u such that*

$$\mathbf{Error}(\sigma_{\min}^2(M_q)) = \llbracket n(\log N + D + n) \rrbracket.$$

PROOF. By Proposition 4.3.12, $\mathbf{Error}(\sigma_{\min}(M_q)) = \llbracket n(\log N + D + n) \rrbracket$. Hence, we have

$$\begin{aligned} |\mathbf{fl}(\sigma_{\min}^2(M_q)) - \sigma_{\min}^2(M_q)| &\leq 2|\sigma_{\min}(M_q)|\llbracket n(\log N + D + n) \rrbracket + \llbracket n(\log N + D + n) \rrbracket^2 \\ &\leq \llbracket n(\log N + D + n) \rrbracket + \llbracket n(\log N + D + n) \rrbracket, \end{aligned}$$

since, by Proposition 4.3.15, $|\sigma_{\min}(M_q)| \leq \|M_q\| \leq 1$. Thus,

$$\mathbf{Error}(\sigma_{\min}^2(M_q)) = \llbracket n(\log N + D + n) \rrbracket. \quad \square$$

PROOF OF PROPOSITION 4.3.19. One has

$$\begin{aligned} \mathbf{fl}(\Delta\tau) &= \mathbf{fl}\left(\frac{\lambda}{\alpha D^{3/2}\mu_{\text{norm}}^2(q, x)}\right) \\ &= \frac{\lambda}{D^{3/2}} \mathbf{fl}\left(\frac{\sigma_{\min}^2(M_q)}{\alpha}\right)(1 + \theta_{\mathcal{O}(1)}) \\ &= \frac{\lambda \mathbf{fl}(\sigma_{\min}^2(M_q))}{\alpha D^{3/2}}(1 + \theta_{\mathcal{O}(\log N)}), \end{aligned}$$

the last equality being from Lemma 4.3.8, and thus by Lemma 4.3.20

$$\begin{aligned} \mathbf{Error}(\Delta\tau) &= \frac{\lambda}{\alpha D^{3/2}}(\llbracket n(\log N + D + n) \rrbracket + \llbracket \log N \rrbracket) \\ &= \frac{\lambda}{\alpha D^{3/2}}\llbracket n(\log N + D + n) \rrbracket. \end{aligned}$$

If u satisfies (4.10) with a value of \mathbf{f} small enough, the term $\llbracket n(\log N + D + n) \rrbracket$ may be bounded by

$$\llbracket n(\log N + D + n) \rrbracket \leq \frac{1}{4\mu_{\text{norm}}^2(q, x)},$$

and consequently

$$\mathbf{Error}(\Delta\tau) \leq \frac{\lambda}{4\alpha D^{3/2}\mu_{\text{norm}}^2(q, x)} = \frac{1}{4}\Delta\tau. \quad \square$$

4.3.6 Bounding the distance between \tilde{q}_τ and q_τ

We evaluate here the error in the computation of q_τ , given f, g , and τ .

Proposition 4.3.21. *There exists a universal constant \mathbf{g} such that the following holds. Let $f, g \in S(\mathcal{H}_d)$ with $d_S(f, g) \leq \frac{\pi}{2}(1 + 1/6)$ be given with roundoff error u . Let $\tau \in [0, 1]$. Then for all pair (q, x) ,*

$$u \leq \frac{\mathbf{g} \cdot C(1 - \varepsilon)}{12(1 + \varepsilon)^5 D^{3/2} \mu_{\text{norm}}^2(q, x) \log N}$$

implies

$$\|\mathbf{f1}(q_\tau) - q_\tau\| \leq \frac{C(1 - \varepsilon)}{12(1 + \varepsilon)^5 D^{3/2} \mu_{\text{norm}}^2(q, x)}.$$

We first bound the distance between the points $tf + (1 - t)g$ and $t\mathbf{f1}(f) + (1 - t)\mathbf{f1}(g)$, without taking into account the error in the computation of t .

Proposition 4.3.22. *Assume that $f, g, \tilde{f}, \tilde{g} \in S(\mathcal{H}_d)$ are such that $d_S(f, g) \leq \frac{\pi}{2}(1 + 1/6)$ and $\|f - \tilde{f}\| \leq 1/6$, $\|g - \tilde{g}\| \leq 1/6$. For $t \in [0, 1]$ define $q = tf + (1 - t)g$ and $\tilde{q} = t\tilde{f} + (1 - t)\tilde{g}$. Then*

$$d_S(q, \tilde{q}) \leq 2 \max \left\{ \|f - \tilde{f}\|, \|g - \tilde{g}\| \right\}.$$

To prove Proposition 4.3.22 we rely on the following lemmas.

We prove here Lemma 4.3.24.

Lemma 4.3.23. *Let $p, q \in \mathcal{H}_d$ with $\|p\|, \|q\| \geq \alpha > 0$, $\|p - q\| \leq \beta$ with $\alpha \geq \beta/2$. Then*

$$\frac{\langle p, q \rangle}{\|p\|\|q\|} \geq 1 - \frac{\beta^2}{2\alpha^2}. \quad (4.11)$$

PROOF. Pick any $p, q \in \mathcal{H}_d$ with $\|p\|, \|q\| \geq \alpha$, $\|p - q\| \leq \beta$, denote $r = (p + q)/2$, and let $s \in \mathcal{H}_d$ be such that $\|s\| = \|p - q\|/2$, $s \perp r$. Then from the orthogonality of r and s we have

$$\|r + s\|^2 = \|r - s\|^2 = \|r\|^2 + \|s\|^2 = \frac{\|p\|^2 + \|q\|^2}{2} \geq \|p\|\|q\| \geq \alpha^2;$$

also,

$$\|(r + s) - (r - s)\| = 2\|s\| = \|p - q\| \leq \beta.$$

Therefore,

$$\frac{\langle r + s, r - s \rangle}{\|r + s\|\|r - s\|} = \frac{\|r\|^2 - \|s\|^2}{4\|p\|\|q\|} \leq \frac{\|p + q\|^2 - \|p - q\|^2}{4\|p\|\|q\|} = \frac{\langle p, q \rangle}{\|p\|\|q\|}.$$

Since $\|r + s\| = \|r - s\|$, we have

$$\min_{\substack{\|p\|, \|q\| \geq \alpha \\ \|p - q\| \leq \beta}} \frac{\langle p, q \rangle}{\|p\|\|q\|} = \min_{\substack{\|p\| = \|q\| \geq \alpha \\ \|p - q\| \leq \beta}} \frac{\langle p, q \rangle}{\|p\|\|q\|}. \quad (4.12)$$

Now assume that $p, q \in \mathcal{H}_d$ with $\|p\| = \|q\| \geq \alpha$, $\|p - q\| \leq \beta$. Let

$$\begin{aligned} p' &= \frac{\beta}{2} \cdot \frac{p - q}{\|p - q\|} + \frac{\sqrt{4\alpha^2 - \beta^2}}{2} \cdot \frac{p + q}{\|p + q\|}; \\ p' &= \frac{\beta}{2} \cdot \frac{p - q}{\|p - q\|} + \frac{\sqrt{4\alpha^2 - \beta^2}}{2} \cdot \frac{p + q}{\|p + q\|}. \end{aligned}$$

It is not difficult to check that $\|p'\| = \|q'\| = \alpha$ and $\|p' - q'\| = \beta$. Moreover,

$$\frac{\langle p', q' \rangle}{\|p'\| \|q'\|} = 1 - \frac{\beta^2}{2\alpha^2} \leq \frac{\|q\|^2 + \|p\|^2}{2\|p\| \|q\|} - \frac{\|p - q\|^2}{2\|p\| \|q\|} = \frac{\langle p, q \rangle}{\|p\| \|q\|}.$$

Therefore,

$$\min_{\substack{\|p\| = \|q\| \geq \alpha \\ \|p - q\| \leq \beta}} \frac{\langle p, q \rangle}{\|p\| \|q\|} = \min_{\substack{\|p\| = \|q\| = \alpha \\ \|p - q\| = \beta}} \frac{\langle p, q \rangle}{\|p\| \|q\|}. \quad (4.13)$$

From (4.12) and (4.13) we have

$$\min_{\substack{\|p\|, \|q\| \geq \alpha \\ \|p - q\|}} \frac{\langle p, q \rangle}{\|p\| \|q\|} = \min_{\substack{\|p\| = \|q\| = \alpha \\ \|p - q\| = \beta}} \frac{\langle p, q \rangle}{\|p\| \|q\|} = 1 - \frac{\beta^2}{2\alpha^2},$$

which shows (4.11). \square

Lemma 4.3.24. *Let $p, q \in \mathcal{H}_d$ with $\|p - q\| \leq \min\{\|p\|, \|q\|\}$. Then*

$$d_{\mathbb{S}}(p, q) < \frac{2}{\sqrt{3}} \cdot \frac{\|p - q\|}{\min\{\|p\|, \|q\|\}}. \quad (4.14)$$

PROOF. From Lemma 4.3.23 we have

$$\cos d_{\mathbb{S}}(p, q) = \frac{\langle p, q \rangle}{\|p\| \|q\|} \geq 1 - \frac{\beta^2}{2\alpha^2},$$

where $\beta = \|p - q\|$, $\alpha = \min\{\|p\|, \|q\|\}$. From the Taylor expansion for cos we obtain

$$\cos d_{\mathbb{S}}(p, q) \leq 1 - \frac{d_{\mathbb{S}}^2(p, q)}{2} + \frac{d_{\mathbb{S}}^4(p, q)}{24},$$

therefore

$$\frac{d_{\mathbb{S}}^4(p, q)}{12} - d_{\mathbb{S}}^2(p, q) + \frac{\beta^2}{\alpha^2} \geq 0.$$

Solving the relevant quadratic equation for $d_{\mathbb{S}}^2(p, q)$, we have

$$d_{\mathbb{S}}^2(p, q) \in \left(-\infty, 6 \left(1 - \sqrt{1 - \frac{\beta^2}{3\alpha^2}} \right) \right] \cup \left[6 \left(1 + \sqrt{1 - \frac{\beta^2}{3\alpha^2}} \right), \infty \right). \quad (4.15)$$

By our assumption $\beta/\alpha \leq 1$, therefore,

$$6 \left(1 + \sqrt{1 - \frac{\beta^2}{3\alpha^2}} \right) > \pi^2,$$

and the interval on the right-hand side of (4.15) is irrelevant (as $d_{\mathbb{S}}(q, p) \leq \pi$ anyway). We have (using $\beta/\alpha \leq 1$ again)

$$d_{\mathbb{S}}^2(q, p) \leq 6 \left(1 - \sqrt{1 - \frac{\beta^2}{3\alpha^2}} \right) = \frac{2}{1 + \sqrt{1 - \frac{\beta^2}{3\alpha^2}}} \cdot \frac{\beta^2}{\alpha^2} < \frac{4\beta^2}{3\alpha^2},$$

which yields (4.14). \square

Lemma 4.3.25. *Let $f, g \in \mathcal{H}_{\mathbf{d}}$, $\|g\| = \|f\| = 1$, and $d_{\mathbb{S}}(f, g) \leq \frac{\pi}{2}(1 + \delta)$. Then, given $t \in [0, 1]$, $q(t) = tf + (1 - t)g$ satisfies*

$$\|q(t)\| \geq \sqrt{1 - \frac{(1 + \delta)^2}{4}}. \quad (4.16)$$

PROOF. Consider the function $\varphi : \mathbb{R} \rightarrow \mathbb{R}$ defined as follows:

$$\varphi(t) = \|q(t)\|^2 = \|g\|^2 + 2t\Re\langle g, f - g \rangle + t^2\|f - g\|^2.$$

Observe that $\min_{t \in \mathbb{R}} \varphi(t)$ is attained at

$$t^* = -\frac{2\Re\langle g, f - g \rangle}{2\|f - g\|^2} = \frac{\|g - f\|^2 - \|g\|^2 - \|f\|^2}{2\|f - g\|^2} = \frac{1}{2},$$

and $\varphi(t^*) = \frac{1}{4}\|f + g\|^2$. We then have

$$\|q(t)\|^2 \geq \frac{1}{4}\|f + g\|^2 = 1 - \frac{\|f - g\|^2}{4} \geq 1 - \frac{d_{\mathbb{S}}^2(f, g)}{\pi^2} \geq 1 - \frac{(1 + \delta)^2}{4},$$

which gives us (4.16). \square

PROOF OF PROPOSITION 4.3.22. Observe that

$$\begin{aligned} \|q - \tilde{q}\| &= \|tf + (1 - t)g - t\tilde{f} - (1 - t)\tilde{g}\| \\ &\leq t\|f - \tilde{f}\| + (1 - t)\|g - \tilde{g}\| \\ &\leq \max\{\|f - \tilde{f}\|, \|g - \tilde{g}\|\} \leq \frac{1}{6}. \end{aligned}$$

From Lemma 4.3.25 applied with $d = \frac{1}{6}$ we have

$$\|q\| \geq \sqrt{1 - \frac{(1 + 1/6)^2}{4}} = \frac{\sqrt{95}}{12},$$

and hence

$$\|\tilde{q}\| \geq \|q\| - \|q - \tilde{q}\| \geq \frac{\sqrt{95}}{12} - 1/6 > \frac{7}{12}.$$

Now applying Lemma 4.3.24, we have

$$d_{\mathbb{S}}(q, \tilde{q}) \leq \frac{2}{\sqrt{3}} \cdot \frac{\|q - \tilde{q}\|}{\min\{\|q\|, \|\tilde{q}\|\}} \leq \frac{2}{\sqrt{3}} \cdot \frac{\max\{\|f - \tilde{f}\|, \|g - \tilde{g}\|\}}{\frac{7}{12}} \leq 2 \max\{\|f - \tilde{f}\|, \|g - \tilde{g}\|\}. \quad \square$$

PROOF OF PROPOSITION 4.3.21. Let us denote $\tilde{f} = \mathbf{fl}(f)$, $\tilde{g} = \mathbf{fl}(g)$ and $\tilde{t} = \mathbf{fl}(t)$. Let \tilde{q}_τ denote $\mathbf{fl}(q_\tau)$ and \hat{q}_τ the system $t\tilde{f} + (1-t)\tilde{g}$. By hypothesis, both $\|f - \tilde{f}\|$ and $\|g - \tilde{g}\|$ are not greater than u .

Thus, by Proposition 4.3.22, if $u \leq 1/6$,

$$\|\hat{q}_\tau - q_\tau\| \leq 2u.$$

From Proposition 4.3.9, $\tilde{t} = t(1 + t_{\mathcal{O}(\log N)})$. thus, there exists a constant \mathbf{g} such that for all A , $u \leq \frac{\mathbf{g}A}{\log N}$ implies

$$\|\hat{q}_\tau - \tilde{q}_\tau\| \leq A/2.$$

By taking $\mathbf{g} \leq 1/6$, $u \leq \frac{\mathbf{g}A}{\log N}$ ensures

$$\|\tilde{q}_\tau - q_\tau\| \leq \|\tilde{q}_\tau - \hat{q}_\tau\| + \|\hat{q}_\tau - q_\tau\| \leq \frac{5}{6}A < A.$$

Proposition 4.3.21 follows by taking $A = \frac{C(1+\varepsilon)}{12(1+\varepsilon)^5 D^{3/2} \mu_{\text{norm}}^2(q, x)}$. \square

4.3.7 Computing u

Along our homotopy, the precision utilized varies with the system q considered. We want to keep this precision at all times within the interval $[\frac{1}{2}\mathbf{B}(q, x), \mathbf{B}(q, x)]$ with

$$\mathbf{B}(q, x) := \frac{k_2}{nD^2(\log N + D + n)\mu_{\text{norm}}^2(q, x)}, \quad (4.17)$$

where k_2 is a universal positive constant (that will be specified in Definition 4.3.29).

Now, since q and x vary at each iteration, one has to update the precision as well. To do so one faces an obstacle. When computing u we actually obtain a quantity $\mathbf{fl}(u)$ which depends on the current precision and this current precision has been computed in the previous iteration. Proposition 4.3.26 below shows that this obstacle can be overcome.

Proposition 4.3.26. *If $u \leq (1 + \varepsilon)^6 \mathbf{B}(q, x)$ then*

$$\text{Error} \left(\frac{3}{4} \mathbf{B}(q, x) \right) \leq \frac{1}{4} \mathbf{B}(q, x).$$

In particular, when computing $u := \frac{3}{4} \mathbf{B}(q, x)$ the computed quantity satisfies $\mathbf{fl}(u) \in [\frac{1}{2} \mathbf{B}(q, x), \mathbf{B}(q, x)]$.

Towards the proof of the proposition above we define

$$B(q, x) := \frac{1}{nD^2(\log N + D + n)\mu_{\text{norm}}^2(q, x)}$$

so that $\mathbf{B}(q, x) = k_2 B(q, x)$. Our first lemma bounds the error in the computation of $B(q, x)$.

Lemma 4.3.27. *Assume $u \leq \frac{1}{2n(\log N + D + n)}$. Then for any $x \in S(\mathbb{C}^{n+1})$ and $q \in S(\mathcal{H}_d)$, one can compute $B(q, x)$ such that*

$$\mathbf{Error}(B(q, x)) = \mathcal{O}\left(\frac{u}{D^2}\right).$$

PROOF. From Lemma 4.3.20, we can compute $\sigma_{\min}^2(M_q)$ with error $\llbracket n(\log N + D + n) \rrbracket$; we can compute $B(q, x)$ such that

$$\mathbf{fl}(B(q, x)) = \frac{\mathbf{fl}(\sigma_{\min}^2(M_q))}{nD^2(\log N + D + n)}(1 + \theta_6),$$

and thus

$$\mathbf{Error}(B(q, x)) = \frac{\llbracket n(\log N + D + n) \rrbracket}{nD^2(\log N + D + n)}.$$

Thus, when $u \leq \frac{1}{2n(\log N + D + n)}$, one has

$$\mathbf{Error}(B(q, x)) = \frac{\mathcal{O}(un(\log N + D + n))}{nD^2(\log N + D + n)} = \mathcal{O}\left(\frac{u}{D^2}\right). \quad \square$$

Let k_3 be such that with the conditions of the previous lemma,

$$\mathbf{Error}(B(q, x)) \leq \frac{k_3 u}{D^2}.$$

Corollary 4.3.28. *Let k_2 be a positive constant such that $k_2 \leq \frac{1}{4k_3(1+\varepsilon)^6}$. The condition $u \leq (1 + \varepsilon)^6 k_2 B(q, x)$ ensures that*

$$\mathbf{Error}\left(\frac{3}{4}k_2 B(q, x)\right) \leq \frac{1}{4}k_2 B(q, x).$$

PROOF. If u is less than or equal to $(1 + \varepsilon)^6 B(q, x)$, since $\mu_{\text{norm}}(q, x)$ is always greater than 1, if we choose k_2 not greater than $\frac{1}{2(1+\varepsilon)^6}$, u will be less than or equal to $\frac{1}{2n(\log N + D + n)}$. Thus, one has

$$\mathbf{Error}(B(q, x)) \leq \frac{k_3 u}{D^2} \leq \frac{k_2 k_3 (1 + \varepsilon)^6}{D^2} B(q, x).$$

Taking $k_2 \leq \frac{1}{4k_3(1+\varepsilon)^6}$ one has $\mathbf{Error}\left(\frac{3}{4}k_2 B(q, x)\right) \leq \mathbf{Error}(k_2 B(q, x)) \leq \frac{1}{4}k_2 B(q, x)$. \square

We now have all the conditions that the constant k_2 must fulfill. Recall that the constants \mathbf{f} , \mathbf{e} and \mathbf{g} were introduced in Propositions 4.3.19, 4.3.16 and 4.3.21 respectively.

Definition 4.3.29. Let k_2 be the minimum of the following values.

- (i) \mathbf{f} . This way, the condition $u \leq \mathbf{B}(q, x)$ is sufficient to apply Proposition 4.3.19.
- (ii) $\frac{\mathbf{e}}{(1+\varepsilon)^4}$. This way, the condition $u \leq \mathbf{B}(q, x) \cdot (1 + \varepsilon)^4$ is sufficient to apply Proposition 4.3.16.
- (iii) \mathbf{g} . This way, $u \leq \mathbf{B}(q, x)$ allows to apply Proposition 4.3.21.

(iv) $\frac{1}{4k_3(1+\varepsilon)^6}$. One can thus apply Corollary 4.3.28.

The first three conditions ensure that the precision will be good enough for the computation of the values of $\Delta\tau$, of the Newton operator and of q_τ . The fourth condition is needed for the computation of $B(q, x)$ itself.

PROOF OF PROPOSITION 4.3.26. From (4.17), the bound $\mathbf{B}(q, x)$ equals $k_2B(q, x)$ and the result now follows from Corollary 4.3.28. \square

4.4 Analysis of the Homotopy

We next describe with more detail our procedure ALHF – Adaptive Linear Homotopy with Finite precision – to follow the path $\{(q_\tau, \zeta_\tau) \mid \tau \in [0, 1]\}$.

All the certifications on an execution of ALHF will be for inputs satisfying certain conditions. We thus define the notion of admissible input for ALHF.

Definition 4.4.1. An *admissible input* for algorithm ALHF consists of

- A procedure `read_input()`, that returns an approximation of a system $f \in S(\mathcal{H}_d)$ with the current round-off unit. That is, the instruction `read_input()` returns a system f' such that the coefficients a'_α of the polynomials f'_i satisfy

$$|a'_\alpha - a_\alpha| \leq u|a_\alpha|,$$

where a_α is the coefficient of the monomial of the same degree α of f_i . In particular, this implies that

$$\|f - f'\| \leq u\|f\|.$$

- An auxiliary system $g \in S(\mathcal{H}_d)$, supposed to be given exactly,
- an approximate zero $x \in S(\mathbb{C}^{n+1})$ of g satisfying

$$d_{\mathbb{P}}(\zeta, x) \leq \frac{C}{D^{3/2}\mu_{\text{norm}}(g, \zeta)}$$

for its associated zero ζ , and

- an initial round-off unit $u \in \mathbb{R}_+$ such that

$$u \leq \mathbf{B}(g, x).$$

For clarity, we denote such a tuple (f, g, x, u) and we refer to it as an input to ALHF even though f is not given directly and the precision u is not passed as a parameter (it is a global variable in MDF).

Define $\lambda := \frac{2C(1-\varepsilon)}{5(1+\varepsilon)^4} \approx 5.37 \cdot 10^{-3}$.

Algorithm ALHF
input (f, g, x, u)
 $\tilde{f} := \text{read_input}()$
if $d_{\mathbb{S}}(\tilde{f}, g) \geq \frac{\pi}{2}$ then $g := -g$
 $\tau := 0, \tilde{q}_{\tau} := g$
repeat
 $\Delta\tau := \frac{\lambda}{d_{\mathbb{S}}(\tilde{f}, g)D^{3/2}\mu_{\text{norm}}^2(\tilde{q}_{\tau}, x)}$
 $\tau := \min\{1, \tau + \Delta\tau\}$
 $\tilde{f} := \text{read_input}()$
 $\tilde{q}_{\tau} := t(\tau)\tilde{f} + (1 - t(\tau))g$
 $\tilde{q}_{\tau} := \frac{\tilde{q}_{\tau}}{\|\tilde{q}_{\tau}\|}$
 $x := N_{\tilde{q}_{\tau}}(x)$
 $x := \frac{x}{\|x\|}$
 $u := \frac{3}{4}\mathbf{B}(\tilde{q}_{\tau}, x)$
until $\tau = 1$
RETURN x

Remark 4.4.2. The algorithm ALHF is a finite-precision adaptation of the algorithm ALH in [12]. It has a slightly smaller stepsize parameter λ . By the parameter f given to ALHF, we mean, that the algorithm is given as input the procedure `read_input` that returns finite precision approximations of f .

We may use ALHF to define a finite precision version MDF of MD.

Algorithm MDF
input $f \in \mathcal{H}_{\mathbf{d}}$
 $u := \frac{k_2}{nD^2(\log N + D + n)2(n+1)^D}$
run ALHF on **input** (f, \bar{U}, z_1)

To a pair $f \in S(\mathcal{H}_{\mathbf{d}})$ and $(g, \zeta) \in V_{\mathbb{P}}$ we associate the number

$$\mu^*(f, g, \zeta) := \max_{\tau \in [0, 1]} \mu_{\text{norm}}(q_{\tau}, \zeta_{\tau}).$$

Theorem 4.4.3. *Let (f, g, x, u) be an admissible input of ALHF. Then:*

- (i) *If the algorithm ALHF stops on input (f, g, x) , it returns an approximate zero of f .*
- (ii) *Assume ALHF stops on input (f, g, x) . Then, the number of iterations $K(f, g, x)$ performed by ALHF satisfies*

$$K(f, g, x) \leq B(f, g, \zeta) + B(-f, g, \zeta)$$

where

$$B(f, g, \zeta) := 408 d_{\mathbb{S}}(f, g)D^{3/2} \int_0^1 \mu_{\text{norm}}^2(q_{\tau}, \zeta_{\tau})d\tau.$$

Consequently the number of performed arithmetic operations $\text{cost}_{\text{ALHF}}(f, g, x)$ is bounded by

$$\text{cost}_{\text{ALHF}}(f, g, x) \leq \mathcal{O}(N)(B(f, g, \zeta) + B(-f, g, \zeta)).$$

If ALHF does not stop then either $B(f, g, \zeta)$ or $B(-f, g, \zeta)$ is unbounded, and either the segment $E_{g, f}$ or $E_{g, -f}$ intersects Σ .

(iii) Furthermore, the finest precision $u_*(f, g, x)$ required during the execution is bounded from below by

$$u_*(f, g, x) = \Omega\left(\frac{1}{nD^2(\log N + D + n)(\mu^*(f, g, \zeta))^2}\right).$$

Remark that Dedieu, Malajovich and Shub [17] proposed an improved version of ALH, implementing an idea from [42], where the step size $\Delta\tau$ is computed exactly as large as possible to ensure that x is a sufficiently good approximate zero of $q_{\tau+\Delta\tau}$. Thanks to this they can replace the $\mu_{\text{norm}}^2(q_\tau, \zeta_\tau)$ factor in bound on the complexity of the homotopy by the better bound $\mu_{\text{norm}}(q_\tau, \zeta_\tau)(\|\dot{q}_\tau\|^2 + \|\dot{\zeta}_\tau\|^2)^{\frac{1}{2}}$. For our finite precision purposes, we stick to the simpler method.

4.4.1 Bounding errors in the homotopy

We begin with a simple consequence of Proposition 4.2.3.

Proposition 4.4.4. *Assume $D \geq 2$. Let $p_0, p_1 \in S(\mathcal{H}_d)$, let ζ be a zero of p_0 , and A a positive constant not greater than C such that*

$$d_{\mathbb{S}}(p_0, p_1) \leq \frac{A}{(1 + \varepsilon)D^{3/2}\mu_{\text{norm}}^2(p_0, \zeta)}.$$

Then the path E_{p_0, p_1} can be lifted to a path in $V_{\mathbb{P}}$ starting in (p_0, ζ) . In addition, the zero χ of p_1 in this lifting satisfies

$$d_{\mathbb{P}}(\zeta, \chi) \leq \frac{A}{D^{3/2}\mu_{\text{norm}}(p_1, \chi)}.$$

Finally, for all $p_\tau \in E_{p_0, p_1}$, if ζ_τ denotes the zero of p_τ in this lifting, we have

$$\frac{1}{1 + \varepsilon}\mu_{\text{norm}}(p_0, \zeta) \leq \mu_{\text{norm}}(p_\tau, \zeta_\tau) \leq (1 + \varepsilon)\mu_{\text{norm}}(p_0, \zeta).$$

PROOF. For each $\tau \in [0, 1]$, let p_τ be the point of the segment $[p_0, p_1]$ such that $d_{\mathbb{S}}(p_0, p_\tau) = \tau d_{\mathbb{S}}(p_0, p_1)$.

Let τ_* be such that $\int_0^{\tau_*} \mu_{\text{norm}}(p_\tau, \zeta_\tau) \|\dot{p}_\tau\| d\tau = \frac{A}{D^{3/2}\mu_{\text{norm}}(p_0, \zeta)}$, or $\tau_* = 1$, or the path E_{p_0, p_1} cannot be lifted to V beyond τ_* , whichever is the smallest. Then, for all $\tau \in [0, \tau_*]$, using that $\|\dot{\zeta}_\tau\| \leq \mu_{\text{norm}}(p_\tau, \zeta_\tau) \|\dot{p}_\tau\|$ (cf. [6, §12.3-12.4]) we have

$$\begin{aligned} d_{\mathbb{P}}(\zeta, \zeta_\tau) &\leq \int_0^\tau \|\dot{\zeta}_s\| ds \leq \int_0^{\tau_*} \mu_{\text{norm}}(p_s, \zeta_s) \|\dot{p}_s\| ds \\ &\leq \frac{A}{D^{3/2}\mu_{\text{norm}}(p_0, \zeta)}. \end{aligned}$$

It is therefore enough to show that $\tau_* = 1$. Suppose to the contrary, that $\tau_* < 1$.

Since $\mu_{\text{norm}}(p_\tau, \zeta_\tau) \geq 1$, for every τ ,

$$d_{\mathbb{S}}(p_0, p_\tau) \leq d_{\mathbb{S}}(p_0, p_1) \leq \frac{A}{D^{3/2} \mu_{\text{norm}}(p_0, \zeta)}.$$

Since $A \leq C$ the bounds on $d_{\mathbb{S}}(p_0, p_\tau)$ and $d_{\mathbb{P}}(\zeta, \zeta_\tau)$ allow us to apply Proposition 4.2.3 and to deduce, for all $\tau \in [0, \tau_*]$,

$$\frac{\mu_{\text{norm}}(p_0, \zeta)}{1 + \varepsilon} \leq \mu_{\text{norm}}(p_\tau, \zeta_\tau) \leq (1 + \varepsilon) \mu_{\text{norm}}(p_0, \zeta). \quad (4.18)$$

We have

$$\begin{aligned} \frac{A}{D^{3/2} \mu_{\text{norm}}(p_0, \zeta)} &= \int_0^{\tau_*} \mu_{\text{norm}}(p_\tau, \zeta_\tau) \|\dot{p}_\tau\| d\tau \quad (\text{by definition of } \tau_*) \\ &\leq (1 + \varepsilon) \mu_{\text{norm}}(p_0, \zeta) \int_0^{\tau_*} \|\dot{p}_\tau\| d\tau \quad (\text{by (4.18)}) \\ &= d_{\mathbb{S}}(p_0, p_{\tau_*}) (1 + \varepsilon) \mu_{\text{norm}}(p_0, \zeta), \end{aligned}$$

and thus

$$d_{\mathbb{S}}(p_0, p_{\tau_*}) \geq \frac{A}{(1 + \varepsilon) D^{3/2} \mu_{\text{norm}}^2(p_0, \zeta)} \geq d_{\mathbb{S}}(p_0, p_1),$$

which leads to a contradiction with $\tau_* < 1$, and finishes the proof. \square

The next proposition puts together many of the results obtained thus far. The general idea for its proof closely follows [12, Theorem 3.1] (which in turn is a constructive version of the main result in [42]) making some room for errors.

Let (f, g, x, u) be an admissible input for algorithm ALHF.

As remarked in Section 4.2.4, the segment $E_{g,f}$ can be lifted to a path given by a continuous function mapping $\tau \mapsto (q_\tau, \zeta_\tau)$.

Let $0 = \bar{\tau}_0 < \bar{\tau}_1 < \bar{\tau}_2 < \dots$, $x = \bar{x}_0, \bar{x}_1, \bar{x}_2, \dots$, and $\bar{u}_0, \bar{u}_1, \bar{u}_2, \dots$, be the sequences of τ -values, points in $S(\mathbb{C}^{n+1})$ and precisions generated by the algorithm ALHF on the admissible input (f, g, x, u) . Let \tilde{f}_i be the approximation of the input f on the i th iteration.

Let $E_{g,f}$ be the path with endpoints g and f . To simplify notation we write q_i instead of $q_{\bar{\tau}_i}$ and ζ_i instead of $\zeta_{\bar{\tau}_i}$. Similarly, we denote by \tilde{q}_i the computed approximation of q_i – that is, $\tilde{q}_i = \mathbf{fl}(t(\tau_i) \tilde{f}_i + (1 - t(\tau_i))g)$ –, by x_{i+1} the exact value of $N_{q_i}(\bar{x}_i)$, and by τ_{i+1} the exact value of $\bar{\tau}_i + \Delta\tau$.

Proposition 4.4.5. *Let (f, g, x, u) be an admissible input for ALHF. Let k be the number of iterations of ALHF on input (f, g, x, u) – that is, either $k = \infty$ or $\tau_k = 1$, $q_k = f$. With the notations above, for all $i \in \{0, \dots, k - 1\}$, the following inequalities are true:*

- (a) $d_{\mathbb{P}}(\bar{x}_i, \zeta_i) \leq \frac{C}{D^{3/2} \mu_{\text{norm}}(q_i, \zeta_i)}$
- (u) $\frac{\mathbf{B}(\tilde{q}_i, \bar{x}_i)}{2} \leq \bar{u}_i \leq \mathbf{B}(\tilde{q}_i, \bar{x}_i)$

- (x) $d_{\mathbb{S}}(q_i, \tilde{q}_i) \leq \frac{C(1-\varepsilon)}{12(1+\varepsilon)D^{3/2}\mu_{\text{norm}}(q_i, \zeta_i)}$
- (c) $d_{\mathbb{S}}(q_i, q_{i+1}) \leq \frac{(1-\varepsilon)C}{2(1+\varepsilon)D^{3/2}\mu_{\text{norm}}(q_i, \zeta_i)}$
- (d) $d_{\mathbb{P}}(\zeta_i, \zeta_{i+1}) \leq \frac{(1-\varepsilon)C}{2(1+\varepsilon)D^{3/2}\mu_{\text{norm}}(q_i, \zeta_i)}$
- (e) \tilde{q}_{i+1} has a zero $\tilde{\zeta}_{i+1}$ such that $d_{\mathbb{P}}(\overline{x}_i, \tilde{\zeta}_{i+1}) \leq \frac{C((1+\varepsilon) + 7/12(1-\varepsilon))}{D^{3/2}\mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})}$

Inequalities **(a)**, **(u)**, and **(x)** hold for k as well.

Proposition 4.4.5 puts together all the needed bounds to ensure the proper work of ALHF. Statement **(a, i)** ensures that \overline{x}_i is “close enough” to ζ_i . That is, \overline{x}_i is not just an approximate zero of q_i , but also an approximate zero for polynomials in a certain neighborhood of q_i on $E_{g,f}$. Statements **(c, i)** and **(d, i)** show that (taking into account computational errors) our step along the homotopy is so small that the next polynomial q_{i+1} belongs to this neighborhood. We hence arrive at **(e, i)**, which essentially means that \overline{x}_i is an approximate zero of q_{i+1} associated with ζ_{i+1} . Therefore, the Newton step (with computational errors accounted for) brings the next iterate \overline{x}_{i+1} close enough to ζ_{i+1} to ensure that **(a, $i+1$)** holds again. Making sure that **(u)** holds on every iteration, we guarantee that computational errors are small enough to allow all the other steps of the proof (**(a)**, **(c)**, **(d)** and **(e)**) to be carried through.

PROOF OF PROPOSITION 4.4.5. We proceed by induction by showing, that **(a, i)**, **(u, i)** and **(x, i)** imply successively **(c, i)**, **(d, i)**, **(x, $i+1$)**, **(e, i)**, and finally **(a, $i+1$)** and **(u, $i+1$)**.

Inequalities **(a)** and **(u)**, for $i = 0$ hold by hypothesis, and **(x, 0)** is obvious since $\tilde{q}_0 = q_0 = g$.

This gives us the induction base. Assume now that **(a)**, **(u)** and **(x)** hold for some $i \leq k-1$.

We now show **(c, i)** and **(d, i)**.

Observe that together with **(a, i)** and **(x, i)**, Proposition 4.2.3 implies

$$\frac{\mu_{\text{norm}}(\tilde{q}_i, \overline{x}_i)}{(1+\varepsilon)} \leq \mu_{\text{norm}}(q_i, \zeta_i) \leq (1+\varepsilon)\mu_{\text{norm}}(\tilde{q}_i, \overline{x}_i). \quad (4.19)$$

By **(u, i)** and Definition 4.3.29 our precision \overline{u}_i satisfies (4.10) for the pair $(\tilde{q}_i, \overline{x}_i)$. Therefore, by Proposition 4.3.19 and the definition of $\Delta\tau$ in ALHF we have

$$\begin{aligned} \alpha(\overline{\tau}_{i+1} - \overline{\tau}_i) &\leq \alpha(\mathbf{Error}(\Delta\tau) + \tau_{i+1} - \overline{\tau}_i) \\ &\leq \alpha \frac{5}{4} \Delta\tau \leq \frac{\lambda(1 + \frac{1}{4})}{D^{3/2}\mu_{\text{norm}}^2(\tilde{q}_i, \overline{x}_i)}. \end{aligned}$$

So, using (4.19) and since $\lambda := \frac{2C(1-\varepsilon)}{5(1+\varepsilon)^4}$, we obtain

$$d_{\mathbb{S}}(q_i, q_{i+1}) = \alpha(\overline{\tau}_{i+1} - \overline{\tau}_i) \leq \frac{C(1-\varepsilon)}{2(1+\varepsilon)^4 D^{3/2} \mu_{\text{norm}}^2(\tilde{q}_i, \overline{x}_i)} \leq \frac{C(1-\varepsilon)}{2(1+\varepsilon)^2 D^{3/2} \mu_{\text{norm}}^2(q_i, \zeta_i)}.$$

Since $\mu_{\text{norm}}(q_i, \zeta_i)$ is always greater than or equal to 1, (\mathbf{c}, i) holds, and (\mathbf{d}, i) is the direct consequence of Proposition 4.4.4 applied to (q_i, q_{i+1}) and ζ_i , with $A = \frac{C(1-\varepsilon)}{2(1+\varepsilon)}$.

This application of Proposition 4.4.4 furthermore ensures that, for all $\tau \in [\bar{\tau}_i, \bar{\tau}_{i+1}]$,

$$\frac{\mu_{\text{norm}}(q_i, \zeta_i)}{1 + \varepsilon} \leq \mu_{\text{norm}}(q_\tau, \zeta_\tau) \leq (1 + \varepsilon)\mu_{\text{norm}}(q_i, \zeta_i), \quad (4.20)$$

and, in particular,

$$\frac{\mu_{\text{norm}}(q_i, \zeta_i)}{1 + \varepsilon} \leq \mu_{\text{norm}}(q_{i+1}, \zeta_{i+1}) \leq (1 + \varepsilon)\mu_{\text{norm}}(q_i, \zeta_i). \quad (4.21)$$

Since $u \leq \mathbf{B}(\tilde{q}_i, \bar{x}_i)$ and from Definition 4.3.29, we can apply Proposition 4.3.21 and we get

$$\begin{aligned} d_{\mathbb{S}}(q_{i+1}, \tilde{q}_{i+1}) &\leq \frac{C(1 - \varepsilon)}{12(1 + \varepsilon)^5 D^{3/2} \mu_{\text{norm}}^2(\tilde{q}_i, \bar{x}_i)} \\ &\leq \frac{\frac{C(1-\varepsilon)}{12(1+\varepsilon)^3}}{D^{3/2} \mu_{\text{norm}}^2(q_i, \zeta_i)} \quad (\text{from (4.19)}) \end{aligned} \quad (4.22)$$

and, hence, using (4.21),

$$d_{\mathbb{S}}(q_{i+1}, \tilde{q}_{i+1}) \leq \frac{\frac{C(1-\varepsilon)}{12(1+\varepsilon)}}{D^{3/2} \mu_{\text{norm}}^2(q_{i+1}, \zeta_{i+1})}. \quad (4.23)$$

Since $\mu_{\text{norm}}(q_{i+1}, \zeta_{i+1}) \geq 1$ this shows $(\mathbf{x}, i + 1)$.

We can now use (\mathbf{x}, i) , (\mathbf{c}, i) , and (4.22) to bound $d_{\mathbb{S}}(\tilde{q}_i, \tilde{q}_{i+1})$ as follows,

$$\begin{aligned} d_{\mathbb{S}}(\tilde{q}_i, \tilde{q}_{i+1}) &\leq d_{\mathbb{S}}(\tilde{q}_i, q_i) + d_{\mathbb{S}}(q_i, q_{i+1}) + d_{\mathbb{S}}(q_{i+1}, \tilde{q}_{i+1}) \\ &\leq \frac{\frac{C(1-\varepsilon)}{12(1+\varepsilon)}}{D^{3/2} \mu_{\text{norm}}(q_i, \zeta_i)} + \frac{\frac{C(1-\varepsilon)}{2(1+\varepsilon)}}{D^{3/2} \mu_{\text{norm}}(q_i, \zeta_i)} + \frac{\frac{C(1-\varepsilon)}{12(1+\varepsilon)^3}}{D^{3/2} \mu_{\text{norm}}^2(q_i, \zeta_i)} \\ &< \frac{\frac{C(1-\varepsilon)}{(1+\varepsilon)}}{D^{3/2} \mu_{\text{norm}}(q_i, \zeta_i)} \\ &\leq \frac{C(1 - \varepsilon)}{D^{3/2} \mu_{\text{norm}}(\tilde{q}_i, \bar{x}_i)} \end{aligned} \quad (4.24)$$

the third inequality using $\mu_{\text{norm}}(q_i, \zeta_i) \geq 1$ and the last from (4.19). We can similarly bound distances between zeros and their approximations. Indeed, using (4.23), Proposition 4.4.4 applied to $(q_{i+1}, \tilde{q}_{i+1})$ and ζ_{i+1} , with $A = \frac{C(1-\varepsilon)}{12}$, ensures the existence of a zero $\tilde{\zeta}_{i+1}$ of \tilde{q}_{i+1} such that

$$d_{\mathbb{P}}(\zeta_{i+1}, \tilde{\zeta}_{i+1}) \leq \frac{C(1 - \varepsilon)}{12D^{3/2} \mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})}. \quad (4.25)$$

Next we use the triangle inequality to obtain

$$\begin{aligned} d_{\mathbb{P}}(\bar{x}_i, \tilde{\zeta}_{i+1}) &\leq d_{\mathbb{P}}(\bar{x}_i, \zeta_i) + d_{\mathbb{P}}(\zeta_i, \zeta_{i+1}) + d_{\mathbb{P}}(\zeta_{i+1}, \tilde{\zeta}_{i+1}) \\ &\leq \frac{C \left(1 + \frac{1-\varepsilon}{2(1+\varepsilon)}\right)}{D^{3/2} \mu_{\text{norm}}(q_i, \zeta_i)} + \frac{C \frac{1-\varepsilon}{12}}{D^{3/2} \mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})} \quad (\text{by } (\mathbf{a}, i), (\mathbf{d}, i) \text{ and (4.25)}) \\ &\leq \frac{C(1 + \varepsilon + \frac{7}{12}(1 - \varepsilon))}{D^{3/2} \mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})}, \quad (\text{by (4.21)}) \end{aligned}$$

which proves (\mathbf{e}, i) .

Note that $(\mathbf{x}, i+1)$ and (4.25), together with Proposition 4.2.3, imply that $\mu_{\text{norm}}(\tilde{q}_{i+1}, \tilde{\zeta}_{i+1}) \leq (1+\varepsilon)\mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})$. Also, that we have $C(1+\varepsilon)(1+\varepsilon + \frac{7}{12}(1-\varepsilon)) \leq \nu_0 \approx 0.3542$ and hence $d_{\mathbb{P}}(\bar{x}_i, \tilde{\zeta}_{i+1}) \leq \frac{\nu_0}{D^{3/2}\mu_{\text{norm}}(\tilde{q}_{i+1}, \tilde{\zeta}_{i+1})}$. We can therefore use Theorem 4.2.2 to deduce that \bar{x}_i is an approximate zero of \tilde{q}_{i+1} associated with its zero $\tilde{\zeta}_{i+1}$. Therefore, $x_{i+1} = N_{\tilde{q}_{i+1}}(\bar{x}_i)$ satisfies

$$d_{\mathbb{P}}(x_{i+1}, \tilde{\zeta}_{i+1}) \leq \frac{1}{2} d_{\mathbb{P}}(\bar{x}_i, \tilde{\zeta}_{i+1}) \leq \frac{C(1+\varepsilon + \frac{7}{12}(1-\varepsilon))}{2D^{3/2}\mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})}, \quad (4.26)$$

where the last inequality is due to (\mathbf{e}, i) , and thus

$$\begin{aligned} d_{\mathbb{P}}(x_{i+1}, \zeta_{i+1}) &\leq d_{\mathbb{P}}(x_{i+1}, \tilde{\zeta}_{i+1}) + d_{\mathbb{P}}(\tilde{\zeta}_{i+1}, \zeta_{i+1}) \\ &\leq \frac{\frac{1}{2}C(1+\varepsilon + \frac{7}{12}(1-\varepsilon)) + \frac{1}{12}C(1-\varepsilon)}{D^{3/2}\mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})} \\ &= \frac{C(\frac{1}{2}(1+\varepsilon) + \frac{3}{8}(1-\varepsilon))}{D^{3/2}\mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})}. \end{aligned} \quad (4.27)$$

Now we are ready to prove the last two implications. We first show $(\mathbf{a}, i+1)$.

Inequality (4.24) allows us to use once more Proposition 4.2.3 to deduce

$$\frac{1}{1+\varepsilon}\mu_{\text{norm}}(\tilde{q}_i, \bar{x}_i) \leq \mu_{\text{norm}}(\tilde{q}_{i+1}, \bar{x}_i) \leq (1+\varepsilon)\mu_{\text{norm}}(\tilde{q}_i, \bar{x}_i). \quad (4.28)$$

Since \bar{u}_i is less than or equal to $\mathbf{B}(\tilde{q}_i, \bar{x}_i)$ (by (\mathbf{u}, i)), from the choice of the constant k_2 in Definition 4.3.29(ii) one has

$$\begin{aligned} \bar{u}_i &\leq \frac{\mathbf{e}}{(1+\varepsilon)^2 n D^2 (\log N + D + n) \mu_{\text{norm}}^2(\tilde{q}_i, \bar{x}_i)} \\ &\leq \frac{\mathbf{e}}{n D^2 (\log N + D + n) \mu_{\text{norm}}^2(\tilde{q}_{i+1}, \bar{x}_i)} \quad (\text{by (4.28)}). \end{aligned}$$

The condition on u (for the pair $(\tilde{q}_{i+1}, \bar{x}_i)$) of Proposition 4.3.16 is thus verified, and applying this proposition we obtain

$$\|\bar{x}_{i+1} - x_{i+1}\| = \mathbf{Error}(N_{\tilde{q}_{i+1}}(\bar{x}_i)) \leq \frac{C(1-\varepsilon)}{4\pi(1+\varepsilon)^2 D^{3/2} \mu_{\text{norm}}(\tilde{q}_{i+1}, \bar{x}_i)}. \quad (4.29)$$

The proof of (4.24) implicitly shows that $d_{\mathbb{S}}(q_i, \tilde{q}_{i+1}) \leq \frac{C(1-\varepsilon)}{D^{3/2}\mu_{\text{norm}}(q_i, \zeta_i)}$. Together with (\mathbf{a}, i) we are in the hypothesis of Proposition 4.2.3 and we can deduce

$$\frac{1}{1+\varepsilon}\mu_{\text{norm}}(q_i, \zeta_i) \leq \mu_{\text{norm}}(\tilde{q}_{i+1}, \bar{x}_i) \leq (1+\varepsilon)\mu_{\text{norm}}(q_i, \zeta_i).$$

This inequality, together with (4.21), yields

$$\frac{1}{(1+\varepsilon)^2}\mu_{\text{norm}}(q_{i+1}, \zeta_{i+1}) \leq \mu_{\text{norm}}(\tilde{q}_{i+1}, \bar{x}_i) \leq (1+\varepsilon)^2\mu_{\text{norm}}(q_{i+1}, \zeta_{i+1}) \quad (4.30)$$

and using these bounds (4.29) becomes

$$\|\overline{x_{i+1}} - x_{i+1}\| \leq \frac{C(1-\varepsilon)}{4\pi D^{3/2} \mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})}. \quad (4.31)$$

We now use this bound and the triangle inequality to bound $d_{\mathbb{P}}(\overline{x_{i+1}}, \zeta_{i+1})$ as follows

$$\begin{aligned} d_{\mathbb{P}}(\overline{x_{i+1}}, \zeta_{i+1}) &\leq d_{\mathbb{P}}(\overline{x_{i+1}}, x_{i+1}) + d_{\mathbb{P}}(x_{i+1}, \zeta_{i+1}) \\ &\leq \frac{\pi}{2} \|\overline{x_{i+1}} - x_{i+1}\| + d_{\mathbb{P}}(x_{i+1}, \zeta_{i+1}) \\ &\leq \frac{C(1-\varepsilon)}{8D^{3/2} \mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})} + \frac{C(1+\varepsilon+3/4(1-\varepsilon))}{2D^{3/2} \mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})} \quad (\text{by (4.31) and (4.27)}) \\ &= \frac{C(\frac{1}{2}(1+\varepsilon) + \frac{3}{8}(1-\varepsilon) + \frac{1}{8}(1-\varepsilon))}{D^{3/2} \mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})} = \frac{C}{D^{3/2} \mu_{\text{norm}}(q_{i+1}, \zeta_{i+1})}, \end{aligned}$$

which proves (a) for $i+1$.

It remains to show (u, $i+1$). To do so note that we may use (a, $i+1$) and (x, $i+1$) together with Proposition 4.2.3 to obtain (4.19) for $i+1$ (just as we obtained it for i). Consequently,

$$\begin{aligned} \mu_{\text{norm}}(\tilde{q}_{i+1}, \overline{x_{i+1}}) &\leq (1+\varepsilon) \mu_{\text{norm}}(q_{i+1}, \zeta_{i+1}) \\ &\leq (1+\varepsilon)^2 \mu_{\text{norm}}(q_i, \zeta_i) \quad (\text{by (4.21)}) \\ &\leq (1+\varepsilon)^3 \mu_{\text{norm}}(\tilde{q}_i, \overline{x}_i) \quad (\text{by (4.19)}). \end{aligned}$$

Using this bound along with (u, i) we obtain

$$\begin{aligned} \bar{u}_i &\leq \mathbf{B}(\tilde{q}_i, \overline{x}_i) = \frac{k_2}{nD^2(\log N + D + n) \mu_{\text{norm}}^2(\tilde{q}_i, \overline{x}_i)} \\ &\leq \frac{k_2(1+\varepsilon)^6}{nD^2(\log N + D + n) \mu_{\text{norm}}^2(\tilde{q}_{i+1}, \overline{x}_{i+1})} = (1+\varepsilon)^6 \mathbf{B}(\tilde{q}_{i+1}, \overline{x}_{i+1}). \end{aligned}$$

We can therefore apply Proposition 4.3.26 with the pair $(\tilde{q}_{i+1}, \overline{x}_{i+1})$ to deduce that $\text{Error}(\frac{3}{4} \mathbf{B}(\tilde{q}_{i+1}, \overline{x}_{i+1})) \leq \frac{1}{4} \mathbf{B}(\tilde{q}_{i+1}, \overline{x}_{i+1})$, and consequently

$$\left| \bar{u}_{i+1} - \frac{3}{4} \mathbf{B}(\tilde{q}_{i+1}, \overline{x}_{i+1}) \right| \leq \frac{1}{4} \mathbf{B}(\tilde{q}_{i+1}, \overline{x}_{i+1}),$$

which proves (u, $i+1$). \square

4.4.2 Proof of Theorem 4.4.3

(i) Since (f, g, x, u) is an admissible input for ALHF we can use Proposition 4.4.5 (and the notation therein). The estimate $d_{\mathbb{P}}(\overline{x}_k, \zeta_k) \leq \frac{C}{D^{3/2} \mu_{\text{norm}}(q_k, \zeta_k)}$ shown as (a, k) in that proposition implies by Theorem 4.2.2 that the returned point \overline{x}_k is an approximate zero of $q_k = f$ with associated zero ζ_1 .

(ii) The first instruction in ALHF swaps f by $-f$ if $d_{\mathbb{S}}(\tilde{f}, g) \geq \frac{\pi}{2}$. The reason to do so is that for nearly antipodal instances of f and g the difference $d_{\mathbb{S}}(f, \tilde{f})$ may be arbitrarily

magnified in $d_{\mathbb{S}}(q_{\tau}, \tilde{q}_{\tau})$. This does not occur under the assumption of infinite precision and this is why such swap is not in the algorithms described in [5, 12].

Let h be either $-f$ or f (according to whether ALHF did the swap or not), $K(h, g, x)$ be the number of iterations performed by ALHF, and $\{(q_{\tau}, \zeta_{\tau})\}$ be the lifting of the path $E_{g,h}$.

Let $k \leq K(h, g, x)$ be a positive integer and consider any $i \in \{0, \dots, k-1\}$. Using Proposition 4.4.4 for q_i, q_{i+1} together with (4.19) implies that, for all $\tau \in [\bar{\tau}_i, \bar{\tau}_{i+1}]$,

$$\frac{\mu_{\text{norm}}(\tilde{q}_i, \bar{x}_i)}{(1+\varepsilon)^2} \leq \mu_{\text{norm}}(q_{\tau}, \zeta_{\tau}) \leq (1+\varepsilon)^2 \mu_{\text{norm}}(\tilde{q}_i, \bar{x}_i). \quad (4.32)$$

Therefore,

$$\begin{aligned} \int_{\bar{\tau}_i}^{\bar{\tau}_{i+1}} \mu_{\text{norm}}^2(q_{\tau}, \zeta_{\tau}) d\tau &\geq \int_{\bar{\tau}_i}^{\bar{\tau}_{i+1}} \frac{\mu_{\text{norm}}^2(\tilde{q}_i, \bar{x}_i)}{(1+\varepsilon)^4} d\tau \\ &= \frac{\mu_{\text{norm}}^2(\tilde{q}_i, \bar{x}_i)}{(1+\varepsilon)^4} (\bar{\tau}_{i+1} - \bar{\tau}_i) \\ &\geq \frac{\mu_{\text{norm}}^2(\tilde{q}_i, \bar{x}_i)}{(1+\varepsilon)^4} \frac{3\lambda}{4\alpha D^{3/2} \mu_{\text{norm}}^2(\tilde{q}_i, \bar{x}_i)} \quad (\text{by Proposition 4.3.19}) \\ &= \frac{3\lambda}{4(1+\varepsilon)^4 \alpha D^{3/2}}. \end{aligned}$$

If $k = K(h, g, x) < \infty$ this implies

$$\int_0^1 \mu_{\text{norm}}^2(q_{\tau}, \zeta_{\tau}) d\tau \geq \left(\frac{3\lambda}{4(1+\varepsilon)^4} \right) k \frac{1}{\alpha D^{3/2}} \geq k \frac{1}{408 \alpha D^{3/2}},$$

which proves that

$$K(h, g, x) \leq 408 d_{\mathbb{S}}(h, g) D^{3/2} \int_0^1 \mu_{\text{norm}}^2(q_{\tau}, \zeta_{\tau}) d\tau = B(h, g, \zeta). \quad (4.33)$$

It follows that the number of iterations $K(f, g, x)$ satisfies either $K(f, g, x) \leq B(f, g, \zeta)$ or $K(f, g, x) \leq B(-f, g, \zeta)$. Certainly – and this introduces a factor of 2 but simplifies the exposition –

$$K(f, g, x) \leq B(f, g, \zeta) + B(-f, g, \zeta).$$

In case $K(h, g, x) = \infty$ (a non-halting computation) it implies that $\int_0^1 \mu_{\text{norm}}^2(q_{\tau}, \zeta_{\tau}) d\tau = \infty$.

The bound for $\text{cost}_{\text{ALHF}}$ follows from the $\mathcal{O}(N)$ cost of each iteration of ALHF mentioned in §4.2.2.

(iii) For $i = 1, \dots, k-1$, due to (u,i),

$$\bar{u}_i \geq \frac{\mathbf{B}(\tilde{q}_i, \bar{x}_i)}{2} = \Omega \left(\frac{1}{nD^2(\log N + D + n) \max_{\tau \in [\bar{\tau}_i, \bar{\tau}_{i+1}]} \mu_{\text{norm}}^2(q_{\tau}, \zeta_{\tau})} \right)$$

the last by (4.32). The statement now follows from the equalities

$$u_*(f, g, \zeta) = \min_{i < k} \bar{u}_i \quad \text{and} \quad \mu^*(f, g, \zeta) = \max_{i < k} \max_{\tau \in [\bar{\tau}_i, \bar{\tau}_{i+1}]} \mu_{\text{norm}}(q_{\tau}, \zeta_{\tau}). \quad \square$$

4.5 Proof of Theorem 4.1.1

We follow here the proof of the corresponding result for MD in [12] and begin by recalling two facts from this article. The first estimates the mean square condition number on the path when an extremity is fixed.

Theorem 4.5.1 (Theorem 10.1 in [12]). *For $g \in S(\mathcal{H}_d) \setminus \Sigma$ we have*

$$\mathbb{E}_{f \in S(\mathcal{H}_d)} \left(d_S(f, g) \int_0^1 \mu_2^2(q_\tau) d\tau \right) \leq 818 D^{3/2} N(n+1) \mu_{\max}^2(g) + 0.01. \quad \square$$

The second bounds the condition of \bar{U} .

Lemma 4.5.2 (Lemma 10.5 in [12]). *The maximum of the condition numbers $\mu_{\max}(\bar{U}) := \max_{z: \bar{U}(z)=0} \{\mu_{\text{norm}}(\bar{U}, z)\}$ satisfies*

$$\mu_{\max}^2(\bar{U}) \leq 2n \max_{i \leq n} \frac{1}{d_i} (n+1)^{d_i-1} \leq 2(n+1)^D. \quad \square$$

The following proposition bounds the maximum $\mu^*(f, g, \zeta)$ of the condition number along a path from (g, ζ) to f in terms of the number of iterations of ALHF to follow this path and of the condition number $\mu_{\text{norm}}(g, \zeta)$ of the initial pair.

Proposition 4.5.3. *Let $f, g \in S(\mathcal{H}_d)$ and ζ a zero of g . The largest condition number $\mu^*(f, g, \zeta)$ along the path from (g, ζ) to f satisfies*

$$\mu^*(f, g, \zeta) \leq (1 + \varepsilon)^{K(f, g, \zeta)} \mu_{\text{norm}}(g, \zeta).$$

PROOF. Write $k := K(f, g, \zeta)$ and let $\mu_i^* := \max_{\tau \in [\bar{\tau}_i, \bar{\tau}_{i+1}]} \mu_{\text{norm}}(q_\tau, \zeta_\tau)$. With this notation, we have $\mu^*(f, g, \zeta) = \max_{i=0, \dots, k-1} \mu_i^*$. Furthermore, (4.20) states that, for all $i \leq k-1$,

$$\mu_i^* \leq (1 + \varepsilon) \mu_{\text{norm}}(q_i, \zeta_i)$$

and an immediate recursion yields

$$\mu^*(f, g, \zeta) = \max_{i \in \{1, \dots, k-1\}} \mu_i^* \leq (1 + \varepsilon)^k \mu_{\text{norm}}(g, \zeta). \quad \square$$

We remark that from the unitary invariance of our setting, for any unitary transformation $\nu \in \mathcal{U}(n+1)$ and any $g \in \mathcal{H}_d$ and $x \in \mathbb{P}^n$,

$$\mu_{\text{norm}}(g, x) = \mu_{\text{norm}}(g \circ \nu^{-1}, \nu x).$$

Furthermore, for any execution of ALHF on an admissible input (f, g, x) , the number of iterations $K(f, g, x)$ during the execution satisfies $K(f, g, x) = K(f \circ \nu^{-1}, g \circ \nu^{-1}, \nu x)$ for any unitary transformation $\nu \in \mathcal{U}(n+1)$.

But one can remark also that any zero z_i of \bar{U} is the image of $z_1 = \frac{1}{\sqrt{2n}}(1, \dots, 1)$ by a unitary transformation ν_i that leaves \bar{U} invariant. Thus, $K(f, \bar{U}, z_1) = K(f \circ \nu_j^{-1}, \bar{U}, z_i)$ for all zeros z_i of \bar{U} , and $\mu_{\max}(\bar{U}) = \mu_{\text{norm}}(\bar{U}, z_1)$.

We also obtain immediately

$$K(f, \bar{U}, \mathbf{z}_1) = \frac{1}{D} \sum_{j=1}^D K(f \circ \nu_j^{-1}, \bar{U}, \mathbf{z}_j). \quad (4.34)$$

But for all measurable functions $\varphi: S(\mathcal{H}_d) \rightarrow \mathbb{R}$ and all $\nu \in \mathcal{U}(n+1)$ we have

$$\mathbb{E}_{f \in S(\mathcal{H}_d)} \varphi(f) = \mathbb{E}_{f \in S(\mathcal{H}_d)} \varphi(f \circ \nu),$$

due to the isotropy of the uniform measure on $S(\mathcal{H}_d)$.

Therefore, (4.34) implies

$$\mathbb{E}_{f \in S(\mathcal{H}_d)} K(f, \bar{U}, \mathbf{z}_1) = \mathbb{E}_{f \in S(\mathcal{H}_d)} \frac{1}{D} \sum_{j=1}^D K(f, \bar{U}, \mathbf{z}_j). \quad (4.35)$$

PROOF OF THEOREM 4.1.1. From Lemma 4.5.2, $\mu_{\text{norm}}(\bar{U}, \mathbf{z}_1) \leq \sqrt{2}(n+1)^{D/2}$, and thus the initial value for u in algorithm MDF is less than or equal to $\mathbf{B}(\bar{U}, \mathbf{z}_1)$. Therefore, the tuple $(f, \bar{U}, \mathbf{z}_1, u)$ given to ALHF during the execution of MD is a admissible input, and we can apply Theorem 4.4.3 to that execution of ALHF. In particular, it follows that MDF stops generically and when it does so, it returns an approximate zero of f .

We next bound the average cost of MDF. Recall, we denoted by $K(f, \bar{U}, \mathbf{z}_1)$ the number of iterations of ALHF during the execution of MDF with input f . Again by Theorem 4.4.3, for any root \mathbf{z}_j of \bar{U} we have

$$K(f, \bar{U}, \mathbf{z}_j) \leq B(f, \bar{U}, \mathbf{z}_j) + B(-f, \bar{U}, \mathbf{z}_j).$$

But we obviously have that $\mathbb{E}_{f \in S(\mathcal{H}_d)} B(f, \bar{U}, \mathbf{z}_j) = \mathbb{E}_{f \in S(\mathcal{H}_d)} B(-f, \bar{U}, \mathbf{z}_j)$, and thus from (4.35),

$$\begin{aligned} \mathbb{E}_{f \in S(\mathcal{H}_d)} K(f, \bar{U}, \mathbf{z}_1) &\leq 2 \mathbb{E}_{f \in S(\mathcal{H}_d)} \frac{1}{D} \sum_{j=1}^D B(f, \bar{U}, \mathbf{z}_j) \\ &= 816D^{3/2} \mathbb{E}_{f \in S(\mathcal{H}_d)} d_{\mathbb{S}}(f, \bar{U}) \int_0^1 \frac{1}{D} \sum_{j=1}^D \mu_{\text{norm}}^2(q_\tau, \zeta_\tau^{(j)}) d\tau \\ &= 816D^{3/2} \mathbb{E}_{f \in S(\mathcal{H}_d)} d_{\mathbb{S}}(f, \bar{U}) \int_0^1 \mu_2^2(q_\tau) d\tau, \end{aligned}$$

the last line by the definition of the mean square condition number (4.2).

Applying successively Theorem 4.5.1 and Lemma 4.5.2, we get

$$\begin{aligned} \mathbb{E}_{f \in S(\mathcal{H}_d)} K(f, \bar{U}, \mathbf{z}_1) &\leq 816D^{3/2}(818D^{3/2}N(n+1)\mu_{\max}^2(\bar{U}) + 0.01) \\ &\leq 816D^{3/2}(818D^{3/2}N(n+1) \cdot 2(n+1)^D + 0.01) \\ &= 667488D^3N(n+1)^{D+1} + 8.16D^{3/2} \\ &\leq 667489D^3N(n+1)^{D+1}. \end{aligned} \quad (4.36)$$

The bound for the average of cost_{MDF} follows from the $\mathcal{O}(N)$ cost of each iteration of ALHF.

We finally bound the average of the precision needed. From Theorem 4.4.3 (iii), the finest precision $u_*(f, \bar{U}, \mathbf{z}_1)$ along the execution of ALHF (and therefore, along that of MDF) satisfies, for some universal constant c ,

$$u_*(f, \bar{U}, \mathbf{z}_1) \geq \frac{1}{cnD^2(\log N + D + n)(\mu^*(f, \bar{U}, \mathbf{z}_1))^2}$$

and, hence,

$$\begin{aligned} |\log u_*(f, \bar{U}, \mathbf{z}_1)| &\leq \log \left(cnD^2(\log N + D + n) (\mu^*(f, \bar{U}, \mathbf{z}_1))^2 \right) \\ &= 2 \log \mu^*(f, \bar{U}, \mathbf{z}_1) + \log(cnD^2(\log N + D + n)) \\ &\leq 2K(f, \bar{U}, \mathbf{z}_1) \log(1 + \varepsilon) + 2 \log \mu_{\text{norm}}(\bar{U}, \mathbf{z}_1) + \log(cnD^2(\log N + D + n)). \end{aligned}$$

Using Lemma 4.5.2 and (4.36), we finally obtain

$$\begin{aligned} \mathbb{E}_{f \in S(\mathcal{H}_d)} |\log u_*(f, \bar{U}, \mathbf{z}_1)| &\leq 2 \mathbb{E}_{f \in S(\mathcal{H}_d)} \left(K(f, \bar{U}, \mathbf{z}_1) \log(1 + \varepsilon) + \log \mu_{\text{norm}}(\bar{U}, \mathbf{z}_1) \right. \\ &\quad \left. + \log(cnD^2(\log N + D + n)) \right) \\ &\leq \log(1 + \varepsilon) \cdot 1334978D^3N(n+1)^{D+1} \\ &\quad + D \log(\sqrt{2}(n+1)) + 2 \log(cnD^2(\log N + D + n)) \\ &= \mathcal{O}(D^3N(n+1)^{D+1}). \end{aligned}$$

We observe that the initial precision $u = \frac{k_2}{nD^2(\log N + D + n)2(n+1)^D}$ also satisfies this inequality. \square

4.6 How to compute the initial precision in ALHF

In algorithm MDF, the computation of the initial precision needed to ensure the correctness of ALHF is easy, since from Lemma 4.5.2 we know a bound on $\mathbf{B}(\bar{U}, \mathbf{z}_1)$ which is easy to compute. But for an initial pair (g, x) for which nothing is known, we cannot just set $u := \mathbf{B}(g, x)$ since the relative error on the computation of $\mathbf{B}(g, x)$ can be arbitrarily large – indeed, Proposition 4.3.26 is only valid when the precision u is already well fitted for the considered system.

We here propose an algorithm SET_PRECISION that computes a precision u satisfying the requirements of ALHF, with no requirements on the initial precision. Its consistency is showed in Proposition 4.6.1 below. Recall, that k_2 and k_3 are the universal constants appearing in the definition of \mathbf{B} (Section 4.3.7).

Algorithm SET_PRECISION

input $g \in S(\mathcal{H}_a)$, $\zeta \in S(\mathbb{C}^{n+1})$

$u := \frac{1}{2n(\log S + D + n)}$

while $u \geq \frac{D^2 \mathbf{B}(g, \zeta)}{4k_2 k_3}$ **do**

$u := u/2$

$u := 3/4 \mathbf{B}(g, \zeta)$

Proposition 4.6.1. *The while loop in algorithm SET_PRECISION takes at most $\log(8k_3\mu_{\text{norm}}^2(g, \zeta))$ iterations. The value of $\mathbf{fl}(u)$ at the end of the loop ensures that*

$$\frac{\mathbf{B}(g, \zeta)}{2} \leq \frac{3\mathbf{fl}(\mathbf{B}(g, \zeta))}{4} \leq \mathbf{B}(g, \zeta).$$

PROOF OF PROPOSITION 4.6.1. Since u is initialized with the value $\frac{1}{2n(\log N + D + n)}$, the conditions of Lemma 4.3.27 are fulfilled, and thus the error in the computation of $\mathbf{B}(g, \zeta)$ is less or equal to $k_2 k_3 u / D^2$.

Thus, the computed value $\mathbf{fl}(\mathbf{B}(g, \zeta))$ is in the interval

$$[\mathbf{B}(g, \zeta) - k_2 k_3 u / D^2, \mathbf{B}(g, \zeta) + k_2 k_3 u / D^2].$$

At the end of the **while** loop, we have: $\frac{k_2 k_3 u}{D^2} \leq \mathbf{fl}(\mathbf{B}(g, \zeta)) / 4$, and thus, $\mathbf{fl}(\mathbf{B}(g, \zeta))$ is in the interval $[\mathbf{B}(g, \zeta) - 1/4 \mathbf{fl}(\mathbf{B}(g, \zeta)), \mathbf{B}(g, \zeta) + 1/4 \mathbf{fl}(\mathbf{B}(g, \zeta))]$. By subtracting $\mathbf{fl}(\mathbf{B}(g, \zeta)) / 4$, we deduce that $3/4 \mathbf{fl}(\mathbf{B}(g, \zeta)) \leq \mathbf{B}(g, \zeta)$, and by dividing by $3/4$, that

$$\begin{aligned} 3/4 \mathbf{fl}(\mathbf{B}(g, \zeta)) &\geq 3/4 \mathbf{B}(g, \zeta) - 3/16 \mathbf{fl}(\mathbf{B}(g, \zeta)) \\ &\geq 3/4 \mathbf{B}(g, \zeta) - 1/4 \mathbf{B}(g, \zeta) = 1/2 \mathbf{B}(g, \zeta). \end{aligned}$$

Finally, at the end of the loop, we have $3/4 \mathbf{fl}(\mathbf{B}(g, \zeta)) \in [1/2 \mathbf{B}(g, \zeta), \mathbf{B}(g, \zeta)]$.

Furthermore, at the end of the **while** loop, u satisfies

$$\frac{k_2 k_3 u}{D^2} \geq \mathbf{B}(g, \zeta) / 16. \quad (4.37)$$

Indeed, otherwise, one would have $k_2 k_3 (2u) / D^2 \leq \mathbf{B}(g, \zeta) / 8$, and hence at the previous iteration of the loop, the value $\mathbf{fl}(\mathbf{B}(g, \zeta))$ computed there with precision $2u$ would have verified that $\mathbf{Error}(\mathbf{B}(g, \zeta)) \leq \mathbf{B}(g, \zeta) / 8$. Consequently, we would have

$$\frac{k_2 k_3 (2u)}{D^2} \leq \mathbf{fl}(\mathbf{B}(g, \zeta)) / 8 + \mathbf{Error}(\mathbf{B}(g, \zeta)) / 8 \leq \mathbf{fl}(\mathbf{B}(g, \zeta)) / 4,$$

and finally the **while** loop would have ended one iteration before.

Since (4.37) holds at the end of the loop, and since u is initialized with $\frac{1}{2n(\log N + D + n)}$, the value u decreases from a multiplicative factor at most $\frac{\mathbf{B}(g, \zeta) n (\log N + D + n) D^2}{8k_2 k_3}$ during the **while** loop. Since at each iteration the value of u is divided by two, the number of iterations is at most

$$\log_2 \left(\frac{8k_2 k_3}{n(\log N + D + n) D^2 \mathbf{B}(g, \zeta)} \right) = \log(8k_3 \mu_{\text{norm}}^2(g, \zeta)). \quad \square$$

Chapter 5

Conclusion and perspectives

The first part of this thesis dealt with the polynomial associated to a boolean formula as a link between counting problems and polynomial evaluation problems.

In the second chapter, we studied the problem of computing the polynomials associated to boolean formulas, depending on the basic constraints used to express the formulas. This study followed a framework initiated by Schaefer [40] for the satisfiability problem and by Creignou and Hermann [13] for the problem of counting the satisfying assignments of a boolean formula. We showed that similarly as in the two previous settings, the evaluation of the associated polynomials presented a dichotomy (Theorem 2.2.2). An interesting fact in this result is that large affine constraints belong to the hard cases, whereas they belonged to the easy cases in the decision and counting dichotomies. Many boolean constraint satisfaction problems presenting similar dichotomies have been studied, and the frontier between hard and easy cases presents large variations in these various results. A vague but interesting question would be to understand the link between the problems and the frontier between hard and easy cases: is it possible to characterize the problems that lead to a certain frontier?

This result is based on several new proofs of VNP-completeness. It also enlightens the link between counting reductions and p -projections, as remarked in Section 2.7: p -projections between polynomials can be seen as 1-Turing reductions between the “weighted counting” problems, and can lead to similar reductions between the counting problems. This result is an example of how the parallel study of questions in a boolean and algebraic context can benefit to both theories. In particular, it seems promising to study reduction issues simultaneously on the decision, counting and weighted counting problems, since results in one setting have repercussions in the other settings.

The third chapter was devoted to a lower bound result on the permanent. We presented a new proof that the permanent family (PER_n) cannot be expressed as polynomials associated to formulas of bounded tree-width (Theorem 3.3.3). This result was first proved by Koiran and Meer [27], and was motivated by Theorem 3.1.2, which they also proved. This theorem states that families of polynomials associated to polynomial size formulas of bounded tree-width are not only computable by polynomial size arithmetic formulas, but characterize all such p -families via p -projections. Recently, Stefan Mengel [36] showed similar characterizations of the classes VP_{ws} , VP and VNP , by considering polynomials

associated to non boolean CSPs of bounded path-width, of bounded tree-width and without restrictions respectively, and still by using p -projections. It would be interesting to see how far it is possible to express the permanent as a polynomial associated to a CSP according to Mengel's settings, and in particular if one can find a lower bound on the tree-width of such CSP.

The second part of this thesis focused on the precision analysis of algebraic algorithms. We presented finite precision versions of the algorithms ALH and MD from [12], for the search of approximate zeros of square systems (Theorems 4.1.1 and 4.4.3). The arithmetic cost of the finite precision algorithms is the same as the cost of their algorithmic counterparts up to a constant factor, and the number of bits required during the computation is of the same order as the arithmetic cost. A distinctive feature of these finite precision algorithms is that they set the required precision for each computational step by themselves; the correctness of the computations is thus ensured.

A natural extension to this work would be to study, also in finite precision, Beltrán and Pardo's randomized algorithm, which randomly guesses a starting pair and runs ALH with this entry. The study of the procedure used to pick the starting system and its zero still remains; then, our finite precision analysis of ALHF could be used, with the slight difference with our setting that the initial pair (g, ζ) would not be computed exactly any more. This does not change drastically the analysis since algorithm ALHF is still correct with an approximation of ζ satisfying equation $(\mathbf{a}, \mathbf{0})$ from Proposition 4.4.5, and since introducing an error in g would increase the error in the computation of q_τ , which is already taken into account.

Bibliography

- [1] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22(3):317–330, 1983.
- [2] C. Beltrán and A. Leykin. Robust certified numerical homotopy tracking. To appear, arXiv:1105.5992, 2011.
- [3] C. Beltrán and L. M. Pardo. On Smale’s 17th problem: a probabilistic positive solution. *Foundations of Computational Mathematics*, 8(1):1–43, 2008.
- [4] C. Beltrán and L. M. Pardo. Smale’s 17th problem: average polynomial time to compute affine and projective solutions. *Journal of American Mathematics Society*, 22(2):363–385, 2009.
- [5] C. Beltrán and L. M. Pardo. Fast linear homotopy to find approximate zeros of polynomial systems. *Foundations of Computational Mathematics*, 11(1):95–129, 2011.
- [6] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.
- [7] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers. *Bulletin of the American Math Society*, 21:1–46, 1989.
- [8] I. Briquel, P. Koiran, and K. Meer. On the expressive power of CNF formulas of bounded tree- and clique- width. *Discrete Applied Mathematics*, 159:1–14, 2011.
- [9] A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.
- [10] P. Bürgisser. On the structure of Valiant’s complexity classes. *Discrete Mathematics and Theoretical Computer Science*, 3:73–94, 1999.
- [11] P. Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*. Number 7 in Algorithms and Computation in Mathematics. Springer, 2000.
- [12] P. Bürgisser and F. Cucker. On a problem posed by Steve Smale. *Annals of Mathematics*. To appear.
- [13] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125:1–12, 1996.

- [14] N. Creignou, S. Khanna, and M. Sudan. *Complexity classification of boolean constraint satisfaction problems*. SIAM monographs on discrete mathematics. 2001.
- [15] F. Cucker, T. Krick, G. Malajovich, and M. Wschebor. A numerical algorithm for zero counting, I: Complexity and accuracy. *Journal of Complexity*, 24:582–605, 2008.
- [16] F. Cucker and S. Smale. Complexity estimates depending on condition and round-off error. *Journal of the American Mathematics Society*, 46:113–184, 1999.
- [17] J.-P. Dedieu, G. Malajovich, and Mike Schub. Adaptive step size selection for homotopy methods to solve polynomial equations. To appear, arxiv: 1104-2084, 2011.
- [18] R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg, 2005.
- [19] F. M. Dong, M. D. Hendy, K. L. Teo, and C. H. C. Little. The vertex-cover polynomial of a graph. *Discrete Mathematics*, 250(1-3):71–78, 2002.
- [20] I. Dumer, D. Micciancio, and M. Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Trans. Inform. Theory*, 49:22–37, 2003.
- [21] A. Durand, M. Hermann, and P. G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theoretical Computer Science*, 340(3):496–513, 2005.
- [22] E. Fisher, J. Makowsky, and E. V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 154(3):511–529.
- [23] G. Golub and C. Van Loan. *Matrix Computations*. John Hopkins University Press, 3rd edition, 1996.
- [24] N. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 96.
- [25] M. Jerrum. Two-dimensional monomer-dimer systems are computationally intractable. *Journal of Statistical Physics*, 48:121–134, 1987.
- [26] M. Jerrum and M. Snir. Some exact complexity results for straight-line computations over semirings. *Journal of the ACM*, 29(3):874–897, 1982.
- [27] P. Koiran and K. Meer. On the expressive power of CNF formulas of bounded tree- and clique- width. In *Proceedings of WG'08 (34th International Workshop on Graph-Theoretic Concepts in Computer Science)*, LNCS 5344. Springer, 2008.
- [28] E. Korach and N. Solel. Tree-width, path-width, and cutwidth. *Discrete Applied Mathematics*, 43(1):97–101, 1993.
- [29] M. Krause, C. Meinel, and S. Waack. Separating the eraser Turing machine classes L_e , NL_e , $co-NL_e$ and P_e . *Theoretical Computer Science*, 86:267–275, 1991.
- [30] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.
- [31] R. E. Ladner. On the structure of polynomial time reductibility. *Journal of the ACM*, 22(1):155–171, 1975.

- [32] N. Linial. Hard enumeration problems in geometry and combinatorics. *SIAM Journal of Algebraic and Discrete Methods*, 7(2):331–335, 1986.
- [33] M. Lotz and J. A. Makowsky. On the algebraic complexity of some families of coloured Tutte polynomials. *Advances in Applied Mathematics*, 32(1):327–349, January 2004.
- [34] G. Malod. Computing the partial permanent in characteristic 2. Unpublished, 2011.
- [35] Guillaume Malod. *Polynômes et coefficients*. PhD thesis, Université Claude Bernard - Lyon I, 2003.
- [36] S. Mengel. Characterizing arithmetic circuit classes by Constraint Satisfaction Problems. In *Proc. ICALP 2011*, pages 700–711, 2011. Extended Abstract.
- [37] Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston.
- [38] B. Poizat. *Les petits cailloux*. Aléas, 1995.
- [39] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. of Comp.*, 12(4):777–788, 1983.
- [40] T. J. Schaefer. The complexity of satisfiability problems. In *Conference Record of the 10th Symposium on Theory of Computing*, pages 216–226, 1978.
- [41] M. Shub. Some remarks on Bezout’s theorem and complexity theory. In New York Springer, editor, *From Topology to Computation: Proceedings of the Smalefest (Berkeley, CA, 1990)*, pages 443–455, 1993.
- [42] M. Shub. Complexity of Bézout’s theorem VI: Geodesics in the condition (number) metric. *Foundations of Computational Mathematics*, 9(2):171–178, 2009.
- [43] M. Shub and S. Smale. Complexity of Bézout’s theorem I: Geometric aspects. *Journal of the American Mathematics Society*, 6(2):459–501, 1993.
- [44] M. Shub and S. Smale. Complexity of Bézout’s theorem II: volumes and probabilities. In F. Eyssette and A. Galligo, editors, *Computational Algebraic Geometry*, volume 109, pages 265–285. Birkhäuser, 1993.
- [45] M. Shub and S. Smale. Complexity of Bézout’s theorem III: condition number and packing. *Journal of Complexity*, 9:4–14, 1993.
- [46] M. Shub and S. Smale. Complexity of Bézout’s theorem V: polynomial time. *Theoretical Computer Science*, 133:141–164, 1994.
- [47] M. Shub and S. Smale. Complexity of Bézout’s theorem IV: probability of success; extensions. *SIAM Journal of Numerical Analysis*, 33:128–148, 1996.
- [48] S. Smale. Newton’s method estimates from data at one point. In K. Gross R. Ewing and C. Martin, editors, *The Merging of Disciplines: New Directions in Pure, Applied, and Computational Mathematics*, pages 265–285. Springer-Verlag, 1986.

- [49] S. Smale. Mathematical problems for the next century. In *Mathematics: frontiers and perspectives*, pages 271–294. Amer. Math. Soc., Providence, RI, 2000.
- [50] S. Toda. *Computational complexity of counting complexity classes*. PhD thesis, Tokyo institute of Technology, 1991.
- [51] S. Toda and O. Watanabe. Polynomial-time 1-Turing reductions from PH to #P. *Mathematics of Computation*, 100:205–221, 1992.
- [52] L. G. Valiant. Completeness classes in algebra. In *Proc. 11th ACM Symposium on Theory of Computing*, pages 249–261, 1979.
- [53] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
- [54] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8(3):410–421, 1979.
- [55] L. G. Valiant. Quantum circuits that can be simulated classically in polynomial time. *SIAM Journal of Computing*, 5(31):1229–1254, 2002.
- [56] A. Vardy. Algorithmic complexity in coding theory and the minimum distance problem. In *Proceedings of the twenty-ninth annual ACM Symposium on Theory of computing*, pages 92–109, 1997.
- [57] I. Wegener. *Branching Programs and Binary Decision Diagrams : Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications. 2000.
- [58] A. Yao. Some complexity questions related to distributive computing. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–213, 1979.
- [59] V. Zankó. #P-completeness via many-one reductions. *International Journal of Foundations of Computer Science*, 2(1):77–82, 1991.