UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

GABRIEL MATTOS LANGELOH

Unrestricted dynamic Gröbner Basis algorithms

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Marcus Rolf Peter Ritt

Porto Alegre
March 2019

CIP — CATALOGING-IN-PUBLICATION

Langeloh, Gabriel Mattos

Unrestricted dynamic Gröbner Basis algorithms / Gabriel Mattos Langeloh. – Porto Alegre: PPGC da UFRGS, 2019.

93 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2019. Advisor: Marcus Rolf Peter Ritt.

1. Gröbner Bases. 2. Dynamic Algorithm. 3. Monomial Ordering. I. Ritt, Marcus Rolf Peter. II. Título.

AGRADECIMENTOS

Primeiramente, agradeço aos meus pais, por todo o apoio durante o mestrado. Agradeço também a meu orientador e aos colegas do laboratório de pesquisa por inúmeras discussões muito produtivas.

O presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil.

ABSTRACT

Gröbner bases are a necessary tool to solve many problems involving polynomial ideals, including applications such as nonlinear polynomial system solving, integer programming and cryptography. Traditional Gröbner Basis algorithms are static, in the sense that they receive a monomial order as input and it is fixed during the entire execution of the algorithm. Dynamic algorithms, in contrast, allow this monomial ordering to change to generate smaller output bases and, hopefully, fewer polynomial reductions.

All but one of the previously proposed dynamic algorithms are restricted, meaning that once they choose a leading monomial for a certain polynomial, that choice cannot be unmade. In this work, we focus on exploring unrestricted dynamic algorithms, studying the relation of monomial orderings to Newton polyhedra and proposing four new unrestricted algorithms that avoid evaluating too many monomial orderings by using a neighborhood construction for monomial orders. We also propose a new heuristic, called the Mixed heuristic, for monomial order evaluation in dynamic algorithms.

Our experiments show that although the restricted algorithms perform better with respect to running time, our unrestricted algorithms find orders that lead to smaller Gröbner Bases for many instances and significantly lower degree polynomials in average. Additionally, we provide a comparison between the previously defined Hilbert and Betti heuristics and our Mixed heuristic, showing it performs better than the Betti heuristic in most aspects and is competitive with the Hilbert heuristic overall.

Keywords: Gröbner Bases. Dynamic Algorithm. Monomial Ordering.

Algoritmos dinâmicos irrestritos para cálculo de Bases de Gröbner

RESUMO

Bases de Gröbner são uma ferramenta necessária para resolver diversos problemas envolvendo ideais polinomiais, incluindo aplicações como resolução de sistemas polinomiais não-lineares, programação inteira e criptografia. Algoritmos tradicionais de cálculo de Bases de Gröbner são estáticos, no sentido que eles recebem uma ordem monomial como entrada e essa ordem é então mantida fixa durante toda a execução do algoritmo. Algoritmos dinâmicos, pelo contrário, permitem que a ordem monomial mude para gerar bases menores e, espera-se, realizar menos reduções polinomiais.

Com apenas uma exceção, todos os algoritmos dinâmicos previamente propostos são restritos, o que significa que uma vez que eles escolhem um monômio líder para um certo polinômio, essa escolha não pode ser desfeita. No presente trabalho, exploramos algoritmos dinâmicos irrestritos, estudando a relação entre ordens monomiais e poliedros de Newton e propondo quatro novos algoritmos irrestritos que evitam avaliar muitas ordens usando um conceito de vizinhança para ordens monomiais. Também propomos uma nova heurística, chamada de heurística Mista, para a avaliação de ordens monomiais em algoritmos dinâmicos.

Nossos experimentos mostram que apesar de os algoritmos restritos terem melhor desempenho em termos de tempo de execução, nossos algoritmos irrestritos encontram ordens que levam a Bases de Gröbner menores para muitas instâncias e significativamente reduzem o grau máximo dos polinômios na base em média. Adicionalmente, fornecemos uma comparação entre as heurísticas de Hilbert e Betti, previamente propostas, e nossa heurística Mista, mostrando que ela tem desempenho melhor que a heurística de Betti na maioria dos aspectos e é competitiva com a heurística de Hilbert em geral.

Palavras-chave: Bases de Gröbner. Algoritmo Dinâmico. Ordem Monomial.

LIST OF SYMBOLS

k	A field
R	The polynomial ring $k[x_1, \ldots, x_n]$
n	The number of variables of R
$\mathcal{F}$	A finite set of polynomials $\{f_1, \ldots, f_m\} \subset R$
m	The cardinality of $\mathcal{F}$
I	The ideal $\langle \mathcal{F} \rangle$ of R generated by $\mathcal{F}$
$LM(f)$	The leading (monic) monomial of $f \in R$ with respect to some fixed monomial order
$LC(f)$	The leading coefficient of $f \in R$ with respect to some fixed monomial order
$LT(f)$	The leading term (monomial and coefficient) of $f \in R$ with respect to some fixed monomial order
$\mathrm{Supp}(f)$	The set of monomials appearing in $f \in R$
$f^{\mathcal{F}}$	The remainder of the division of f by $\mathcal{F}$
$[l]$	The set $\{1, \ldots, l\}$

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Let $f_1, \dots, f_m$ be polynomials in n variables and let $<$ be a monomial ordering, that is, a total ordering in the set of monomials in n variables compatible with monomial multiplication. It is possible to compute a set $G = \{g_1, \dots, g_t\}$ of algebraic combinations of the f_i such that, for any algebraic combination

$$f = \sum_{i=1}^{m} a_i f_i$$

the leading monomial $LM(f)$ of f, that is, its largest monomial with respect to $<$, is divisible by one of the $LM(g_i)$. This is what we call a *Gröbner Basis* of the polynomial ideal $I = \langle f_1, \dots, f_m \rangle$.

In polynomial ideal theory, Gröbner Bases are a key object allowing for the computation of ideal membership, invariants, ideal operations and more — virtually all computations involving ideals either depend on or are made easier by the computation of a Gröbner Basis. Applications outside of pure mathematics also abound, mostly because Gröbner Bases are one of the main tools in multivariate polynomial system solving. Examples include integer programming (THOMAS, 1998), the design of cryptographic schemes (PATARIN, 1996) and algebraic cryptanalysis (PETIT; KOSTERS; MESSENG, 2016), (FAUGÈRE; JOUX, 2003).

In our description of Gröbner Bases above, note the dependence on the choice of monomial order — different orders lead to distinct bases, each with its own properties. Some of these are necessary in certain applications, such as elimination orders for polynomial system solving. Many applications, however, depend on no such properties. It becomes desirable, then, to obtain smaller Gröbner Bases, or Gröbner Bases whose polynomials are of low degree. Even in cases where a specific monomial ordering is needed, it is sometimes advantageous to compute the Gröbner Basis in a more efficient ordering and then use an algorithm to change orders, such as FGLM (FAUGÈRE et al., 1993) or the Gröbner Walk (COLLART; KALKBRENER; MALL, 1997).

Traditional Gröbner Basis algorithms, such as Buchberger's algorithm (BUCHBERGER, 2006), F4 (FAUGÈRE, 1999) and F5 (FAUGÈRE, 2002) are highly customizable, as one can choose, for example, how to compute polynomial reductions and how to select polynomials to be reduced. Usually, these algorithms take a monomial ordering as part of the input and compute a Gröbner Basis with respect to it — for this reason, we call these algorithms *static*. Bayer and Stillman (BAYER; STILLMAN, 1987) have shown that the $grevlex$ monomial ordering is "optimal" in a certain sense, which, in the literature, is often taken to mean that the $grevlex$ order leads to faster Gröbner Basis computations than other orders. This is not true in general, and, specially, if one desires a *small* Gröbner Basis, as is the case in many applications, it is possible to choose better orderings in a case by case basis. This, however, is not easy to do *a priori*, so it would make sense to develop algorithms to do this task during the computation of the Gröbner Basis. This is exactly the idea behind the *dynamic* Gröbner Basis algorithms, proposed by (GRITZMANN; STURMFELS, 1993) and (CABOARA, 1993) — they are variants of traditional Gröbner Basis algorithms that allow for the monomial order to change during the computation, often leading to smaller final Gröbner Bases and, sometimes, to better performance.

To exemplify, there are instances where a static algorithm with the $grevlex$ ordering as input returns a Gröbner Basis with 443 polynomials and 79897 monomials of degree no more than 19, while there exists a dynamic algorithm returning a basis with 117 polynomials composed of 25763 monomials of degree at most 13 — in about a third of the time, in our experiments. While this example is somewhat extreme, it shows that there are cases where dynamic algorithms are very advantageous with respect to static ones.

Dynamic Gröbner Basis algorithms were also studied by (CABOARA; PERRY, 2014), (HASHEMI; TALAASHRAFI, 2016) and (PERRY, 2017), who proposed and implemented various modifications for these algorithms. In addition, (GOLUBITSKY, 2006) proposed a classification of dynamic algorithms in two classes — restricted and unrestricted, with all previous algorithms but that of (GRITZMANN; STURMFELS, 1993) being the former. According to this classification, restricted algorithms cannot change leading monomials of polynomials that have already been inserted in a partial Gröbner Basis during the execution, while unrestricted algorithms can. In practice, the search space for monomial orders in restricted algorithms narrows at each iteration, while in unrestricted ones, the search space only becomes larger.

The main reason unrestricted dynamic algorithms are still unexplored is that the amount of candidate monomial orders grows very fast in the original unrestricted algorithm (GRITZMANN; STURMFELS, 1993), which makes it unviable for instances of even moderate sizes. This algorithm evaluates *every* candidate monomial order in order to choose the best one at each iteration, *but this is not necessary* — one can also navigate the space of all candidate monomial orders without visiting all of them, as long as one drops the requirement of choosing a global optimal order. This is not a problem, however, as orders would be evaluated heuristically either way, and this process would be done at each iteration. Our main contributions are various algorithms to explore this space of monomial orders, leading to unrestricted dynamic algorithms that do not depend on visiting all orders at any given step and, thus, are applicable to a wider variety of instances if compared to the original unrestricted algorithm.

In Chapter 2, we provide an overview of Gröbner Bases and polynomial system solving, while in Chapter 3 we describe previous Dynamic Gröbner Basis algorithms and the heuristics used in them to evaluate monomial orders. We also propose a new heuristic, intended to combine the strengths of the previous ones. In Chapter 4 we develop a theory for neighborhoods of monomial orders and apply it to introduce four new unrestricted dynamic algorithms. Chapter 5 presents experimental results comparing most previous dynamic algorithms to our new algorithms, using various heuristic functions to evaluate the monomial orders.

2 POLYNOMIAL SYSTEM SOLVING AND GRÖBNER BASES

Given a field k and a multivariate polynomial ring $R = k[x_1, \dots, x_n]$ over k, we consider the problem of solving a system $\mathcal{F} = \{f_1, \dots, f_m\} \subset R$, where *solving* means finding all points $(a_1, \dots, a_n) \in k^n$ (the *solutions* of $\mathcal{F}$) such that

$$f_1(a_1, \dots, a_n) = f_2(a_1, \dots, a_m) = \dots = f_m(a_1, \dots, a_n) = 0,$$

in case the set of solutions is finite, or somehow describing the infinite solution set (for example, parametrically). We denote $I = \langle \mathcal{F} \rangle$ the ideal generated by $\mathcal{F}$ which, informally (and, imprecisely), represents the set of polynomials equivalent to the input $\mathcal{F}$ with respect to the solution set.

This chapter is an introduction to polynomial system solving, describing algorithms, concepts and known results, focusing on the theory of Gröbner Bases as its main tool. Sections 2.1 through 2.4 present the basic theory and algorithms, while the remaining sections focus on complexity and alternative methods.

2.1 Insights from the linear case

In this section, we will outline a generic approach to multivariate polynomial system solving based on insights from the more familiar linear case. Consider the following linear system $\mathcal{F}$ over $R = \mathbb{R}[x, y, z]$:

$$\begin{cases} x + 10y - 3z = 0 \\ 2x + 19y + z = 3 \\ -x + 3y + 20z = -5 \end{cases} \quad \Rightarrow M = \begin{bmatrix} 1 & 10 & -3 & 0 \\ 2 & 19 & 1 & 3 \\ -1 & 3 & 20 & -5 \end{bmatrix}$$

In order to solve this system, we may start by computing the row echelon form of its matrix M, obtaining:

$$M_{red} = \begin{bmatrix} 1 & 10 & -3 & 0 \\ 0 & -1 & 7 & 3 \\ 0 & 0 & 108 & 34 \end{bmatrix}$$

In the process of row reduction of the matrix, each new line corresponds to a polynomial in the ideal generated by $\mathcal{F}$, the original system. Each intermediate matrix

Table 2.1: Steps of linear and non-linear cases of multivariate polynomial system solving.

	Linear case	Non-Linear case
1	Order the variables	Order the monomials
2	Compute row echelon form	Compute a Gröbner Basis
3	Solve univariate linear equation	Find roots of a univariate polynomial
4	Extend the solutions to more variables	Extend the solutions to more variables

appearing in the reduction corresponds to a system $\mathcal{F}'$ that is also a generating set of $I = \langle \mathcal{F} \rangle$. We may call the system $\mathcal{F}'$ corresponding to the matrix M_{red} above a *Gröbner Basis* of I. This concept will generalize to the non-linear case (see Section 2.3) allowing non-linear polynomial system solving.

Note that in the last row of M_{red}, corresponding to the equation $108z = 34$, we have *eliminated* the variables x, y, and in the second row we have eliminated the variable x. Implicitly, we considered a *variable ordering* which we used to decide the order the variables would be eliminated. We will also have to generalize these concepts to solve the non-linear case.

In the linear case example, we would then proceed to solve the univariate linear equation $108z = 34$. This will usually also happen in the non-linear case, except the corresponding univariate polynomial may be of higher degree. Then, if $k = \mathbb{R}$ or $k = \mathbb{C}$ one can use numerical methods to obtain the roots of the polynomial or, in case k is a finite field, it is possible to use a specialized algorithm such as Cantor-Zassenhaus (CANTOR; ZASSENHAUS, 1981) to obtain the roots in (probabilistic) polynomial time.

Finally, with the solution $z = 17/54$ to the equation corresponding to the last line of M_{red}, we *extend* this solution to the other variables by substitution in the equations corresponding to the remaining rows. This extension process will also apply in the non-linear case.

Table 2.1 summarizes the conceptual steps in this solution to a linear system of equations and the corresponding ideas of the non-linear case that will be developed in Sections 2.2 through 2.4.

2.2 Monomial orders

We will now develop the concept of *monomial order*, which will ultimately allow the elimination of variables and the computation of a non-linear analogue to the row echelon form of the matrix of a linear system of equations, a Gröbner Basis. Our initial definitions and notation follow (COX; LITTLE; O'SHEA, 2015; COX; LITTLE; O'SHEA, 2005).

Let $\mathcal{M}$ be the set of all monomials over R, that is, the set of products $x_1^{\alpha_1}x_2^{\alpha_2}\dots x_n^{\alpha_n}$ for $\alpha_i \in \mathbb{Z}_{\geq 0}$. Clearly, there is a bijection between $\mathcal{M}$ and $\mathbb{Z}^n_{\geq 0}$ given by the exponents of the monomials. We will denote this bijective function by $\log$. In the following, we will use this bijection to define a monomial order. For simplicity, we denote $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n) \in \mathbb{Z}^n_{\geq 0}$ and $x^\alpha = x_1^{\alpha_1}x_2^{\alpha_2}\dots x_n^{\alpha_n}$, so $\log x^\alpha = \alpha$.

Definition 1. An order $<$ over $\mathbb{Z}^n_{\geq 0}$ is a monomial order if it satisfies the following conditions:

1. $<$ is a total order;
2. For any $\gamma \in \mathbb{Z}^n_{\geq 0}$, if $\alpha < \beta$ then $\alpha + \gamma < \beta + \gamma$;
3. $<$ is a well-ordering, that is, any subset $S \subset \mathbb{Z}^n_{\geq 0}$ has a minimum.

Condition 2 in the above definition corresponds to the restriction that a monomial order should be preserved by monomial multiplication, that is, if $x^\alpha < x^\beta$ and x^γ is any monomial, it must be true that $x^\alpha x^\gamma < x^\beta x^\gamma$ while, whenever condition 2 holds, condition 3 is equivalent to specifying that $\alpha > 0$ for any $\alpha \neq 0$ in $\mathbb{Z}^n_{\geq 0}$ or, in terms of monomials, $x^\alpha \geq 1$ for any α.

Notation: Let $<$ be a monomial order, and $f \in R$ a polynomial. We denote $\mathrm{Supp}(f)$ the *support* of f, that is, the set of monomials appearing in f, $LM(f)$ the *leading monomial* of f, the largest monomial with respect to $<$ appearing in f, and $LC(f)$ the *leading coefficient* of f, that is, the coefficient of $LM(f)$ in f. Also, we define the *leading term* $LT(f) = LC(f)LM(f)$. Finally, for $G \subseteq R$, let $\langle LT(G)\rangle$ be the ideal generated by $\{LT(g) \mid g \in G\}$.

We denote $|\alpha| = \sum_{i=1}^n \alpha_i$ the *degree* of a monomial, and $\deg f$ the degree of a polynomial f, defined as the maximum of the degrees of its monomials. A monomial order $<$ is *graded* if $x^\alpha < x^\beta$ when $|\alpha| < |\beta|$. We now define two usual monomial orders.

Example 2 (Lexicographical order)**.** The *lexicographical order*, denoted $<_{lex}$, is the order where $x^\alpha > x^\beta$ if and only if the leftmost nonzero entry of $\alpha - \beta$ is positive.

lex is not a graded order. In $k[x_1, x_2]$, for example, $x_1 > x_2^5$. It is often the order used to eliminate variables in a polynomial system.

Example 3 (Degree reverse lexicographical order)**.** The *degree reverse lexicographical order*, or $<_{grevlex}$, is defined by: $x^\alpha > x^\beta$ if $|\alpha| > |\beta|$ or $|\alpha| = |\beta|$ and the rightmost nonzero entry of $\alpha - \beta$ is negative. It is a graded order.

In $grevlex$, taking $R = k[x_1, x_2]$, $x_2^5 > x_1$ by the degree criterion and $x_1x_2^2 < x_1^2x_2$ by the tiebreaker criterion. This order is often used to speed up Gröbner Basis computations (see Sections 2.3, 2.7 and 2.8).

More generally, one can define a monomial order from a $s \times n$ matrix M (for some $s \in \mathbb{Z}_{\geq 0}$) with real entries by setting $x^\alpha >_M x^\beta$ when $M\alpha >_{lex} M\beta$, as long as $\ker M \cap \mathbb{Z}^n = 0$ and the the first nonzero entry of every column of M is positive. The following matrices M_{lex} and $M_{grevlex}$ correspond to the lex and $grevlex$ orders, respectively.

$$M_{lex} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, M_{grevlex} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Graded orders are exactly those with the same weight for every variable in the first row.

It is often useful to define a monomial order with respect to a weight vector $w \in \mathbb{R}^n$, instead of a full matrix. However, this does not define an order completely, as ties may happen. A solution to this issue is to break ties using another monomial order, such as $grevlex$. We denote such an order by $<_{w,grevlex}$.

In fact, Robbiano proved in (ROBBIANO, 1985) that every monomial order arises as a matrix with entries in $\mathbb{R}$. For $v \in \mathbb{R}^n$, define $d(v)$ as the dimension of the $\mathbb{Q}$-subspace of $\mathbb{R}$ spanned by the coordinates of v and let $A(d_i)$ be the quotient set $B(d_i)/\sim$, where $B(d_i)$ is the set of vectors v in $\mathbb{R}^n$ with $d(v) = d_i$ and $\sim$ is the equivalence relation given by $v \sim v'$ when there exists a nonzero $\lambda \in \mathbb{R}$ with $v = \lambda v'$. Then, the following classification theorem holds.

Theorem 4. *The monomial orders $<$ on R are classified by:*

- *The* type *of $<$, an integer s such that $1 \leq s \leq n$;*
- *The* partition type *of $<$, that is, a partition $\sum_{i=1}^{s} d_i = n$;*
- *Vectors $w_1, w_2, \ldots, w_s$ with $w_i \in A(d_i)$ such that*

  1. *for $i \in [s]$, if G_{i-1} is the $\mathbb{Q}$-subspace of $\mathbb{Q}^n$ of the vectors orthogonal to $w_1, \ldots, w_{i-1}$, with $G_0 = \mathbb{Q}^n$, then $w_i \in (G_{i-1} \otimes_{\mathbb{Q}} \mathbb{R})$.*
  2. *for every $v \in \mathbb{Z}^n_{\geq 0} \backslash \{0\}$, the first nonzero coordinate of $(v \cdot w_1, v \cdot w_2, \ldots, v \cdot w_s)$ is positive.*

The vectors $w_1, \ldots, w_s$ of this theorem give the rows of the matrix M defining the matrix order $<_M$. Condition 2 in the classification theorem is clearly necessary, otherwise one could have elements $\alpha \in \mathbb{Z}^n_{\geq 0}$ with $\alpha <_M 0$, contradicting the definition of a monomial order. The orthogonality imposed by condition 1 guarantees that $\ker M = 0$. Note, however, that the classification theorem does not imply that every matrix M defining a monomial order has to be orthogonal, merely that the same order could be defined by an orthogonal matrix (see Proposition 9). The following examples show the influence of the type and the partition type of a monomial order.

Example 5. There is a single monomial order over $R = k[x]$, namely, the order where $x^i < x^j$ whenever $i < j$. To see this, note that the only partition of 1 is $d_1 = 1$, so any monomial order over R has type 1. Moreover, $\#A(1) = 1$, as every element of $\mathbb{R} \setminus \{0\}$ is equivalent with respect to $\sim$.

Example 6. Let $w = (1, \sqrt{2}) \in \mathbb{R}^2$. Then $d(w) = 2$, because 1 and $\sqrt{2}$ are linearly independent over $\mathbb{Q}$. From the classification theorem, it follows that w completely defines a monomial order over $R = k[x_1, x_2]$. This order has type 1, as only one vector is needed to define it, and partition type (2).

An ordering of type 1 is also called an *archimedean ordering*.

Example 7. Let $w_1 = (\pi, 1, 5) \in \mathbb{R}^3$. Then $d(w_1) = 2$, because π and 1 are linearly independent over $\mathbb{Q}$ but 1 and 5 are not. As $d(w_1) = 2$ is not a partition of $n = 3$, the vector w_1 does not completely define a monomial order over $R = k[x_1, x_2, x_3]$. To complete the definition of a monomial order, we need a vector w_2 with $d(w_2) = 1$, completing a partition of 3. This process of adding another vector will *break ties* that would occur with an order defined solely by w_1.

For example, if we take $v_1 = x^{10}y^5z$ and $v_2 = x^{10}y^{10}$ then $w_1 \cdot v_1 = 10\pi + 10 = w_1 \cdot v_2$, so a tie happens. If we set $w_2 = (1, 1, 1)$, we have $d(w_1) + d(w_2) = n$ and we break the above tie by scalar multiplication by w_2, so $w_2 \cdot v_1 = \pi + 6 < \pi + 10 = w_2 \cdot v_2$. The classification theorem now guarantees there will be no more ties. This order has type 2 and partition type $(2, 1)$.

Remark 8. If we restrict ourselves to rational (or integer) entries in the vectors w_i composing the matrix order, then all orders will have type n (that is, n vectors will be needed to completely define an ordering) and partition type $(1, 1, \ldots, 1)$.

Distinct matrices may, however, define the same order. In the following proposition, we show that matrices that can be row reduced to each other without swapping rows define the same order. It appears as an exercise (Tutorial 9) in (KREUZER; ROBBIANO, 2000).

Proposition 9. *Let $w_1, \dots, w_s \in \mathbb{R}^n$ be such that $M = [w_1; \dots; w_s]$, the matrix with the w_i as its rows, defines a monomial order. Fix $i \in [s]$. Then if $w_i' = \lambda w_i$ or $w_i' = w_i + \lambda w_j$, for $j < i$ and $\lambda > 0$, $M' = [w_1; \dots; w_i'; \dots w_s]$ defines the same monomial order as M.*

Proof. Let $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$. To prove the first case, let $w_i' = \lambda w_i$ for some $\lambda \in \mathbb{R}$. Then $w_i \cdot \alpha > w_i \cdot \beta$ if and only if $\lambda(w_i \cdot \alpha) > \lambda(w_i \cdot \beta)$ and, by the properties of the inner product, this is equivalent to $w_i' \cdot \alpha > w_i' \cdot \beta$.

Now suppose $w_i' = w_i + \lambda w_j$, for $\lambda \in \mathbb{R}$ and $j < i$. If α and β do not tie with respect to w_j, then $M'\alpha >_{lex} M'\beta$ if, and only if, $M\alpha >_{lex} M\beta$. If α and β tie when compared with respect to w_j, that is, $\alpha \cdot w_j = \beta \cdot w_j$, then

$$\alpha \cdot w_i' = \alpha \cdot w_i + \lambda(\alpha \cdot w_j) > \beta \cdot w_i + \lambda(\beta \cdot w_j) = \beta \cdot w_i'$$

exactly when $\alpha \cdot w_i > \beta \cdot w_i$. □

2.3 Polynomial reduction and Gröbner Bases

In the linear example of Section 2.1, new polynomials in the ideal $I = \langle \mathcal{F} \rangle$ were generated by linear combinations of previous ones in the process of computing a row-echelon form of the matrix representing the input system. In the non-linear case, as we allow higher degree polynomials, these linear combinations are replaced by algebraic combinations of the form

$$f = \sum_{i=1}^{m} a_i f_i$$

for $a_i \in R$. Before approaching the problem of deciding which algebraic combinations of the input generate useful polynomials (corresponding, in the linear case, to those in the row-echelon form matrix) we define a multivariate division algorithm, generalizing the usual univariate polynomial division.

Proposition 10 (Multivariate division algorithm). *Let $f \in R$, $\mathcal{F} = \{f_1, \dots, f_m\}$ and fix a monomial order $<$. Then we can write*

$$f = \sum_{i=1}^{m} a_i f_i + r$$

for some $a_i, r \in R$ with either $a_i f_i = 0$ or $LT(a_i f_i) \leq LT(f)$, and no monomial of r being divisible by any $LT(f_i)$.

The polynomial r is then called the remainder *of the division of f by $\mathcal{F}$ or the* normal form *of f with respect to $\mathcal{F}$. We will also denote $r = \overline{f}^{\mathcal{F}}$.*

Clearly, if $r = 0$ in the division algorithm, then $f \in I = \langle \mathcal{F} \rangle$. The converse, however, is false, as it may happen that r is also in I. This suggests it may be useful to consider a particular basis G of I such that every monomial of $\langle LT(I) \rangle$ is in $\langle LT(G) \rangle$. This leads us to the definition of a Gröbner Basis, which will turn out to be our central object of study.

Definition 11. $G = \{g_1, \dots, g_t\}$ is said to be a *Gröbner Basis* of the ideal $I = \langle G \rangle$ (with respect to a monomial order $<$) if $\langle LT(G) \rangle = \langle LT(I) \rangle$.

Alternatively, G is a Gröbner Basis of I if and only if every $f \in I$ has normal form $\overline{f}^{G} = 0$, which establishes a converse to the statement that $f \in I$ if and only if its remainder in the division algorithm is 0.

Example 12. Let $k = \mathbb{F}_{32003}$ and $I = \langle \{f, g, h\} \rangle$ with

$$\begin{aligned} f &= z^2 + 17983x + 6683y + 8704z + 6113, \\ g &= 5763xy + 30567yz + 7999z^2 + 14968y + 23433z, \\ h &= 477yz + 8393z^2 + 26904x + 9921y + 15994z \end{aligned}$$

over $R = k[x, y, z]$. A Gröbner Basis of I with respect to the $grevlex$ ordering is

$$\begin{aligned} G = \{&x^2 + 8129xz + 19771x + 10160y + 2070z + 11415, \\ &xy + 27808x + 3172y + 5036z + 15620, \\ &y^2 + 9739xz + 12367x + 19043y + 26414z + 2658, \\ &yz + 8365x + 1117y + 20518z + 14346, \\ &z^2 + 17983x + 6683y + 8704z + 6113\} \end{aligned}$$

It is possible to show that every ideal over R has a Gröbner Basis. Indeed, algorithms that compute such a basis will be introduced in Section 2.4. Gröbner Bases are also unique in a certain sense, as will be made clear by the next definition.

Definition 13. A Gröbner Basis G is *reduced* if

1. $LC(g) = 1$ for all $g \in G$;
2. No monomial of any $g \in G$ is in $\langle LT(G \setminus \{g\}) \rangle$.

An ideal over R has a unique reduced Gröbner Basis with respect to any fixed monomial order $<$. It can be easily obtained from any Gröbner Basis G by successively replacing each polynomial $g \in G$ by $\overline{g}^{G \setminus \{g\}}$ in order of increasing leading terms with respect to $<$. This process is sometimes called *interreduction*.

Example 14. Let $k = \mathbb{F}_2$ and define, over $R = k[x, y, z]$, an ideal I generated by the following polynomials:

$$\{x^7y^2, x^7y^2 + z^9 + xz, xy^5z^4 + x^2y^6\}$$

Then, the reduced Gröbner Basis of I with respect to the $grevlex$ ordering has 26 polynomials with 347 monomials in total, counted with multiplicity. It contains a polynomial with 40 monomials and a polynomial of degree 19. The leading monomials of the polynomials in the Gröbner Basis are:

$$\begin{aligned}\{&z^{19}, x^3y^{14}, x^8z^9, x^4y^4z^9, y^8z^9, x^5z^{12}, x^3y^{12}z, x^2y^{13}z, x^2y^{12}z^2, x^{12}yz^3,\\ &x^{12}z^4, x^{11}z^5, x^6z^{10}, x^6y^9, x^5y^{10}, x^4y^{11}, x^{14}z, x^{13}yz, x^4y^{10}z, x^4yz^{10},\\ &xz^{13}, x^8yz^4, y^2z^{10}, x^5y^4z, xy^5z^4, x^7y^2\}\end{aligned}$$

Note that even the reduced Gröbner Basis is much larger than the input ideal, both in terms of number of polynomials, monomials and in the maximum degree appearing in the basis.

Different monomial orders have potentially distinct Gröbner Bases that may satisfy various properties. Some of these will be explored in more detail in Section 2.7. For now, however, we will link the computation of Gröbner Bases to polynomial system solving through some properties of the lex order.

Suppose I is an ideal and some power of the variable x_n, say x_n^s, is in $\langle LT(I)\rangle$, with s minimal. Let G be the Gröbner Basis of I in $<_{lex}$. Then there exists $g \in G$ with $LM(g) = x_n^s$ and, moreover, $g \in k[x_n]$, because any monomial smaller than x_n^s in the lex order cannot contain any other variable. This reasoning can then be repeated to infer that there are elements of G in $k[x_i, \ldots, x_n]$ for each $i \in [n]$ whenever there is $g \in G$ with leading monomial in $k[x_i, \ldots, x_n]$. Thus, informally, we can say that a lex Gröbner Basis is a generating set of I with the elimination property we were looking for to be able to solve polynomial systems. More precisely, we have:

Theorem 15 (Elimination theorem)**.** *Let I be an ideal, G be a Gröbner Basis of I with respect to the lex order and $I_l = I \cap k[x_{l+1}, \ldots, x_n]$. Then $G_l = G \cap k[x_{l+1}, \ldots, x_n]$ is a lex Gröbner Basis of I_l.*

Let $P = (a_{l+1}, \ldots, a_n) \in k^{n-l}$ be a *partial solution* of a system $\mathcal{F}$, that is, a zero of every polynomial in I_l. We define the morphism

$$\begin{aligned} \phi_P : k[x_1, \ldots, x_n] &\to k[x_1, \ldots, x_l] \\ f(x_1, \ldots, x_n) &\mapsto f(x_1, \ldots, x_l, a_{l+1}, \ldots, a_n), \end{aligned}$$

the application of P to f.

We denote the set of solutions of $\mathcal{F}$ by

$$V(\mathcal{F}) = \{(a_1, \ldots, a_n) \in k^n \mid f_i(a_1, \ldots, a_n) = 0 \, \forall i \in [m]\}.$$

Suppose that $V(\mathcal{F})$ is finite and k is algebraically closed. Then we can use Gröbner Bases and the Elimination theorem above as in Algorithm 1 to compute $V(\mathcal{F})$, solving the multivariate polynomial system problem in this case. The same ideas can be applied to solve systems over non algebraically closed fields as well with a few adaptations, such as dealing with the case where a partial solution cannot be extended to more variables.

```
Input: A polynomial set $\mathcal{F}$ with a finite number of solutions
Output: $V(\mathcal{F})$, the set of solutions of $\mathcal{F}$
$G :=$ a Gröbner Basis of $\langle \mathcal{F} \rangle$ with respect to $lex$
for $i := 1, \ldots, n$ do
    $V_i := \emptyset$
end
for $i := n-1, \ldots, 1$ do
    $G_i := G \cap k[x_{i+1}, \ldots, x_n]$
    for $(a_{i+1}, \ldots, a_n) \in V_{i+1}$ do
        $F_i := \phi_{(a_{i+1}, \ldots, a_n)}(G_i)$
        $V_i := V_i \cup \{(a_i, \ldots, a_n) \in k^n \mid g(a_i) = 0 \, \forall g \in F_i\}$
    end
end
$V(\mathcal{F}) := V_1$
```

Algorithm 1: Solving multivariate polynomial systems with Gröbner Bases.

2.4 Algorithms for computing a Gröbner Basis

In the previous section, we defined Gröbner Bases of an ideal and described some of their properties. Now, we will briefly present three main families of algorithms for computing Gröbner Bases: the Buchberger family, introduced in Buchberger's Thesis (BUCHBERGER, 2006), the F4 family of matrix-based algorithms (FAUGÈRE, 1999) and the signature-based family based on F5 (FAUGÈRE, 2002).

2.4.1 Buchberger's algorithm

Intuitively, to obtain a Gröbner Basis of $I = \langle \mathcal{F} \rangle$ from $\mathcal{F}$, we will need to generate polynomials with new leading terms that are not multiples of previous ones. The main tool to achieve this is the S-polynomial.

Definition 16. The S-polynomial of $f, g \in R$, denoted by $S(f, g)$ is defined by

$$S(f,g) = \frac{\lambda}{LT(f)} f - \frac{\lambda}{LT(g)} g$$

where $\lambda = \operatorname{lcm}(LM(f), LM(g))$.

Note that if $f, g \in I$ then $S(f, g) \in I$. Also, the following theorem relates S-polynomials to Gröbner Bases.

Theorem 17 (Buchberger's criterion)**.** *$G = \{g_1, \ldots, g_t\} \subset R$ is a Gröbner Basis of $I = \langle G \rangle$ if and only if $\overline{S(g_i, g_j)}^G = 0$ for all $i, j \in [t]$.*

This theorem gives a criterion for the termination of a Gröbner Basis algorithm. In fact, by the division algorithm, $\overline{S(f,g)}^G$ for G a finite subset of R and $f, g \in G$ is either 0 or has a leading term not in $\langle LT(G) \rangle$. This gives Algorithm 2, known as Buchberger's algorithm. Subalgorithm $choose$ takes an element from the set P according to some criteria, for example, choosing the minimal element with respect to the given monomial order.

Various implementation questions arise when trying to program Buchberger's algorithm, such as how to represent the data structures involved (ROUNE; STILLMAN, 2012), such as the set P (which is usually implemented as a priority queue, and will be referred to as such in the following), how to choose an element from P (GIOVINI et al., 1991), how to compute polynomial reductions and how, if possible, to predict that certain

```
**Input:** $F = \{f_1, \dots, f_m\} \subset R$, $<$ a monomial order over $R$
**Output:** $G$, a Gröbner Basis of $I = \langle F \rangle$ with respect to $<$
$G := \emptyset$
$P := F$
**while** $P \neq \emptyset$ **do**
    $f := choose(P)$
    $P := P \setminus \{f\}$
    **if** $\overline{f}^G \neq 0$ **then**
        $P := P \cup \{S(\overline{f}^G, g) \mid g \in G)\}$
        $G := G \cup \{\overline{f}^G\}$
    **end**
**end**
```

Algorithm 2: Buchberger's algorithm

S-polynomials will reduce to zero in order to simply remove them without the computational cost of a reduction (GEBAUER; MÖLLER, 1988). All of these factors have a significant impact on the practical performance of the algorithm.

2.4.2 Faster reduction with matrices: F4

Even with a good implementation of Buchberger's algorithm, two bottlenecks may appear in the performance of the computation of a Gröbner Basis: the polynomial reduction itself and the amount of polynomials reduced to zero, which waste computational time. The F4 family of algorithms was created to address the former problem by reducing multiple polynomials simultaneously in a structured matrix and, at the same time, profit from well studied sparse linear algebra.

It is possible to represent a set of polynomials as a matrix by seeing them as linear combinations over k of their monomials, as shown in the following example.

Example 18. If $R = \mathbb{Q}[x, y, z]$ and $\mathcal{F} = \{f_1, f_2, f_3\}$ with $f_1 = xyz + 2x + 3, f_2 = x^3 + 5xy + 7xz, f_3 = y^2 + 10x - 2z$ in the $grevlex$ order, we can build the matrix

$$\begin{array}{c|cccccccc|}
 & x^3 & xyz & xy & y^2 & xz & x & z & 1 \\
f_1 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 3 \\
f_2 & 1 & 0 & 5 & 0 & 7 & 0 & 0 & 0 \\
f_3 & 0 & 0 & 0 & 1 & 0 & 10 & -2 & 0
\end{array}$$

by taking the monomials of f_1, f_2, f_3, in $grevlex$ order, as the columns, and each polynomial in $\mathcal{F}$ as a row. Note that, in many applications, the polynomials will not have many monomials in common, so often the generated matrices will be sparse.

It can also be shown that

Proposition 19. *For $I = \langle f_1, \ldots, f_m \rangle$ homogeneous, every element of degree d in I for any $d \in \mathbb{Z}$ is a linear combination of some $x^\alpha f_i$, with $|\alpha| + \deg(f_i) = d$.*

Proof. See (COX; LITTLE; O'SHEA, 2015), Chapter 10, §1, Lemma 7. □

This result suggests that there is a choice of polynomials of the form $x^\alpha f_i$ such that every polynomial of the given degree in I, including S-polynomials, is generated as a linear combination. This is one of the key ideas linking Buchberger's algorithm to a matrix-based reduction step, as it can also be shown that the leading monomials appearing in the row reduction of a suitable matrix include all leading monomials of the target degree in the ideal.

Algorithm 3 shows a high-level view of F4, based on (COX; LITTLE; O'SHEA, 2015). The set P represents, as before, the S-polynomials, but represented as pairs of indices (i, j). We can see that the overall structure of the algorithm is similar to Buchberger's, the main difference being in the reduction step, based on the construction and row reduction of a matrix. For simplicity, suppose the input system $\mathcal{F}$ is homogeneous and that the choice strategy implemented by the $chooseSubset$ algorithm is taking all elements of P of minimal degree (this is often called the *normal strategy*).

Then, the procedure $makeMatrix$ constructs a matrix as previously described from the polynomials $\frac{\operatorname{lcm}(g_i, g_j)}{LT(g_i)} g_i$ for $g_i, g_j \in G$, that is, the components of S-polynomials, as well as some other polynomials in order to guarantee a similar condition to that of Proposition 19. Then, polynomials obtained from row reduction of M that add new information about a Gröbner Basis, which are those with new leading monomials with respect to M, are added to the basis and further considered to build the matrices of subsequent iterations.

Note that the F4 algorithm, as presented in Algorithm 3 also works for non-homogeneous ideals. However, the construction of the matrices is not as simple and some phenomena that never happen in the homogeneous case may occur. For example, when applying the normal strategy in the homogeneous case as above, we proceed degree by degree — once polynomials of degree $d' > d$ are processed, it is guaranteed that no more polynomials of degree d will be added to the Gröbner Basis. This is not true when the

```
Input: $F = \{f_1, \dots, f_m\} \subset R$, $<$ a monomial order over $R$
Output: $G$, a Gröbner Basis of $I = \langle F \rangle$ with respect to $<$
$G := F$
$P := \{(i, j) \mid 1 \leq i < j \leq m\}$
while $P \neq \emptyset$ do
    $P' := chooseSubset(P)$
    $P := P \setminus P'$
    $M := makeMatrix(P')$
    $N := rowEchelonForm(M)$
    for $f \in rows(N)$ do
        if $LM(f) \notin \langle LM(rows(M)) \rangle$ then
            $G := G \cup \{f\}$
            $m := m + 1$
            $P := P \cup \{(i, m) \mid 1 \leq i \leq m - 1\}$
        end
    end
end
```

Algorithm 3: F4 algorithm.

input generators are not homogeneous, as the elimination of leading monomials may generate new polynomials with leading monomials of lower degrees. This is called a *degree fall* and is sometimes used as a measure of complexity.

2.4.3 Avoiding reductions to zero using signatures: F5

Buchberger's algorithm often generates S-polynomials that are eventually reduced to zero. Various criteria can be added to the algorithm to predict and avoid zero reductions, but their coverage is usually not complete. Signature-based algorithms, originated from F5 (FAUGÈRE, 2002), use additional algebraic structure of the input system to eliminate useless S-polynomials and have the advantage of guaranteeing no reductions to zero if the input system is regular (see example 36 for a definition). More recently, the survey (EDER; FAUGÈRE, 2017) unified the algorithms of this signature-based family in a single framework with various input parameters that can be seen as instantiations of the algorithms. In this subsection, we will present the main ideas of the signature-based family of algorithms based on this survey.

Fix $I = \langle \mathcal{F} \rangle$ with $\mathcal{F} = \{f_1, \dots, f_m\}$. Let R^m be the rank m free module over R

with the standard basis $\{\mathbf{e_i} \mid 1 \leq i \leq m\}$, and equip it with a module homomorphism

$$\begin{aligned} \pi : R^m &\to I \\ \sum_{i=1}^{m} g_i \mathbf{e_i} &\mapsto \sum_{i=1}^{m} g_i f_i \end{aligned}$$

so that elements of R^m can be associated to formal algebraic combinations of the elements of $\mathcal{F}$. Elements of R^m will be written in **bold**. In signature-based algorithms, we will use such formal combinations to keep track of the way intermediate polynomials occurring in the computations are written in terms of the input system and to predict reductions to zero.

A *module monomial* is an element of R^m of the form $x^\alpha \mathbf{e_i}$ where x^α is a monomial of R. A *module monomial order* is a total order over the monomials of R^m that is a well ordering and respects multiplication by elements of R — if $\mathbf{T_1}, \mathbf{T_2}$ are module monomials with $\mathbf{T_1} < \mathbf{T_2}$, then $x^\alpha \mathbf{T_1} < x^\alpha \mathbf{T_2}$ for any $\alpha \in \mathbb{Z}^n_{\geq 0}$.

Example 20. We define the position-over-term order $<_{pot}$ over R^m by $x^\alpha \mathbf{e_i} >_{pot} x^\beta \mathbf{e_j}$ if and only if $i < j$ or $i = j$ and $x^\alpha > x^\beta$ over R. It is a module monomial order.

Instead of storing the entire representation of a module element of R^m to track the origin of polynomials in the execution of the algorithm, F5-based algorithms store a single module monomial associated to each polynomial. This reduces memory consumption and, as we will see, is enough to predict many reductions to zero. These stored module monomials are called *signatures*.

Definition 21. The *signature* of $\mathbf{f} \in R^m$, denoted $\mathfrak{s}(\mathbf{f})$, is the leading monomial of $\mathbf{f}$ with respect to a fixed module monomial order $<_m$.

We may also refer to the signature of a polynomial $f \in I$. In this case, a signature of f, denoted $\mathfrak{s}(f)$, is the signature of some $\mathbf{f} \in R^m$ such that $\pi(\mathbf{f}) = f$.

The concepts of Gröbner Basis and S-polynomials can be generalized to the signature case. The S-polynomials of free module elements are then called *S-pairs* and denoted $spair(\mathbf{f}, \mathbf{g})$ for $\mathbf{f}, \mathbf{g} \in R^m$. An analogue of Buchberger's criterion is also available for S-pairs. One can also define signature reduction steps (called $\mathfrak{s}$-reductions) analogue to the multivariate polynomial division of Section 2.3. These can be required, for example, to keep signatures unchanged, but eliminate lead terms. These reductions allow us to define signature Gröbner Bases.

```
Input: $F = \{f_1, \ldots, f_m\} \subset R$, $<$ a monomial order over $R$, $<_m$ a module
    monomial order over $R^m$
Output: $G$, a Gröbner Basis of $I = \langle F \rangle$ with respect to $<$
$G := \emptyset$
$H := \{f_i \mathbf{e_j} - f_j \mathbf{e_i} \mid 1 \leq i \leq m\}$
$P := \{\mathbf{e_i} \mid 1 \leq i \leq m\}$
while $P \neq \emptyset$ do
    $\mathbf{f} := \min_{<_m} P$
    $P := P \setminus \{\mathbf{f}\}$
    if not $Rewritable(\mathbf{f}, G \cup H)$ then
        if $\overline{\mathbf{f}}^G = 0$ then
            $H := H \cup \{\overline{\mathbf{f}}^G\}$
        else
            $G := G \cup \{\overline{\mathbf{f}}^G\}$
            $P := \{spair(\overline{\mathbf{f}}^G, \mathbf{g}) \mid \mathbf{g} \in G\}$
        end
    end
end
```

Algorithm 4: F5 algorithm.

Definition 22. A finite subset $G \subset R^m$ is a signature Gröbner Basis of I with respect to a module monomial order $<_m$ (and a monomial order $<$ over R) if every element $\mathbf{f} \in R^m$ $\mathfrak{s}$-reduces to zero by G.

The following proposition allows the computation of a Gröbner Basis from a signature Gröbner Basis.

Proposition 23. *If G is a signature Gröbner Basis of I then $\pi(G) = \{\pi(g) \mid g \in G\}$ is a Gröbner Basis of I.*

Algorithm 4, based on the RB framework of (EDER; FAUGÈRE, 2017), can be seen as a generalization of Buchberger's algorithm to the free modules R^m. In actual implementations, it is usual to not store the full module elements, only pairs $\mathfrak{s}(f), f$ for a polynomial f. The $Rewritable$ function implements rewriting criteria as defined in (EDER; FAUGÈRE, 2017), detecting useless S-pairs to avoid reductions. The main idea is that only a single polynomial with each signature has to be reduced, as long as S-pairs are treated in signature-increasing order. So S-pairs with repeated signatures can be discarded. This is sometimes called the *signature criterion*. The $Rewritable$ function can also implement the *syzygy criterion*, that discards S-pairs with signatures that are multiples of previously reduced to zero signatures. These are stored in the set of known syzygies (reductions to zero), H. These two criteria are unified by the Rewrite Bases (RB) framework described in the survey (EDER; FAUGÈRE, 2017).

The F4 and F5 families are not disjoint, and there are algorithms that both use signatures to avoid useless computations and do F4-style matrix reductions (see (EDER; FAUGÈRE, 2017), §13 for an overview).

2.5 The Syzygy module of a monomial ideal and useless S-polynomial elimination

Throughout this section, we let $I = \langle t_1, \ldots, t_m \rangle$ be a monomial ideal. In the context of Gröbner Basis computations, we take the t_i as the leading monomials (with respect to a fixed monomial order) of a partial Gröbner Basis G occurring during the execution of Buchberger's algorithm.

The *syzygy module* of I is defined as

$$Syz(I) = \left\{ (f_1, \ldots, f_m) \in R^m \mid \sum_{i=0}^{m} f_i t_i = 0 \right\}.$$

We also define the syzygies

$$S_{ij} = \frac{\operatorname{lcm}(t_i, t_j)}{t_i} \mathbf{e_i} - \frac{\operatorname{lcm}(t_i, t_j)}{t_j} \mathbf{e_j}$$

where $\mathbf{e_i}$ is the i-th element of the canonical basis of R^m. These syzygies correspond to the S-polynomials appearing in the computation of Gröbner Bases (in fact, the S in S-polynomial stands for syzygy). It can be shown that the syzygies S_{ij} generate $Syz(I)$, although they are possibly not a minimal set of generators. This fact was implicit in the discussion of Buchberger's algorithm in the previous Section. In the remainder of this Section, we will present techniques to compute smaller or, in some cases, minimal, sets of generators of $Syz(I)$. This is equivalent to eliminating most useless S-polynomials in Buchberger's algorithm.

2.5.1 Buchberger's criteria and the Gebauer-Möller algorithm

Buchberger introduced two criteria for eliminating useless S-polynomials in Gröbner Basis computations (BUCHBERGER, 1985). This is equivalent to eliminating redundant generators of the module of syzygies $Syz(I)$, and our presentation of these criteria will be based on eliminating these redundant generators. The criteria are the following:

Proposition 24 (LCM Criterion)**.** *Suppose* $S \subseteq \{S_{ij} \mid 1 \leq i < j \leq m\}$ *generates* $Syz(I)$*. If there are distinct* i, j, k *with* t_k *dividing* $\operatorname{lcm}(t_i, t_j)$ *and* $S_{ik}, S_{jk} \in S$*, then* $S \setminus \{S_{ij}\}$ *also generates* $Syz(I)$*.*

Proposition 25 (GCD Criterion)**.** *Suppose* $S \subseteq \{S_{ij} \mid 1 \leq i < j \leq m\}$ *generates* $Syz(I)$*. If* t_i *and* t_j *are coprime then* $S \setminus \{S_{ij}\}$ *also generates* $Syz(I)$*.*

In (BUCHBERGER, 1985), whenever an S-polynomial is chosen to be reduced, it is checked with respect to the two criteria above and may be discarded, without need for further processing. They do not guarantee to discard all useless S-polynomials.

(GEBAUER; MÖLLER, 1988) proposed an algorithm to update the queue P (see Algorithm 2) of S-polynomials based on Buchberger's criteria, reducing its size, instead of waiting to eliminate useless elements right before they are processed. It is shown in Algorithm 5.

```
**Input:** A generating set $S$ of $Syz(I)$, a monomial $t_{m+1}$
**Output:** A "small" generating set of $Syz(I + \langle t_{m+1} \rangle)$
Eliminate from $S$ all $S_{ij}$ such that $t_{m+1} | \operatorname{lcm}(t_i, t_j)$ and
  $\operatorname{lcm}(t_{m+1}, t_i) \neq \operatorname{lcm}(t_i, t_j) \neq \operatorname{lcm}(t_{m+1}, t_j)$
$S' = \{S_{i(m+1)} \mid 1 \leq i \leq m\}$
Eliminate from $S'$ all $S_{i(m+1)}$ such that there is $j$ with $\operatorname{lcm}(t_j, t_{m+1})$ a proper
  divisor of $\operatorname{lcm}(t_i, t_{m+1})$
For every $l = \operatorname{lcm}(t_j, t_{m+1})$ keep only one element $S_{i(m+1)}$ in $S'$ with
  $\operatorname{lcm}(t_i, t_{m+1}) = l$
Eliminate from $S'$ all $S_{i(m+1)}$ such that $t_i$ and $t_{m+1}$ are coprime
Return $S \cup S'$
```

Algorithm 5: The Gebauer-Möller update algorithm

The idea is updating the priority queue in Buchberger's algorithm when a new polynomial g_{m+1} with leading monomial t_{m+1} is being inserted. This update algorithm is used in various implementations of Buchberger's algorithm, for example, (CABOARA; PERRY, 2014; PERRY, 2017). It is also not guaranteed to return a minimal generating set of $Syz(I)$, although it has been experimentally shown that the queue is much smaller in practice than applying Buchberger's criteria (GEBAUER; MÖLLER, 1988) and that the number of generators of the module of syzygies is often close to minimal (CABOARA; KREUZER; ROBBIANO, 2004).

2.5.2 Graph-based criteria

In addition to Buchberger's criteria and the Gebauer-Möller update algorithm, which are used in most serious implementations of Buchberger's algorithm, there are also graph-based characterizations of useless S-polynomials.

Definition 26. (MILLER; STURMFELS, 2004) The *Buchberger graph* $Buch(I)$ is the graph with vertices $t_1, \ldots, t_m$ and an edge (t_i, t_j) if there is no t_k that divides $\operatorname{lcm}(t_i, t_j)$ and has smaller degree than $\operatorname{lcm}(t_i, t_j)$ in every variable appearing in it.

Remark 27. The Buchberger graph is not invariant with respect to the choice of generators of I. To obtain a uniquely defined graph $Buch(I)$ one must take $t_1, ..., t_m$ to be the minimal generators of I. Whenever we refer to $Buch(I)$, we mean the graph defined by the minimal generators in this way.

It turns out that the edges of $Buch(I)$ generate $Syz(I)$. In order to prove that we will now prove the following lemma.

Lemma 28. *Let t_x, t_y be fixed monomials. For any $i_1, \ldots, i_l, i_{l+1} \in [m]$ with $i_1 = i_{l+1}$ and t_{i_j} dividing $\operatorname{lcm}(t_x, t_y)$ for all $j \in [l]$,*

$$\sum_{j=1}^{l} \frac{\operatorname{lcm}(t_x, t_y)}{\operatorname{lcm}(t_{i_j}, t_{i_{j+1}})} S_{i_j i_{j+1}} = 0$$

Proof. First, note that the above expression makes sense in $Syz(I)$, as t_{i_j} dividing $\operatorname{lcm}(t_x, t_y)$ implies that the least common multiples in the denominators do as well.

Now,

$$\sum_{j=1}^{l} \frac{\operatorname{lcm}(t_x, t_y)}{\operatorname{lcm}(t_{i_j}, t_{i_{j+1}})} S_{i_j i_{j+1}} = \sum_{j=1}^{l} \left(\frac{\operatorname{lcm}(t_x, t_y)}{t_{i_j}} \mathbf{e_{i_j}} - \frac{\operatorname{lcm}(t_x, t_y)}{t_{i_{j+1}}} \mathbf{e_{i_{j+1}}} \right)$$
$$= \frac{\operatorname{lcm}(t_x, t_y)}{t_{i_1}} \mathbf{e_{i_1}} - \frac{\operatorname{lcm}(t_x, t_y)}{t_{i_{l+1}}} \mathbf{e_{i_{l+1}}} + \sum_{j=2}^{l} \left(\frac{\operatorname{lcm}(t_x, t_y)}{t_{i_j}} \mathbf{e_{i_j}} - \frac{\operatorname{lcm}(t_x, t_y)}{t_{i_j}} \mathbf{e_{i_j}} \right) = 0$$

□

Proposition 29. *The set S consisting of the S_{ij} such that (t_i, t_j) is an edge of $Buch(I)$ generates $Syz(I)$.*

Proof. Let i, j be such that (t_i, t_j) is not an edge in $Buch(I)$. Then there exists t_k dividing $\operatorname{lcm}(t_i, t_j)$ with smaller degree than $\operatorname{lcm}(t_i, t_j)$ in every variable appearing in it. Applying

Lemma 28 to $x = i, y = j, i_1 = i, i_2 = j, i_3 = k, i_4 = i$ we have

$$\frac{\operatorname{lcm}(t_i,t_j)}{\operatorname{lcm}(t_i,t_j)}S_{ij} + \frac{\operatorname{lcm}(t_i,t_j)}{\operatorname{lcm}(t_j,t_k)}S_{jk} + \frac{\operatorname{lcm}(t_i,t_j)}{\operatorname{lcm}(t_k,t_i)}S_{ki} = 0$$
$$\implies S_{ij} = -\frac{\operatorname{lcm}(t_i,t_j)}{\operatorname{lcm}(t_j,t_k)}S_{jk} - \frac{\operatorname{lcm}(t_i,t_j)}{\operatorname{lcm}(t_k,t_i)}S_{ki}.$$

Moreover, $\operatorname{lcm}(t_i,t_k) \neq \operatorname{lcm}(t_i,t_j)$ and $\operatorname{lcm}(t_j,t_k) \neq \operatorname{lcm}(t_i,t_j)$ because t_k has smaller degree than $\operatorname{lcm}(t_i,t_j)$ in every variable appearing in it. That means S_{ij} can be written in terms of lower degree syzygies and is thus unnecessary to generate $Syz(I)$. □

Care must be taken in applying this criterion to eliminate useless S-polynomials in Gröbner Basis computations, as the following example shows.

Example 30. Let $f = x^2 + xy, g = x^3 + y, h = x^3 + z \in k[x,y,z]$ and choose the $grevlex$ ordering in what follows. We will compute a Gröbner Basis of $I = \langle G \rangle$, for $G = \{f, g, h\}$ using Buchberger's algorithm and applying the Buchberger graph criterion naively to eliminate S-polynomials. Immediately, from the graph criterion, we can eliminate S_{gh} as $LM(f)$ strictly divides $\operatorname{lcm}(LM(g), LM(h))$ in every variable. Carrying out computations, we see that

$$f_1 = \overline{S_{fh}}^G = -xy^2 - y$$

should be added to G and, similarly,

$$f_2 = \overline{S_{gh}}^G = y - z$$

should also be added to G, so that

$$G = \{x^2 + xy, x^3 + y, x^3 + z, -xy^2 - y, y - z\}.$$

Now, all S_{pf_1} are eliminated by the graph criterion for any $p \in G$, as $LM(f_2)$ divides $\operatorname{lcm}(LM(p), LM(f_1))$, as are S_{gf_2} and S_{hf_2}, with $LM(f)$ as divisor. The only S-polynomial remaining is then S_{ff_2}, but it reduces to zero immediately (this can also be seen from the GCD criterion). So G should be a Gröbner Basis — *but it is not*, as, for example,

$$\overline{S_{hf_2}}^G = -xz^3 - z^2 \neq 0.$$

The problem here is that some leading monomials are multiples of one another, and the correspondence between minimal generating sets of $Syz(I)$ and minimal sets of S-

polynomials breaks in this case. To solve this problem, one has to interreduce the polynomials in the input and in the course of the Gröbner Basis computations.

According to (ROUNE; STILLMAN, 2012), Bayer developed an unpublished characterization of a minimal set of generators of $Syz(I)$. This approach has been implemented in (ROUNE; STILLMAN, 2012) to eliminate S-polynomials right before their reduction, instead of when they are built, due to its relatively high overhead. Postponing the criterion's application in this way may cause a useless S-polynomial to be eliminated by other, lighter criteria before, thus reducing the number of calls to Bayer's graph criterion.

Proposition 31 (Bayer's criterion)**.** *Let t be a monomial and*

$$V_t = \{t_i \mid t_i \textit{ divides } t, 1 \leq i \leq m\}$$

$$E_t = \{(t_i, t_j) \mid \operatorname{lcm}(t_i, t_j) \neq t \textit{ or } S_{ij} \textit{ has been eliminated}\}.$$

Then S_{ij} may be eliminated from a generating set S of $Syz(I)$ if there is a path between t_i and t_j in $G_t = (V_t, E_t)$ with $t = \operatorname{lcm}(t_i, t_j)$.

Furthermore, applying this criterion for all pairs t_i, t_j gives a minimal generating set of $Syz(I)$.

2.6 Hilbert Function, Hilbert Series and Betti Numbers

Counting the monomials of a certain degree d in $\langle LT(I)\rangle$ for some ideal I and the minimal number of generators of $Syz(I)$ are relevant combinatorial questions to the design and complexity analysis of Gröbner Basis algorithms. In order to approach these problems, in this section we will study the graded structure of the polynomial ring R. First, we write R as a graded module

$$R = \bigoplus_{d \geq 0} R_d$$

where each R_d is the k-vector space of homogeneous polynomials of R of degree d. The set of monomials of degree d in R is a basis for R_d so that it is finite dimensional and, more precisely,

$$\dim_k R_d = \binom{d+n-1}{d}.$$

Define also the D-th *graded twist* of R as the graded module

$$R(D) = \bigoplus_{d \geq 0} R_{D+d}$$

which is essentially R as a module over itself, but with a degree $-D$ generator. Suppose that I is a monomial ideal. As $Syz(I)$ is finitely generated, we can write

$$Syz(I) = \bigoplus_{d \geq 0} R(-d)^{\beta_{1,d}}$$

for some $\beta_{1,d} \in \mathbb{N}$, almost all 0. They are called the (first) *Betti numbers* of I, and they count the minimal number of generators of degree d of $Syz(I)$. When we say *the* Betti number of I, we mean the number

$$\beta_1 = \sum_{d=0}^{\infty} \beta_{1,d}.$$

Now, let I be an homogeneous ideal of R (but not necessarily a monomial ideal) and $I_d = I \cap R_d$ be the k-vector space of homogeneous polynomials in I of degree d. We denote by S the k-algebra $S = R/I$ (a k-algebra is simply a ring that is also k-vector space). S has a graded structure inherited from R, that is, we can write

$$S = \bigoplus_{d \geq 0} S_d = \bigoplus_{d \geq 0} \frac{R_d}{I_d}$$

where S_d is the subspace of S of elements of degree d.

The link between these k-algebras and the question of counting the number of monomials of each degree in $\langle LT(I) \rangle$ is given by the following proposition.

Proposition 32. *The image of* $B = \{x^\alpha \in R \mid x^\alpha \notin \langle LT(I) \rangle\}$ *by the quotient map defining* S *is a basis of* S.

Proof. To prove linear independence, suppose there is a finite subset $B' \subseteq B$ such that $f = \sum_{x^\alpha \in B'} c_\alpha x^\alpha = 0$ in S with scalars $c_\alpha \in k$ not all zero. Then $f \in I$. In particular, $LT(f) \in \langle LT(I) \rangle$, contradicting the definition of B.

Now, let $g \in R$ and write $g = \sum_\alpha c_\alpha x^\alpha$. The quotient map $\pi : R \to S$ is linear, and so $\pi(g) = \sum_\alpha c_\alpha \pi(x^\alpha)$. So the $x^\alpha \in \langle LT(I) \rangle$ are taken to 0 in S and it follows that the nonzero monomials of $\pi(g)$ are exactly those not in $\langle LT(I) \rangle$. Thus B is a generating set of S. $\square$

This result motivates the following definitions.

Definition 33. The *Hilbert Function* of an homogeneous ideal I, denoted by HF_I is

$$HF_I(d) = \dim_k S_d = \dim_k R_d - \dim_k I_d$$

and the *Hilbert Series* of I is the generating series of the Hilbert Function, that is,

$$HS_I(t) = \sum_{d \geq 0} HF_I(d) t^d.$$

It is immediate from the definition that $HF_I(d) \leq \binom{d+n-1}{d}$. In fact, better bounds for specific ideals can be easily obtained. The following theorem and its corollary show such a bound and can be applied to compute the Hilbert Series of an ideal from one of its Gröbner Bases. Note that these results hold regardless of the choice of monomial order.

Theorem 34. $HS_I(t) = HS_{\langle LT(I) \rangle}(t)$.

Proof. We have to show that $\dim_k \langle LT(I) \rangle_d = \dim_k I_d$ for any d. The set $L = \{LM(f) \mid f \in I_d\}$ is finite, so we can take $B = \{f_1, \ldots, f_s\}$ with $LM(B) = L$. We suppose, without loss of generality, that $LM(f_i) > LM(f_j)$ for $i < j$. We start by showing B is a basis of I_d.

If $\sum_{i=1}^{s} c_i f_i = 0$ for $c_i \in k$ not all zero, we can take the minimal i such that $c_i \neq 0$. Then $LM(f_i) > LM(f_j) > m$ for any $j > i$ and any monomial m of any f_j. So $LM(f_i)$ cannot be canceled in this sum, a contradiction, and it follows that B is linearly independent.

Let $g_0 \in I_d$. Then $LM(g_0) = LM(f_{i_1})$ for some i_1, and we can write $g_1 = g_0 - c_1 f_{i_1} \in I_d$ for some c_1 cancelling $LM(g_0)$, so that $LM(g_1) < LM(g_0)$. Repeating this construction, we obtain a sequence $LM(g_0) > LM(g_1) > \ldots$ such that $LM(g_m) = 0$ for some m from the well-ordering property of the monomial order. So

$$g_0 - \sum_{j=1}^{m} c_j f_{i_j} = 0$$

and B spans I_d.

To finish the proof, we show that $LM(B)$ is a basis of $\langle LT(I) \rangle_d$. Clearly, any $LM(f_i) \in \langle LT(I) \rangle_d$, and the linear independence of $LM(B)$ comes from the fact that its monomials are all distinct. Also, $LM(B)$ spans $\langle LT(I) \rangle_d$. To see this, just take $f \in I$ with $\deg LM(f) = d$ and note that, because I is homogeneous, $f_{(d)}$, the homogeneous part of f of degree d, is in I. It follows that $f_{(d)} \in I_d$ and $LM(f) = LM(f_{(d)}) \in LM(B)$. $\square$

Corollary 35. *Let G be a finite subset of I. Then $HF_I(d) \leq HF_{\langle LT(G)\rangle}(d)$, with equality for all d exactly when G is a Gröbner Basis of I.*

Proof. To prove the inequality, just note that $\langle LT(G)\rangle \subseteq \langle LT(I)\rangle$. The equality follows from the definition of Gröbner Bases. $\square$

As a first example of an application of Hilbert series, the following example shows how they can be used to characterize a class of polynomial systems.

Example 36 (Regular sequences)**.** The notion of a randomly generated, or generic, system is often useful for the complexity analysis of Gröbner Basis algorithms. A certain class of systems, called *regular sequences* (see, for example, (BARDET, 2004)) is conjectured to be generic in this sense. We can define these systems using Hilbert Series. Let $\mathcal{F} = \{f_1, \ldots, f_m\}$ homogeneous with $d_i = \deg f_i$. Then $\mathcal{F}$ is regular if

$$HS_{\langle \mathcal{F}\rangle}(t) = \frac{\prod_{i=1}^{m}(1-t^{d_i})}{(1-t)^n}.$$

Note that when $n = m$ and none of the f_i is constant, this series is necessarily a polynomial, as $(1-t)$ appears at least once per f_i in the factorization of the numerator.

Theorem 37. *The Hilbert Series of I can be written as*

$$HS_I(t) = \frac{HN_I(t)}{(1-t)^n}$$

for some $HN_I(t) \in \mathbb{Z}[t]$. We will call HN_I the Hilbert numerator *of I.*

Proof. See (COX; LITTLE; O'SHEA, 2015), Chapter 10, §2, Theorem 4. $\square$

These results generalize to non-homogeneous ideals with few changes. In this case, one works instead with $R_{\leq d} = \bigoplus_{0\leq i\leq d} R_i$ and similarly defined $I_{\leq d}$ and $S_{\leq d}$. The affine Hilbert Function is then given by ${}^aHF_I(d) = \dim_k S_{\leq d}$ and the previous theorems hold, except that Theorem 34 requires a graded monomial order. The following result relates the homogeneous and affine cases.

Theorem 38. *Let I be an ideal and I^h be its homogenization with respect to a variable h. Then*

$${}^aHF_I(d) = HF_{I^h}(d)$$

Proof. See (COX; LITTLE; O'SHEA, 2015), Chapter 9, §3, Theorem 12. □

Finally, one can also show that the Hilbert Function is essentially a polynomial, in the sense of the proposition below.

Proposition 39. *Let I be an homogeneous ideal. For sufficiently large t, its Hilbert Function is*

$$HF_I(t) = \sum_{i=0}^{s} b_i \binom{t}{s-i}$$

for some $b_i \in \mathbb{Z}$ and the polynomial on the right-hand side is called the Hilbert Polynomial *of I, denoted HP_I.*

The same result holds for affine ideals, and thus the Hilbert Polynomial may be similarly defined in the general case. The smallest t_0 such that $HF_I(t) = HP_I(t)$ for all $t \geq t_0$ is called the *index of regularity* of I, denoted $i_{reg}(I)$. It can be used to measure the complexity of Gröbner Basis computations over I, as (LAZARD, 1983) has shown that it is an upper bound for the degree of Gröbner Basis polynomials of regular systems for the $grevlex$ order after a generic change of variables. The index of regularity is also an invariant of I, that is, it is independent from the particular set of generators of I or the algorithm used to compute its Gröbner Basis. For more details on complexity, see Section 2.8. There, we will need the following result, obtained by Macaulay (MACAULAY, 1902).

Proposition 40 (Macaulay's bound)**.** *Let I be an ideal generated by $\mathcal{F} = \{f_1, \ldots, f_m\}$ with $\deg f_i = d_i$ for all $i \in [m]$. Then*

$$i_{reg}(I) \leq 1 + \sum_{i=1}^{m} (d_i - 1).$$

2.7 Change of ordering

The definition of a Gröbner Basis takes into account the choice of monomial order, and some results, such as the Elimination Theorem, depend on the properties of specific orders to hold, such as the lexicographical order. Unfortunately, Gröbner Basis computations in these orders are not always efficient, partly because other orders may have much smaller Gröbner Bases, both in terms of number of elements and total number of monomials of the basis. A classic observation (FAUGÈRE et al., 1993) is that the $grevlex$

order is usually much more efficient than lex, but other orders can be much more efficient for specific instances. Table 2.2 compares basis size and number of monomials in the basis with respect to the $grevlex$ order to some weight orders based on $grevlex$, chosen among 5000 randomly generated orders to minimize the number of arithmetic operations in the Gröbner Basis computation. Monomials are counted with repetition, that is, if a monomial appears in multiple polynomials, it is counted multiple times. See Appendix A for more complete information about the instances.

Table 2.2: Comparison between the size of basis and number of monomials in $grevlex$ and weight order Gröbner Bases.

	Grevlex		Weight order		
Instance	Size	Monomials	Weights	Size	Monomials
cyclicnh5	38	538	(6, 0, 3, 3, 2, 4)	10	135
cyclicnh6	99	3502	(7, 3, 5, 5, 5, 5, 5)	28	1207
cyclicnh7	443	79897	(1, 3, 6, 4, 4, 4, 4, 5)	134	25664
jason210	900	290098	(7, 3, 1, 2, 4, 3, 8, 4)	738	3513
hrand2_10	426	59514	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)	426	59514
katsuran10	272	98498	(8, 8, 4, 8, 10, 11, 11, 8, 9, 7)	244	94521

It is clear, from Table 2.2, that it is possible to obtain improvements in performance choosing an adequate weight order instead of $grevlex$ or lex for Gröbner Basis computations. This potentially requires algorithms for the conversion of Gröbner Bases between monomial orders, in order to keep the desired properties of, for example, the lex order for polynomial system solving. Algorithms for change of ordering and the structure of the Gröbner Bases of an ideal with respect to every monomial order will be the subject of the remainder of this section.

2.7.1 Change of ordering in dimension zero: FGLM

When the input ideal has dimension zero, that is, the input system has finite solutions, a simpler and more efficient algorithm exists to change ordering of a Gröbner Basis than in the general case. This is the FGLM algorithm, introduced in (FAUGÈRE et al., 1993).

We start by recalling that I is a zero-dimensional ideal if and only if the k-algebra $S = R/I$ is finite dimensional over k (for k algebraically closed), as the dimension of S essentially counts the number of solutions of the input system, with multiplicities. Each Gröbner Basis induces a basis $B = \{x^\alpha \mid x^\alpha \notin \langle LT(G)\rangle\}$ of S by Proposition 32, so

```
**Input:** $<_s$ a monomial order, $G_s$ a Gröbner Basis (with respect to $<_s$) of a
        zero-dimensional ideal $I$ over $R$, a target order $<_t$
**Output:** $G_t$, a Gröbner Basis of $I$ with respect to $<_t$
$B_t := \emptyset$
$G_t := \emptyset$
**repeat**
    $M := nextMonomial(<_t, G_t)$
    **if** $\overline{M}^{G_t}$ *is linearly dependent on* $B_t$ **then**
        Let $\overline{M}^{G_t} - \sum_{x^\alpha \in B_t} c_\alpha \overline{x^\alpha}^{G_t}$ be a dependence relation
        $g := M - \sum_{x^\alpha \in B_t} c_\alpha x^\alpha$
        $G_t := G_t \cup \{g\}$
    **else**
        $B_t := B_t \cup \{M\}$
    **end**
**until** $G_t$ *is a Gröbner Basis of* $I$
```

Algorithm 6: FGLM algorithm

a Gröbner Basis conversion algorithm is related to a change of basis of S to an initially unknown basis. The FGLM algorithm will compute this basis along with a Gröbner Basis in the target order, using linear algebra over S. Our presentation is based on (COX; LITTLE; O'SHEA, 2005), Chapter 2, §3.

Let $<_s$ be the source monomial order, that is, an order for which we have a Gröbner Basis G_s, and let $<_t$ be the target order, with G_t its Gröbner Basis. Also, denote B_t the basis of S induced by G_t. Algorithm 6 shows the FGLM algorithm. The idea is to proceed by processing monomials in increasing order with respect to $<_t$, building G_t and B_t incrementally. The function $nextMonomial$ returns the next monomial M with respect to $<_t$ that is not a multiple of any $LT(g)$ for $g \in G_t$, in increasing order. Then, if $\overline{M}^{G_t} - \sum_{x^\alpha \in B_t} c_\alpha \overline{x^\alpha}^{G_t} = 0$ is a linear dependence relation, $g = M - \sum_{x^\alpha \in B_t} c_\alpha x^\alpha \in I$ and we can add this polynomial to G_t. Note that $LM(G_t) = M$. If, however, linear independence happens in line 5, M can be added to the basis B_t. The algorithm terminates when G_t is a Gröbner Basis of I.

One can also formulate FGLM in terms of matrices, as in the original paper. This formulation allows for further optimizations as it is known that the matrices are usually sparse, and techniques have been proposed to build them efficiently (FAUGÈRE; MOU, 2011) and to exploit their sparsity to change the ordering with better complexity (FAUGÈRE; MOU, 2017).

2.7.2 The Gröbner Fan of an ideal

For any given ideal, distinct monomial orders determine, in general, different (reduced) Gröbner Bases. We will now describe the set of all reduced Gröbner Bases of an ideal. Our presentation is based on (COX; LITTLE; O'SHEA, 2005).

Definition 41. A *marked Gröbner Basis* of I is a reduced Gröbner Basis G of I (with respect to some ordering) with a distinguished term $t_g = LT(g)$ for every $g \in G$.

This definition is independent of specific choices of monomial order and is used to take into account the fact that I may have G as reduced Gröbner basis for two monomial orders that choose distinct leading terms on G. It holds that

Proposition 42. *The set of marked Gröbner Bases of any ideal I is finite.*

Proof. By (COX; LITTLE; O'SHEA, 2005), Chapter 8, §4, Theorem 4.1, the set

$$Mon(I) = \{\langle LT_>(I)\rangle \mid > \text{ is a monomial order}\}$$

is finite. We will show there is a bijection

$$\{\text{marked Gröbner Bases of } I\} \longleftrightarrow Mon(I).$$

To see this, note that given a marked Gröbner Basis $G = \{g_1, \dots, g_t\}$ of I with marked terms $M = \{x^{\alpha(1)}, \dots, x^{\alpha(t)}\}$ the ideal $J = \langle M\rangle \in Mon(I)$ has minimal basis M, as the $x^{\alpha(i)}$ cannot divide each other because G is reduced (with respect to some order).

Conversely, any $M = \{x^{\alpha(1)}, \dots, x^{\alpha(t)}\}$ that is a minimal basis of $J \in Mon(I)$ extends to a marked basis $G = \{g_1, \dots, g_t\}$ — it suffices to take any $g_i \in I$ with $LT(g_i) = x^{\alpha(i)}$ with respect to some order defining J and interreducing the g_i, as this process does not affect leading monomials. □

Let $G = \{g_1, \dots, g_t\}$ be a marked Gröbner Basis of I with $x^{\alpha(i)}$ the associated leading monomial of g_i, for all $i \in [t]$. Then we can write

$$g_i = x^{\alpha(i)} + \sum_{\beta} c_{i,\beta} x^{\beta}$$

for $i \in [t]$. The *Gröbner cone* of G is

$$C_G = \{w \in \mathbb{R}^n_+ \mid w \cdot (\alpha(i) - \beta) \geq 0 \text{ when } c_{i,\beta} \neq 0\}$$

and the collection of cones of I and with their faces (intersections with hyperplanes) is called the *Gröbner fan* of I. This concept was introduced in (MORA; ROBBIANO, 1988), which also includes an algorithm to compute a Gröbner fan with better performance than a naive search, but that is still impractical for even moderate inputs.

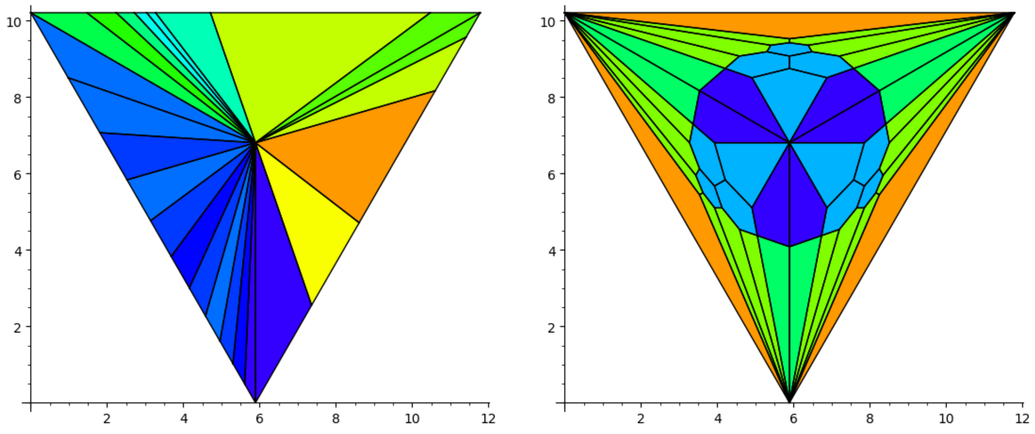
Note that if w is the first row of a matrix order $<_M$ and G is the marked Gröbner Basis with respect to this order, then $w \in C_G$ so there is a link between the cones and the weight orders. There is a partial converse to this: any matrix order with first row w in the interior (not on the boundary) of a cone C_G has G as its marked Gröbner Basis. Also, the Gröbner cones of an ideal partition the positive orthant $\mathbb{R}^n_+$, so the Gröbner fan essentially classifies the monomial orders into a finite number equivalence classes given by their marked Gröbner Bases.

Example 43. Let $R = \mathbb{F}_{32003}[x, y, z]$. Define I as an ideal of R generated by homogeneous binomials of the form

$$x^{\alpha_1}y^{\beta_1}z^{\gamma_1} - x^{\alpha_2}y^{\beta_2}z^{\gamma_2}$$

of degree 26, that is, $\alpha_1 + \beta_1 + \gamma_1 = \alpha_2 + \beta_2 + \gamma_2 = 26$. Also, define J as an ideal of R with random coefficients in $\mathbb{F}_{32003}$ — J is generic, as in Example 36. Let $Gf(I)$ and $Gf(J)$ be their respective Gröbner fans. Then Figure 2.1 shows the projection on $\mathbb{R}^2$ of the intersection of each of these Gröbner fans with the plane $x + y + z = 1$.

Figure 2.1: Gröbner fans as projections on $\mathbb{R}^2$ of their intersections with a plane in $\mathbb{R}^3$.



(a) The Gröbner fan of the binomial ideal I. (b) The Gröbner fan of the generic ideal J.

In the images, the upper right corner of the triangle is the intersection with the x-axis, the upper left with the y-axis and the bottom corner with the z-axis. These correspond to lexicographical monomial orders where the leading variable is respectively

x, y and z. The center of the triangle corresponds to a monomial order that can be represented as a matrix with first row $(1, 1, 1)$, such as $grevlex$. The color of a region C_G (that is actually a cone of the Gröbner fan) indicates the size of the reduced Gröbner Basis G with respect to a monomial order in the interior of the cone C_G — "hotter" colors, such as orange, correspond to the smallest bases while "cooler" colors, such as dark blue, correspond to the largest bases.

One can use the notion of Gröbner fan to develop an algorithm to change the ordering of an input Gröbner Basis, like FGLM, but without the restriction of zero-dimensionality of the input ideal. This algorithm is called a *Gröbner Walk* and was introduced in (COLLART; KALKBRENER; MALL, 1997). The main idea is walking from the cone defined by the input order $<_s$ to the cone containing the target order $<_t$ in piecewise linear paths, recomputing a Gröbner Basis with respect to intermediate orders every time the walk passes through another cone or boundary. These recomputations, however, are expected to be more efficient than a full Gröbner Basis computation, because they can be obtained from input systems with predictably fewer monomials. For more details on the Gröbner Walk algorithm, see (COX; LITTLE; O'SHEA, 2005), Chapter 8, §5.

2.8 Known complexity results

There are many known results on the complexity of the computation of Gröbner Basis, including upper bounds for certain algorithms and some particular families of instances, as well as some general, algorithm-independent results. This section covers some of the main known results in this topic.

The classic result is that computing the Gröbner Basis of an ideal has doubly exponential worst-case complexity. This fact is more precisely stated in the following theorem.

Theorem 44. *(MÖLLER; MORA, 1984) Let $I = \langle \{f_1, \ldots, f_m\} \rangle$ be an ideal over $R = k[x_1, \ldots, x_n]$ with $d = \max_{1 \leq i \leq m} \deg f_i$ and s be the degree of the Hilbert Polynomial of I. Then the maximum degree D of a polynomial appearing in a Gröbner Basis of I with respect to the lexicographic order is bounded by*

$$D \leq ((n+1)(d+1)+1)^{2^{s+1}(n+1)}.$$

This worst case is attained by some ideals. There are, however, good reasons to assume the $grevlex$ order is more efficient than lex for many cases. For example, it is known that at most m variables may appear in the leading terms of a (partial) reduced Gröbner Basis with respect to the $grevlex$ order (see, for example, (EISENBUD, 1995), Chapter 15, §7 for more details, or (BAYER; STILLMAN, 1987) for a complete characterization).

A simple, algorithm-dependent complexity result for the $grevlex$ order can be obtained by observing that, similarly to the F4 algorithm, one can obtain Gröbner Bases of homogeneous ideals by row reducing certain large matrices built from the product of certain monomials by the polynomials of the input ideal basis. By bounding the size and amount of matrices to be reduced, we obtain the following bound.

Theorem 45. *(BARDET; FAUGÈRE; SALVY, 2015) Let I be an homogeneous ideal. The number of field operations necessary to compute a Gröbner Basis of I with respect to a graded order up to degree D is bounded by*

$$O\left(mD\binom{n+D-1}{D}^{\omega}\right)$$

where $2 \leq \omega < 3$ is the exponent of the complexity of matrix multiplication.

When the input systems are generic, this result can be combined with the Macaulay bound on the index of regularity given by Proposition 40 to obtain an upper bound on D. In fact, many applications where the input systems are not generic tend to have lower maximum degrees than the Macaulay bound, so it usually also holds for many non-regular inputs.

If the input ideal is zero-dimensional, one can use the FGLM algorithm, combined with the worst-case complexity of the computation in the $grevlex$ order to show that a Gröbner Basis with respect to any monomial ordering can be computed in simply exponential time. More precisely,

Theorem 46. *Let I be a zero-dimensional ideal generated by $\mathcal{F} = \{f_1, \ldots, f_m\}$, with $d = \max_{1\leq i\leq m} \deg f_i$. Then:*

1. *a Gröbner Basis of I with respect to $grevlex$ can be computed in polynomial time in d^{n^2}. If the number of solutions at infinity is also finite ($\dim I^h = 0$) then the $grevlex$ Gröbner Basis can be computed in polynomial time in d^n.*
2. *FGLM runs in $O(nD^3)$, where D is the* degree *of I, the dimension of the k-algebra R/I. So a Gröbner Basis of I with respect to any monomial order can be computed by FGLM in polynomial time in d^{n^2} or d^n, as in the $grevlex$ order above.*

More optimized variants of the FGLM algorithm are known with improved worst-case complexity. For example, the Wiedemann-based FGLM introduced in (FAUGÈRE; MOU, 2017) runs probabilistically in $O(D(N_1 + n\log D^2))$, where N_1 is a sparsity parameter of the FGLM matrices, and the generic algorithm of (FAUGÈRE; MOU, 2017) computes a change of ordering of regular systems in $O\left(\sqrt{\frac{6}{\pi n}}D^{2+\frac{n-1}{n}}\right)$.

In (BARDET; FAUGÈRE; SALVY, 2015), a tighter bound is proved for the MatrixF5 algorithm, a simple variant of F5 using matrix-based reduction, but few other optimizations (see, for example, (EDER; FAUGÈRE, 2017) for a description of MatrixF5). In particular, a formula is obtained to approximate the number of field arithmetic operations required for the computation of a Gröbner Basis of a regular system is obtained. The Main Theorem giving the result is the following:

Theorem 47. *(BARDET; FAUGÈRE; SALVY, 2015) Let $\mathcal{F} = \{f_1, \dots, f_m\}$ be a regular system with $\delta = \max_{1\le i\le m} \deg f_i$, $\delta \ge 2$, $m = n - l$ for some $l \in \mathbb{N}$. Then the number of field operations of MatrixF5 over $\mathcal{F}$ behaves asymptotically as*

$$B(\delta)^n n\left(A(\delta, l) + O\left(\frac{1}{n}\right)\right), n \to \infty$$

where

$$B(\delta) = \frac{(\frac{\lambda_0+1}{\lambda_0})^{2\delta} - 1}{\frac{1}{\lambda_0^2} - \frac{1}{(\lambda_0+1)^2}} \qquad\qquad A(\delta, l) = \frac{1-\delta^{-1}}{2\pi}\frac{(1+\lambda_0^{-1})^3 - 1}{(1+\lambda_0)^{1+l}}$$

and λ_0 is the unique positive root in the interval $(\frac{\delta-1}{2}, \delta - 1)$ of

$$\left(\frac{\lambda+1}{\lambda}\right)^{2\delta} = \frac{1}{1 - \delta\frac{(\lambda+1)^2-\lambda^2}{(\lambda+1)^3-\lambda^3}}.$$

Also, $\delta^3 \le B(\delta) \le 3\delta^3$.

In some applications, the practical success of Gröbner Basis computations as a method for polynomial system solving also has a theoretical explanation based on its complexity in these specific cases. One particularly interesting case is that of the HFE systems from cryptography, defined in (PATARIN, 1996), that was successfully attacked by a variant of F5 in (FAUGÈRE; JOUX, 2003). In (BARDET; FAUGÈRE; SALVY, 2003), it was empirically shown that the index of regularity of HFE systems grows much more slowly than that of (semi-)regular systems, and so not even the simply exponential complexity bounds described above are tight.

3 DYNAMIC ALGORITHMS FOR GRÖBNER BASES

Classical Gröbner Basis algorithms take a monomial order as input, in addition to a polynomial system. Usually, in practice, the $grevlex$ ordering is recommended, but, as we have shown in Table 2.2, this choice is sometimes not advisable in running time or size of the output basis. However, as we will show, one cannot choose well an ordering a priori just analyzing the input polynomial system. This has led to the development of *dynamic* Gröbner Basis algorithms, that return a monomial order along with a Gröbner Basis, without requiring an order as input. In this context, classical Gröbner Basis algorithms that require a monomial order as input and return a Gröbner Basis with respect to that ordering are called *static*.

Dynamic algorithms are still relatively unexplored and the literature on them is limited. Below, we briefly state the main contributions of all works that, to our knowledge, have dealt with this topic.

1. (GRITZMANN; STURMFELS, 1993) introduced the idea of a dynamic Buchberger algorithm and the Hilbert heuristic, along with the first (and only) unrestricted dynamic algorithm (see Section 3.1).
2. (CABOARA, 1993) presented a restricted dynamic algorithm (see section 3.2) along with the first computational results of dynamic algorithms.
3. (GOLUBITSKY, 2006) defined the distinction between unrestricted and restricted algorithms and showed that unrestricted algorithms could obtain smaller bases than restricted ones.
4. (CABOARA; PERRY, 2014) developed the disjoint cones and boundary vectors criteria to optimize the restricted algorithm.
5. (HASHEMI; TALAASHRAFI, 2016) first implemented the exact boundary vectors criterion.
6. (PERRY, 2017) explored alternative heuristics and variants of the restricted algorithm.

In short, dynamic algorithms are based (and compatible with) classical static algorithms such as Buchberger's algorithm and F4, the main difference being that after each reduction step, a new monomial order may be chosen to continue the computations. This choice of new order is done heuristically, and will be detailed later.

In the following, we will present the unrestricted (section 3.1) and restricted (section 3.2) versions of the dynamic Gröbner Bases algorithms.

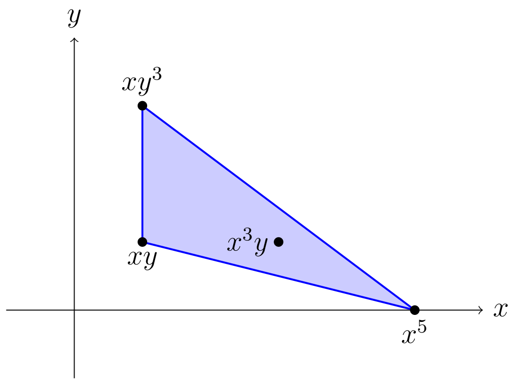
3.1 Unrestricted algorithms

A polynomial set G being a Gröbner Basis with respect to some ordering is directly related to the leading monomials of G chosen by the ordering. In fact, one can ask what are the equivalence classes of orderings over a polynomial set G, where two orderings are considered equivalent if they choose the same leading monomials. If one could compute this set of equivalence classes, it would be possible to evaluate each of them with respect to some heuristic function and choose the one minimizing the heuristic as a promising ordering. This is the idea behind the *unrestricted* dynamic algorithms, that evaluate all distinguishable orders on a polynomial set at each iteration of the Gröbner Basis computation. It turns out that this set of equivalence classes has a particular geometrical interpretation as a certain polyhedron in Euclidean space, the *Newton polyhedron* of G. We will briefly present the construction of this polyhedron.

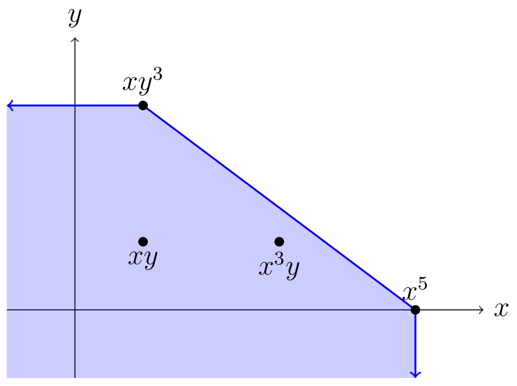
Definition 48. The *Newton polytope* of a polynomial $f \in R$ is the convex hull of the points in $\mathbb{R}^n$ determined by the exponents of its monomials. It is denoted by $np(f)$.

Example 49. The Newton polytope of $f = xy + x^5 + xy^3 + x^3y \in k[x, y]$ is the convex hull of the points $(1, 1), (5, 0), (1, 3), (3, 1)$ in $\mathbb{R}^2$. It is shown in Figure 3.1a. Note that each vertex of this convex hull corresponds to a monomial of f, but the converse is false, as the point $(3, 1)$, corresponding to the monomial x^3y, is in the interior of the polytope.

Figure 3.1: Newton polytope and affine Newton polyhedron of $f = xy + x^5 + xy^3 + x^3y$.



(a) Newton polytope of f.



(b) Newton polyhedron of f.

In order to obtain a polyhedron corresponding to a set of polynomials, we will need to join together the polytopes of multiple polynomials. The operation that allows this is the Minkowski sum.

Definition 50. The *Minkowski sum* of two polyhedra P, Q over $\mathbb{R}^n$ is the polyhedron

$$P + Q = \{x + y \in \mathbb{R}^n \mid x \in P, y \in Q\}.$$

Minkowski sums may be computed simply as convex hulls of all possible sums of vertices of their summands. Other, more efficient algorithms exist, for example (FUKUDA, 2004), although the number of vertices of the result, and thus the complexity of the algorithm, may still be exponential in the dimension of the space.

Definition 51. The *affine Newton polyhedron* of a polynomial $f \in R$ is the Minkowski sum of its Newton polytope with the negative orthant of $\mathbb{R}^n$, that is,

$$NP(f) = np(f) + \mathbb{R}^n_-.$$

The relation between Newton polyhedra and monomial orders will be given by the normal vectors at the vertices of the polyhedra. To make this more precise, we now define the outward normals of a polyhedron at a point.

Definition 52. Let P be a polyhedron in $\mathbb{R}^n$ and $x \in P$. Then $w \in \mathbb{R}^n$ is an *outward normal* (or simply a *normal*) of P at x if

$$w \cdot x \geq w \cdot y \quad \forall y \in P.$$

Example 53. The affine Newton polyhedron of the polynomial of Example 49 is shown in Figure 3.1b. Note that all of its vertices have normals in the positive orthant of $\mathbb{R}^n$, and that these vertices form a subset of the vertices of the Newton polytope of Figure 3.1a.

The vertices of the affine Newton polyhedron of a polynomial correspond exactly to the monomials that can be its leading monomials with respect to some ordering. Any normal vector to one of its vertices determines the first row of a valid ordering choosing the corresponding monomial as lead. We now proceed to define the Newton polyhedron corresponding to a polynomial set, instead of a single polynomial.

Definition 54. Let $G \subset R$ be a finite set of polynomials. The affine Newton polyhedron of G is the Minkowski sum of the polyhedra of its elements, that is,

$$NP(G) = \sum_{g \in G} NP(g) = \mathbb{R}^n_- + \sum_{g \in G} np(g).$$

The vertices of the affine Newton polyhedron of a polynomial set G are in bijection with the compatible choices of leading monomials for G. Each vertex v of this Minkowski sum decomposes uniquely into vertices of its summands, that correspond to the leading monomials chosen when one picks v from the sum. A weight vector corresponding to an ordering making this choice of leading monomials can be computed as a normal vector to v in the affine Newton polyhedron. With this, we have the unrestricted dynamic algorithm, as introduced by (GRITZMANN; STURMFELS, 1993), in Algorithm 7. It can be called from Buchberger's algorithm every time a new polynomial is added to a partial Gröbner Basis G. Note, however, that whenever the leading monomials of polynomials in G change, the queue of remaining S-polynomials has to be rebuilt, as S-polynomials that were previously considered useless may not reduce to zero in the new ordering.

```
Input: $G = \{g_1, \dots, g_m\} \subset R$, a heuristic function $H$
Output: A monomial ordering $w$ over $G$
Compute the set $V$ of vertices of $NP(G)$
$v_{min} = \min_{v \in V} H(v)$
Compute a normal vector $w$ of $v_{min}$ in $NP(G)$
```

Algorithm 7: The "dynamic engine" of the unrestricted algorithm of (GRITZMANN; STURMFELS, 1993).

The unrestricted algorithm has some interesting properties. Firstly, if, during the execution of the algorithm, one arrives at a partial Gröbner Basis G that is a Gröbner Basis with respect to *some* monomial order, the algorithm is able to detect this property (with an adequate heuristic, such as the *Hilbert heuristic*, introduced in section 3.3) and returns such a monomial order. In theory, this could end the execution of a Gröbner Basis algorithm much sooner. Also, (GOLUBITSKY, 2006) has shown that there are cases where the unrestricted algorithm returns a smaller basis than the restricted algorithms from section 3.2. Originally, (GRITZMANN; STURMFELS, 1993) proved that, when the number of variables is fixed, such as during the execution of the algorithm, the number of vertices in $NP(G)$ can only grow polynomially. In practice, however, this is not enough to guarantee the good performance of the algorithm — even for relatively small examples, this polynomial growth is of high degree and the number of vertices of $NP(G)$ quickly becomes very large, making the unrestricted dynamic algorithm much slower than the static algorithm.

3.2 Restricted algorithms

The unrestricted dynamic algorithm has two major flaws in practice - it must evaluate a potentially very large number of orderings and, by changing previously chosen leading monomials, it may generate many new S-polynomials in the queue of the Gröbner Basis computation. *Restricted* dynamic Gröbner Basis algorithms deal with these problems by choosing new orderings without changing previous choices of leading monomials, reducing both the number of viable orders to evaluate and eliminating the need to recompute S-polynomials. The trade-off is that potentially good orders may be lost, as a heuristic choice that appeared advisable at the beginning of the execution may be found later to be unwise, but it cannot be changed.

The basic restricted algorithm introduced in (CABOARA, 1993) uses linear programming to keep track of the previous choices of leading monomials and to ensure their compatibility. For each polynomial f with leading monomial t, one adds the linear constraints

$$w \cdot t > w \cdot u \quad \forall u \in \mathrm{Supp}(f) \setminus \{t\}$$

where w is a vector of positive real variables representing the monomial order. Geometrically, this determines an open cone with apex at the origin, and any feasible w chooses the correct leading monomials. Further choices of leading monomials add more linear constraints of this form, and thus “narrow” the cone. That is the reason the restricted dynamic algorithm is sometimes said to be a *narrowing cone algorithm*. We will denote the cone determined by choosing t as leading monomial by C_t, and the cone determined by choosing a set T of leading monomials by C_T.

Whenever a new polynomial is added to the basis, one must choose a leading monomial for it that is compatible with previous choices. To do that, one may simply add the linear constraints corresponding to each choice and verify which ones have a nonempty feasible region. If there are multiple feasible candidates, they can be compared with respect to some heuristic function, like in the unrestricted case, and the one optimizing the heuristic value will be chosen. Its corresponding linear constraints are then permanently added to the linear programming model.

This algorithm can also be understood in terms of Minkowski sums: if, at any given step, we have the partial Gröbner Basis G and the currently chosen order corresponds to the vertex v of the affine Newton polyhedron P of G, upon inserting a new polynomial g in G, the restricted algorithm will only look at the vertices of $P + np(g)$

that can be written as sums of v with some vertex of $np(g)$.

There are many optimizations that can be made to the restricted algorithm. In the same paper where it was introduced, Caboara proposed two of them, based on the following propositions.

Proposition 55. *If t, m are monomials and m divides t, with $t \neq m$, then $t > m$ in any monomial order.*

We call a monomial t of $f \in R$ a *potential leading monomial* of f with respect to a set $F = \{f_1, \dots, f_m\}$ with leading monomials $t_1, \dots, t_m$ if there exists a monomial ordering choosing $t, t_1, \dots, t_m$ as the leading monomials of their respective polynomials. Alternatively, t is a potential leading monomial with respect to F if the exponent vector of $t \prod_{i=1}^{m} t_i$ is a vertex of $NP(\{f\} \cup F)$. Then:

Proposition 56. *Let $f_1, \dots, f_m$ be a polynomial system with respective leading monomials $t_1, \dots t_{m-1}$, and t_m be a potential leading monomial of f_m with respect to $\{f_1, \dots, f_{m-1}\}$. Define*

$$M = \{t \in \operatorname{Supp}(f_m) \mid \text{ t is a potential leading monomial w.r.t. } f_1, \dots, f_{m-1}\}.$$

Then

$$t_m > t \quad \forall t \in \operatorname{Supp}(f_m) \setminus \{t_m\}$$

is equivalent to

$$t_m > t \quad \forall t \in M.$$

In short, one may ignore monomials that strictly divide other monomials in the same polynomial and only add one linear constraint for every potential leading monomial of a polynomial, ignoring monomials that are known to be incompatible with previous choices of leading monomials. Although these optimizations greatly reduce the number of constraints of the linear programming model, (CABOARA; PERRY, 2014) showed it is possible to optimize even further, using the disjoint cones criterion and the boundary vectors criterion.

Proposition 57 (Disjoint cones criterion)**.** *If T is the set of chosen leading monomials of G, g is a polynomial being inserted in G and $t \in \operatorname{Supp}(g)$ is incompatible with t, then any monomial m such that $C_m \subseteq C_t$ is incompatible with T as well.*

In practice, we can record the linear constraints that would be inconsistent with T and ignore any monomials that would add a subset of these constraints later during the execution of the algorithm. This technique reduces the number of times one has to find a feasible solution of a linear program by reducing the amount of potential leading monomials.

Definition 58. A d-boundary vector of a cone C is an extreme point of the intersection of the closure of C with the hyperplane $\sum_{i=1}^{n} y_i = d$ in $\mathbb{R}^n$.

In theory, the choice of d in the definition of boundary vectors does not matter, as the cones appearing in the execution of the restricted algorithm have their apex at the origin. From an implementation perspective, sometimes it is useful to work with integers, and to do so one must choose d to be potentially large.

Proposition 59 (Boundary vectors criterion)**.** *Let T be the set of leading monomials of G (with respect to some monomial order w) and Ω be the set of boundary vectors of C_T. If t is the leading monomial of g with respect to w and $u \in \operatorname{Supp}(g)$, u is incompatible with T if $\omega \cdot t > \omega \cdot u$ for every $\omega \in \Omega$.*

The boundary vectors criterion reduces the number of potential leading monomials even further, leading to fewer calls to a linear programming solver. If, furthermore, we add linear constraints only corresponding to potential leading monomials as detected by the boundary vectors criterion, we can also reduce the number of constraints.

Boundary vectors can be computed approximately or exactly: in (CABOARA; PERRY, 2014), they are computed approximately, while (HASHEMI; TALAASHRAFI, 2016) and (PERRY, 2017) also provide experimental results for the exact computation. The experimental results of (PERRY, 2017) suggest that neither strictly dominates the other in terms of time or output basis size, and that in the case of some instances, computing boundary vectors exactly can become very expensive.

3.3 Heuristic functions

All dynamic algorithms proposed until now depend on the use of a heuristic function to determine whether taking a certain monomial order is advisable or not. These algorithms do not depend on the properties of the chosen heuristic for their correctness, and thus may use any heuristic function. We present heuristics that were used in previous works in Subsection 3.3.1 and 3.3.2 and introduce a new heuristic, based on the previous ones in Subsection 3.3.3.

3.3.1 The Hilbert heuristic

The most common heuristic for dynamic algorithms is the *Hilbert heuristic*, first proposed in (GRITZMANN; STURMFELS, 1993). For a given ordering $<$ and a partial basis G, it computes the Hilbert series or the Hilbert polynomial of $\langle LT_<(G)\rangle$. The idea behind the use of the Hilbert function as a heuristic is that it measures how close a set G is to being a Gröbner Basis — it is non-increasing during the execution of a Gröbner Basis algorithm and G is a Gröbner Basis if and only if its Hilbert function coincides with that of $I = \langle G\rangle$. In the following, we denote by HS, HP, HN the Hilbert series, polynomial and numerator of an ideal, respectively.

The comparison between Hilbert series or polynomials may be done in various ways. (GRITZMANN; STURMFELS, 1993) showed that, for homogeneous inputs, computing the first $2D$ coefficients of the Hilbert series is enough to distinguish between orders, where D is the maximum of the degrees of the elements of G. This suggests the following comparison:

Definition 60 (Hilbert heuristic, 1)**.** (GRITZMANN; STURMFELS, 1993) Let $<_1$ and $<_2$ be monomial orders and $G \subset R$ be a finite set of polynomials of degree at most D. Then $<_1$ is preferable to $<_2$ if the first (in increasing degree order) nonzero entry of

$$HS(\langle LM_{<_1}(G)\rangle) - HS(\langle LM_{<_2}(G)\rangle)$$

is negative. Also, one may compare only the first $2D$ coefficients.

The Hilbert polynomial coincides with the Hilbert function in high enough degree and, roughly, represents the growth of the number of monomials not in an ideal. A heuristic can then be designed based primarily on the Hilbert polynomial and, in particular, the degree and coefficient of its leading terms. Comparing Hilbert polynomials and breaking ties by the numerator HN of the Hilbert series is sometimes used as the definition of the Hilbert heuristic. More precisely, we have:

Definition 61 (Hilbert heuristic, 2)**.** (PERRY, 2017) Let $<_1$ and $<_2$ be monomial orders, $G \subset R$ be a finite set of polynomials, $h_1 = HN(\langle LM_{<_1}(G)\rangle)$, $h_2 = HN(\langle LM_{<_2}(G)\rangle)$, $p_1 = HP(\langle LM_{<_1}(G)\rangle)$ and $p_2 = HP(\langle LM_{<_2}(G)\rangle)$. Then $<_1$ is preferable to $<_2$ if the leading term of $p_1 - p_2$ is negative or $p_1 = p_2$ and the trailing term of $h_1 - h_2$ is positive.

More simply, in order to reduce the overhead of the computation of the heuristic function, we can accept ties between Hilbert polynomials as in the following heuristic, which is a simplification of Definition 61.

Definition 62 (Hilbert heuristic, 3)**.** Let $<_1$ and $<_2$ be monomial orders, $G \subset R$ be a finite set of polynomials and $p_1 = HP(\langle LM_{<_1}(G)\rangle)$, $p_2 = HP(\langle LM_{<_2}(G)\rangle)$. Then $<_1$ is preferable to $<_2$ if $\deg p_1 < p_2$ or if $\deg p_1 = \deg p_2$ and $LC(p_1) < LC(p_2)$.

Note that using this definition of Hilbert heuristic, ties may still happen, as two distinct monomial orders may have the same Hilbert polynomial. The tiebreaking criterion will usually be, in this case, to keep the current ordering.

(PERRY, 2017) proposed, as an alternative to the Hilbert heuristic above, the use of the *graded* Hilbert heuristic. It is obtained by replacing the standard grading of R by the grading given by a weight vector w. More precisely, the w-degree of a polynomial $f \in R$ is

$$\deg_w(f) = \max\{w \cdot \log t \mid t \in \operatorname{Supp}(f)\}$$

and the w-graded part of R is the k-vector space

$$R_d = \{f \in R \mid \deg_w f = d\}$$

and $I_d = R_d \cap I$. Then the w-graded Hilbert function of I is

$$grHF_I(d) = \dim_k \frac{R_d}{I_d}.$$

Similarly to the case of the standard grading, one can compute the w-graded Hilbert numerator $grHN_w$ of a monomial ideal I. The degree of the Hilbert polynomial in the standard grading measures the dimension of I, and this is a useful measure regardless of grading. For this reason, the graded Hilbert heuristic uses the Hilbert polynomial in the standard grading and, additionally, breaks ties using the graded Hilbert numerator. Precisely, we define the graded Hilbert heuristic in the following way:

Definition 63 (Graded Hilbert heuristic)**.** Let $<_1$ and $<_2$ be monomial orders compatible with weight vectors w_1, w_2 respectively, $G \subset R$ be a finite set of polynomials, $h_1 = grHN_{w_1}(\langle LM_{<_1}(G)\rangle)$, $h_2 = grHN_{w_2}(\langle LM_{<_2}(G)\rangle)$, $p_1 = HP(\langle LM_{<_1}(G)\rangle)$ and $p_2 = HP(\langle LM_{<_2}(G)\rangle)$. Then $<_1$ is preferable to $<_2$ if the leading term of $p_1 - p_2$ is negative or $p_1 = p_2$ and the trailing term of $h_1 - h_2$ is positive.

In a dynamic Gröbner Basis algorithm, this heuristic should represent the behavior of monomial orders more accurately, as it directly uses the grading determined by the current ordering. Unfortunately, (PERRY, 2017) reports that, in practice, the performance of this graded Hilbert heuristic is worse than the ungraded one, although no theoretical explanation for that is known.

3.3.2 The Betti heuristic

An alternative to Hilbert function based heuristics, proposed by (PERRY, 2017), is to try to minimize the amount of remaining S-polynomials to be computed. This is algorithm-based, because the size of the queue depends on the criteria to eliminate useless S-polynomials. In fact, one could use no such criteria at all, and in this case every choice of leading monomial would be equally advisable according to the Betti heuristic. In good implementations of Buchberger's algorithm, however, the Gebauer-Möller update algorithm (GEBAUER; MÖLLER, 1988) is usually used to manage the queue. This algorithm tries to compute the smallest possible queue for the monomial order, and this minimal queue size is directly related to the first Betti number, of the input ideal, defined in Section 3.3.2. In general, one can use any of the techniques described in Section 2.5 to compute, approximately or, in the case of Bayer's criterion, exactly, the first Betti number of $LM_<(G)$ for candidate monomial orders $<$. (PERRY, 2017) uses the following Betti heuristic:

Definition 64 (Betti heuristic)**.** Let $<_1$ and $<_2$ be monomial orders, $G \subset R$ be a finite set and $g \in R$ be a polynomial being added to G by Buchberger's algorithm. Then $<_1$ is preferable to $<_2$ according to the Betti heuristic if fewer S-polynomials are added to the queue in order $<_1$ than $<_2$ using the Gebauer-Möller update algorithm. This comparison is done degree by degree, from smallest to largest.

Note that this has to be slightly modified to hold for unrestricted dynamic algorithms, as in this case the S-polynomial queue is reconstructed. We can, then, compare the sizes of the reconstructed queues.

According to (PERRY, 2017), the Betti heuristic is a good alternative in practice to the ungraded Hilbert heuristic. In his implementation of the restricted dynamic algorithm, the Betti heuristic is occasionally more efficient and, sometimes, less efficient than the Hilbert heuristic, depending on the instance. The precise results of the Betti heuristic were not reported.

(PERRY, 2017) also proposed a graded version of the Betti heuristic by extrapolating reasonable behavior from the ungraded case. This approach looks unpromising, however, and led to bad performance in practice.

3.3.3 Mixed heuristic

We now introduce a new heuristic based on both the Hilbert and the Betti heuristics. On one hand, the degree of the Hilbert polynomial is a good intuitive measure of how close one is to obtaining a Gröbner Basis of an ideal I, but on the other hand, this value usually has a small range, and often leads to many ties as a heuristic — if one is computing a Gröbner Basis of an ideal of dimension d over a polynomial ring in n variables, only the values from d to n can appear as the Hilbert degree during the execution of the algorithm.

The Betti heuristic has the desirable property of reducing the amount of S-polynomials to be computed in the short term (and, possibly, in the long term as well). This seems particularly useful when dealing with an unrestricted dynamic algorithm, that may have to rebuild the entire queue of S-polynomials multiple times during the execution of the algorithms.

The advantages of each of these heuristics can be complementary, as it always seems a good idea to pick monomial orders minimizing the degree of the Hilbert polynomial, while when choosing between orders that lead to the same Hilbert degree, it would seem advisable to minimize the amount of S-polynomials to be computed instead. This is the *Mixed heuristic*.

Definition 65 (Mixed heuristic)**.** Let $<_1$ and $<_2$ be monomial orders, $G \subset R$ be a finite set and $p_1 = HP(\langle LM_{<_1}(G)\rangle)$ and $p_2 = HP(\langle LM_{<_2}(G)\rangle)$. Then $<_1$ is preferable to $<_2$ according to the Mixed heuristic if $\deg p_1 < \deg p_2$ or if $\deg p_1 = \deg p_2$ and $<_1$ is preferable to $<_2$ according to the Betti heuristic.

4 UNRESTRICTED DYNAMIC GRÖBNER BASIS ALGORITHMS

The original unrestricted dynamic algorithm described in Section 3.1 and introduced by (GRITZMANN; STURMFELS, 1993) quickly becomes impractical even for some small instances due to the complexity of computing the vertices of the Minkowski sums involved. In order to propose more viable unrestricted algorithms, it is necessary to work without computing the entire Newton polyhedron of a set of polynomials. In a way, that is what the restricted algorithms of Section 3.2 do, but they consider only vertices that have a previously computed vertex v as summand, thus heavily restricting the choices of monomial order.

For a simple example where the issues with restricted algorithms are easily seen, suppose that the first polynomial processed by a restricted dynamic algorithm is $\sum_{i=1}^{n} x_i$. Then, during its first iteration, the restricted algorithm has to choose one of the variables x_i to be the leading monomial of this polynomial. However, at this point, the algorithm has no information to make this decision, and all variables will tie when compared with respect to the Hilbert or Betti heuristics. Whatever choice the algorithm makes may be seen to be inadvisable later on during the execution, but then the leading monomial is already fixed and cannot be changed anymore, because the algorithm is restricted. Unrestricted algorithms have no such problems, as even if they make some bad choices of leading monomials early on, these choices can be fixed in further iterations.

In the remainder of this chapter, we describe a simple neighborhood structure for monomial orders based on the vertices of the Newton polyhedron (Section 4.1) and develop unrestricted dynamic algorithms applying this structure (Sections 4.2, 4.3, 4.4) that will be evaluated experimentally in Chapter 5.

4.1 Neighborhoods of monomial orders

In this section, we will describe precisely what the neighbors of a vertex v in a Minkowski sum P represent, where two vertices u, v are neighbors if $\mathrm{Conv}\{u, v\}$ (the convex hull of u and v) is an edge of P. This will allow us to establish relations between the heuristic values (with respect to either the ungraded Hilbert or Betti heuristics) of vertices that are "close" in P. Propositions 67 and 69 appear in previous works on Minkowski sums (for example, (WEIBEL, 2007)) outside the context of Newton polyhedra, but, Proposition 69 is usually not emphasized. For this reason, and because we could not find a proof of it, we provide one here. We start by giving a precise definition of a *face* of a polyhedron.

Definition 66. Let P be a polyhedron (in $\mathbb{R}^n$) and $w \in \mathbb{R}^n$. Then the face of P determined by w is

$$P_w = \{x \in P \mid w \cdot x \geq w \cdot y \, \forall y \in P\}$$

and w is a *normal* of P_w.

Note that P is a face of itself, as $P = P_0$. We will also consider that the empty face $\emptyset$ is a face of every polyhedron P.

Through the rest of this section, we will suppose that the face P_w exists, that is, that P is bounded in the direction of w. This holds in the case of Newton polyhedra when $w > 0$, which is exactly the relevant case for our application to monomial orderings.

We can relate the faces of a Minkowski sum to faces of its summands using the following proposition. It gives a precise meaning to our observation in Section 3.1 that a vertex of a Newton polyhedron of a polynomial set G corresponds to a choice of leading monomial for each $g \in G$.

Proposition 67. *Let $P_1, \dots, P_k$ be polyhedra and $P = \sum_{i=1}^k P_i$ be their Minkowski sum. Then, for any $w \in \mathbb{R}^n$,*

$$P_w = \sum_{i=1}^{k} (P_i)_w.$$

Proof. We will prove this for $k = 2$, as the general case follows by induction. First, let $x_1 \in (P_1)_w, x_2 \in (P_2)_w$. Then, for any $y_1 \in (P_1)_w, y_2 \in (P_2)_w$, we have

$$\begin{aligned} w \cdot x_1 &\geq w \cdot y_1 \\ w \cdot x_2 &\geq w \cdot y_2 \\ \implies w \cdot (x_1 + x_2) &\geq w \cdot (y_1 + y_2) \end{aligned}$$

so $(x_1 + x_2) \in P_w$.

To prove the converse, let, for any polyhedron Q,

$$a_Q(w) = \max\{w \cdot x \mid x \in Q\}$$

and note that

$$a_P(w) = a_{P_1}(w) + a_{P_2}(w).$$

Then, if $(x_1 + x_2) \in P_w$ with $x_1 \in P_1, x_2 \in P_2$ we have

$$a_P(w) = a_{P_1}(w) + a_{P_2}(w) = w \cdot (x_1 + x_2) = w \cdot x_1 + w \cdot x_2$$

so if $x_1 \notin (P_1)_w$, there would exist $y_1 \in P_1$ with $w \cdot y_1 > w \cdot x_1$ and $w \cdot (y_1 + x_2) > a_P(w)$ a contradiction. Thus $x_1 \in (P_1)_w$, and similarly for x_2. □

As we are primarily interested in understanding vertices (faces of dimension 0) and edges (faces of dimension 1) it is useful to have a precise definition of the dimension of a polyhedron P. To do this, we will define an *affinely independent set* in P as a set of points $x_1, \dots, x_l$ of P such that

$$\lambda_1 x_1 + \cdots + \lambda_l x_l = \mu_1 x_1 + \cdots + \mu_l x_l \implies \lambda_i = \mu_i \quad \forall i \in [l]$$

where $\sum_{i=1}^{l} \lambda_i = \sum_{i=1}^{l} \mu_i = 1$. We will also define a chain of faces of P as a collection of distinct faces $F_1, F_2, \dots, F_l$ of P such that $F_1 \subset F_2 \subset \dots \subset F_l$.

Definition 68. The *dimension* of a polyhedron P is the size of a maximal affinely independent set in P minus one.

Equivalently, the dimension of P is the length of a maximal chain of faces in P minus one.

The dimension of a Minkowski sum (and its faces) is related to the dimension of its summands by the following proposition.

Proposition 69. *If $P_1, \dots, P_k$ are polyhedra and $P = \sum_{i=1}^{k} P_i$, then*

$$\max\{\dim P_i \mid i \in [k]\} \leq \dim P \leq \sum_{i=1}^{k} \dim P_i.$$

Proof. We will prove this for $k = 2$, as the rest follows by induction. Furthermore, without loss of generality, suppose $\dim P_1 = \max\{\dim P_1, \dim P_2\}$. Let $x_1, \dots, x_l \in P_1$ be a maximal affinely independent set in P_1. Then, for scalars $\lambda_1, \dots, \lambda_l$, $\mu_1, \dots, \mu_l$ we have

$$\sum_{i=1}^{l} (\lambda_i - \mu_i) x_i = 0 \qquad \sum_{i=1}^{l} \lambda_i = 1 \qquad \sum_{i=1}^{l} \mu_i = 1.$$

Now, take $y \in P_2$. Then

$$\sum_{i=1}^{l}(\lambda_i - \mu_i)(x_i + y) = \sum_{i=1}^{l}(\lambda_i - \mu_i)x_i + \sum_{i=1}^{l}\lambda_i y - \sum_{i=1}^{l}\mu_i y = 0$$

so $\{x_i + y \mid i \in [l]\}$ is an affinely independent set in $P = P_1 + P_2$. This proves the first inequality.

To prove the second inequality, we will proceed by induction on the dimension of P. If $\dim P = 0$, then by the first part of this proposition, its summands P_i are all of dimension 0 as well. Suppose now that $\dim P \leq \sum_{i=1}^{k} P_i$ for all Minkowski sums such that $\dim P = d$. If $\dim P = d + 1$, then there exists a maximal chain of faces of P

$$\emptyset \subset F_0 \subset \ldots \subset F_d \subset P$$

and there exists $w \in \mathbb{R}^n$ such that $F_d = P_w = \sum_{i=1}^{k}(P_i)_w$, that is, F_d is a Minkowski sum of dimension d. So

$$\dim P = \dim F_d - 1 \leq \sum_{i=1}^{k}(P_i)_w \implies \dim P \leq \sum_{i=1}^{k} P_i$$

as there is some $i \in [k]$ such that $(P_i)_w$ is strictly contained in P_i, otherwise we would have $F_d = P$. □

Corollary 70. *An edge of a Minkowski sum $P = \sum_{i=1}^{k} P_i$ is the Minkowski sum of edges and vertices of the P_i.*

Proof. Apply the first inequality of proposition 69 to an edge $P_w = \sum_{i=1}^{k}(P_i)_w$ of P. □

We can in fact show more - an edge of a Minkowski sum is *usually* the sum of a single edge and multiple vertices, the exceptional case being described by the next proposition.

Proposition 71. *Let $v_1, v_2, w_1, w_2 \in \mathbb{R}^n$, $v_1 \neq v_2, w_1 \neq w_2$. If $P_1 = conv\{v_1, v_2\}$ and $P_2 = conv\{w_1, w_2\}$ are edges, then $P = P_1 + P_2$ is an edge if and only if $v_1 - v_2 = \mu(w_1 - w_2)$ for some $\mu \neq 0$.*

Proof. Without loss of generality, suppose $\mu > 0$ (if $\mu < 0$, we can just reorder and relabel w_1 and w_2) and let $\lambda = \frac{\mu}{\mu+1}$. Clearly, $0 \leq \lambda \leq 1$. A direct computation shows that

$$v_1 + w_2 = \lambda(v_1 + v_2) + (1 - \lambda)(w_1 + w_2)$$

that is, $v_1 + w_2 \in P$ and similarly, if $\lambda = \frac{1}{1+\mu}$ we can show that $v_2 + w_1 \in P$. So $P = conv\{v_1 + w_1, v_2 + w_2\}$ is an edge.

Now, if $x_1 = v_1 + w_1, x_2 = v_1 + w_2, x_3 = v_2 + w_1$ and

$$(\lambda_1 - \mu_1)x_1 + (\lambda_2 - \mu_2)x_2 + (\lambda_3 - \mu_3)x_3 = 0$$

is an affine combination, one can show that

$$(\mu_3 - \lambda_3)(v_1 - v_2) = (\lambda_2 - \mu_2)(w_1 - w_2)$$

so if there is no $\lambda \neq 0$ such that $v_1 - v_2 = \lambda(w_1 - w_2)$, then $\lambda_3 = \mu_3, \lambda_2 = \mu_2$ and $\lambda_1 = \mu_1$, that is, $\{x_1, x_2, x_3\}$ is an affinely independent set. By definition, then, $\dim P \geq 2$, so P is not an edge. □

This means that, in the Newton polyhedron of a polynomial set G, changing from a vertex to one of its neighbors corresponds to a choice of monomial order that chooses the same leading terms in G, with, usually, exactly one exception. We can generalize this idea in the following way: two vertices share a face of dimension d in the Newton polyhedron if their associated orders agree on $|G| - d$ choices of leading monomials.

We now define a neighborhood of a positive vector $w \in \mathbb{R}^n$ and, consequently, the neighborhood of a monomial order that can be represented as a matrix (as in Section 2.2) with first row w.

Definition 72. Let w be a positive vector in $\mathbb{R}^n$, G be a polynomial set and $v \in NP(G)$ be a vertex with w as normal vector. The neighborhood $\mathcal{N}(w)$ is the set of vectors of $\mathbb{R}^n$ that are normal to one of the neighbors of v.

We will now show how the Hilbert and Betti heuristics behave with respect to this definition of neighborhood, that is, how the heuristic values change when an order is exchanged by one of its neighbors.

Proposition 73. *Let $t_1, \ldots t_m$ be monomials and $I = \langle t_1, \ldots, t_m \rangle$. For any $i \in [m]$, let $\overline{t_i}$ be a monomial, $s = \deg \overline{t_i}$. Then*

$$HS_{I+\langle \overline{t_i} \rangle}(t) = HS_I(t) - t^s HS_{I:\overline{t_i}}(t).$$

Proof. See, for example, (COX; LITTLE; O'SHEA, 2015), Chapter 10, §2, Lemma 5. □

The above proposition implies that the terms of degree smaller than $\min\{r, s\}$ of the Hilbert series are unchanged when one replaces the leading monomial t_i of degree r by $\overline{t_i}$ of degree s. Intuitively, this means changing a monomial order by one of its neighbors in the Newton polyhedron will often cause only small changes in the Hilbert function, so the Hilbert heuristic has a somewhat local effect compatible with our definition of neighborhoods of monomial orders.

To see the effects of the neighborhood in the Betti heuristic, suppose that we are approximating Betti numbers using the Buchberger graph of Definition 26. Then, changing the leading monomial t_i by $\overline{t}_i$ in I consists of

1. removing the vertex t_i from $Buch(I)$ along with its incident edges;
2. reinserting all edges (t_j, t_k) such that t_i was the only monomial in $\{t_1, \ldots, t_m\}$ strictly dividing $\operatorname{lcm}(t_j, t_k)$ in every variable;
3. inserting the vertex $\overline{t_i}$ in $Buch(I)$ along with edges $(\overline{t_i}, t_j)$ whenever there is no t_k strictly dividing $\operatorname{lcm}(\overline{t_i}, t_j)$ in every variable;
4. removing all edges (t_j, t_k) of $Buch(I)$ such that $\overline{t_i}$ strictly divides $\operatorname{lcm}(t_j, t_k)$ in every variable.

It is possible, then, to compute the new value of the Betti heuristic after the change of ordering if, for every non-edge of $Buch(I)$ one stores the list of vertices that eliminated it. This approach has the advantage of also providing enough data to rebuild the queue of S-polynomials in the execution of Buchberger's algorithm.

4.2 Simplified unrestricted algorithms

In this section, we will propose two simple unrestricted algorithms that do not depend on the computation of Newton polyhedra. They are intended to be easy to implement and to have small overheads, without necessarily being competitive with more sophisticated algorithms.

The first simplified algorithm is not based on any locality structure for monomial orders. Instead, we simply generate a number of random weight vectors w at each iteration of the dynamic algorithm and evaluate them with respect to one of the heuristics, continuing the computation using the one optimizing the chosen heuristic. This is expected to be particularly problematic for the Hilbert heuristic, as tiny improvements in the heuristic value would lead to a complete reconstruction of the S-polynomial queue at potentially every iteration of the algorithm, causing a very bad performance. This problem would not occur when using the Betti heuristic, however, as one could only keep an ordering if it reduces the size of the queue compared to its current size, regardless of how many S-polynomials have been processed since the last time the queue was rebuilt. In our implementation, 10 new random orderings were generated and evaluated during each iteration of the algorithm. This number was chosen based on the performance of the algorithm on a few instances, but other values could also be tested.

The second simplified algorithm is based on perturbations of a weight order w. This is loosely connected to our definition of neighborhood (Definition 72). More precisely, starting from an initial weight vector w_s, corresponding to either the $grevlex$ ordering or a random ordering, we can generate "close" weight vectors w by choosing $i \in [n]$, $\epsilon \in \mathbb{R}$ and setting

$$w = w_p + \epsilon \mathbf{e_i}$$

where w_p is the previous weight vector and $\mathbf{e_i}$ is the i-th canonical basis vector of $\mathbb{R}^n$. For small enough ϵ, w should be in the neighborhood of w_p, although guaranteeing this is not necessary and would, in fact, cause a larger overhead. It is also possible that w and w_p induce equivalent orders over the polynomial set G. In practice, it is cheaper to simply fix a value of ϵ and do no checks, regardless of w generating an order that is distinct from w_p or being in its neighborhood. The chosen heuristic can then be applied to w, which is chosen as the new weight vector if it improves the heuristic value. Multiple values of i (and ϵ) could be used in each iteration of the dynamic algorithm and the only extra cost is the computation of the heuristic function for each of these candidates.

In our implementation of this perturbation algorithm, we decided to start from the $grevlex$ ordering and to try perturbations over all values of $i \in [n]$ both increasing and decreasing each coordinate of the vector w. Also, at each iteration, we set ϵ to a random number between 1 and the maximum of the degrees in the partial basis G.

4.3 Simplex-based algorithm

The perturbation-based algorithm of the previous section used some of the neighborhood structure of monomial orders we described in Section 4.1, but in an imprecise manner — the perturbations did not guarantee that the order would indeed change nor that, if it did, the new order would be a neighbor of the previous. In this section we will describe a more sophisticated method to “walk” in neighborhoods of a monomial order.

First, we note that finding neighboring orders is the same as finding a normal vector to a neighbor of a vertex in $NP(G)$. We can, then, use classical algorithmic tools to walk in a polyhedron, such as the Simplex method. To do this, we will need a linear model for $NP(G)$.

Let $G = \{g_1, \ldots, g_m\}$, l_i be the number of vertices of $NP(g_i)$ and α_{ij} be the j-th vertex of $NP(g_i)$. Then, for any linear function $w \cdot x$ with $w > 0$ there is a vertex x^* of $NP(G)$ maximizing it subject to the constraints given by linear program 4.1. This model comes directly from the definition of a Minkowski sum, as points in the sum are sums over i of convex combinations of the α_{ij}. This linear program consists of $n + \sum_{i=1}^{m} l_i$ real variables and $n + m + \sum_{i=1}^{m} l_i$ constraints.

$$\begin{aligned} &\sum_{i=1}^{m}\sum_{j=1}^{l_i} \lambda_{ij}\alpha_{ij} = x, \\ &\sum_{j=1}^{l_i} \lambda_{ij} = 1, \ \forall i \in [m] \\ &\lambda_{ij} \geq 0, \ \forall i \in [m] \, \forall j \in [l_i] \end{aligned} \tag{4.1}$$

Given a weight order w, then, maximizing $w \cdot x$ under the constraints of the linear model above gives a vertex $x^* \in NP(G)$ with normal vector w. To find a neighbor of x^*, one could apply Simplex pivoting operations in the optimal dictionary of the linear program. This leads to some problems, however — due to the presence of the slack variables in the Simplex method, the polyhedron appearing in it is higher dimensional than $NP(G)$, and many of its vertices do not correspond to vertices of $NP(G)$. In fact, even the neighborhood relations between vertices may not be the same. In addition, a vertex of $NP(G)$ is represented by many distinct bases, and any pivoting-based algorithm would have to also deal with cycling issues.

Another approach to find a neighbor of x^* is to use sensitivity analysis. Classical sensitivity analysis can be applied to the optimal dictionary of the linear program to obtain a range $[t_{min}, t_{max}]$ where x^* is still optimal for the modified objective

$$\overline{w} = w + t\mathbf{e_j} \quad \forall t \in [t_{min}, t_{max}]$$

for some fixed j and we would expect that, by choosing t slightly outside this range, the corresponding $\overline{w}$ would cease to be the normal of x^* and be instead, usually, normal to one of the neighbors of x^*. The issue is that this does not happen, as the range $[t_{min}, t_{max}]$ given by classical sensitivity analysis is often strictly smaller than the maximum range of the perturbations that would keep x^* optimal. Fortunately, this problem has been studied previously by (JANSEN et al., 1997) and has a relatively simple solution using linear programming.

Given a linear programming problem of the form

$$\begin{aligned} \max \quad & c^T \cdot x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

with an optimal solution x^* we can obtain the full sensitivity range $[t_{min}, t_{max}]$ for a change

$$\bar{c} = c + t\mathbf{e_j}$$

in the j-th coefficient of c by solving the following two linear programs:

$$\begin{aligned} \min \quad & \gamma \\ \text{s.t.} \quad & Ay \geq c + \gamma\mathbf{e_j} \\ & b^T y = c^T x^* + \gamma x_j^* \end{aligned} \qquad\qquad \begin{aligned} \max \quad & \gamma \\ \text{s.t.} \quad & Ay \geq c + \gamma\mathbf{e_j} \\ & b^T y = c^T x^* + \gamma x_j^* \end{aligned}$$

where t_{min} is an optimal solution of the first program and t_{max} an optimal solution of the second. Intuitively, these two linear programs model the values of γ such that the dual of the original program with the modified objective function remain feasible.

Although this method adds a significant overhead, due to the fact that one must solve at least two extra linear programs during each iteration, we note that all variables appearing in them take real values, and that their values can be warm started from a dual solution to the original problem, that would have to be solved either way in order to obtain the vertex of $NP(G)$ corresponding to the current monomial order.

The Simplex-based unrestricted dynamic algorithm is summarized in Algorithm 8. The subroutine $linearProgram$ builds the linear program 4.1 from G, while the subroutine $changeIndex$ chooses an index of the weight vector to be altered and the subroutine $sensitivityRange$ computes the sensitivity range of the linear program as described above. The value $iterations$ is the (fixed) maximum number of neighbors visited per iteration of the dynamic algorithm.

All linear programs involved can be built incrementally, as variables and con-

```
**Input:** A polynomial set $G$, a weight vector $w$, an heuristic function $H$
**Output:** A weight vector $w \in \mathbb{R}^n$ optimizing $H$
$lp := linearProgram(G)$
$x^* := solve(lp)$
**for** *iterations* **do**
    $j := changeIndex()$
    $t_{min}, t_{max} := sensitivityRange(lp, x^*, j)$
    $w_{new} := w + t\mathbf{e}_j$ for some $t \notin [t_{min}, t_{max}]$
    **if** $H(w_{new}) < H(w)$ **then**
        return $w_{new}$
    **end**
**end**
return $w$
```

Algorithm 8: The Simplex-based unrestricted dynamic algorithm.

straints are only added at each iteration, and can all be warm started from previous solutions. In Algorithm 8, these implementation details are left to the subroutines $linearProgram$ and $sensitivityRange$.

Also, the Simplex-based algorithm has two potentially nondeterministic steps, the choice of the index of weight vector to be changed and whether to choose $t \notin [t_{min}, t_{max}]$ above or below the sensitivity interval. In theory, both steps can be done randomly, but, instead, in the first step, we consider each index, one at a time, setting the number of iterations of the main loop in Algorithm 8 to be the number of variables of the polynomial ring. Similarly, we make no choice between taking $t < t_{min}$ or $t > t_{max}$ — we simply compute the heuristic function for both and decide whether to increase the parameter, decrease it or keep it unchanged.

4.4 The Restricted-with-regrets algorithm

The basic variant of the restricted algorithm proposed by (CABOARA, 1993) works fairly well, with the issue that, once a leading monomial is chosen, it can never be changed, even if that would cause the heuristic value to decrease. This happens in practice, as the heuristics are not very good at predicting which choices are advisable early on during the execution, as not much information is available at the time. It may happen, then, that the insertion of further polynomials in the basis G implies that a previous choice was not advisable. A simple way to deal with this problem is to choose a previously inserted polynomial at each iteration of the dynamic algorithm, remove it from G and reinsert it, updating the related linear program consistently, as this can lead to a different choice of leading monomial this time. We call this the *Restricted-with-regrets* algorithm. It applies the neighborhood structure of Section 4.1 by potentially changing to a neighboring order upon each insertion in G.

In order to make an impact with this strategy, it makes sense to prioritize removing and reinserting recently added $g \in G$, as older polynomials in the basis will usually have been used to eliminate many more candidate leading monomials in more recent insertions and, thus, will often not be subject to change alone. One can also take the heuristic into account when choosing which polynomial will be reinserted — for example, if the ungraded Hilbert heuristic is being used, the following proposition holds.

Proposition 74. *Let $J = \langle t_1, \dots, t_m \rangle$ be a monomial ideal, fix $i \in [m]$ and let $I = \langle t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_m \rangle$. Then $HS_I = HS_J$ if and only if t_i is a multiple of some t_j, $j \in [m]$, $j \neq i$.*

Proof. Apply the identity

$$HS_I(t) = HS_{I+\langle f \rangle}(t) - t^r HS_{I:f}(t)$$

that holds for any homogeneous I and f, $\deg f = r$, to I and $J = I + \langle t_i \rangle$. The result follows if and only if $HS_{I:t_i} = 0$. But this happens exactly when $I : t_i = \langle 1 \rangle$, that is, when $\langle t_i \rangle \subseteq I$. This is equivalent to t_i being a multiple of one of the generators of I. $\square$

Using this proposition, we can determine which leading monomials are not contributing to the value of the Hilbert heuristic and attempt to modify them applying the Restricted-with-regrets strategy.

One disadvantage of the Restricted-with-regrets algorithm when compared to the optimized restricted algorithms of more recent works (CABOARA; PERRY, 2014) and (PERRY, 2017) is that many of the techniques developed to reduce the size and number of linear programs involved do not work, as certain constraints are eliminated under the assumption that a previous leading monomial would not change, and this assumption is broken by Restricted-with-regrets.

We consider this algorithm to be unrestricted because all monomial orders are still reachable in further iterations, even when a choice of leading monomial for a certain polynomial is made, contrarily to what happens in the restricted algorithms described in Section 3.2.

5 COMPUTATIONAL RESULTS

In this chapter, we will report the results of our experiments involving the algorithms described in Chapter 3 and those proposed in Chapter 4. In Section 5.1, we describe the instances and computational environment of the experiments while in Section 5.2 we approach the experiments and results specifically.

The objectives of the experiments are:

- to evaluate all three heuristics (Hilbert, Betti and Mixed) with respect to the size of the output bases.
- to evaluate the performance of the new unrestricted algorithms.
- to compare unrestricted algorithms to each other, restricted algorithms and the Static algorithm, in order to find whether they lead to smaller Gröbner Bases.

Subsections 5.2.1, 5.2.2 and 5.2.3 deal, respectively, with each of the three objectives. Additionally, Appendix B shows results of each algorithm individually, in a format that is not suited to direct comparisons among them, but includes data such as how many timeouts occurred for each algorithm, as well as how well the algorithm performed over the instances for which it did not time out.

5.1 Instances and computational environment

All algorithms were implemented in the Sage computer algebra environment (The Sage Developers, 2018), version 8.3, based on Caboara and Perry's implementation of their dynamic algorithm (CABOARA; PERRY, 2014) available at

<http://www.math.usm.edu/perry/Research/dynamic_gb.pyx>.

In fact, all of our experimental results involving the static, original restricted algorithm (CABOARA, 1993) and restricted algorithm with boundary vectors and disjoint cones criteria (CABOARA; PERRY, 2014) use the implementation from (CABOARA; PERRY, 2014). The remaining algorithms also share their implementation of the base functionality of Buchberger's algorithm, Gebauer-Möller criteria and polynomial reduction.

All experiments were run on an AMD FX-8150 Eight-core Processor (3.6 GHz) with 32 GB of memory and parallelized using GNU Parallel (TANGE, 2018).

Each algorithm was run on 141 instances with 2 to 8 variables. A summary of their characteristics is given in Table 5.1. A more complete description of each instance is presented in Appendix A.

5.2 Experiments and results

We have ran all of the following algorithms over each of the 141 instances detailed in Appendix A, using the Hilbert, Betti and Mixed heuristics in all dynamic algorithms. In the remainder of this Section, we will refer to the algorithms as follows:

- Static is the Buchberger algorithm with fixed $grevlex$ ordering.
- Caboara (CABOARA, 1993) is the restricted algorithm with no disjoint cones nor boundary vectors criteria.
- CP (CABOARA; PERRY, 2014) is the restricted algorithm with both disjoint cones and boundary vectors criteria.
- GS is our implementation of the original unrestricted algorithm of (GRITZMANN; STURMFELS, 1993). It uses Sage's classes for working with polyhedra to compute Minkowski sums directly.
- Random is our implementation of the random walk algorithm proposed in Section 4.2
- Perturb is our implementation of the perturbation-based algorithm proposed in Section 4.2
- Simplex is our implementation of the Simplex-based algorithm proposed in Section 4.3
- Regrets is our implementation of the Restricted-with-regrets algorithm described in Section 4.4
- GS-then-CP is an algorithm that runs GS during m iterations (the number of input polynomials) and then runs CP until obtaining a Gröbner Basis. This is similar to using GS to choose a good initial order, and then running CP.

All algorithms use the *sugar* strategy (GIOVINI et al., 1991) to choose S-polynomials to be reduced, the Gebauer-Möller algorithm to update the queue of S-polynomials and the same reduction algorithm. As in (CABOARA; PERRY, 2014), all implementations are meant as proofs of concept, and are not necessarily efficient nor do they use the best known data structures. For instance, the implementation of the polynomial reduction procedure is not optimized, and no data structures were used to cache and speed up monomial division queries. This, however, is not a problem for the comparisons between algorithms, as all of them share these components.

In all tables that follow, we compare algorithms pairwise, with respect to running time, number of polynomials of the output basis, number of monomials in the output basis, maximum degree of a polynomial in the output basis and number of S-reductions. To compare algorithms A1 and A2, we take the geometric mean of the ratios of the values for A1 and A2 with respect to each parameter (for example, running time) over all instances in which none of A1 and A2 timed out. Thus, if these geometric means are smaller than 1, A1 is preferred with respect to the parameter being measured, otherwise, A2 is preferred. More specific details will be explained case by case, as we use multiple table formats. In all cases, the timeout time chosen was 30 minutes.

5.2.1 Evaluation of the heuristics

We start by evaluating the Hilbert, Betti and Mixed heuristics. To do this, for each pair of heuristics and each algorithm, we compare various basis size and performance parameters. Each table contains the following data:

- Algorithm 1 and Algorithm 2 — the algorithms being compared, including the heuristic being used
- Instances — the number of instances in which neither of Algorithm 1 and Algorithm 2 timed out
- t, $|G|$, $|\operatorname{Supp}|$, $\deg$, #S-red — the geometric mean of the ratios of Algorithm 1 by Algorithm 2 with respect to running time, number of polynomials in basis, number of monomials in basis, maximum degree of a polynomial in basis and number of S-reductions, respectively

In short, if the values are smaller than 1, then Algorithm1 is preferable to Algorithm2 in the instances they do not timeout, and conversely, if the ratios are larger than 1, Algorithm2 is preferable.

From Table 5.2, we observe that the Mixed heuristic performs better than the Betti heuristic in every aspect but running time. This indicates that the use of the degree of the Hilbert polynomial indeed improves the final basis in terms of size, and even in number of S-reductions in most cases. A possible explanation for the fact that the Betti heuristic was faster is that to compute the Hilbert heuristic it is necessary both to compute the Hilbert polynomial and to approximate a Betti number for each monomial order considered, while the Betti heuristic only has to do the latter computation.

Table 5.3 shows that the Hilbert heuristic performs better both in running time and number of polynomials in the output basis than the Mixed heuristic. The differences in running time is probably due to the fact that the Betti numbers in our implementation of the Mixed heuristic are computed by our own, non-optimized code in Sage, while the Hilbert series are computed by Singular, that is much faster. The results for basis size suggest that using the Hilbert series is often a better tiebreaking criterion than using our approximate Betti numbers, but not by a very large margin. In fact, for many algorithms, the bases returned by the Mixed heuristic tend to have fewer monomials, smaller degrees or be generated with fewer S-reductions. In particular, for the restricted algorithms, the Mixed heuristic seems overall preferable if one simply aims to obtain smaller bases, and it also seems plausible that if we used a more efficient implementation of the Betti component of the heuristic, the running times would be much closer as well.

Comparing the Hilbert to the Betti heuristic, as in Table 5.4, we see that Hilbert is preferable in almost every aspect for almost every algorithm. It is interesting to note that (PERRY, 2017) reported that Betti could be a good alternate heuristic (in the context of an implementation of the CP algorithm) and indeed, in average, it performs similarly to the Hilbert heuristic in this case. However, each returns bases significantly smaller than the other in certain instances — for example, in katsuranh6, cp-betti returns a basis of size 32, while cp-hilbert returns a basis of size 66 and, conversely, in econ8, the former returns a basis with 34 polynomials while the latter returns a basis with 13 polynomials.

Overall, we can conclude that the Betti heuristic is worse than both the Hilbert and Mixed heuristics in terms of output basis size. For this reason, in the following, we consider only the Hilbert and Mixed heuristics, reducing the size of the comparison tables.

5.2.2 Algorithm comparison with respect to time

We now proceed to compare all algorithms with respect to running time and S-reductions, as this is the other measure that should be most closely related to the running time. Each of the remaining tables is an square matrix in one of the two following formats:

- Each entry a_{ij} of the matrix is the geometric mean of the ratios of the parameter (running time or S-reductions) of the algorithm in row i by the algorithm in row j. In order to avoid redundancy and reduce the amount of tables, instead of duplicating the results by having $a_{ij} = \frac{1}{a_{ji}}$, we present results for the Hilbert heuristic when $i < j$ (above the main diagonal) and results for the Mixed heuristic when $i > j$ (below the main diagonal). Examples are Tables 5.5 and 5.7 — in captions, it is referred as *geometric mean of ratios*.

- Each entry a_{ij} of the matrix contains three values x_{lt}, x_{eq}, x_{gt}, which correspond to the number of instances where none of the algorithms in row i and column j timed out and i had a respectively lower, equal or greater value of the comparison parameter than j. It uses the same partition of the table between the Hilbert and Mixed heuristics of the previous format. Tables 5.6 and 5.8 are examples of this case — in captions, it is referred as *number of instances*.

It is not surprising that the results of Tables 5.5 and 5.6 show that the Static algorithm is the fastest, as it has no dynamic overhead, and this fact was previously observed in implementations. It is also expected that GS is the slowest algorithm, as it evaluates every potential ordering. The fact that all newly proposed unrestricted dynamic algorithms are faster than it for most instances is a first indicator that they work as designed, being viable for more instances. This is also shown by the fact GS times out much more often than any other algorithm — this is seen more clearly from the data in Appendix B.

All unrestricted algorithms, however, performed poorly on average when compared to the restricted algorithms Caboara and CP. As some of these unrestricted algorithms, such as Random, should have very low overhead, it seems reasonable to conclude this worse behavior is due to the reconstruction of the S-polynomial queues that could lead to many more S-reductions. This is confirmed by the data in Tables 5.7 and 5.8. It is worth noting, however, that there is a non-negligible amount of instances where each unrestricted algorithm performs better than the restricted algorithms with respect to both time and S-reductions.

Also, the fact that the number of S-reductions is sometimes much lower in the restricted algorithms when compared to Static suggests that, if these algorithms could be implemented more efficiently, with lower overhead, they would be reasonably competitive with Static in terms of running time. From Table 5.7, it does not look like the same would happen with the unrestricted algorithms.

Although none of the new unrestricted algorithms seem particularly promising from the running time perspective, we note that they time out about as often as the restricted algorithm Caboara does, and much less often than the original unrestricted algorithm GS. They also perform better than Caboara and GS-then-CP in many instances, even though their average performances are worse.

5.2.3 Algorithm comparison with respect to the final basis

In this Subsection, we evaluate all algorithms with respect to the output basis — in particular, its number of polynomials, monomials and maximum degree. All tables are presented in the same formats described in Subsection 5.2.2, containing either geometric means of ratios or number of instances where each algorithm performs better with respect to the given parameter.

For both the Hilbert and the Betti heuristics, we can see from Tables 5.9 and 5.11 that all dynamic algorithms have smaller bases on average than Static. This means all of these algorithms achieve the primary goal of returning smaller bases. It is also easily seen that the opposite happens with the GS algorithm — no other dynamic algorithm is able to return smaller bases than it in terms of polynomials nor monomials. This is expected, as it explores many more monomial orders and consequently times out for many more instances.

The second most successful algorithm in terms of basis size is GS-then-CP, as according to Tables 5.9 and 5.11 it is better on average than every other algorithm, except GS. Interestingly, this behavior is significantly different from the original CP algorithm, that returns larger bases than Caboara and every unrestricted algorithm in terms of monomials and often also in terms of polynomials. This means that the choice of a good initial monomial ordering can improve the CP algorithm in many cases, with some overhead, as seen in the previous subsection.

Among the new unrestricted algorithms, Random performs better in average. This is somewhat surprising, as the algorithm is extremely simple. We note, from Tables 5.10 and 5.12 that Random returns smaller bases than the other new unrestricted algorithms in relatively few instances. This must mean it works particularly well for these instances. A possible explanation for this is that all other algorithms start from the $grevlex$ ordering and do some kind of local search or cone narrowing around it, while the Random algorithm can choose orderings very far from $grevlex$ fast. It would be interesting to design algorithms that both do local searches and are able to explore distant parts of the monomial ordering search space like Random does.

It is interesting to compare the results of Simplex and Perturb, as they are very similar algorithms, differing mostly by the fact that Simplex does its local search in a more precise way than Perturb does. Surprisingly, Perturb works better in average, and Tables 5.10 and 5.12 show that they tie very often. This means that the extra overhead of the Simplex algorithm is mostly unnecessary, and a very similar effect can be achieved by the simpler local search procedure of Perturb.

The Regrets algorithm, unfortunately, is worse than every other algorithm in terms of basis size. Its reinsertion procedure does not seem to improve anything with respect to the Caboara algorithm on which it is based.

Comparing the Caboara and CP algorithms we can see that, although Subsection 5.2.2 has shown that the optimizations in CP improve its running time, they lead to worse output bases in general. This may happen due to the way boundary vectors are computed approximately, instead of exactly, in this implementation of CP, as these approximations may cause it to "lose" some orderings it should evaluate.

With respect to the maximum degree of a polynomial in basis, Tables 5.13 and 5.14 show that the restricted dynamic algorithms are usually worse than Static, and Perturb and Simplex are both better, while Random is much better than the restricted algorithms but in average ties with Static in this regard. The fact that GS also performs better with respect to the maximum degree is, again, not very surprising, as it visits many orderings and the heuristics are at least indirectly designed to minimize degrees — the Hilbert heuristic does this explicitly, by comparing the coefficients of the Hilbert series lexicographically, but we note that this behavior of minimizing degrees also happens in the Mixed heuristic. Overall, both the Perturb and Simplex algorithms lead to smaller or equal maximum degrees in the basis in most instances when compared to Static, Caboara and CP.

Experimentally, we have shown that the restricted algorithm GS-then-CP leads to smaller bases than the previous restricted algorithms and the new unrestricted algorithms very often. While our new unrestricted algorithms return larger bases in average than it and the previous restricted algorithm Caboara, they also return significantly smaller bases for many instances, and the degrees of the polynomials in their output bases tend to be smaller. This means that the behavior of unrestricted algorithms often complements that of restricted algorithms, in the sense that they perform well for different groups of instances.

Table 5.1: Summarized instance data, grouped by number of variables.

- n — number of variables of the input system
- Instances — number of instances
- m — average number of polynomials of the instances
- $|\operatorname{Supp}|$ — average number of monomials of the instances
- $\deg$ — average maximum degree of the instances
- hom — number of homogeneous instances

n	Instances	m	$\lvert\operatorname{Supp}\rvert$	$\deg$	hom
2	3	2.00	10.67	3.33	0
3	12	2.33	15.25	6.08	3
4	20	3.95	158.45	5.40	10
5	30	4.37	137.63	4.73	19
6	23	5.26	151.70	4.22	16
7	25	7.40	325.32	5.12	13
8	28	8.14	323.07	4.75	14

Table 5.2: Comparison between Betti and Mixed heuristics

Algorithm 1	Algorithm 2	Instances	t	$\lvert G\rvert$	$\lvert\operatorname{Supp}\rvert$	$\deg$	#S-red
caboara-betti	caboara-mixed	84	0.94	1.03	1.05	1.07	1.10
cp-betti	cp-mixed	97	1.04	1.01	1.04	1.09	1.09
gs-betti	gs-mixed	51	0.57	1.07	1.11	1.13	0.90
gs-then-cp-betti	gs-then-cp-mixed	90	0.99	1.12	1.20	1.16	1.30
perturb-betti	perturb-mixed	79	1.30	1.25	1.29	1.05	1.29
random-betti	random-mixed	81	0.78	1.32	1.37	1.08	1.02
regrets-betti	regrets-mixed	74	0.82	1.13	1.14	1.00	1.15
simplex-betti	simplex-mixed	81	1.11	1.30	1.35	1.08	0.91

Table 5.3: Comparison between Hilbert and Mixed heuristics

Algorithm 1	Algorithm 2	Instances	t	$\lvert G\rvert$	$\lvert\operatorname{Supp}\rvert$	$\deg$	#S-red
caboara-hilbert	caboara-mixed	83	0.75	0.99	1.02	1.08	1.04
cp-hilbert	cp-mixed	100	0.78	1.00	1.05	1.04	1.05
gs-hilbert	gs-mixed	51	0.69	0.99	0.99	0.95	0.81
gs-then-cp-hilbert	gs-then-cp-mixed	93	0.85	0.96	1.02	0.98	0.93
perturb-hilbert	perturb-mixed	85	0.69	0.92	0.87	0.98	1.24
random-hilbert	random-mixed	82	0.48	0.98	0.94	0.95	0.96
regrets-hilbert	regrets-mixed	76	0.68	0.98	1.00	1.05	1.03
simplex-hilbert	simplex-mixed	84	0.66	0.94	0.92	0.99	0.89

Table 5.4: Comparison between Hilbert and Betti heuristics

Algorithm 1	Algorithm 2	Instances	t	$\lvert G\rvert$	$\lvert \text{Supp} \rvert$	$\deg$	#S-red
caboara-hilbert	caboara-betti	87	0.74	0.96	0.97	1.03	0.97
cp-hilbert	cp-betti	97	0.77	1.00	1.00	0.95	0.97
gs-hilbert	gs-betti	53	1.29	0.91	0.88	0.84	0.98
gs-then-cp-hilbert	gs-then-cp-betti	91	0.87	0.86	0.83	0.84	0.71
perturb-hilbert	perturb-betti	77	0.63	0.72	0.66	0.92	0.91
random-hilbert	random-betti	80	0.62	0.74	0.68	0.85	0.95
regrets-hilbert	regrets-betti	74	0.84	0.86	0.87	1.04	0.89
simplex-hilbert	simplex-betti	81	0.66	0.72	0.68	0.92	1.03

Table 5.5: Pairwise comparison between dynamic algorithms with respect to time (geometric mean of ratios)

	static	caboara	cp	gs	gs-then-cp	perturb	random	regrets	simplex
static	1.00	0.25	0.40	0.06	0.21	0.21	0.18	0.15	0.13
caboara	4.89	1.00	1.67	0.16	0.81	0.70	0.63	0.62	0.46
cp	3.10	0.51	1.00	0.13	0.51	0.47	0.42	0.36	0.31
gs	16.98	7.38	10.59	1.00	3.57	2.75	3.07	5.01	3.31
gs-then-cp	5.86	1.12	2.18	0.23	1.00	0.92	0.85	0.84	0.73
perturb	9.94	1.81	3.41	0.25	1.57	1.00	0.93	0.99	0.82
random	13.71	2.78	5.30	0.36	2.43	1.52	1.00	1.07	0.89
regrets	8.06	1.67	3.50	0.18	1.36	0.86	0.60	1.00	0.78
simplex	11.32	2.35	4.43	0.31	1.96	1.35	0.88	1.50	1.00

Table 5.6: Pairwise comparison between dynamic algorithms with respect to time (number of instances)

	static	caboara	cp	gs	gs-then-cp	perturb	random	regrets	simplex
static	0/127/0	102/0/2	105/0/6	56/0/2	102/0/6	97/0/5	99/0/4	90/0/1	86/0/2
caboara	4/0/80	0/84/0	29/0/74	56/0/2	65/0/37	66/0/27	71/0/25	69/0/21	73/0/14
cp	7/0/93	67/0/17	0/100/0	57/0/1	89/0/18	80/0/16	90/0/9	77/0/14	82/0/6
gs	2/0/51	2/0/51	1/0/52	0/53/0	22/0/36	12/0/46	17/0/41	3/0/53	13/0/45
gs-then-cp	5/0/89	25/0/58	14/0/80	37/0/16	0/94/0	45/0/50	53/0/45	37/0/53	45/0/42
perturb	0/0/89	15/0/67	4/0/85	45/0/8	40/0/46	0/89/0	54/0/43	27/0/59	42/0/43
random	0/0/87	12/0/69	2/0/85	45/0/8	37/0/48	37/0/48	0/87/0	28/0/58	38/0/47
regrets	0/0/76	13/0/62	5/0/71	45/0/4	45/0/30	51/0/22	56/0/16	0/76/0	60/0/23
simplex	0/0/86	18/0/63	6/0/80	42/0/11	41/0/44	41/0/42	34/0/48	22/0/52	0/86/0

Table 5.7: Pairwise comparison between dynamic algorithms with respect to number of S-reductions (geometric mean of ratios)

	static	caboara	cp	gs	gs-then-cp	perturb	random	regrets	simplex
static	1.00	1.20	1.12	1.11	1.35	0.85	0.77	1.04	0.82
caboara	0.75	1.00	0.94	0.78	1.11	0.70	0.62	0.86	0.65
cp	0.82	1.09	1.00	0.86	1.19	0.73	0.67	0.90	0.70
gs	0.98	1.44	1.35	1.00	1.47	0.76	0.83	1.17	0.88
gs-then-cp	0.75	1.00	0.93	0.69	1.00	0.60	0.55	0.76	0.58
perturb	1.00	1.34	1.21	0.95	1.32	1.00	0.97	1.32	1.03
random	1.26	1.70	1.55	1.12	1.69	1.26	1.00	1.42	1.10
regrets	0.96	1.21	1.11	0.84	1.19	0.94	0.74	1.00	0.76
simplex	1.33	1.80	1.65	1.16	1.78	1.38	1.09	1.44	1.00

Table 5.8: Pairwise comparison between dynamic algorithms with respect to number of S-reductions (number of instances)

	static	caboara	cp	gs	gs-then-cp	perturb	random	regrets	simplex
static	0/127/0	43/19/42	47/20/44	15/18/25	41/17/50	30/48/24	46/29/28	33/26/32	25/44/19
caboara	39/20/25	0/84/0	31/58/14	21/24/13	33/29/40	45/21/27	59/19/18	32/34/24	38/21/28
cp	38/21/41	11/51/22	0/100/0	22/22/14	26/29/52	41/20/35	51/20/28	32/30/29	38/20/30
gs	18/14/21	11/18/24	14/17/22	0/53/0	4/32/22	26/19/13	25/21/12	17/21/18	27/18/13
gs-then-cp	43/16/35	25/31/27	36/25/33	22/28/3	0/94/0	45/20/30	60/21/17	42/24/24	42/19/26
perturb	18/53/18	23/21/38	34/22/33	21/14/18	30/16/40	0/89/0	41/32/24	28/24/34	25/42/18
random	22/27/38	14/25/42	24/20/43	16/19/18	15/23/47	22/28/35	0/87/0	20/21/45	26/24/35
regrets	22/25/29	20/23/32	25/19/32	22/15/12	22/17/36	21/25/27	32/20/20	0/76/0	35/25/23
simplex	15/46/25	19/19/43	26/21/39	16/14/23	26/14/45	13/45/25	26/27/29	20/24/30	0/86/0

Table 5.9: Pairwise comparison between dynamic algorithms with respect to number of polynomials in basis (geometric mean of ratios)

	static	caboara	cp	gs	gs-then-cp	perturb	random	regrets	simplex
static	1.00	1.40	1.31	1.57	1.48	1.30	1.36	1.23	1.28
caboara	0.76	1.00	0.93	1.20	1.05	0.92	0.97	0.95	0.90
cp	0.78	1.05	1.00	1.27	1.14	1.02	1.06	1.00	0.98
gs	0.62	0.85	0.82	1.00	0.94	0.88	0.93	0.82	0.83
gs-then-cp	0.72	0.98	0.94	1.13	1.00	0.89	0.93	0.88	0.86
perturb	0.82	1.08	1.05	1.22	1.13	1.00	1.04	0.96	0.96
random	0.74	0.99	0.96	1.15	1.04	0.92	1.00	0.92	0.92
regrets	0.90	1.10	1.04	1.34	1.12	1.05	1.13	1.00	1.02
simplex	0.83	1.11	1.08	1.25	1.14	1.02	1.11	0.96	1.00

Table 5.10: Pairwise comparison between dynamic algorithms with respect to number of polynomials in basis (number of instances)

	static	caboara	cp	gs	gs-then-cp	perturb	random	regrets	simplex
static	0/127/0	29/20/55	32/22/57	2/21/35	22/22/64	4/55/43	13/36/54	22/26/43	5/49/34
caboara	46/20/18	0/84/0	28/61/14	4/28/26	22/43/37	29/32/32	31/30/35	24/41/25	26/31/30
cp	50/21/29	13/52/19	0/100/0	3/26/29	19/39/49	25/30/41	31/27/41	22/34/35	29/28/31
gs	30/22/1	21/27/5	21/28/4	0/53/0	16/39/3	24/30/4	16/38/4	24/26/6	25/29/4
gs-then-cp	54/23/17	30/35/18	40/31/23	6/32/15	0/94/0	37/33/25	36/34/28	31/43/16	34/32/21
perturb	30/53/6	24/24/34	29/25/35	1/28/24	18/28/40	0/89/0	23/48/26	32/34/20	19/58/8
random	45/29/13	27/32/22	38/26/23	2/29/22	21/39/25	27/37/21	0/87/0	34/30/22	31/39/15
regrets	29/29/18	17/34/24	22/29/25	4/22/23	17/25/33	18/29/26	16/26/30	0/76/0	24/34/25
simplex	30/50/6	24/26/31	28/25/33	4/27/22	23/27/35	12/60/11	21/37/24	25/28/21	0/86/0

Table 5.11: Pairwise comparison between dynamic algorithms with respect to number of monomials in basis (geometric mean of ratios)

	static	caboara	cp	gs	gs-then-cp	perturb	random	regrets	simplex
static	1.00	1.36	1.21	1.67	1.38	1.36	1.42	1.19	1.31
caboara	0.79	1.00	0.92	1.28	1.04	1.00	1.06	0.96	0.97
cp	0.81	1.05	1.00	1.36	1.15	1.15	1.17	1.02	1.06
gs	0.62	0.84	0.82	1.00	0.91	0.89	0.97	0.80	0.84
gs-then-cp	0.75	0.97	0.94	1.13	1.00	0.99	1.04	0.95	0.96
perturb	0.81	1.04	1.01	1.22	1.11	1.00	1.03	0.91	0.93
random	0.74	0.96	0.93	1.15	1.03	0.93	1.00	0.87	0.91
regrets	0.92	1.08	1.03	1.29	1.08	1.05	1.11	1.00	1.06
simplex	0.83	1.09	1.06	1.22	1.11	1.02	1.09	0.96	1.00

Table 5.12: Pairwise comparison between dynamic algorithms with respect to number of monomials in basis (number of instances)

	static	caboara	cp	gs	gs-then-cp	perturb	random	regrets	simplex
static	0/127/0	37/15/52	44/13/54	10/13/35	35/14/59	4/53/45	18/32/53	27/25/39	5/48/35
caboara	44/15/25	0/84/0	28/53/22	9/22/27	30/37/35	28/25/40	34/23/39	25/32/33	28/24/35
cp	49/15/36	16/46/22	0/100/0	8/20/30	25/33/49	25/21/50	34/17/48	24/26/41	30/18/40
gs	29/12/12	21/23/9	21/24/8	0/53/0	17/37/4	25/21/12	19/25/14	25/19/12	25/20/13
gs-then-cp	51/17/26	27/31/25	37/27/30	6/32/15	0/94/0	34/23/38	36/25/37	27/33/30	30/22/35
perturb	29/52/8	29/19/34	35/19/35	13/17/23	27/22/37	0/89/0	29/42/26	37/28/21	25/52/8
random	42/29/16	27/27/27	39/20/28	8/23/22	26/31/28	28/34/23	0/87/0	37/28/21	31/35/19
regrets	29/27/20	26/21/28	28/17/31	13/15/21	26/16/33	18/29/26	21/23/28	0/76/0	22/30/31
simplex	31/47/8	32/20/29	35/18/33	15/17/21	33/19/33	14/58/11	25/32/25	26/28/20	0/86/0

Table 5.13: Pairwise comparison between dynamic algorithms with respect to maximum degree of polynomials in basis (geometric mean of ratios)

	static	caboara	cp	gs	gs-then-cp	perturb	random	regrets	simplex
static	1.00	0.75	0.78	1.17	0.84	1.05	1.00	0.82	1.05
caboara	1.09	1.00	1.04	1.16	1.14	1.38	1.31	0.99	1.33
cp	1.17	1.05	1.00	1.20	1.08	1.31	1.25	0.96	1.30
gs	0.88	0.89	0.85	1.00	0.94	0.94	0.98	0.81	0.94
gs-then-cp	1.09	0.96	0.93	1.09	1.00	1.22	1.11	0.87	1.16
perturb	0.98	0.89	0.82	1.05	0.88	1.00	0.96	0.77	0.99
random	1.02	0.92	0.86	1.06	0.92	1.04	1.00	0.79	1.02
regrets	1.07	1.05	1.02	1.20	1.07	1.11	1.11	1.00	1.29
simplex	0.96	0.88	0.83	1.04	0.89	0.99	0.95	0.89	1.00

Table 5.14: Pairwise comparison between dynamic algorithms with respect to maximum degree of polynomials in basis (number of instances)

	static	caboara	cp	gs	gs-then-cp	perturb	random	regrets	simplex
static	0/127/0	41/42/21	45/47/19	7/25/26	38/41/29	12/61/29	21/51/31	28/44/19	10/55/23
caboara	21/32/31	0/84/0	14/72/17	2/32/24	10/56/36	7/49/37	7/46/43	15/57/18	6/44/37
cp	19/39/42	9/57/18	0/100/0	2/31/25	17/56/34	6/48/42	8/47/44	22/51/18	8/39/41
gs	23/24/6	23/25/5	22/27/4	0/53/0	13/45/0	16/36/6	8/45/5	25/31/0	14/37/7
gs-then-cp	28/31/35	22/40/21	30/37/27	3/34/16	0/94/0	14/51/30	14/53/31	31/54/5	16/43/28
perturb	20/58/11	32/38/12	41/38/10	7/29/17	32/37/17	0/89/0	17/63/17	36/43/7	14/58/13
random	23/40/24	26/44/11	32/41/14	4/35/14	28/46/11	15/48/22	0/87/0	37/42/7	16/52/17
regrets	15/38/23	13/41/21	16/39/21	3/22/24	17/31/27	6/43/24	11/36/25	0/76/0	7/48/28
simplex	21/53/12	32/36/13	38/36/12	7/31/15	32/36/17	12/62/9	23/45/14	25/40/9	0/86/0

6 CONCLUSIONS AND FURTHER WORK

In this work, we have introduced a notion of neighborhood for monomial orderings over a set of polynomials and applied it to develop multiple unrestricted dynamic algorithms for the computation of Gröbner Bases. These algorithms use the idea of a local search to evaluate multiple neighboring monomial orderings allowing for previously chosen leading monomials to change.

Additionally, we introduced a new heuristic for dynamic algorithms, the Mixed heuristic, using components from both the Betti and Hilbert heuristics. Our experiments show that this heuristic tends to lead to smaller bases than the Betti heuristic in average. However, the Hilbert heuristic seems preferable to both the Betti and the Mixed heuristics in most cases, according to our results.

To our knowledge, this is the first work on dynamic Gröbner Basis algorithms reporting results for a large number of instances (141 instances in total). Our experiments include most previously proposed algorithms, including the original unrestricted algorithm of (GRITZMANN; STURMFELS, 1993), that was not previously implemented or tested in practice. In our experiments, we compared these algorithms to the classical Static Buchberger algorithm and our new unrestricted algorithms. Although the restricted algorithms performed better on average than our unrestricted algorithms with respect to the running time, the results for basis size and number of monomials show that the unrestricted algorithms have comparable results to the Caboara algorithm, and outperform every restricted algorithm in terms of the maximum degree of the polynomials in the output basis. Our experiments also showed that the restricted and unrestricted algorithms perform well in different groups of instances, so their behavior complement each other. Future works could try to develop algorithms mixing traits from both of these classes, in order to perform better in as many cases as possible.

Our unrestricted Random-walk based algorithm performed surprisingly well, and it seems reasonable to propose an algorithm that uses some kind of neighborhood structure, such as that used in our Perturb and Simplex algorithms, along with elements of the Random algorithm. Given that Perturb also performed well, this could lead to a dynamic algorithm with relatively low overhead, when compared to one of the restricted algorithms, as those depend on solving linear programs. Such an algorithm might scale better, for larger instances, than the restricted algorithms, as (PERRY, 2017) reported that in some cases the overhead of running the restricted algorithms is high. It would also be possible to adjust the number of iterations of the Random and Perturb components of the algorithm, allowing for even lower overhead by reducing the amount of orderings to be evaluated. Another advantage would be that the lower number of monomials and lower degrees of these unrestricted algorithms could speed up computations for large instances where the performance bottlenecks are associated to the polynomial reduction process.

Another path for research in dynamic algorithms is to adapt both restricted and unrestricted algorithms to F4 and F5, as well as providing an optimized implementation of the dynamic algorithms and heuristics.

REFERENCES

BARDET, M. **Etude des systèmes algébriques surdéterminés: applications aux codes correcteurs et à la cryptographie**. Tese (Doutorado) — Paris 6, 2004.

BARDET, M.; FAUGÈRE, J.-C.; SALVY, B. **Complexity of Gröbner basis computation for Semi-regular Overdetermined sequences over F_2 with solutions in F_2**. [S.l.], 2003.

BARDET, M.; FAUGÈRE, J.-C.; SALVY, B. On the Complexity of the F5 Gröbner Basis Algorithm. **Journal of Symbolic Computation**, p. 49–70, 2015. Disponível em: <http://algo.inria.fr/seminars/.>

BAYER, D.; STILLMAN, M. A theorem on refining division orders by the reverse lexicographic order. **Duke Math. J**, v. 55, n. 2, p. 321–328, 1987.

BUCHBERGER, B. Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory. In: **Multidimensional systems theory**. [S.l.: s.n.], 1985.

BUCHBERGER, B. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. **Journal of symbolic computation**, Elsevier, v. 41, n. 3, p. 475–511, 2006.

CABOARA, M. A Dynamic Algorithm for Gröbner Basis Computation. **Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation**, p. 275–283, 1993.

CABOARA, M.; KREUZER, M.; ROBBIANO, L. Efficiently computing minimal sets of critical pairs. **Journal of Symbolic Computation**, v. 38, n. 4, p. 1169–1190, 2004. ISSN 07477171.

CABOARA, M.; PERRY, J. Reducing the size and number of linear programs in a dynamic Gröbner basis algorithm. **Applicable Algebra in Engineering, Communications and Computing**, v. 25, n. 1-2, p. 99–117, 2014. ISSN 09381279.

CANTOR, D. G.; ZASSENHAUS, H. A new algorithm for factoring polynomials over finite fields. **Mathematics of Computation**, 1981. ISSN 0025-5718.

COLLART, S.; KALKBRENER, M.; MALL, D. Converting bases with the Gröbner walk. **Journal of Symbolic Computation**, 1997. ISSN 07477171.

COX, D.; LITTLE, J.; O'SHEA, D. **Using algebraic geometry**. 2nd. ed. Springer, 2005. 596 p. ISBN 0387207333, 9780387207339. Disponível em: <http://link.springer.com/10.1007/b138611>.

COX, D.; LITTLE, J.; O'SHEA, D. **Ideals, varieties, and algorithms**. 4th. ed. [S.l.]: Springer, 2015. ISBN 9780387356501.

EDER, C.; FAUGÈRE, J. C. A survey on signature-based algorithms for computing Gröbner bases. **Journal of Symbolic Computation**, Elsevier Ltd, v. 80, p. 719–784, 2017. ISSN 07477171.

EISENBUD, D. **Commutative algebra with a view toward algebraic geometry**. New York: Springer-Verlag, 1995. ISSN 0273-0979. ISBN 9780387942698.

FAUGÈRE, J.-C. A new efficient algorithm for computing Gröbner bases (F4). **Journal of pure and applied algebra**, Elsevier, v. 139, n. 1, p. 61–88, 1999.

FAUGÈRE, J.-C. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). **Proceedings of the 2002 international symposium on Symbolic and algebraic computation - ISSAC '02**, ACM Press, p. 75–83, 2002.

FAUGÈRE, J.-C. et al. Efficient computation of zero-dimensional Gröbner bases by change of ordering. **Journal of Symbolic Computation**, Elsevier, v. 16, n. 4, p. 329–344, 1993.

FAUGÈRE, J.-C.; JOUX, A. Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In: **Advances in Cryptology-CRYPTO 2003**. [S.l.]: Springer, 2003. p. 44–60. ISBN 978-3-540-40674-7.

FAUGÈRE, J.-C.; MOU, C. Fast algorithm for change of ordering of zero-dimensional Gröbner bases with sparse multiplication matrices. **Proceedings of the 36th international symposium on Symbolic and algebraic computation - ISSAC '11**, p. 115, 2011. Disponível em: <http://portal.acm.org/citation.cfm?doid=1993886.1993908>.

FAUGÈRE, J. C.; MOU, C. Sparse FGLM algorithms. **Journal of Symbolic Computation**, Elsevier Ltd, v. 80, p. 538–569, 2017. ISSN 07477171. Disponível em: <http://dx.doi.org/10.1016/j.jsc.2016.07.025>.

FUKUDA, K. From the zonotope construction to the Minkowski addition of convex polytopes. **Journal of Symbolic Computation**, v. 38, n. 4, p. 1261–1272, 2004. ISSN 07477171.

GEBAUER, R.; MÖLLER, H. M. On an installation of Buchberger's algorithm. **Journal of Symbolic Computation**, 1988. ISSN 07477171.

GIOVINI, A. et al. "One sugar cube, please" or selection strategies in the Buchberger algorithm. In: **Proceedings of the 1991 international symposium on Symbolic and algebraic computation - ISSAC '91**. [S.l.: s.n.], 1991. ISBN 0897914376.

GOLUBITSKY, O. Converging term order sequences and the dynamic Buchberger algorithm. **Preprint**, 2006. Disponível em: <https://pdfs.semanticscholar.org/815e/b8cd6ae91f2718532a25ddf98a16c6d160b5.pdf>.

GRITZMANN, P.; STURMFELS, B. Minkowski Addition of Polytopes: Computational Complexity and Application to Gröbner Bases. **SIAM J. Disc. Math.**, v. 6, n. 2, p. 246–269, 1993.

HASHEMI, A.; TALAASHRAFI, D. A Note on Dynamic Gröbner Bases Computation. **International Workshop on Computer Algebra in Scientific Computing**, p. 276–288, 2016. ISSN 0302-9743. Disponível em: <http://link.springer.com/10.1007/978-3-540-75187-8>.

JANSEN, B. et al. Sensitivity analysis in linear programming : just be careful ! **European Journal of Operational Research**, v. 101, n. 1, p. 15–28, 1997.

KREUZER, M.; ROBBIANO, L. **Computational Commutative Algebra 1**. [s.n.], 2000. 1 – 327 p. ISBN 978-3-540-67733-8. Disponível em: <http://link.springer.com/10.1007/978-3-540-70628-1>.

LAZARD, D. Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations. In: SPRINGER. **European Conference on Computer Algebra**. [S.l.], 1983. p. 146–156.

MACAULAY, F. S. Some formulae in elimination. **Proceedings of the London Mathematical Society**, Wiley Online Library, v. 1, n. 1, p. 3–27, 1902.

MILLER, E.; STURMFELS, B. **Combinatorial Commutative Algebra**. [S.l.]: Springer, 2004. ISBN 0387223568.

MÖLLER, H. M.; MORA, F. Upper and lower bounds for the degree of Gröbner bases. In: SPRINGER. **International Symposium on Symbolic and Algebraic Manipulation**. [S.l.], 1984. p. 172–183.

MORA, T.; ROBBIANO, L. The Gröbner Fan of an Ideal. **Journal of Symbolic Computation**, v. 6, p. 183–208, 1988.

PATARIN, J. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms. In: SPRINGER. **Advances in Cryptology—EUROCRYPT'96**. [S.l.], 1996. p. 33–48.

PERRY, J. Exploring the Dynamic Buchberger Algorithm. In: **Proceedings of the 2017 International Symposium on Symbolic and Algebraic Computation**. [S.l.: s.n.], 2017. p. 365–372. ISBN 9781450350648.

PETIT, C.; KOSTERS, M.; MESSENG, A. Algebraic Approaches for the Elliptic Curve Discrete Logarithm Problem over Prime Fields. In: **Public-Key Cryptography–PKC 2016**. [S.l.]: Springer, 2016. p. 3–18.

ROBBIANO, L. Term orderings on the polynomial ring. In: **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**. [S.l.: s.n.], 1985. ISBN 9783540159841. ISSN 16113349.

ROUNE, B. H.; STILLMAN, M. Practical Gröbner Basis Computation. **Proceedings of the 2012 International Symposium on Symbolic and Algebraic Computation**, p. 17, 2012.

TANGE, O. **GNU Parallel 2018**. Ole Tange, 2018. Disponível em: <https://doi.org/10.5281/zenodo.1146014>.

The Sage Developers. **SageMath, the Sage Mathematics Software System (Version 8.3)**. [S.l.], 2018.

THOMAS, R. R. Gröbner Bases in Integer Programming. **Handbook of Combinatorial Optimization**, Springer, Boston, MA, p. 533–572, 1998.

WEIBEL, C. **Minkowski Sums of Polytopes: Combinatorics and Computation**. 1–114 p. Tese (Doutorado) — École Polytechnique Fédérale de Lausanne, 2007.

Appendices

Appendix A DETAILED INSTANCE DATA

Tables A.1 and A.2 include more detailed data about every instance used in the experiments of Chapter 5. Instances were extracted from Christian Eder's repository

<https://github.com/ederc/singular-benchmarks>

and were also used, for example, in (EDER; FAUGÈRE, 2017).

The data shown in the tables for each instance is:

- n — the number of variables of the polynomial ring
- m — the number of polynomials of the input system
- char(k) — the characteristic of the base field of the polynomial ring
- $|\operatorname{Supp}|$ — the number of monomials in the input system
- $\deg$ — the maximum degree of the polynomials of the input system
- hom — whether the input ideal is homogeneous.

Table A.1: Instances used in the experiments of Chapter 5 (part 1).

name	n	m	char(k)	$\lvert\operatorname{Supp}\rvert$	deg	hom
ahml	5	4	32003	64	4	no
ahmlh	6	4	32003	64	4	yes
assur44	8	8	32003	102	3	no
aubry2	7	12	32003	541	7	no
berth	3	2	32003	64	26	no
berthh	4	2	32003	64	26	yes
binomial_4_2_4_1	4	4	32003	8	2	yes
binomial_5_3_4_4	5	4	32003	8	3	yes
binomial_5_3_4_6	5	4	32003	8	3	yes
binomial_5_3_5_3	5	5	32003	10	3	yes
binomial_5_3_5_5	5	5	32003	10	3	yes
binomial_6_2_6_7	6	6	32003	12	2	yes
binomial_6_3_6_6	6	6	32003	12	3	yes
binomial_7_3_7_9	7	7	32003	14	3	yes
binomial_7_4_7_8	7	7	32003	14	4	yes
binomial_8_3_8_16	8	8	32003	16	3	yes
binomial_8_4_8_9	8	8	32003	16	4	yes
buchberger87	6	3	7583	6	3	yes
butcher8	8	8	32003	150	4	no
cohn3	4	4	32003	88	6	no
cyclicn2	2	2	32003	4	2	no
cyclicn3	3	3	32003	8	3	no
cyclicn4	4	4	32003	17	4	no
cyclicn5	5	5	32003	47	5	no
cyclicn6	6	6	32003	97	6	no
cyclicn7	7	7	32003	209	7	no
cyclicn8	8	8	32003	406	8	no
cyclicnh2	3	2	32003	4	2	yes
cyclicnh3	4	3	32003	8	3	yes
cyclicnh4	5	4	32003	17	4	yes
cyclicnh5	6	5	32003	47	5	yes
cyclicnh6	7	6	32003	97	6	yes
cyclicnh7	8	7	32003	209	7	yes
czapor87_1h	5	4	32003	40	2	yes
czapor87_2h	5	4	32003	48	2	yes
dessin1	8	10	32003	139	3	no
dl	8	11	32003	89	6	no
ducos7_3	7	12	32003	1018	7	no
ducos7_5	7	13	32003	1612	7	no
econ2	2	2	32003	4	1	no
econ3	3	3	32003	8	2	no
econ4	4	5	32003	21	3	no
econ5	5	6	32003	32	3	no
econ6	6	7	32003	49	3	no
econ7	7	8	32003	81	3	no
econ8	8	9	32003	120	3	no
econh2	3	2	32003	4	2	yes
econh3	4	3	32003	8	3	yes
econh4	5	4	32003	15	3	yes
econh5	6	5	32003	24	3	yes
econh6	7	6	32003	40	3	yes
econh7	8	7	32003	65	3	yes
extcyc5	6	6	32003	124	5	no
extcyc5h	7	6	32003	124	5	yes
extcyc6	7	7	32003	283	6	no
extcyc6h	8	7	32003	283	6	yes
fateman	4	3	32003	33	5	yes
fatemanh	5	3	32003	33	5	yes
fmtm	4	3	32003	6	4	yes
fmtm_non_hom	3	2	32003	4	3	no
fmtmdeh	3	3	32003	6	4	no
gerdt93	6	3	7583	10	3	yes
hemmecke	5	3	32003	16	15	yes
hfe_segers	8	14	2	96	3	no
ilias12	8	11	32003	345	5	no
ilias13	7	10	32003	187	5	no

Table A.2: Instances used in the experiments of Chapter 5 (part 2).

name	n	m	char(k)	|Supp|	deg	hom
ilias13h	8	10	32003	187	5	yes
ilias_k_2	8	9	32003	38	5	no
issac97	4	4	32003	48	2	no
jason210	8	3	32003	8	6	yes
joswig101-trimmed	5	5	101	23	23	no
katsuran2	3	2	32003	5	2	no
katsuran3	4	3	32003	12	2	no
katsuran4	5	4	32003	26	2	no
katsuran5	6	5	32003	47	2	no
katsuran6	7	6	32003	73	2	no
katsuran7	8	7	32003	117	2	no
katsuranh2	4	2	32003	5	2	yes
katsuranh3	5	3	32003	12	2	yes
katsuranh4	6	4	32003	26	2	yes
katsuranh5	7	5	32003	47	2	yes
katsuranh6	8	6	32003	73	2	yes
lichtblau	3	2	32003	23	11	no
liu	6	4	2	16	2	yes
mckay	4	9	32003	1342	10	no
noon7	7	7	32003	56	3	no
noon8	8	8	32003	72	3	no
nya	4	9	32003	1342	10	no
r5_2_2	5	5	32003	55	2	yes
r5_2_2_h	5	5	32003	55	2	yes
r5_2_4	5	5	32003	580	4	no
r5_2_4_h	5	5	32003	82	4	yes
r5_2_6	5	5	32003	2260	6	no
r5_2_6_h	5	5	32003	396	6	yes
r6_2_2	6	6	32003	96	2	yes
r6_2_2_h	6	6	32003	96	2	yes
r6_2_4	6	6	32003	1187	4	no
r6_2_4_h	6	6	32003	270	4	yes
r6_2_6_h	6	6	32003	948	6	yes
r7_2_2	7	7	32003	154	2	yes
r7_2_2_h	7	7	32003	154	2	yes
r7_2_4_h	7	7	32003	406	4	yes
r7_2_6_h	7	7	32003	2156	6	yes
r8_2_2	8	8	32003	232	2	yes
r8_2_2_h	8	8	32003	232	2	yes
r8_2_4_h	8	8	32003	912	4	yes
redcyc6	6	6	32003	97	10	no
redcyc6h	7	6	32003	97	11	yes
redcyc7	7	7	32003	209	12	no
redcyc7h	8	7	32003	209	13	yes
reimer5	5	5	32003	66	6	no
reimer5h	6	5	32003	66	6	yes
reimer6	6	6	32003	153	7	no
reimer6h	7	6	32003	153	7	yes
reimer7	7	7	32003	271	8	no
reimer7h	8	7	32003	271	8	yes
rose	3	3	32003	29	9	no
roseh	4	3	32003	29	9	yes
rpbl	7	6	32003	100	3	no
rpblh	8	6	32003	100	3	yes
safey	8	9	32003	4305	16	no
schiele	4	2	32003	6	2	no
schieleh	5	2	32003	6	2	yes
schrans_troost	8	8	32003	127	2	no
sendra	2	2	32003	24	7	no
sendrah	3	2	32003	24	7	yes
solotarev	4	4	32003	15	2	no
solotarevh	5	4	32003	15	3	yes
sparsesym5	5	5	32003	16	10	no
sparsesym5h	6	5	32003	12	10	yes
sym33	4	3	7583	11	4	yes
syz1	3	2	32003	4	2	no
trinks	7	6	32003	37	3	yes
uteshev_bikker	5	4	7583	75	3	yes
vermeer	5	5	32003	37	5	no
virasoro	8	8	32003	127	2	no
wang16	4	6	32003	89	4	no
wang16h	5	4	32003	48	4	yes
weispfenning94	4	3	7583	17	5	yes
wu90	5	5	32003	29	3	no
wu90h	6	5	32003	30	3	yes

Appendix B ADDITIONAL EXPERIMENTAL RESULTS

In this appendix, we present experimental results for all algorithms of Section 5.2 non-comparatively, that is, in a format not well suited to compare algorithms directly. For each triple (algorithm, heuristic, instance) the following data was measured and grouped by number of variables of the instance, for convenience. Every experiment was run with a timeout of 30 minutes.

- n — number of variables of this group of instances
- t_{outs} — number of timeouts in this group
- t_1 — the average running time of instances with n variables, counting instances that timed out as having taken 30 minutes
- t_2 — the average running time of instances with n variables that did not time out
- O — the average overhead of instances with n variables that did not time out, computed as the time the algorithm spends inside functions that are specific to dynamic algorithms
- $|G|$ — the average size of the final (reduced) Gröbner Basis of instances with n variables that did not time out
- $|\operatorname{Supp}|$ — the average number of monomials in the final (reduced) Gröbner Basis of instances with n variables that did not time out
- $\deg$ — the average maximum degree of polynomials in the final (reduced) Gröbner Basis of instances with n variables that did not time out
- #S-red — the average number of S-reductions in the computation of instances with n variales that did not time out

The format above is not well suited to compare algorithms directly, as timeouts are not considered in the averages. This means an algorithm with fewer timeouts was usually able to finish computing Gröbner Bases for harder instances, and so its averages increase when compared to an algorithm that timed out on these same instances.

Table B.1: Experimental results for the Static algorithm

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.10	0.10	0.00	3.33	64.67	4.67	4.33
3	0	0.27	0.27	0.00	6.92	243.33	5.58	19.42
4	0	35.72	35.72	0.00	21.30	1583.65	8.45	131.00
5	2	135.57	16.68	0.00	32.79	8102.96	7.89	119.96
6	2	197.74	45.14	0.00	47.29	4809.90	8.90	239.48
7	3	469.49	288.05	0.00	111.27	12381.73	9.18	904.05
8	7	665.74	287.65	0.00	169.71	33345.43	10.76	1079.05

Table B.2: Experimental results for the Caboara algorithm

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.17	0.17	0.07	3.33	64.33	5.33	3.67
3	0	0.53	0.53	0.36	3.67	65.83	10.67	12.17
4	3	280.41	12.25	9.20	13.41	1127.53	8.06	51.71
5	3	229.30	54.78	40.02	24.41	1860.70	9.41	112.19
6	5	421.81	38.99	28.70	23.61	734.22	21.94	171.00
7	12	903.75	76.45	70.76	38.15	1551.77	11.23	178.69
8	14	974.14	148.29	81.13	53.93	2894.93	18.29	455.36

(a) Experimental results for the Caboara algorithm with the Hilbert heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.12	0.12	0.01	3.33	78.00	6.67	5.67
3	0	7.05	7.05	6.74	2.92	38.67	18.00	14.75
4	4	364.90	6.12	5.89	9.06	617.69	8.56	18.94
5	6	370.44	13.05	12.65	13.83	235.08	7.92	44.96
6	8	703.44	118.61	117.76	18.40	318.20	7.87	73.87
7	14	1150.27	323.34	320.70	34.00	1091.27	12.09	151.45
8	21	1374.39	97.54	95.29	26.86	704.57	7.29	130.00

(b) Experimental results for the Caboara algorithm with the Betti heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.15	0.15	0.06	3.33	78.00	6.67	5.67
3	0	0.47	0.47	0.35	3.83	65.75	8.17	8.75
4	4	369.09	11.37	11.15	9.25	618.38	8.31	18.81
5	6	370.77	13.46	13.07	13.12	243.58	7.25	43.58
6	9	709.11	7.83	7.39	16.93	198.36	7.50	57.43
7	15	1180.02	250.05	247.65	29.40	788.40	10.20	134.80
8	23	1482.51	22.08	21.08	24.20	431.00	5.80	102.80

(c) Experimental results for the Caboara algorithm with the Mixed heuristic

Table B.3: Experimental results for the CP algorithm

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\mathrm{Supp}\|$	deg	#S-red
2	0	0.17	0.17	0.07	3.33	64.33	5.33	3.67
3	0	0.38	0.38	0.18	3.67	65.83	10.67	12.17
4	0	148.14	148.14	4.09	15.80	2964.40	12.40	158.20
5	3	202.53	25.04	3.85	26.41	2186.00	9.52	121.48
6	3	284.46	57.13	30.62	41.35	6557.80	16.95	237.15
7	11	838.81	83.59	45.02	58.79	4338.71	11.07	300.86
8	12	848.68	135.19	45.20	79.12	10903.56	13.62	554.88

(a) Experimental results for the CP algorithm with the Hilbert heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\mathrm{Supp}\|$	deg	#S-red
2	0	0.12	0.12	0.00	3.33	78.00	6.67	5.67
3	0	0.88	0.88	0.67	3.08	44.17	11.75	11.58
4	3	272.27	2.67	2.34	8.76	587.88	9.53	24.59
5	4	267.20	31.38	27.61	17.85	1229.15	9.04	72.85
6	7	608.36	87.02	85.09	18.44	224.88	9.94	97.94
7	13	956.03	41.72	37.10	40.17	1715.67	12.67	177.92
8	17	1247.40	393.38	375.60	56.18	3976.00	14.91	334.82

(b) Experimental results for the CP algorithm with the Betti heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\mathrm{Supp}\|$	deg	#S-red
2	0	0.14	0.14	0.04	3.33	78.00	6.67	5.67
3	0	0.20	0.20	0.09	3.83	65.75	8.17	8.75
4	3	271.90	2.24	1.94	8.94	587.35	8.88	23.88
5	4	271.18	35.98	33.91	16.77	709.85	7.69	64.77
6	5	520.93	165.63	155.18	35.00	3301.39	10.94	167.67
7	12	916.68	101.31	94.66	45.08	1941.08	12.31	206.54
8	17	1200.52	274.04	259.97	56.27	4000.55	9.73	307.00

(c) Experimental results for the CP algorithm with the Mixed heuristic

Table B.4: Experimental results for the GS algorithm

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\mathrm{Supp}\|$	deg	#S-red
2	0	32.79	32.79	32.44	2.00	34.00	16.33	18.67
3	1	150.19	0.20	0.06	2.82	36.09	8.45	4.45
4	4	446.39	107.98	107.14	6.00	537.25	8.81	37.00
5	12	786.45	110.75	110.32	11.06	148.94	5.50	68.44
6	15	1223.88	143.67	143.46	9.62	59.88	6.38	27.50
7	22	1600.80	139.99	139.71	10.00	53.67	5.33	40.33
8	26	1671.70	3.82	3.64	8.50	17.00	5.00	28.00

(a) Experimental results for the GS algorithm with the Hilbert heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\mathrm{Supp}\|$	deg	#S-red
2	0	30.30	30.30	29.96	2.00	34.00	16.33	18.67
3	1	150.19	0.21	0.06	2.91	36.27	7.91	5.27
4	4	442.45	103.06	102.40	7.31	592.06	9.06	41.38
5	12	812.69	154.49	153.95	11.56	166.67	5.89	81.11
6	15	1224.51	145.47	145.25	9.62	59.88	6.38	29.50
7	22	1600.24	135.31	135.03	10.00	53.67	5.33	41.67
8	26	1671.65	3.07	2.90	8.50	17.00	5.00	28.00

(b) Experimental results for the GS algorithm with the Betti heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\mathrm{Supp}\|$	deg	#S-red
2	0	3.63	3.63	2.91	3.33	78.00	6.67	94.33
3	1	151.15	1.25	1.06	2.55	18.91	7.27	16.91
4	6	578.86	55.51	55.05	4.71	543.07	7.36	54.29
5	16	970.90	23.35	23.10	6.93	79.43	4.86	31.79
6	15	1232.20	167.57	167.29	8.00	68.12	4.50	40.88
7	24	1728.09	2.28	2.12	7.00	14.00	3.00	23.00
8	26	1673.97	35.54	35.32	8.50	17.00	5.00	63.50

(c) Experimental results for the GS algorithm with the Mixed heuristic

Table B.5: Experimental results for the GS-then-CP algorithm

n	t_{outs}	t_1	t_2	O	$\lvert G\rvert$	$\lvert\,\mathrm{Supp}\,\rvert$	deg	#S-red
2	0	0.25	0.25	0.09	3.00	81.00	7.00	6.67
3	0	0.39	0.39	0.16	3.00	38.00	10.00	11.17
4	2	185.20	5.78	0.77	10.06	1135.67	8.67	43.22
5	2	160.58	43.48	2.15	23.68	2613.71	10.04	121.39
6	3	261.07	30.23	7.18	37.85	5250.85	19.70	234.60
7	11	860.54	122.38	21.01	39.86	1920.79	22.93	297.64
8	13	960.75	233.39	127.58	71.93	9782.27	12.13	513.53

(a) Experimental results for the GS-then-CP algorithm with the Hilbert heuristic

n	t_{outs}	t_1	t_2	O	$\lvert G\rvert$	$\lvert\,\mathrm{Supp}\,\rvert$	deg	#S-red
2	0	0.17	0.17	0.02	3.33	79.00	7.00	6.00
3	0	0.99	0.99	0.73	3.25	54.00	11.42	11.83
4	3	276.97	8.20	7.77	7.88	162.53	10.06	24.41
5	6	363.98	4.97	1.94	16.25	2358.79	9.67	54.79
6	7	604.03	80.80	77.85	22.25	280.44	11.38	116.56
7	13	993.67	120.14	114.15	44.92	1661.42	11.83	202.25
8	21	1400.85	203.42	194.92	32.00	1101.29	15.86	204.14

(b) Experimental results for the GS-then-CP algorithm with the Betti heuristic

n	t_{outs}	t_1	t_2	O	$\lvert G\rvert$	$\lvert\,\mathrm{Supp}\,\rvert$	deg	#S-red
2	0	0.25	0.25	0.08	3.00	81.00	7.33	6.67
3	0	0.37	0.37	0.19	3.50	53.00	7.92	9.00
4	3	270.64	0.75	0.51	9.82	228.00	9.18	23.65
5	5	302.95	3.54	2.59	14.60	731.36	7.12	47.00
6	5	504.40	144.52	136.07	28.72	2074.72	10.67	144.39
7	13	974.58	80.37	76.09	39.25	1190.42	10.42	169.92
8	21	1406.34	225.36	211.52	39.29	5804.43	11.14	194.14

(c) Experimental results for the GS-then-CP algorithm with the Mixed heuristic

Table B.6: Experimental results for the Random algorithm

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\mathrm{Supp}\|$	deg	#S-red
2	0	0.79	0.79	0.58	3.33	71.67	5.33	27.00
3	0	2.02	2.02	0.86	3.42	43.33	6.17	74.92
4	3	272.93	3.45	1.70	7.47	728.59	7.65	106.29
5	5	320.82	24.98	5.38	28.08	7862.68	7.12	137.12
6	7	573.62	37.08	9.85	28.38	1048.94	8.69	317.62
7	10	830.12	183.54	33.21	59.87	3977.07	8.47	611.27
8	12	905.39	234.43	35.68	82.62	10810.00	12.31	906.25

(a) Experimental results for the Random algorithm with the Hilbert heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\mathrm{Supp}\|$	deg	#S-red
2	0	0.14	0.14	0.04	3.33	64.67	4.67	4.33
3	0	8.17	8.17	7.79	6.33	225.17	5.67	24.67
4	3	294.07	28.31	27.13	16.06	766.12	9.41	67.24
5	5	341.17	49.40	46.31	19.08	1312.24	7.92	92.12
6	9	735.36	50.95	49.90	16.29	172.50	7.64	73.14
7	17	1290.85	208.90	204.28	22.88	500.12	9.75	174.12
8	23	1541.24	350.95	342.59	32.60	702.20	5.80	228.00

(b) Experimental results for the Random algorithm with the Betti heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\mathrm{Supp}\|$	deg	#S-red
2	0	0.31	0.31	0.20	3.67	85.67	5.67	8.33
3	0	4.52	4.52	4.24	3.83	55.75	6.75	33.17
4	3	393.67	145.50	137.69	8.71	813.47	8.06	325.35
5	5	405.75	126.90	122.05	13.56	481.80	6.96	189.68
6	9	837.96	219.51	217.17	17.57	251.71	7.36	134.43
7	15	1145.99	164.98	162.76	31.80	923.20	8.40	139.60
8	22	1453.78	184.33	182.19	25.83	662.83	5.83	131.83

(c) Experimental results for the Random algorithm with the Mixed heuristic

Table B.7: Experimental results for the Perturb algorithm

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.34	0.34	0.20	3.33	64.33	5.33	13.33
3	0	1.00	1.00	0.46	3.92	51.17	7.00	46.58
4	3	361.18	107.27	7.00	9.18	269.29	7.76	2077.35
5	5	322.65	27.19	5.82	27.44	7830.72	7.00	237.40
6	6	479.70	13.71	6.42	30.29	1122.35	7.12	207.88
7	8	747.66	252.44	65.09	65.76	5267.88	8.41	536.65
8	16	1099.89	166.41	53.30	98.75	16951.58	9.42	556.58

(a) Experimental results for the Perturb algorithm with the Hilbert heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.11	0.11	0.01	3.33	64.67	4.67	4.33
3	0	8.29	8.29	7.89	7.17	256.42	6.00	26.58
4	4	368.31	10.38	10.00	13.00	436.81	7.75	36.62
5	5	380.04	96.05	92.49	21.00	1405.68	7.72	105.64
6	9	799.49	156.31	154.78	22.00	244.07	8.00	108.93
7	19	1388.39	84.95	83.63	18.17	289.00	5.50	105.00
8	23	1545.52	374.93	371.60	32.20	680.80	5.00	178.40

(b) Experimental results for the Perturb algorithm with the Betti heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.23	0.23	0.12	3.67	85.67	5.67	9.67
3	0	0.48	0.48	0.35	4.00	64.00	6.42	13.50
4	3	274.56	5.36	4.86	10.24	948.76	8.00	34.65
5	6	425.66	82.07	80.40	13.12	192.08	6.46	96.83
6	7	670.94	176.97	175.57	18.25	252.31	7.38	109.25
7	15	1126.46	116.16	114.54	31.80	919.30	6.90	127.00
8	21	1434.82	339.27	336.53	32.57	970.86	5.43	158.86

(c) Experimental results for the Perturb algorithm with the Mixed heuristic

Table B.8: Experimental results for the Regrets algorithm

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	2.44	2.44	2.30	3.00	81.00	7.33	13.33
3	1	153.98	4.34	4.08	3.27	42.64	9.36	17.91
4	3	291.24	24.98	24.16	16.00	943.71	10.00	41.59
5	6	390.42	38.02	34.38	23.25	1195.62	8.88	84.79
6	8	682.01	85.75	76.82	17.33	222.40	10.93	113.60
7	13	948.53	26.10	23.72	36.17	1179.33	9.00	144.75
8	19	1331.87	343.59	321.59	56.56	1910.33	15.00	395.33

(a) Experimental results for the Regrets algorithm with the Hilbert heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.23	0.23	0.12	3.67	85.67	5.67	5.67
3	5	750.08	0.13	0.03	3.00	36.43	3.86	3.29
4	5	452.37	3.16	2.98	10.40	248.80	9.27	23.93
5	7	433.55	17.68	17.33	16.09	233.09	8.00	50.30
6	9	713.19	14.53	14.01	21.00	237.50	8.14	72.21
7	18	1337.04	146.57	144.42	34.43	592.43	8.14	147.71
8	23	1504.70	146.32	144.65	37.80	487.20	6.40	170.80

(b) Experimental results for the Regrets algorithm with the Betti heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.39	0.39	0.29	3.33	70.00	5.33	4.67
3	4	654.71	0.26	0.16	3.00	36.43	3.86	3.29
4	4	365.92	7.39	7.19	10.62	298.50	8.75	24.12
5	7	444.75	32.28	31.88	13.70	210.74	8.17	47.83
6	9	711.34	11.49	11.05	16.64	196.07	7.79	59.86
7	17	1283.08	184.62	182.41	33.00	690.25	7.12	138.62
8	23	1510.81	180.55	178.79	38.80	592.20	9.00	178.40

(c) Experimental results for the Regrets algorithm with the Mixed heuristic

Table B.9: Experimental results for the Simplex algorithm

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.24	0.24	0.14	3.33	64.67	4.67	4.33
3	0	0.77	0.77	0.51	4.67	84.58	6.67	26.75
4	3	284.87	17.49	10.90	8.29	388.65	8.06	316.94
5	7	426.91	9.01	6.43	12.04	156.48	6.22	113.09
6	9	773.29	113.26	109.86	19.86	646.21	6.50	145.07
7	14	1056.53	110.30	86.59	29.64	845.82	7.45	775.36
8	20	1303.77	63.19	51.91	34.25	989.88	5.88	453.75

(a) Experimental results for the Simplex algorithm with the Hilbert heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.17	0.17	0.07	3.33	64.67	4.67	4.33
3	0	3.13	3.13	2.85	6.92	243.33	5.58	19.50
4	4	423.86	79.83	78.90	17.88	735.06	10.19	48.50
5	5	394.56	113.48	110.32	24.28	1532.36	8.96	88.76
6	9	758.37	88.76	87.82	20.86	232.93	7.36	80.14
7	18	1302.36	22.70	21.92	21.00	332.43	5.71	87.00
8	20	1423.99	483.98	478.26	42.50	1322.75	6.00	226.88

(b) Experimental results for the Simplex algorithm with the Betti heuristic

n	t_{outs}	t_1	t_2	O	$\|G\|$	$\|\text{Supp}\|$	deg	#S-red
2	0	0.23	0.23	0.12	3.33	70.00	5.33	6.00
3	0	0.74	0.74	0.57	4.58	85.92	5.83	23.83
4	3	317.85	56.30	53.19	10.29	288.29	8.00	209.18
5	8	526.84	63.87	61.89	11.05	140.95	6.45	119.50
6	9	773.40	113.44	109.73	17.93	183.43	7.21	336.64
7	14	1052.36	100.82	99.26	31.55	872.18	7.27	134.18
8	21	1385.91	143.63	141.28	32.71	1100.14	6.14	141.57

(c) Experimental results for the Simplex algorithm with the Mixed heuristic