Spring 2020

A Dynamic F5 Algorithm

Candice Mitchell

A DYNAMIC F5 ALGORITHM

by

Candice Bardwell Mitchell

A Dissertation
Submitted to the Graduate School,
the College of Arts and Sciences
and the School of Mathematics and Natural Sciences
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Approved by:

Dr. John Perry, Committee Chair
Dr. Bernd Schröder
Dr. Karen Kohl
Dr. Jiu Ding
Dr. Rajeev Agrawal

Dr. John Perry	Dr. Bernd Schröder	Dr. Karen S. Coats
Committee Chair	Director of School	Dean of the Graduate School

May 2020

ABSTRACT

Gröbner bases are a "nice" representation for nonlinear systems of polynomials, where by "nice" we mean they have good computation properties. They have many useful applications, including decidability (whether the system has a solution or not), ideal membership (whether a given polynomial is in the system or not), and cryptography. Traditional Gröbner basis algorithms require as input an ideal and an admissible term ordering. They then determine a Gröbner basis with respect to the given ordering. Some term orderings lead to a smaller basis, but finding them traditionally requires testing many orderings and hoping for better results. A dynamic algorithm requires as input only the ideal and allows the term ordering to vary at each step of the algorithm. Previous work has shown that this often produces a smaller basis and/or finds a basis in a shorter time frame. Since some Gröbner bases under certain term orderings are extremely large, it is advantageous to find ways to compute smaller bases. The F5 algorithm is a traditional algorithm that computes a Gröbner basis in a way that attempts to avoid computing S-polynomials that reduce to zero. Since S-polynomials that reduce to zero do not add any useful information, avoiding these computations can drastically reduce the amount of work done. This work describes a dynamic F5 algorithm to compute Gröbner bases. The algorithm combines the advantages of a traditional F5 algorithm by avoiding the majority of S-polynomials that reduce to zero as well as the decrease in size that can be gained using a dynamic algorithm.

ACKNOWLEDGEMENTS

There are many who helped me to this point and so many different ways that they helped. I am grateful to the Department of Mathematics at the University of Southern Mississippi for allowing me to be a graduate assistant throughout my time here. I would not have been able to pursue, much less complete, a PhD without the generosity of the Mississippi Space Grant Consortium (MSSGC) Graduate Fellowship Program. I also want to thank my committee for sticking with me through this process and their helpful guidance along the way.

I am extremely thankful for my advisor, John Perry. He sets high expectations but gives me the tools I need to meet them. Dr. Perry has given me guidance and encouragement when I needed it and left me to my own devices when it was necessary for my growth. He has freely given his so much of his time and wisdom that I could never repay him.

My parents have supported me even though I squandered opportunities in my past. I thank them for not only continuing to believe in me but also for all their help and support with my family. My husband, Ben, has probably endured the most torture during this process. He has somehow continued to be loving and supportive, and I will be in his debt for many years to come. Finally, I am grateful for my boys, without whom I would have never pursued this path: No matter how far you fall, you can still get up and reach for the stars. I love you to the moon . . . and back!

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

LIST OF ABBREVIATIONS

DC	-	Divisibility Criterion
EDC	-	Extended Divisibility Criterion
gcd	-	Greatest Common Divisor
LC	-	Leading Coefficient
lcm	-	Least Common Multiple
LM	-	Leading Monomial
LT	-	Leading Term
$\mathbb{M}$	-	the set of monomials in $x_1, \ldots, x_n$
Supp	-	Support
t_p	-	$\mathrm{LM}(p)$
$t_{p,q}$	-	$\mathrm{lcm}(t_p, t_q)$

NOTATION AND GLOSSARY

General Usage and Terminology

For notation, we generally follow the example set in [5]. Unfortunately, there is little in the way of a "standard" notation in Computational Algebra. We will use angle brackets, $\langle \cdot \rangle$ to represent ideals, but many authors have used parenthesis $(\cdot)$ to represent them. We use $|\cdot|$ to represent the sum of a vector's elements as well as to represent a set's cardinality. As usual, $\mathbb{Z}$ represents the integers. We will use $\mathbb{Z}_{\geq 0}$ to represent the nonnegative integers. As is typical, $\emptyset$ represents the empty set. Capital letters are used to represent ideals and bases while lowercase letters are used as much as possible to represent fields and basis elements; however, neither are used exclusively for those purposes. When a notation may be unfamiliar to the reader, we define what it is intended to represent.

Algebra Basics

Definition 0.0.1 (Definition 1 of §1.2 in [5])**.** A **monomial** in $x_1, \ldots, x_n$ is a product of the form

$$\prod_{i=1}^{n} x_i^{\alpha_i}$$

where $\alpha_i \in \mathbb{Z}_{\geq 0}$, the set of nonnegative integers.

For simplicity, we can represent monomials in the following way. Let $\alpha = (\alpha_1, \ldots, \alpha_n)$, then

$$x^{\alpha} = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \cdots \cdot x_n^{\alpha_n}.$$

We will also define the **standard degree (deg)** of a monomial x^{α} as

$$|\alpha| = \sum_{i=1}^{n} \alpha_i.$$

Finally, we will represent the set of all monomials in $x_1, \ldots, x_n$ as $\mathbb{M}$.

Definition 0.0.2 (Definition 2 of §1.2 in [5])**.** A **polynomial** f in $x_1, \ldots, x_n$ with coefficients in a field k is a finite linear combination of monomials and can be represented as

$$f = \sum_{\alpha} a_{\alpha} x^{\alpha}, \quad a_{\alpha} \in k$$

where the sum is over a finite set of n-tuples $\alpha = (\alpha_1, \ldots, \alpha_n)$.

The set of all polynomials in $x_1, \ldots, x_n$ whose coefficients come from k will be written as $k[x_1, \ldots, x_n]$.

Definition 0.0.3 (Definition 1 of §1.4 in [5])**.** A subset $I \subseteq k[x_1, \ldots, x_n]$ is an **ideal** if it satisfies the following:

(i) $0 \in I$

(ii) if $f, g \in I$, then $f + g \in I$

(iii) if $f \in I$ and $h \in k[x_1, \ldots, x_n]$, then $hf \in I$.

Definition 0.0.4 (Definition 2 of §1.4 in [5])**.** Let $f_1, \ldots, f_s$ be polynomials in $k[x_1, \ldots, x_n]$. Then we set

$$\langle f_1, \ldots, f_s \rangle = \left\{ \sum_{i=1}^{s} h_i f_i \mid h_i \in k[x_1, \ldots, x_n] \right\}.$$

We call $\langle f_1, \ldots, f_s \rangle$ the **ideal generated by** $f_1, \ldots, f_s$. The proof that $\langle f_1, \ldots, f_s \rangle$ is an ideal of $k[x_1, \ldots, x_n]$ can be found in Lemma 3 of §1.4 in [5].

Remark 0.0.1. Notice that $\langle f_1, \ldots, f_s \rangle$ is an ideal generated by a finite number of polynomials.

Definition 0.0.5 (Definition 1 of §1.2 in [5])**.** Let $\langle f_1, \ldots, f_s \rangle \subseteq k[x_1, \ldots, x_n]$ be a polynomial ideal. We refer to the set of common zeros of I as the **solutions** of I. These are the set

$$\{(a_1, \ldots, a_s) \in k^n \mid f_i(a_1, \ldots, a_s) = 0 \text{ for all } 1 \leq i \leq s\}.$$

These $(a_1, \ldots, a_s)$ are also called the **affine variety** of $f_1, \ldots, f_s$.

Chapter 1
Gröbner Bases

1.1 Background

Given a polynomial ideal

$$I = \langle x^3 - 2xy, x^2y - 2y^3 + x \rangle = \langle f_1, f_2 \rangle \tag{1.1}$$

some questions about the ideal arise naturally.

- Are there any solutions to I? That is, can we find (a_1, a_2) such that

  $$f_1(a_1, a_2) = f_2(a_1, a_2) = 0?$$

- Are other polynomials like

  $$x^2 + y^2 \tag{1.2}$$

  or

  $$x^2 - xy \tag{1.3}$$

  contained in the ideal?

Finding solutions to polynomial ideals has applications in many fields, including coding theory, encryption, graph coloring, and robotics. Gröbner bases give us a "nice" form of ideals, meaning a representation with good computational properties, that make answering the questions above easier. We will investigate these further in Section 1.3.2.

In order to understand a Gröbner basis, we will begin with a concept that is more familiar, linear systems. Given the linear system

$$2x - 3y = -1$$

$$3x + 3y = 6$$

a basic background in algebra tells us that we can multiply the first equation by 3 and the second by -2 then add the two equations in order to solve for y.

$$6x - 9y = -3$$

$$-6x - 6y = -12$$

$$-15y = -15$$

Now, division by -15 tells us that

$$y = 1.$$

At this point, we simply substitute $y = 1$ into one of the original equations to see that

$$x = 1.$$

So, we have found the solution $(1,1)$.

You might be thinking, "Wouldn't it have been easier to add the original equations to solve for x?" Indeed, doing it this way saves us from having to first multiply either equation before adding.

$$\begin{aligned} 2x - 3y &= -1 \\ 3x + 3y &= 6 \\ \hline 5x &= 5 \end{aligned}$$

Division by 5 again tells us $x = 1$ and substitution gives us the same $y = 1$. We have found the solution but with less work.

Going back to the polynomials in Equation (1.1), we can try a similar approach by setting them both equal to zero and forming the system

Example 1.1.1.

$$x^3 - 2xy = 0 \tag{1.4}$$
$$x^2y - 2y^3 + x = 0 \tag{1.5}$$

At this point in the linear example, we simply multiplied the second equation by a constant that would eliminate the term containing x in the first equation. However, with our new system, it is not quite as obvious how to accomplish this goal, and it certainly can't be accomplished with multiplication by a constant.

We could try to cancel the terms by multiplying Equation (1.4) by y and Equation (1.5) by x then subtracting to get

$$\begin{aligned} x^3y - 2xy^2 &= 0 \\ -x^3y + 2xy^3 - x^2 &= 0 \\ \hline 2xy^3 - 2xy^2 - x^2 &= 0 \end{aligned}$$

Unfortunately, this did not give us helpful results like we obtained in the linear case. Luckily, Bruno Buchberger addressed this issue by introducing Gröbner bases in his 1965 PhD thesis [2]. Before we get to that, we wish to continue with this example a little more to demonstrate other possibilities for eliminating terms. That requires some definitions.

Definition 1.1.1 (Definition 3 of §1.2 in [5])**.** Given a polynomial

$$f = \sum_{\alpha} a_\alpha x^\alpha$$

(i) a_α is the **coefficient** of x^α.

(ii) When $a_\alpha \neq 0$, $a_\alpha x^\alpha$ is a **term** of f.

(iii) The **total degree** of f, $\operatorname{tdeg}(f)$, is the maximum $|\alpha|$ where $a_\alpha \neq 0$.

Different authors interchange the meanings of monomial and term, but we will use the notations found in [5]. An example of their meanings can be seen by looking at Equation (1.5). The monomials in this equation are

$$x^2y, \ y^3 \ \text{ and } \ x.$$

Their respective coefficients are

$$1, \ -2 \ \text{ and } \ 1.$$

The terms are

$$x^2y, \ -2y^3 \ \text{ and } \ x.$$

Finally, the degrees of the monomials are

$$3, \ 3 \ \text{ and } \ 1.$$

which gives us $\operatorname{tdeg}(f) = 3$. Notice that this degree is obtained by two of the monomials, both x^2y and y^3 have degree of 3.

Going back to Example 1.1.1, what if we were to rewrite the system of equations in the following way?

Example 1.1.2.

$$x^3 - 2xy = 0 \tag{1.6}$$

$$-2y^3 + x^2y + x = 0 \tag{1.7}$$

Now, proceeding in exactly the same fashion as before, if we multiply Equation (1.6) by $2y^3$ and Equation (1.7) by x^3 then add them together, we get

$$\begin{aligned} 2x^3y^3 - 4xy^4 &= 0 \\ -2x^3y^3 + x^5y + x^4 &= 0 \\ \hline x^5y - 4xy^4 + x^4 &= 0\ . \end{aligned}$$

While this result still doesn't seem helpful, we note that it is quite different from the one obtained earlier, including having a higher total degree of 6 compared with 4 earlier. Recall that in the linear example, we were able to do less work by eliminating y instead of x. A similar situation occurs when constructing a Gröbner basis; however, there is a little more to it. In Gröbner basis computation, we are eliminating the leading monomial and choosing the "right" leading monomial often leads not only to less work but also to better results, as we will see in Section 1.3.2. It then becomes important to explicitly define leading monomial, which requires us to first define monomial orderings.

1.2 Monomial Orderings

If asked to write the polynomial

$$f = 3x + x^5 + 1,$$

in descending order, we would simply reorder the terms from highest degree of x to lowest degree of x, like so

$$f = x^5 + 3x + 1.$$

This rewritten polynomial has an easily identifiable leading term (LT),

$$\mathrm{LT}(f) = x^5.$$

When we look at a polynomial with more than one variable, like

$$f = z^5 + 2xyz^3 + y^5, \tag{1.8}$$

rewriting it in descending order becomes more difficult. One might intuitively want to order it as

$$f = 2xyz^3 + y^5 + z^5,$$

so that the term with x comes before the term with only y and the term with only y before the one with only z. Notice, however, that all three monomials have the same standard degree

of 5. In the following discussion, we will show that this polynomial can be ordered in a number of different ways.

We define orderings on the exponent vector referenced in Definition 0.0.2. Throughout the remainder of this section, let $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\beta = (\beta_1, \ldots, \beta_n)$, and $\gamma = (\gamma_1, \ldots, \gamma_n)$ be in $\mathbb{Z}^n_{\geq 0}$ with α and β distinct.

To define a monomial ordering, we must first define a total ordering.

Definition 1.2.1. $\succ$ is a **total ordering** if the following are satisfied:

(i) Either $x^\alpha \succ x^\beta$ or $x^\beta \succ x^\alpha$. That is, we must be able to compare every pair of monomials and determine which is larger.

(ii) If $x^\alpha \succ x^\beta$ and $x^\beta \succ x^\gamma$, we must have $x^\alpha \succ x^\gamma$, meaning $\succ$ is transitive.

We will define a monomial ordering following the example in [5] as follows.

Definition 1.2.2 (Definition 2 of §2.2 in [5])**.** A **monomial ordering** $>$ on $\mathbb{M}$ is a relation on the set of monomials satisfying:

(i) $>$ is a total ordering on $\mathbb{Z}^n_{\geq 0}$.

(ii) If $\alpha > \beta$, then $\alpha + \gamma > \beta + \gamma$. When this is true, we say the ordering is "compatible with multiplication."

(iii) $>$ is a well-ordering on $\mathbb{Z}^n_{\geq 0}$. Here, well-ordering means if $A \subseteq \mathbb{Z}^n_{\geq 0}$ and $A \neq \emptyset$, there exists $\alpha \in A$ with $\beta > \alpha$ for every $\beta \neq \alpha$ in A.

Now, that we have a formal definition of monomial ordering, we will take a little time to look at some examples of different monomial orderings. Proofs that each of the following relations is a monomial order can be found in §2.2 of [5].

1.2.1 Lexicographic Order

As lexicography refers to the writing of dictionaries, we can infer that a lexicographic (lex) order is an "alphabetical" order.

Definition 1.2.3 (Definition 3 of §2.2 of [5])**.** If the leftmost nonzero entry of the difference $\alpha - \beta \in \mathbb{Z}^n$ is positive, we say

$$\alpha >_{lex} \beta$$

and we write

$$x^\alpha >_{lex} x^\beta.$$

Going back to our example with Equation (1.8), we see that the exponent vectors for the monomials are as follows

$$\begin{aligned} z^5 &\mapsto (0,0,5) \\ xyz^3 &\mapsto (1,1,3) \\ y^5 &\mapsto (0,5,0). \end{aligned}$$

Using these, we can easily see that

$$\begin{aligned} (1,1,3)-(0,5,0) &= (1,-4,3) \\ (1,1,3)-(0,0,5) &= (1,1,-2) \\ (0,5,0)-(0,0,5) &= (0,5,-5) \end{aligned}$$

so we have

$$xyz^3 >_{lex} y^5 >_{lex} z^5.$$

1.2.2 Graded Lex Order

When we are talking about monomial orders, "graded" indicates that we should look at the degree of the monomial first. So, a graded lex order will sort by degree and break ties using lex order.

Definition 1.2.4 (Definition 5 of §2.2 of [5])**.** If

$$|\alpha| > |\beta|$$

or

$$|\alpha| = |\beta| \quad \text{and} \quad \alpha >_{lex} \beta,$$

we say

$$\alpha >_{grlex} \beta$$

and we write

$$x^\alpha >_{grlex} x^\beta.$$

For Equation (1.8), grlex sorts the monomials exactly the same as lex, so we will look at a different polynomial

$$f = x^3y^2 + xy^2z^4 + xyz^5.$$

The polynomial is already in lex order, so let's inspect the exponent vectors to see how grlex order will sort it

$$x^3y^2 \mapsto (3,2,0)$$
$$xy^2z^4 \mapsto (1,2,4)$$
$$xyz^5 \mapsto (1,1,5)$$

We work out that

$$|(1,2,4)| = |(1,1,5)| = 7 > |(3,2,0)| = 5 \tag{1.9}$$
$$(1,2,4) - (1,1,5) = (0,1,-1). \tag{1.10}$$

Equation (1.9) shows us that

$$xy^2z^4 >_{grlex} x^3y^2 \quad \text{and} \quad xyz^5 >_{grlex} x^3y^2.$$

Since the total degrees of xy^2z^4 and xyz^5 are equal, we need to check the lex order of these two monomials which Equation (1.10) shows us is

$$xy^2z^4 >_{lex} xyz^5.$$

We have found the grlex order to be

$$xy^2z^4 >_{grlex} xyz^5 >_{grlex} x^3y^2.$$

1.2.3 Graded Reverse Lex Order

Since this is a graded order, we will first sort by degree. However, we will not break ties by sorting in reverse alphabetical order as the name might make one think. We will break ties by reversing the direction we search through the difference $\alpha - \beta$.

Definition 1.2.5 (Definition 6 of §2.2 of [5])**.** If

$$|\alpha| > |\beta|$$

or

$$|\alpha| = |\beta| \text{ and the rightmost nonzero entry of } \alpha - \beta \text{ is negative,}$$

we say

$$\alpha >_{grevlex} \beta$$

and we write

$$x^\alpha >_{grevlex} x^\beta.$$

Going back to Equation (1.8), we know that all the polynomials have the same total degree, so we will again inspect the exponent vectors.

$$\begin{aligned}(0,5,0)-(0,0,5)&=(0,5,-5)\\(0,5,0)-(1,1,3)&=(-1,4,-3)\\(1,1,3)-(0,0,5)&=(1,1,-2)\end{aligned}$$

shows us that

$$y^5 >_{grevlex} xyz^3 >_{grevlex} z^5.$$

Another way to look at this order is in the following way

(i) If $|\alpha| > |\beta|$, then $x^\alpha >_{grevlex} x^\beta$.

(ii) If $|\alpha| = |\beta|$, remove the x_i with the largest index in both monomials and recheck the total order. Continue in this fashion until all the monomials are sorted.

Again, we look at Equation (1.8), and we know that we need to start with Step (ii). So, we remove z from each monomial to obtain

$$\begin{aligned}z^5 &\mapsto 1 \text{ and } \deg(1)=0\\xyz^3 &\mapsto xy \text{ and } \deg(xy)=2\\y^5 &\mapsto y^5 \text{ and } \deg(y^5)=5.\end{aligned}$$

We end up sorting them in exactly the same order as before.

There are many other monomial orderings, and we will discuss some others in Section 2.2. For now, we only need one more definition to finish our discussion on monomial orderings.

Definition 1.2.6 (Definition 7 of §2.2 of [5])**.** Let f be a polynomial and let $>$ be a monomial order.

(i) The **degree** of f is

$$\deg(f) = \max(\alpha \in \mathbb{Z}^n_{\geq 0} \mid a_\alpha \neq 0)$$

(the maximum is taken with respect to the ordering $<$).

(ii) The **leading monomial** of f is

$$\mathrm{LM}(f) = x^\alpha, \text{ where } \alpha > \beta \text{ for all } \beta \neq \alpha.$$

For easy of reading, we will sometimes write t_f to represent of $\mathrm{LM}(f)$.

(iii) The **leading coefficient** of f is

$$\mathrm{LC}(f) = a_\alpha \text{ for all } \beta \neq \alpha.$$

(iv) The **leading term** of f is

$$\mathrm{LT}(f) = \mathrm{LC}(f) \cdot \mathrm{LM}(f).$$

As before, let $f = z^5 + 2xyz^3 + y^5$ and let $>$ be lex order. Then

$$\begin{aligned} \deg(f) &= (1,1,3) \\ \mathrm{LM}(f) &= xyz^3 \\ \mathrm{LC}(f) &= 2 \\ \mathrm{LT}(f) &= 2xyz^3. \end{aligned}$$

Notice that $\deg(f)$ is not the same as $\mathrm{tdeg}(f)$, which is 5 in this case.

1.3 Gröbner Basis Computation

Before getting into the computation, we will present a formal definition of a Gröbner basis.

Definition 1.3.1 (Definition 5 of §2.5 in [5])**.** Fix a monomial order on the polynomial ring $k[x_1,\ldots,x_n]$. A finite subset $G = \{g_1,\ldots,g_t\}$ of an ideal $I \subseteq k[x_1,\ldots,x_n]$, where $G \neq \emptyset$, is said to be a **Gröbner basis** if

$$\langle \mathrm{LT}(g_1),\ldots,\mathrm{LT}(g_t)\rangle = \langle \mathrm{LT}(I)\rangle.$$

So, if the ideal generated by the leading terms of the basis is equal to the ideal generated by the leading terms of the ideal, we have a Gröbner basis.

There are many ways to compute a Gröbner basis, and we will explore others in later chapters. We wish to begin with the most common algorithm for Gröbner basis computation. As mentioned in Section 1.1, Buchberger's PhD thesis presented us with Gröbner bases, but it also gave us an algorithm to compute them which we call "Buchberger's Algorithm" and present in Section 1.3.1.

1.3.1 Buchberger's Algorithm

Let $I = \langle f_1,\ldots,f_s\rangle \neq \{0\}$ be a polynomial ideal. Buchberger's algorithm outputs a Gröbner basis for I in a finite number of steps. The Algorithm 1 is an adaptation of Theorem 2 in §2.7 of [5].

Algorithm 1 Buchberger's Algorithm

Input: $F = \{f_1, \ldots, f_s\}$ and a monomial ordering, $>$

Output: a Gröbner basis for F with respect to $>$, $G = \{g_1, \ldots, g_t\}$, with $F \subseteq G$

1. **let** $G = F$
2. **repeat**
   (a) $G' = G$
   (b) **for** each pair $\{g_i, g_j\}$, $g_i \neq g_j$ and $i > j$ in G' **do**
      i. $S(g_i, g_j) \underset{G'}{\longrightarrow} r$
      ii. **if** $r \neq 0$ **then** $G = G \cup \{r\}$

   until $G = G'$
3. **return** G

Line 2(b)i of Algorithm 1 shows $S(g_i, g_j) \underset{G'}{\longrightarrow} r$ which we have not yet seen. To understand what this means, we will need the following algorithm and definition.

First, we need to know how to do reduction modulo a set in $k[x_1, \ldots, x_n]$. We present a modified version of the division algorithm presented in Theorem 3 of §2.3 in [5] which returns the reduction modulo G instead of the quotient and remainder after division by G.

Let $>$ be a monomial order on $\mathbb{Z}^n_{\geq 0}$, $G = \{g_1, \ldots, g_s\}$ be a set of monic polynomials in $k[x_1, \ldots, x_n]$, and $f \in k[x_1, \ldots, x_n]$. Algorithm 2 returns the reduction of f modulo G. We will write this as

$$f \underset{G}{\longrightarrow} r,$$

and we will say

$$f \text{ reduces modulo } G \text{ to } r.$$

Finally, to understand what Algorithm 1 means by $S(p, q)$, we need to define an S-polynomial.

Definition 1.3.2 (Definition 4 of §2.6 in [5])**.** Let $f, g \in k[x_1, \ldots, x_n]$ be nonzero polynomials.

(i) If $\deg(f) = \alpha$ and $\deg(g) = \beta$, let $\gamma = (\gamma_1, \ldots, \gamma_n)$, where

$$\gamma_i = \max(\alpha_i, \beta_i), \text{ for each } i.$$

We say x^γ is the **least common multiple** of $\mathrm{LM}(f)$ and $\mathrm{LM}(g)$ and we write

$$x^\gamma = t_{f,g}.$$

Algorithm 2 Reduction Algorithm in $k[x_1,\ldots,x_n]$

Input: G, f with g monic for all $g \in G$
Output: r such that no term of r is divisible by $\mathrm{LM}(g)$ for any $g \in G$

do

1. **let** $r = 0$
2. **while** $f \neq 0$
   (a) **if** $\mathrm{LM}(g)$ divides $\mathrm{LM}(f)$ for some $g \in G$
      i. $f = f - \frac{\mathrm{LT}(f)}{\mathrm{LM}(g)} \cdot g$
   (b) **else**
      i. $r = r + \mathrm{LT}(f)$
      ii. $f = f - \mathrm{LT}(f)$

return r

(ii) The S-**polynomial** of f and g is

$$S(f,g) = \frac{x^\gamma}{\mathrm{LT}(f)} \cdot f - \frac{x^\gamma}{\mathrm{LT}(g)} \cdot g.$$

(Notice that we are dividing by the leading term of each polynomial which causes their leading coefficients to go to 1, so that $\mathrm{LT}(S(f,g)) < x^\gamma$.)

When we reduce an S-polynomial using Algorithm 2, we will call that an S-**reduction** modulo G. We will omit "modulo G" when it is obvious from context.

Example 1.3.1. For a quick example of how these work, let $<$ be lex order,

$$f = 2x^2y^2 + 4y^4,$$

$$g = x^3y - x^2y^2, \text{ and}$$

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

First we compute the S-polynomial of f and g. Since $\deg(f) = (2,2)$ and $\deg(g) = (3,1)$, we get

$$\gamma = (3,2)$$

and

$$t_{f,g} = x^3y^2.$$

The S-polynomial of f and g is then

$$\begin{aligned} S(f,g) &= \frac{x^3y^2}{2x^2y^2}\cdot(2x^2y^2+4y^4) - \frac{x^3y^2}{x^3y}\cdot(x^3y-x^2y^2) \\ &= \frac{x}{2}\cdot(2x^2y^2+4y^4) - y\cdot(x^3y-x^2y^2) \\ &= (x^3y^2+2xy^4) - (x^3y^2-x^2y^3) \\ &= x^2y^3+2xy^4. \end{aligned}$$

Notice that $\mathrm{LT}(S(f,g)) = x^2y^3 < x^\gamma = x^3y^2$.

To compute the S-reduction, we will walk through Algorithm 2. For ease of reading, we will say

$$g_1 = x^2y - y^2,$$

$$g_2 = x + y,$$

and

$$S = S(f,g) = x^2y^3 + 2xy^4.$$

Loop 1. $\mathrm{LM}(g_1) = x^2y$ divides $\mathrm{LM}(S) = x^2y^3$ so

$$\begin{aligned} S &= (x^2y^3+2xy^4) - \frac{x^2y^3}{x^2y}\cdot(x^2y-y^2) \\ &= (x^2y^3+2xy^4) - y^2(x^2y-y^2) \\ &= 2xy^4+y^4 \end{aligned}$$

Loop 2. $\mathrm{LM}(g_2) = x$ divides $\mathrm{LM}(S) = xy^4$ so

$$\begin{aligned} S &= (2xy^4+y^4) - \frac{2xy^4}{x}\cdot(x+y) \\ &= (2xy^4+y^4) - 2y^4(x+y) \\ &= -2y^5+y^4 \end{aligned}$$

Loop 3. $\mathrm{LM}(g_1)$ and $\mathrm{LM}(g_2)$ do not divide $\mathrm{LM}(S) = y^5$ so

$$r = -2y^5$$

and

$$S = y^4.$$

Loop 4. $\mathrm{LM}(g_1)$ and $\mathrm{LM}(g_2)$ do not divide $\mathrm{LM}(S) = y^4$ so

$$r = -2y^5 + y^4$$

and

$$S = 0.$$

Now, the algorithm returns

$$S(f,g) \underset{G}{\to} -2y^5 + y^4.$$

Section 1.3.2 will walk through a simple example using Buchberger's algorithm as well as Example 1.1.1 from Section 1.1, and we will return to Example 1.1.2 in Section 3.1. Before the examples, we wish to add two improvements to Algorithm 1 as it is only a basic outline. First, we will remove from the final basis any polynomials whose leading term is divisible by the leading term of another basis element.

Lemma 1.3.1 (Lemma 3 in §2.7 of [5])**.** *Let G be a Gröbner basis of $I \subseteq k[x_1,\ldots,x_n]$. Let $g \in G$ be such that $\mathrm{LM}(g) \in \langle \mathrm{LM}(G\backslash\{g\})\rangle$. Then $G\backslash\{g\}$ is also a Gröbner basis for I.*

Lemma 1.3.1 allows us to eliminates unnecessary generators to compute a *minimal Gröbner basis*.

The second improvement reduces all terms of the polynomials in the final basis by the leading monomials of basis elements. We first define $\mathrm{Supp}(f)$ to be the terms of f, then

Definition 1.3.3. A **reduced Gröbner basis** for a polynomial ideal I is a Gröbner basis G for I such that for all $g \in G$ and all $m \in \mathrm{Supp}(g)$

$$\mathrm{LM}(h) \text{ does not divide } m$$

for any $h \in G\backslash\{g\}$.

We now proceed with our examples.

1.3.2 Examples of Gröbner basis computation

Example 1.3.2. Let $<$ be grevlex order,

$$I = \langle x^2 + y^2, xy - 1\rangle,$$

$$\text{where } f_1 = x^2 + y^2,$$

$$\text{and } f_2 = xy - 1.$$

We will work through the algorithm leaving out some of the computations that we have already demonstrated previously in the Chapter.

Loop 1. $G = \{x^2+y^2, xy-1\} = \{f_1, f_2\}$

$G' = G = \{g_1, g_2\}$

There is only one distinct pair in G', so we compute

$$\begin{aligned} S(g_2, g_1) &= x \cdot g_2 - y \cdot g_1 \\ &= -y^3 - x, \end{aligned}$$

so

$$S(g_2, g_1) \underset{G'}{\longrightarrow} y^3 + x.$$

We will call this new polynomial g_3 and add it to G to get

$$G = \{x^2+y^2, xy-1, y^3+x\}.$$

Loop 2. Now,

$$G' = G = \{g_1, g_2, g_3\},$$

and we have two new distinct pairs, $\{g_1, g_3\}$ and $\{g_2, g_3\}$. For the first pair we compute

$$\begin{aligned} S(g_3, g_1) &= x^2 \cdot g_3 - y^3 \cdot g_1 \\ &= -y^5 + x^3 \\ &= -y^2 g_3 + x g_1 \end{aligned}$$

so

$$S(g_3, g_1) \underset{G'}{\longrightarrow} 0. \tag{1.11}$$

We have nothing to add to G, so we move to the second pair and find

$$\begin{aligned} S(g_3, g_2) &= x \cdot g_3 - y^2 \cdot g_2 \\ &= x^2 + y^2 \\ &= g_1. \end{aligned}$$

Clearly

$$S(g_3, g_2) \underset{G'}{\longrightarrow} 0 \tag{1.12}$$

Again we have nothing new to add to G, and we have no pairs in G' to consider. Notice that $G = G'$, so we now have a Gröbner basis

$$G = \{x^2+y^2, xy-1, y^3+x\}.$$

One might wonder how we can be sure that this is a Gröbner basis. For that we will use the generalized Buchberger's Criterion to detect a Gröbner basis.

Theorem 1.3.2 (Theorem 3 in §2.9 of [5])**.** *Let I be a polynomial ideal. Then a basis $G = \{g_1, \ldots, g_t\}$ is a Gröbner basis of I if and only if for all pairs $i \neq j$,*

$$S(g_i, g_j) \underset{G}{\to} 0.$$

From here, we can simply check the pairs of distinct basis elements, $\{g_1, g_2\}, \{g_1, g_3\}$, and $\{g_2, g_3\}$. Notice that $S(g_3, g_1) \underset{G}{\to} 0$ and $S(g_3, g_2) \underset{G}{\to} 0$ as was already computed in Loop 2. We only need to calculate the S-reduction for the first pair:

$$\begin{aligned} S(g_2, g_1) &= x \cdot g_2 - y \cdot g_1 \\ &= -y^3 + x \\ &= -g_3. \end{aligned}$$

Clearly $S(g_2, g_1) \underset{G}{\to} 0$, so we have found a Gröbner basis.

Now, we return to Example 1.1.1 and work through Algorithm 1 on it. We will not indicate the loops and only write the S-reductions found at each step.

Example 1.3.3. Let $<$ be grevlex order,

$$I = \langle x^3 - 2xy, x^2y - 2y^3 + x \rangle.$$

Initially, we have

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

with

$$g_1 = x^3 - 2xy,$$
$$\text{and } g_2 = x^2y - 2y^3 + x.$$

From here we list the S-reductions and indicate the basis elements as we add them, but we wait until the end to show the Gröbner basis.

$$S(g_2, g_1) \underset{G'}{\to} xy^3 - xy^2 - \frac{1}{2}x^2 = g_3 \tag{1.13}$$

$$S(g_3, g_1) \underset{G'}{\to} 0$$

$$S(g_3, g_2) \underset{G'}{\to} y^5 - y^4 - \frac{1}{2}xy^2 = g_4 \tag{1.14}$$

$$S(g_4, g_1) \underset{G'}{\longrightarrow} 0$$

$$S(g_4, g_2) \underset{G'}{\longrightarrow} 0$$

$$S(g_4, g_3) \underset{G'}{\longrightarrow} 0$$

We have no new pairs, so we have computed

$$G = \{x^3 - 2xy, x^2y - 2y^3 + x, xy^3 - xy^2 - \frac{1}{2}x^2, y^5 - y^4 - \frac{1}{2}xy^2\}.$$

A few computations will show that all distinct pairs of the basis reduce to 0, so we have found a Gröbner basis with respect to $<$.

Chapter 2
Leading Monomials

2.1 Introduction

In Chapter 3, we will look at dynamic Gröbner basis computation. One way to perform this computation is by allowing the monomial order to vary at each step of the algorithm. In order to see which ordering is "best," the algorithm looks at each monomial of the newest polynomial, chooses the one that will have the "best" outcome as the leading monomial, and adjusts the ordering to make this "preferred" monomial the leading monomial. We say a monomial is **compatible** when some monomial ordering would make would make it the leading monomial.

For some polynomials, there are monomials that are not compatible under any ordering.

Example 2.1.1. For example, if

$$f = x^2 + xy + y^2, \tag{2.1}$$

then $xy \neq \mathrm{LM}(f)$ for any monomial order, $<$. To see why, we will assume

$$xy = \mathrm{LM}(f)$$

and return to Definition 1.2.2, specifically "compatiblity with multiplication."' If xy is the leading monomial, we must have

$$xy > x^2 \tag{2.2}$$

$$\text{and } xy > y^2. \tag{2.3}$$

For Equation (2.2) to be true, we need $y > x$. Compatibility with multiplication shows us that this implies

$$y^2 > xy,$$

which is a contradiction to Equation (2.3). Similarly, if Equation (2.3) is true, we need $x > y$, which implies

$$x^2 > xy,$$

a contradiction to Equation (2.2).

This shows that xy is not compatible for f under any monomial order.

It would be helpful to know these from the start, so we don't waste time testing them against the others.

2.2 Weighted Monomial Ordering

In Chapter 1, we looked at a few monomial orders, and we now define another class of them called a weighted order.

Definition 2.2.1. Let $\omega = (\omega_1, \ldots, \omega_n) \in \mathbb{Z}^n_{\geq 0}$, and fix a monomial order $<_\sigma$ (like $<_{lex}$ as in Section 1.2) on $\mathbb{Z}^n_{\geq 0}$. For $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$, define $\alpha >_{\omega,\sigma} \beta$ if and only if

$$\omega \cdot \alpha > \omega \cdot \beta \text{ or } (\omega \cdot \alpha = \omega \cdot \beta \text{ and } \alpha >_\sigma \beta).$$

We call $>_{\omega,\sigma}$ the **weighted order** determined by ω and $>_\sigma$. We will omit ω, σ when it is clear from context. We also define the **weighted degree** ($\deg_\omega$) of a monomial x^α under a weighted monomial order as

$$\deg_\omega(x^\alpha) = \omega \cdot \alpha.$$

We omit ω when it is clear from context.

So, a weighted order uses the dot product of the weight vector ω and the exponent vector α to order the monomials. If there are ties, they are broken using $<_\sigma$.

Example 2.2.1. To see how this works, we return to the polynomial from Equation (1.8),

$$f = z^5 + 2xyz^3 + y^5.$$

We already showed that $<_{lex}$ orders the polynomial as $f = 2xyz^3 + y^5 + z^5$ and $<_{grevlex}$ orders it as $f = y^5 + xyz^3 + z^5$. One might wonder if there is a way to order it such that $\mathrm{LM}(f) = z^5$. Indeed, the weight vector

$$\omega = (0,0,1)$$

does the trick, since

$$(0,0,1) \cdot (0,0,5) = 5,$$

$$(0,0,1) \cdot (1,1,3) = 3, \text{ and}$$

$$(0,0,1) \cdot (0,5,0) = 0.$$

This weighted order gives us

$$f = z^5 + 2xyz^3 + y^5.$$

Suppose instead we had $\omega = (1,0,0)$. A simple computation shows this weighted order gives us

$$\deg_\omega(z^5) = 0,$$

$$\deg_\omega(xyz^3) = 1, \text{ and}$$
$$\deg_\omega(y^5) = 0.$$

It is clear that xyz^3 is the leading monomial, and, according to Definition 2.2.1, we can choose a monomial order, say $<_{lex}$, to break the tie between z^5 and y^5. This would give us

$$f = 2xyz^3 + y^5 + z^5.$$

Since we have finitely many monomials, we could achieve this same outcome by adjusting ω in a way that orders all of the monomials and preserves xyz^3 as the leading monomial. For example, let $\hat{\omega} = (5,1,0)$, then

$$\deg_{\hat{\omega}}(z^5) = 0,$$
$$\deg_{\hat{\omega}}(xyz^3) = 6, \text{ and}$$
$$\deg_{\hat{\omega}}(y^5) = 5.$$

This weighted order gives us exactly the same ordering for f as ω using *lex* to break ties. For this reason, throughout the remainder of this work, we will look only at the weight vector and ignore the monomial order used to break ties, as the weight vector can always be adjusted if there are ties.

What does this new class of orders mean for Gröbner bases?

Example 2.2.2. Returning to Example 1.1.2, if we choose $\omega = (2,3)$, we find our polynomials ordered as in Equations (1.6) and (1.7). We then use those to compute a Gröbner basis. After making the polynomials monic, we have

$$G' = \{x^3 - 2xy, y^3 - \frac{1}{2}x^2y - \frac{1}{2}x\}.$$

Then, it is not hard to show that

$$S(g_2, g_1) \underset{G'}{\longrightarrow} 0,$$

so we get as our Gröbner basis

$$G = \{x^3 - 2xy, y^3 - \frac{1}{2}x^2y - \frac{1}{2}x\}.$$

This basis has only two elements as opposed to the four elements we obtained using grevlex order. It also only took one *S*-reduction as opposed to the six we did in Example 1.3.3! In fact, if we use the following Proposition, we would not need to do any reductions.

Proposition 2.2.1 (Proposition 4 of §2.9 in [5]). *Given a finite set $G \subset k[x_1, \ldots, x_n]$, suppose $f, g \in G$ such that the leading monomials of f and g are relatively prime. Then*

$$S(f,g) \underset{G}{\to} 0.$$

Example 2.2.2 shows that there are indeed orders that lead to a smaller basis computed with less work. As mentioned in Section 2.1, a dynamic Gröbner basis searches through the monomials of each new polynomial to identify a preferable leading monomial. Often the support of these polynomials is quite large; obviously the search would be more efficient if we could reduce the search space by removing monomials that are not compatible under any monomial order.

2.3 Background

Gritzmann and Sturmfels [12] described a technique that identifies incompatible leading monomials by determining whether a monomial's exponent vector lies within the polynomial's Newton polyhedron. This criterion is necessary and sufficient, but requires one to solve systems of linear constraints. Caboara [3] identified a simpler "indivisibility criterion":

Given monomials $u, t \in k[x_1, \ldots, x_n]$,

$$u \mid t \implies u \neq \mathrm{LM}(t+u).$$

Unfortunately, it does not identify all incompatible monomials. We have found a more general criterion:

Given monomials $u, t_1, \ldots, t_m \in k[x_1, \ldots, x_n]$,

$$u^m \mid t_1 \cdots t_m \implies u \neq \mathrm{LM}((t_1 + \cdots + t_m) + u).$$

You may notice that the first criterion is a special case of the second when $m = 1$.

If we return to Equation (2.1), we can easily see how this criteria applies. Notice that

$$xy \nmid x^2$$

and

$$xy \nmid y^2.$$

However, we have already shown that xy is not compatible for $f = x^2 + xy + y^2$. The indivisibility criteria failed to detect this, but

$$(xy)^2 \mid (x^2)(y^2),$$

so the new criteria finds that $xy \neq \mathrm{LM}(f)$.

The following Theorem is an adaptation of the indivisibility criterion and leads directly into the newer criterion.

Theorem 2.3.1. *For distinct monomials u,t, $u \mid t$ if and only if u is incompatible for $t+u$.*

From here on, we will write $\mathrm{DC}(t,u)$ to indicate that t and u are distinct monomials such that $u \mid t$. We will write only DC when t and u are clear from context.

Proof. Throughout the proof, we assume u,t are distinct.

Assume $u \mid t$. By definition of divides, there exists a monomial v such that $uv = t$. By Corollary 6 in §1.2 of [5], $1 \leq v$. By substitution and compatibility with multiplication, $u = 1 \cdot u \leq u \cdot v = t$. Finally, since u and t are distinct, we have

$$u < t.$$

Since we used an arbitrary monomial order $<$, u is incompatible for $t+u$ under every monomial order.

For the other direction, we prove the contrapositive. Assume $u \nmid t$, then there exists an indeterminate x_i with $\deg_{x_i}(u) > \deg_{x_i}(t)$. Choose any weighted order with $\omega_i > \omega_j$ for all $j \neq i$. Clearly this gives us

$$u = \mathrm{LM}(t+u),$$

so u is compatible for $t+u$. □

The first part of the proof of Theorem 2.3.1 extends naturally to larger polynomials; however, the second part is no longer true as soon as we move from binomials to trinomials. To see this, simply recall that we already showed that $xy \nmid y^2$, $xy \nmid x^2$, and $xy \neq \mathrm{LM}(x^2 + xy + y^2)$. It turns out that the indivisibility criterion is not necessary for any homogeneous polynomials.

The weighted orders defined in Section 2.2 allow us to define another criterion. Let L be a set of linear constraints in $y_1,\ldots,y_n$ that correspond to constraints on the exponents of monomials in $x_1,\ldots,x_n$. For example, $x_1^2 > x_1x_2$ corresponds to $2y_1 > y_1 + y_2$ or, equivalently, $y_1 - y_2 > 0$. The set of solutions to L is a polyhedral cone, C, whose edges we call *boundary vectors*. The boundary vector criterion was given in [4] as follows.

Theorem 2.3.2 (Corollary 1 of [4])**.** *Let L and C be as described above, $\sigma \in C$, $x^\alpha = \mathrm{LM}_\sigma(f)$ and $x^\beta \in \mathrm{Supp}(f)$ for some polynomial f. If $\omega \cdot \beta < \omega \cdot \alpha$ for every boundary vector $\omega \in C$, then x^β is incompatible for f.*

Theorem 2.3.2 is useful when we compute a Gröbner basis using a "restricted" dynamic algorithm, which ensures all previously-chosen leading monomials are unchanged using a linear program L. This is described in further detail in Chapter 3. For now, set a minimal $L = x_i \geq 0$ for all $i \in \{1,\ldots,n\}$ whose solution C consists of the canonical basis vectors. However, this criterion also fails to detect many incompatible monomials, including the one from our example. Notice that $xy > y^2$ for $\omega = (1,0)$, similarly, $xy > x^2$ for $\omega = (0,1)$. This shows that the boundary vector criterion fails to detect that xy is incompatible for x^2+xy+y^2.

Section 2.4 describes the new criterion which extends the indivisibility criterion to test a monomial u against an arbitrary number of distinct monomials.

2.4 Result

2.4.1 Main Thoerem

Theorem 2.4.1. *Let k be a positive integer. The following are equivalent.*

(a) $k \geq m$

(b) For all distinct monomials $u, t_1, \ldots, t_m$, if $u^k \mid t_1 \cdots t_m$, then u is incompatible for $(t_1 + \cdots + t_m) + u$.

We write $\mathrm{EDC}(t_1,\ldots,t_m,u,k)$ to indicate that $t_1,\ldots,t_m$, and u are distinct monomials and k is a positive integer such that $u^k \mid t_1 \cdots t_m$. We may write EDC if the arguments are obvious from context.

Proof. Corollary 2.4.3 presented below proves (a) $\implies$ (b), and Lemma 2.4.4 proves $\neg$(a) $\implies \neg$ (b). □

This theorem allows us to completely characterize all the powers of u that we need to check for divisibility, and, as we will show later, we can adapt the criterion in a "reductive" manner to test for more incompatible monomials.

2.4.2 Proof of Theorem 2.4.1

Throughout this section, let $f = (t_1 + \cdots + t_m) + u$ where $u, t_1, \ldots, t_m \in k[x_1,\ldots,x_n]$ are distinct monomials.

Lemma 2.4.2. $\mathrm{EDC}(t_1,\ldots,t_m,u,m)$ *implies that u is incompatible for f.*

Proof. Assume EDC; that is assume $u^m \mid t_1 \cdots t_m$. By Definition 0.0.1, we can write $u = \prod_{j=1}^{n} x_j^{\beta_j}$ and $t_i = \prod_{j=1}^{n} x_j^{\alpha_{ij}}$. By hypothesis, we have for each $j \in \{1, \ldots, n\}$,

$$m\beta^j \leq \sum_{i=1}^{m} \alpha_{ij}. \tag{2.4}$$

By way of contradiction, assume there exists a weighted order ω such that $u = \mathrm{LM}_\omega(f)$. Then, for each $i \in \{1, \ldots, m\}, \omega \cdot \beta > \omega \cdot \alpha_i$. In other words,

$$\sum_{j=1}^{n} \omega_j \beta_j > \sum_{j=1}^{n} \omega_j \alpha_{ij}. \tag{2.5}$$

We can multiply Inequality (2.5) by m to obtain

$$\sum_{j=1}^{n} \omega_j (m\beta_j) > m \sum_{j=1}^{n} \omega_j \alpha_{ij}.$$

Using Inequality (2.4), we can rewrite the left-hand side to get the following

$$\sum_{j=1}^{n} \omega_j \left(\sum_{i=1}^{m} \alpha_{ij} \right) > m \sum_{j=1}^{n} \omega_j \alpha_{ij}.$$

Since this is true for every i, we can sum the left- and right-hand sides for all m inequalities to obtain

$$m \sum_{j=1}^{n} \omega_j \left(\sum_{i=1}^{m} \alpha_{ij} \right) > \sum_{i=1}^{m} m \sum_{j=1}^{n} \omega_j \alpha_{ij},$$

which is clearly a contradiction. So, we have $u \neq \mathrm{LM}(f)$. □

Lemma 2.4.2 leads to the following Corollary.

Corollary 2.4.3. *For all $k \geq m$,* $\mathrm{EDC}(t_1, \ldots, t_m, u, k)$ *implies that u is incompatible for f.*

Proof. Assume $\mathrm{EDC}(t_1, \ldots, t_m, u, k)$ for some $k \geq m$. This means that $u^k \mid t_1 \cdots t_m$. By definition of divides, there exists $v \in k[x_1, \ldots, x_n]$ such that $u^k v = t_1 \cdots t_m$. We can rewrite u^k as $u^m u^{k-m}$. Then $u^m(u^{k-m}v) = t_1 \cdots t_m$. Again, by definition of divides, $u^m \mid t_1 \cdots t_m$. This gives us $\mathrm{EDC}(t_1, \ldots, t_m, u, m)$, so Theorem 2.4.2 applies telling us that u is incompatible for f. □

To complete the proof of Theorem 2.4.1, we look at the contrapositive of $(b) \implies (a)$. This means if $k < m$, then there exist $t_1, \ldots, t_m, u \in k[x_1, \ldots, x_n]$ and a monomial order $>$ such that $u^k \mid t_1 \cdots t_m$ and $LM_>(f) = u$. We can restate $u^k \mid t_1 \cdots t_m$ as $\mathrm{EDC}(t_1, \ldots, t_m, u, k)$.

Lemma 2.4.4. *For all $k < m$, there exist distinct monomials $t_1, \ldots, t_m, u$ and a monomial order $>$ such that* $\mathrm{EDC}(t_1, \ldots, t_m, u, k)$ *and* $\mathrm{LM}_>(f) = u$.

Proof. Suppose $k < m$. Let $\alpha = \frac{m(m+1)}{2}, u = x_1^\alpha, t_i = x_1^{\alpha-i}x_2^i$, and $\omega = (1, 0, \ldots, 0)$. Obviously, $u = \mathrm{LM}_\omega((t_1 + \cdots + t_m) + u)$. Notice that

$$\begin{aligned}\deg_{x_1}(t_1 \cdots t_m) = \sum_{i=1}^{m}(\alpha - i) &= \sum_{i=1}^{m}\left(\frac{m(m+1)}{2}\right) - \sum_{i=1}^{m} i \\ &= \frac{m^2(m+1)}{2} - \frac{m(m+1)}{2} \\ &= \frac{m(m+1)(m-1)}{2}.\end{aligned}$$

Since k is strictly less than m and they are both positive integers, we know that $k \leq m-1$. This means we also have

$$\begin{aligned}\deg_{x_1}(u^k) = k\alpha &= k \cdot \frac{m(m+1)}{2} \\ &\leq \frac{m(m-1)(m+1)}{2}\end{aligned}$$

By definition, $u^k \mid \prod t_i$ for every $k \in \{1, \ldots, m-1\}$. Therefore, we have $\mathrm{EDC}(t_1, \ldots, t_m, u, k)$. □

As stated early, along with Corollary 2.4.3, this completes the proof of Theorem 2.4.1.

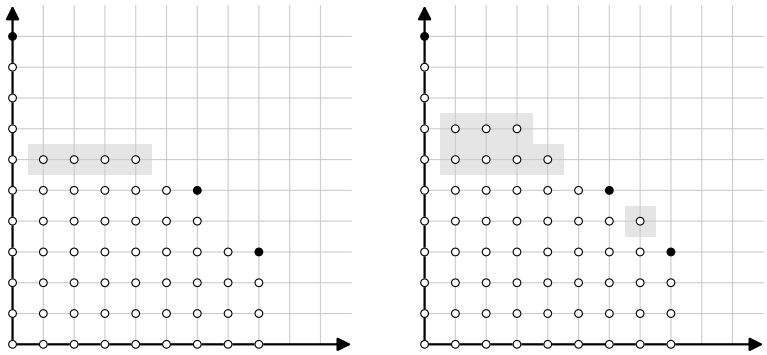
2.4.3 Application

To see the applicability of EDC, we will walk through an example.

Example 2.4.1. Let $t_1 = y^{10}, t_2 = x^6y^5$, and $t_3 = x^8y^3$. The solid dots in Figure 2.1(a) represent these three monomials, the hollow dots show the monomials detected by DC and EDC, and the highlighted dots represent the additional monomials detected by EDC. DC alone detects 52 incompatible monomials (those whose x- and y-values are less than or equal to each of the given monomials). EDC detects 4 additional incompatible monomials that DC did not. For example, xy^6 does not divide any of the given monomials, but

$$(xy^6)^3 \mid (y^{10}) \cdot (x^6y^5) \cdot (x^8y^3).$$

When EDC does not work with all of the monomials of a polynomial, we can still attempt it with a subset of them. Notice that if $u^m \nmid t_1 \cdots t_m$ but $u^{m-1} \mid t_1 \cdots t_{m-1}$, Theorem 2.4.1 says that u is incompatible for $(t_1 + \cdots + t_{m-1}) + u$, but this means that it is also incompatible for



(a) (b)

Figure 2.1: Monomial diagrams illustrating which incompatible monomials u are detectable by DC and EDC.

$(t_1 + \cdots + t_m) + u$. We can continue in this reductive manner with subsets of the Supp(f) until we end up with DC. This method sometimes allows us to find additional incompatible monomials.

Example 2.4.2. Let t_1, t_2, t_3 be as defined in Example 2.4.1. The highlighted dots in Figure 2.1(b) show that applying EDC to subsets of $\{t_1, t_2, t_3\}$ doubles the number of incompatible monomials detected. For example, if $u = x^7y^4$, $u^3 \nmid t_1t_2t_3$ so EDC does not detect that it is incompatible for $(t_1 + t_2 + t_3) + u$. However, if we reduce the monomial subset to $\{t_2, t_3\}$, we see that

$$(x^7y^4)^2 \mid (x^6y^5) \cdot (x^8y^3),$$

and EDC detects the incompatibility.

Similarly, for $u = xy^7$, $u^3 \nmid t_1t_2t_3$ but reducing the monomial subset to $\{t_1, t_2\}$ allows EDC to detect that u is incompatible since

$$(xy^7)^2 \mid (y^{10}) \cdot (x^6y^5).$$

Occasionally, EDC detects nothing that DC would not on its own. For example, given two monomials $t_1 = x$ and $t_2 = y^k$ for some positive integer k, if $u^2 \mid t_1t_2$, it must be true that $u \mid y^k$. This incompatible monomial would have been detected by DC.

Algorithm 3 presents one possible implementation of EDC that checks for it alongside DC. This algorithm also shows one possible implementation of the refinement to subsets of the Supp(f) as described in Example 2.4.2.

To see how the algorithm implements the refinement, we will walk through an example of Step 2(c).

Example 2.4.3. Let t_1, t_2, t_3 be as in Example 2.4.1 and $u = x^7y^4$.

Algorithm 3 Algorithm that detects incompatible monomials using both DC and EDC.

inputs f, a polynomial whose monomials are in $k[x_1,\ldots,x_n]$
outputs $P \subseteq \text{Supp}(f)$ such that $u \notin P$ implies u is incompatible for f

1. let $P = \text{Supp}(f)$
2. for $u \in \text{Supp}(f)$
   (a) let $T = \emptyset$
   (b) for $t \in \text{Supp}(f) \setminus \{u\}$
      i. if $u \mid t$ then remove u from P and quit this inner loop
      ii. else if $\gcd(u,t) \neq 1$ then
         A. add t to T
         B. if $u^{|T|} \mid \prod T$ then remove u from P and quit this inner loop
   (c) *(optional)* while u is in P and $|T| > 1$
      i. select an indeterminate x_i that maximizes $\deg_{x_i}(u^{|T|}) - \deg_{x_i}(\prod T)$
      ii. remove some $t \in T$ that minimizes $\deg_{x_i}(t)$, breaking ties by minimizing $\deg(\gcd(t,u))$
      iii. if $u^{|T|} \mid \prod T$ then remove u from P
3. return P

---

2. $u = x^7y^4$

   (c) $u \in P$ and $T = \{t_1, t_2, t_3\}$

      i. $\deg_x((x^7y^4)^3) - \deg_x(y^{10} \cdot x^6y^5 \cdot x^8y^3) = 21 - 14 = 7$
      $\deg_y((x^7y^4)^3) - \deg_y(y^{10} \cdot x^6y^5 \cdot x^8y^3) = 12 - 18 = -6$
      We choose x is this step since $7 > -6$.

      ii. $\deg_x(y^{10}) = 0$
      $\deg_x(x^6y^5) = 6$
      $\deg_x(x^8y^3) = 8$
      This step chooses t_1, since it has the minimal degree of x. It then removes t_1 from T, making $T = \{t_2, t_3\}$.

      iii. $(x^7y^4)^2 \mid (x^6y^5) \cdot (x^8y^3)$ so we remove u from the list of compatible monomials.

Unfortunately, the algorithm does not capture all situations where the refinement could be applied.

Example 2.4.4. Let $u = x^2yz, t_1 = x^2y^2, t_2 = rx^2z$, and $t_3 = x^2z^2$.

2. $u = x^2yz$

   (c) $u \in P$ and $T = \{t_1, t_2, t_3\}$

      i. $\deg_x((x^2yz)^3) - deg_x(x^2y^2 \cdot rx^2z \cdot x^2z^2) = 6 - 6 = 0$
      $\deg_y((x^2yz)^3) - deg_y(x^2y^2 \cdot rx^2z \cdot x^2z^2) = 3 - 2 = 1$
      $\deg_z((x^2yz)^3) - deg_z(x^2y^2 \cdot rx^2z \cdot x^2z^2) = 3 - 3 = 0$
      $\deg_r((x^2yz)^3) - deg_z(x^2y^2 \cdot rx^2z \cdot x^2z^2) = 0 - 3 = -3$
      We choose y in this step since $1 > 0 > -3$.

      ii. $\deg_y(x^2y^2) = 2$
      $\deg_y(rx^2z) = 0$
      $\deg_y(x^2z^2) = 0$
      Here, t_2 and t_3 are tied, so we look at the degree of the gcd of each with u:
      $\deg(\gcd(t_2, u)) = \deg(x^2z) = 3$
      $\deg(\gcd(t_3, u)) = \deg(x^2z) = 3$
      They are still tied. Suppose the algorithm chooses to remove t_3 making $T = \{t_1, t_2\}$.

      iii. $(x^2yz)^2 \nmid (x^2y^2) \cdot (rx^2z)$ so u gets returned as a compatible monomial.

However, if the algorithm had removed t_2 in Step 2(c)(ii), we would have gotten $(x^2yz)^2 \mid (x^2y^2) \cdot (x^2z^2)$ and seen that u is in fact NOT compatible.

Theorem 2.4.5. *Algorithm 3 terminates correctly with at most $\mathcal{O}(m^3n)$ additions and comparisons of exponents. Removing Step 2(c) allows it to terminate with at most $\mathcal{O}(m^2n)$ additions and comparison of exponents.*

Proof. We only remove monomials from P if they are proven to be incompatible, so we get correctness.

Step 2(b)(i) has complexity $\mathcal{O}(n)$ from the n comparisons of exponents needed to determine if $u \mid t$. Step 2(b)(ii)(A) has complexity of $\mathcal{O}(1)$ as we are only adding an element to T. Step 2(b)(ii)(B) has complexity of $\mathcal{O}(n)$, again due to the n comparisons of exponents needed to check division. This gives us an overall complexity of $\mathcal{O}(n)$ for Step 2(b)(ii). We loop through Step 2(b) at most $m-1$ times; once for each $t \in Supp(f) \setminus \{u\}$. This means that the worst case complexity for Step 2(b) is $\mathcal{O}(mn)$.

Step 2(c) is a little more complicated. Step 2(c)(iii) has complexity $\mathcal{O}(n)$ from determining if $u^{|T|} \mid \Pi T$. Step 2(c)(ii) has complexity $\mathcal{O}(mn)$ due to the use of the gcd to break ties. If

no ties need to be broken, this step only has complexity $\mathcal{O}(m)$. In Step 2(c)(i), we can reuse the previous computations of $u^{|T|}$ and ΠT, so this step has complexity $\mathcal{O}(n)$. We do this for every $u \in P$, which has worst case complexity $\mathcal{O}(m)$. Therefore, the overall complexity of Step 2(c) is $\mathcal{O}(m^2 n)$. We loop through Step 2 for every $u \in P$. At this point, it is easy to see that the overall complexity of the algorithm is $\mathcal{O}(m^3 n)$ if we implement the optional portion and $\mathcal{O}(m^2 n)$ otherwise. □

You might wonder how this compares to the complexity of the boundary vector criterion stated in Theorem 2.3.2. The worst case of the boundary vector criterion would be multiplying all ℓ vectors to each of the $m-1$ other monomials and comparing against the leading monomial's exponent vector of size n. Therefore, the worst case complexity of the boundary vector criterion is $\mathcal{O}(\ell mn)$. This would compare nicely to EDC without Step 2(c) if ℓ and m were of similar size. Unfortunately, in the majority of systems we have studied, ℓ is much smaller than m. In these instances, it would be more efficient to apply the boundary vector criterion and then apply DC and EDC only to the smaller subset of monomials remaining.

As mentioned in Section 2.3, we investigated this problem as a way to improve dynamic computation of Gröbner bases. Algorithm 3 was implemented in a system that uses a dynamic F4 algorithm to compute Gröbner bases [18]. The implementation was performed twice, once before and once after checking the boundary vector criterion. As discussed in the previous paragraph, we expect the "postcheck" to be more efficient that the "precheck" but tested both to see how many incompatible monomials DC and EDC were capable of detecting.

The results of the precheck implementation can be seen in Table 2.1. We see that DC detects the majority of incompatible monomials for inhomogeneous systems but EDC does contribute a small percentage more on all but one of the systems. However, as mentioned in Section 2.3, DC contributes nothing at all for homogeneous systems (those marked with "(h)" in the Table 2.1), but EDC is still able to detect quite a few incompatible monomials.

The postcheck implementation requires no table. When DC and EDC are implemented on these same systems *after* applying the boundary vector criterion, DC is able to detect only a handful of incompatible monomials in total and EDC only detects one in Cyclic-4 (and the same one in Cyclic-4(h)). The monomial EDC detects is precisely the one that led to this work and was seen in Example 2.1, although it is not in terms of x and y when seen in the cyclic systems.

As expected the precheck implementation of Algorithm 3 came at a high cost, especially in the homogeneous Cyclic-n systems which tend to have a large number of monomials and only a few boundary vectors. For example, homogeneous Cyclic-8 was given up on after

	Monomials detected via divisibility		
	DC	EDC	% new
Buchberger85	7	0	0.0
Butcher8	7170	76	1.0
Cyclic-4	8	1	11.1
Cyclic-4 (h)	0	1	100
Cyclic-5	260	50	16
Cyclic-5 (h)	0	31	100
Cyclic-6	5853	499	7.9
Cyclic-6 (h)	0	144	100
Cyclic-7	341165	22919	6.3
Cyclic-7 (h)	0	2686	100
Eco8	15811	763	4.6
Katsura-5	358	7	1.9
Kotsireas	24630	1393	5.4
Noon3	234	14	5.6
Noon5	46508	2058	4.2
s9-1	90	4	4.2
Trinks	85	3	3.4

Table 2.1: Number of incompatible monomials detected when testing Algorithm 3 before the boundary vector criterion ("precheck") in a dynamic F4 algorithm to compute Gröbner bases.

about ten minutes but is computed with the same algorithm using only the boundary vector criterion in around 40 seconds. Other systems saw less penalty for precheck but still reaped little benefit from using DC and EDC postcheck.

Chapter 3

Dynamic Gröbner Basis Computation

3.1 Introduction

As demonstrated in Chapter 2, the outcome of traditional Gröbner basis algorithms is dependent on the choice of monomial order, and some monomial orders lead to smaller bases than others. However, using a traditional algorithm, it is difficult to know which will be better without running the algorithm in its entirety on many different orderings. There are an infinite number of choices for monomial orders, so a brute force search can never guarantee that it has found the smallest Gröbner basis.

Researchers wondered if there was a method that could identify a "better" order. The authors of [17] investigated the Gröbner fan of an ideal. This fan looks at the polyhedral cones associated with sets of monomial orders, Σ, where for each $\sigma \in \Sigma$, the reduced Gröbner bases of the ideal with respect to σ are the same. They also proved that there are only finitely many such cones; however, the number of cones may be very large. They propose an algorithm that computes the basis for each of these fans independently. Of these, the computation that terminates first is said to yield the smallest Gröbner basis for the given ideal. They admit to having no way to determine which computation will terminate first before completion.

Say that the orders σ and τ are related if I has the same reduced Gröbner basis with respect to both σ and τ. It is easy to see that this is an equivalence relation and the previous paragraph's polyhedral cones are the equivalence classes. The reader likely noticed that, for a fixed ideal, there are finitely many equivalence classes of monomials orders. This also means that there are finitely many Gröbner bases. Instead of a brute force search on "all" the monomial orders, we could compute the Gröbner fan, then compute the Gröbner basis for one monomial order in each equivalence class. However, this is also infeasible, as computing a Gröbner fan consumes significant amounts of time and memory.

In 1993, [3] and [12] independently proposed a dynamic Buchberger algorithm that allows the monomial order to vary at the start of each S-polynomial loop if doing so would lead to a smaller Gröbner basis. They minimize the Hilbert function in an attempt to identify this "preferable" order.

Definition 3.1.1 (Definition 2 of §9.3 in [5])**.** Let $I \in k[x_1,\ldots,x_n]$ be an ideal. Let $R_s = k[x_1,\ldots,x_n]_s$ represent the set of polynomials with degree s in $k[x_1,\ldots,x_n]$ and I_s represent the set of polynomials in I with degree s. The **Hilbert function** is the function

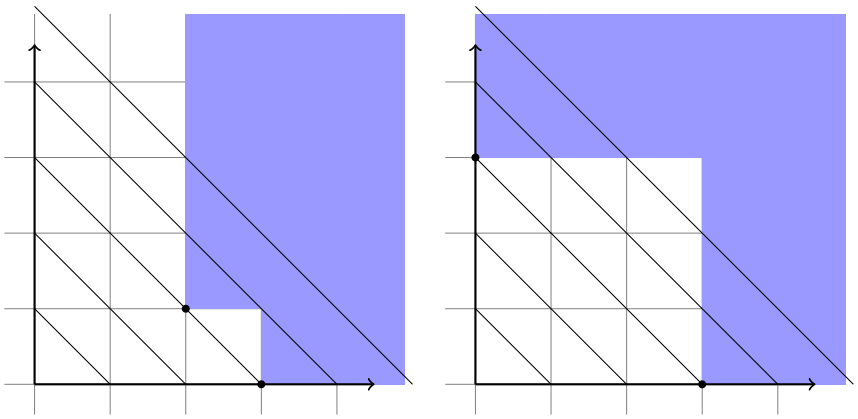
$$HF_{R/I}(s) = \dim R_s/I_s = \dim R_s - \dim I_s.$$

By Definition 3.1.1 and Proposition 3 in §9.3 of [5], we know that given an ideal of monomials in $k[x_1,\ldots,x_n]$, the Hilbert function identifies the number of monomials at each degree that are not in the ideal. By Definition 1.3.1, we know that we have a Gröbner basis for an ideal I when the ideal of leading terms in the basis equals the ideal of leading terms in I.

Example 3.1.1. Let $R = k[x_1,\ldots,x_n]$. Recall from Examples 1.3.3 and 2.2.2 that for the ideal $I = \langle x^3 - 2xy, x^2y - 2y^3 + x\rangle$, we looked at two options for leading monomials. The monomial ideals associated with those are $\langle x^3, x^2y\rangle$ and $\langle x^3, y^3\rangle$ for grevlex order and the weighted order $\omega = (2,3)$, respectively. Figure 3.1 is a geometric interpretation of the Hilbert function for the quotient rings (a) $R/\langle x^3, x^2y\rangle$ and (b) $R/\langle x^3, y^3\rangle$. The quotient rings are represented by the unshaded region, and a monomial $x^\alpha y^\beta$ is represented as an ordered pair (α,β), as described in Definition 0.0.1. The diagonal lines show each degree of the diagram. For example, at degree 1, the diagonal line touches x $(1,0)$ and y $(0,1)$, while at degree 2, the diagonal line touches x^2 $(2,0)$, xy $(1,1)$, and y^2 $(0,2)$. This allows us to more easily count the number of monomials at each degree.

To find the Hilbert function, we will count how many monomials are in the quotient ring R/I at each degree. In Figure 3.1(a), we see for $R/\langle x^3, x^2y\rangle$, we have 1 monomial outside the shaded region as degree 0, 2 at degree 1, 3 at degree 2, 2 at degree 3, and 2 at each degree beyond 3. Notice that we do not count monomials on the edge of the shaded region as they are part of R/I. The corresponding Hilbert function is $(1,2,3,2,2,2,\ldots)$. In a similar fashion, in Figure 3.1(b), we find the Hilbert function for $R/\langle x^3, y^3\rangle$ to be $(1,2,3,2,1,0,\ldots)$. In this case, minimizing the Hilbert function means minimizing the number of monomials outside the ideal once the function stabilizes. The grevlex order starts adding 2 additional monomials for each degree over 3, while the weighted order adds no additional monomials after degree 5. This suggests that the weighted ordering should be 'closer' to a Gröbner basis, so using this ordering should produce a 'smaller' basis. As we saw in Chapter 2, this is in fact the case.

Moreover, Proposition 1.9 of [3] tells us that the Hilbert function for a polynomial ideal I is exactly the Hilbert function for the ideal generated by the leading monomials of I. The Hilbert function is invariant for homogeneous ideals; that is, it has a fixed value regardless



(a) (b)

Figure 3.1: Monomial diagrams illustrating the Hilbert function for $\langle x^3, x^2y \rangle$ (a) and $\langle x^3, y^3 \rangle$ (b).

of the monomial ordering [3]. When the Buchberger algorithm adds a new polynomial to the basis, its leading monomial is not divisible by any leading monomials already in the basis. This shrinks the unshaded regions of the monomial diagrams in Figure 3.1, leaving fewer options of new leading monomials for the algorithm to discover. Therefore, minimizing the Hilbert function should put us closer to a Gröbner basis.

Caboara also discusses that while the Hilbert function is invariant for homogeneous ideals, it varies according to the monomial ordering for inhomogeneous ideals. The following example demonstrates the meaning of this.

Example 3.1.2. Let $I = \langle x^2+z^2, y^2-z^2 \rangle$, $J = \langle x^2+z, y^2-z^2 \rangle$ be ideals over $R = k[x_1, \ldots, x_n]$. Notice that I is homogeneous and J is inhomogeneous. Let $\omega = (3,2,1)$ and $\nu = (1,4,3)$ be weight orderings. Then, by Definition 1.3.1, we have

$$\mathrm{LM}(I)_\omega = \{x^2, y^2\},$$

$$\mathrm{LM}(I)_\nu = \{z^2, y^2\},$$

$$\mathrm{LM}(J)_\omega = \{x^2, y^2\},$$

and

$$\mathrm{LM}(J)_\nu = \{z, y^2\}.$$

By Proposition 2.2.1, we know that under both of these orderings, the generators of I and J are themselves Gröbner bases.

To determine the Hilbert functions for each of these, we will again determine how many monomials at each degree are outside R/I. Here we are referring to the standard degree

from Definition 0.0.1. We show those monomials for through degree 3 in the following table. Since $\mathrm{LM}(I)_\omega = \mathrm{LM}(J)_\omega$, we only include $\mathrm{LM}(I)_\omega$ in the table.

	$\mathrm{LM}(I)_\omega$	$\mathrm{LM}(I)_\nu$	$\mathrm{LM}(J)_\nu$
Degree 0	1	1	1
Degree 1	x, y, z	x, y, z	x, y
Degree 2	xy, xz, yz, z^2	xy, xz, yz, x^2	xy, x^2
Degree 3	xyz, xz^2, yz^2, z^3	xyz, x^2y, x^2z, x^3	x^2y, x^3

Using this table, we can see that the homogeneous ideal I has

$$HF_{R/I} = (1, 3, 4, 4, \ldots)$$

for both weight orderings. However, the inhomogeneous ideal J under ω has

$$HF_{R/J} = (1, 3, 4, 4, \ldots)$$

while, under ν, it has

$$HF_{R/J} = (1, 2, 2, 2, \ldots).$$

Research has been done into two different categories of dynamic algorithms. Following the lead of [11], we call an algorithm that preserves previously chosen leading monomials throughout computation a **restricted** algorithm and an algorithm that allows previous choices to be "undone" an **unrestricted** algorithm. To our knowledge, little work outside of [11], [12], and [15] has been done into unrestricted algorithms. In [15], the author investigates the idea of a neighborhood for monomial orders. He then evaluates neighboring orders using a local search and allowing previously chosen leading monomials to change if doing so might lead to a smaller basis. Although this work is interesting, we will focus on restricted dynamic algorithms.

3.2 Dynamic Buchberger Algorithms

3.2.1 Prior Work on Dynamic Buchberger Algorithms

A number of researchers have investigated techniques for improving the dynamic Buchberger algorithm from [3]. The boundary vector criterion from [4] described in Chapter 2 reduces the number of linear programs that need to be solved by discarding incompatible leading monomials. This same paper describes a method for reducing the number of constraints in each program by only adding constraints for monomials that are compatible for the polynomial. These two refinements greatly reduce the overhead for the dynamic Buchberger

algorithm. In addition, the original algorithm from [3] had so many linear programs and constraints that they grew unmanageable, and the algorithm was forced to stop adding additional constraints and continue as a static algorithm until termination. The boundary vector criterion allows the algorithm to continue in a dynamic fashion until termination.

Alternative heuristics to the Hilbert heuristic described in 3.1 were explored in [19]. The author investigates four possible heuristics. The first is the Hilbert heuristic, which is based on the use of the Hilbert function in [12]. This heuristic requires us to define the Hilbert polynomial, the Hilbert series, and the Hilbert numerator.

Theorem 3.2.1 (Theorem 5.1.21 of [13])**.** *Let $R = k[x_1, \ldots, x_n]$, I be an ideal, and R/I be their quotient ring. Then the Hilbert function $HF_{R/I} : \mathbb{Z} \to \mathbb{Z}$ is an integer function of polynomial type.*

Definition 3.2.1 (Definition 5 of §9.3 in [5])**.** The **Hilbert polynomial** of the quotient ring R/I is the polynomial which equals $HF_{R/I}(s)$ for sufficiently large s. It is written $HP_{R/I}(s)$.

We can use well-known tool from discrete mathematics to rewrite any sequence

$$(a_0, a_1, a_2, \ldots)$$

as a power series

$$A = a_0 + a_1 x + a_2 x^2 + \ldots$$

which is called the *generating function* for $(a_0, a_1, a_2, \ldots)$. A thorough discussion on this can be found in Chapter 49 of [20], which is available online.

Definition 3.2.2 (Definition 5.2.13 of [13])**.** The **Hilbert series** of R/I is the formal power series associated with the Hilbert function of R/I. In other words,

$$HS_{R/I}(z) = \sum_{i \geq 0} HF_{R/I}(i) z^i.$$

Definition 3.2.3 (Definition 5.2.21 of [13])**.** Given a quotient ring R/I and Hilbert series

$$HS_{R/I}(z) = \frac{HN_{R/I}(z)}{(1-z)^n},$$

the polynomial $HN_{R/I}(z)$ is called the **Hilbert numerator** of R/I. (Here, the n in the denominator refers to the number of variables in R.)

Example 3.2.1. Let R be as in Example 3.1.1. We will also use the same ideals of leading monomials, $\langle x^3, x^2 y \rangle$ and $\langle x^3, y^3 \rangle$.

We saw in the previous example that for $s \geq 3$,

$$HF_{R/\langle x^3,x^2y\rangle}(s) = 2,$$

and for $s \geq 5$,

$$HF_{R/\langle x^3,y^3\rangle}(s) = 0.$$

The corresponding Hilbert polynomials are then

$$HP_{R/\langle x^3,x^2y\rangle} = 2$$

and

$$HP_{R/\langle x^3,y^3\rangle} = 0.$$

When $|z| < 1$, the corresponding Hilbert series are

$$HS_{R/\langle x^3,x^2y\rangle} = 1 + 2z + 3z^2 + 2z^3 + \sum_{i\geq 4} 2z^i \tag{3.1}$$

and

$$HS_{R/\langle x^3,y^3\rangle} = 1 + 2z + 3z^2 + 2z^3 + z^4. \tag{3.2}$$

Notice in Equation 3.1, we can rewrite $\sum_{i\geq 4} 2z^i$ as

$$\sum_{i=0}^{\infty} 2z^i - \sum_{i=0}^{3} 2z^i,$$

where the first term is a geometric series. This allows us to rewrite the Hilbert series as

$$\begin{aligned}
HS_{R/\langle x^3,x^2y\rangle} &= 1 + 2z + 3z^2 + 2z^3 + \sum_{i\geq 4} 2z^i \\
&= 1 + 2z + 3z^2 + 2z^3 + \sum_{i=0}^{\infty} 2z^i - \sum_{i=0}^{3} 2z^i \\
&= 1 + 2z + 3z^2 + 2z^3 + \sum_{i=0}^{\infty} 2z^i - 2 - 2z - 2z^2 - 2z^3 \\
&= \frac{2}{1-z} + z^2 - 1 \\
&= \frac{2(1-z) + (z^2-1)(1-z)^2}{(1-z)^2} \\
&= \frac{z^4 - 2z^3 + 1}{(1-z)^2}.
\end{aligned}$$

Equation 3.2 is easier to rewrite, as we only need to multiply by $\frac{(1-z)^2}{(1-z)^2}$,

$$\begin{aligned} HS_{R/\langle x^3,y^3\rangle} &= 1+2z+3z^2+2z^3+z^4 \\ &= \frac{(1+2z+3z^2+2z^3+z^4)(1-z)^2}{(1-z)^2} \\ &= \frac{z^6-2z^3+1}{(1-z)^2} \end{aligned}$$

At this point, it is clear that we have

$$HN_{R/\langle x^3,x^2y\rangle} = z^4-2z^3+1$$

and

$$HN_{R/\langle x^3,y^3\rangle} = z^6-2z^3+1.$$

Now, that we understand the meaning of a Hilbert polynomial and a Hilbert numerator, we can define the first heuristic from [19].

Definition 3.2.4. Let $G=\{g_1,\ldots,g_\ell\}$ be the current basis elements, f be the polynomial we wish to add to the basis, and t_1,t_2 be possible leading monomials of f. Let h_i be the Hilbert numerator of $\langle \mathrm{LM}(G)\cup\{t_i\}\rangle$ and p_i be the Hilbert polynomial of $\langle \mathrm{LM}(G)\cup\{t_i\}\rangle$. For tuples $(h_1,p_1),(h_2,p_2)$, the **Hilbert heuristic** chooses $t_1=\mathrm{LM}(f)$ if

(i) $\mathrm{LC}(p_1-p_2)<0$, or

(ii) $p_1=p_2$ and the "trailing" term of h_1-h_2 is positive.

In part (ii) of the definition, trailing term means the term of smallest degree. If we use this heuristic with Example 3.2.1 for $t_1=x^2y, t_2=y^3$, we see that

$$\mathrm{LC}(p_2-p_1)=0-2<0.$$

Therefore, the Hilbert heuristic would choose y^3 to be the leading term.

The next heuristic is the Betti heuristic which attempts to minimize the number of S-polynomials the algorithm computes by minimizing the number of critical pairs generated.

Definition 3.2.5. Let G,f,t_1,t_2 be as in Definition 3.2.4. Let P_i be the set of critical pairs queued to be computed after potentially choosing $t_i=\mathrm{LM}(f)$ and adding the new pairs associated with this choice. The **Betti heuristic** chooses $t_1=\mathrm{LM}(f)$ if

$$|P_1|<|P_2|.$$

The final two heuristics are the "graded" Hilbert and Betti heuristics. These are simply the previous two heuristics but using the current monomial order instead of the standard degree. The results of implementing these new heuristics show that the graded heuristics are generally outperformed by their "ungraded" counterparts. The standard Betti heuristic however shows promise in some systems and is therefore recognized as a good alternative to the standard Hilbert heuristic.

Algorithm 4 follows the example of Algorithm 1 in [19] and shows one possible implementation of a dynamic Buchberger algorithm. Section 3.2.2 walks through an example computation using the algorithm. Step 2(a) of the algorithm indicates that we should choose a pair with "minimal sugar." This strategy for choosing which S-polynomial to compute comes from [10].

Definition 3.2.6 (From Section 2 of [10])**.** Given a polynomial f, its **sugar** S_f is defined in the following way:

(i) for the initial f_i, $S_{f_i} = \text{tdeg}(f)$ (this is the degree from Definition 1.1.1 not the degree with respect to the monomial order);

(ii) if f is a polynomial and $t = x^\alpha$ a monomial, then $S_{tf} = |\alpha| + S_f$;

(iii) if $f = g + h$, then $S_f = \max(S_g, S_h)$.

If the sugar strategy fails, we will use the normal selection strategy, meaning choose the pair (p,q) with lowest degree of $t_{p,q}$.

3.2.2 Dynamic Buchberger Algorithm Example

For our example, we will use one of the famous Cyclic-n systems, specifically Cyclic-4h where h indicates that the system is homogenized. Throughout the example, we will make the polynomials found in Steps 2(b) and 2(c)(iii) monic for ease of computation.

Example 3.2.2. Let

$$\begin{aligned} f_1 &= x_1 + x_2 + x_3 + x_4 \\ f_2 &= x_1x_2 + x_2x_3 + x_3x_4 + x_1x_4 \\ f_3 &= x_1x_2x_3 + x_2x_3x_4 + x_1x_3x_4 + x_1x_2x_4 \\ f_4 &= x_1x_2x_3x_4 - h^4 \end{aligned}$$

and

$$I = \langle f_1, f_2, f_3, f_4 \rangle.$$

Algorithm 4 A Dynamic Buchberger Algorithm

Input: $I = \langle f_1, \ldots, f_s \rangle$ and a heuristic H
Output: a weighted order ω and a Gröbner basis for I with respect to $>_\omega$, $G = \{g_1, \ldots, g_t\}$, with $I \subseteq G$

1. **let** $G = \emptyset$
   $P = \{(f, 0) \mid f \in I\}$
   $\omega = (1, \ldots, 1)$
2. **while** $P \neq \emptyset$
   (a) choose $(p, q) \in P$ with minimal sugar and remove it
   (b) $S(p, q) \underset{G}{\to} r$
   (c) **if** $r \neq 0$ **then**
      i. **let** $T = \{t_1, \ldots, t_\ell\}$ for $t_k \in \text{Supp}(r)$ such that t_k is compatible for r
      ii. choose t such that $H(t)$ is minimized
      iii. update ω so that $t = \text{LM}(r)$ and $\{\text{LM}(g) \mid g \in G\}$ is constant
      iv. add (r, g_i) to P for all $g_i \in G$
      v. update G and P using ω
      vi. $G = G \cup \{r\}$
3. **return** G, ω

For H, we will use the Hilbert heuristic described in [19]. If there are ties, we will break them by choosing the term that is minimal under grevlex order.

1. $G = \emptyset$
   $P = \{(f_1, 0), (f_2, 0), (f_3, 0), (f_4, 0)\}$
   $\omega = (1, 1, 1, 1, 1)$

Loop 1:

2. $P \neq \emptyset$
   (a) $S(f_1, 0) = f_1$ has minimal sugar of 1
   (b) G is empty, so $r = f_1$
   (c) $r \neq 0$
      i. $T = \{x_1, x_2, x_3, x_4\}$

ii. Unfortunately, all four Hilbert polynomials and all four Hilbert numerators are equal. We choose x_4 as the smallest monomial with respect to grevlex order.

iii. $\omega = (1,1,1,2,1)$

iv. G is empty, so we skip this step

v. Throughout this example, we will neglect to show the updated elements of P and previous elements of G, but you will encounter the updated polynomials in other loops or when the basis is returned.

vi. $G = \{x_4 + x_1 + x_2 + x_3\}$

Loop 2:

2. (a) $S(f_2, 0) = x_1x_4 + x_3x_4 + x_1x_2 + x_2x_3$ has minimal sugar of 2

   (b) $r = x_1^2 + 2x_1x_3 + x_3^2$

   (c) i. $T = \{x_1^2, x_3^2\}$
   (We are able to remove x_1x_3 by Theorem 2.4.1)

   ii. Again, both Hilbert polynomials and both Hilbert numerators are equal. The smallest monomial under grevlex order is x_3^2.

   iii. $\omega = (1,1,2,3,1)$

   iv. By Proposition 2.2.1, we do not need to compute $S(r, g_1)$. We have $P = \{(f_3, 0), (f_4, 0)\}$

   vi. $G = G \cup \{x_3^2 + 2x_1x_3 + x_1^2\}$

Loop 3:

2. (a) $S(f_3, 0) = x_1x_3x_4 + x_2x_3x_4 + x_1x_2x_4 + x_1x_2x_3$ has minimal sugar of 3

   (b) $r = x_1^2x_3 - x_2^2x_3 + x_1^3 - x_1x_2^2$

   (c) i. $T = \{x_1^2x_3, x_2^2x_3\}$

   ii. Again, both Hilbert polynomials and both Hilbert numerators are equal. The smallest monomial under grevlex order is $x_2^2x_3$.

   iii. update $\omega = (1,2,2,3,1)$

   iv. $P = \{(f_4, 0), (r, g_2)\}$

   vi. $G = G \cup \{x_2^2x_3 + x_1x_2^2 - x_1^2x_3 - x_1^3\}$

Loop 4:

2. (a) $S_{S(g_3,g_2)} = 4$ and $S_{f_4} = 4$. The sugar strategy has failed us, so we choose (g_3, g_2), the pair with lowest degree.

   (b) $r = 0$

Loop 5:

2. (a) $(f_4, 0)$ is the only element in P

   (b) $r = x_1^2x_2^2 + x_1^2x_2x_3 + x_1^3x_2 - x_1^3x_3 - x_1^4 - h^4$

   (c) i. $T = \{x_1^2x_2^2, x_1^2x_2x_3, h^4\}$. We will call these t_1, t_2, t_3.

   ii. $h_1 = -t^6 + t^5 + t^4 - t^2 - t + 1 \qquad p_1 = 6t - 3$
   $h_2 = t^7 - 3t^6 + 2t^5 + t^4 - t^2 - t + 1 \qquad p_2 = \frac{1}{2}t^2 + \frac{5}{2}t + 3$
   $h_3 = t^9 - 2t^8 + t^6 + t^4 - t^2 - t + 1 \qquad p_3 = -4t - 6$

   $\mathrm{LT}(p_3 - p_1) = -10t$ and all others are positive, so we choose h^4 as the $\mathrm{LM}(r)$.

   iii. update $\omega = (1,2,2,3,7)$

   iv. We add no new pairs by Proposition 2.2.1, so $P = \emptyset$

   vi. $G = G \cup \{h^4 - x_1^2x_2^2 - x_1^2x_2x_3 - x_1^3x_2 + x_1^3x_3 + x_1^4\}$

3. **return** G, ω

So, we have found a Gröbner basis,

$$G = \{x_4 + x_2 + x_3 + x_1, x_3^2 + 2x_1x_3 + x_1^2, \\ x_2^2x_3 + x_1x_2^2 - x_1^2x_3 - x_1^3, h^4 - x_1^2x_2^2 - x_1^2x_2x_3 - x_1^3x_2 + x_1^3x_3 + x_1^4\},$$

for $\omega = (1,2,2,3,7)$ by computing only 5 S-polynomials, and the basis contains just 4 elements. To put this in perspective, the static Buchberger algorithm using grevlex order computes 12 S-polynomials and returns a Gröbner basis with 7 elements.

3.3 A Dynamic F4 Algorithm

3.3.1 Background

The F4 algorithm was first presented by Faugére in [9]. He used notions from linear algebra to relate the polynomial ring, $k[x_1, \ldots, x_n]$, to a vector space and the Buchberger algorithm to Gaussian elimination using the Macaulay matrix described by [16], which is a generalization of the Sylvester matrix. In the F4 algorithm, the columns of a matrix M correspond to

the monomials in $\mathbb{M}$ in descending order, according to some monomial order. Faugére recommends using grevlex order as it is well known to terminate with a Gröbner basis at a lower degree than other orders. Given an ideal $I = \langle f_1, \ldots, f_m \rangle$, the rows of M correspond to the coefficients of the polynomials f_i in $B \subseteq I$, where $\deg(f_i) = \deg(f_j)$ for all $f_i, f_j \in B$ *as well as* the coefficients of the multiples mf_i with $m \in \mathbb{M}, f_i \in B$. Clearly, this is an infinite matrix. However, if we are restricted to finite submatrices, it becomes more manageable. In [9], Faugére showed that this method was able to compute Gröbner bases for polynomial ideals that were previously intractable.

Algorithm 5 from Proposition 1 in §10.3 of [5] presents a method for computing such a matrix with the following properties:

(i) $L \subseteq H$

(ii) if we have a polynomial $f \in H$ with $x^\alpha \in \text{Supp}(f)$ and there exists $g_\ell \in G$ such that $g_\ell \mid x^\alpha$, then $\dfrac{x^\alpha}{\text{LM}(g_\ell)} g_\ell \in H$.

In the algorithm, $Mon(H)$ is the set of all monomials in the polynomials of H, meaning for every $f \in H$, we have $\text{Supp}(f) \in Mon(H)$.

Algorithm 5 ComputeM: An Algorithm to Compute a Submatrix for F4

Input: L, a set of polynomials, and $G = \{g_1, \ldots, g_t\}$, the current basis
Output: a matrix M

1. **let** $H = L$
   $done = \text{LM}(H)$
2. **while** $done \neq Mon(H)$
   (a) choose the largest $x^\alpha \in Mon(H) \setminus done$
   (b) $done = done \cup \{x^\alpha\}$
   (c) **if** $\text{LM}(g_\ell) \mid x^\alpha$ for some $g_\ell \in G$ **then**
      i. select one such g_ℓ
      ii. $H = H \cup \left\{ \text{Supp}\left(\dfrac{x^\alpha}{\text{LM}(g_\ell)} \cdot g_\ell \right) \right\}$
3. **return** M, the matrix of coefficients of H with respect to $Mon(H)$ columns

---

In order to show that Algorithm 5 terminates correctly, we will need the following lemma regarding well-orderings.

Lemma 3.3.1 (Lemma 2 of §2.2 in [5]). *An order relation $>$ on $\mathbb{Z}^n_{\geq 0}$ is a well-ordering if and only if every strictly decreasing series*

$$\alpha(1) > \alpha(2) > \alpha(3) > \cdots$$

eventually terminates.

Now that we have necessary lemma, we give the following proposition and its proof.

Proposition 3.3.2 (Proposition 1 of §10.3 in [5]). *Algorithm 5 terminates correctly with the two properties mentioned above.*

Proof. Fix a monomial order $>$. For termination, we must show that we eventually have $done = Mon(H)$. First notice that after x^α is chosen in Step 2(a), we append it to *done* in Step 2(b). This means that this monomial is not considered again. In Step 2(c), if we find a $g_\ell \in G$ such that $\mathrm{LM}(g_\ell) \mid x^\alpha$, we add to H the monomials in $\mathrm{Supp}(f)$, where $f = \left(\dfrac{x^\alpha}{\mathrm{LM}(g_\ell)} \cdot g_\ell\right)$. Notice that the leading monomial of f is x^α, then any other monomial in $\mathrm{Supp}(f)$ is smaller under $>$ than x^α. It follows that the x^α chosen successively in Step 2(a) form a strictly decreasing sequence under $>$. By Lemma 3.3.1, we will reach a point where no new monomials are included in H. It will eventually be true that $done = Mon(H)$, so ComputeM will terminate.

Correctness is straightforward. By Step 1, we know we will achieve $L \subseteq H$, and Step 2(c) gives us property (ii). □

Now that we have an algorithm for computing the submatrices, we can define the F4 algorithm. Algorithm 6 comes from §10.3 of [5] and is an algorithm for the entire F4 family, including the variants that came out of [9].

Section 3.3.2 walks through a small portion of an example of Algorithm 6 on the ideal $I = \langle x^2+xy-1, x^2-z^2, xy+1\rangle$ using grevlex order.

3.3.2 Example of Gröbner Basis Computation Using the F4 Algorithm

For ease of reading, we will say $f_1 = x^2+xy-1$, $f_2 = x^2-z^2$, and $f_3 = xy+1$. Suppose we have already completed 2 loops through Step 2 and now also have $f_4 = z^2-2$ and $f_5 = x+2y$. We also currently have $G = \{f_1, f_2, f_3, f_4, f_5\}$ and $P = \{(f_4,f_1),(f_4,f_2),(f_4,f_3),(f_5,f_1),(f_5,f_2),(f_5,f_3),(f_5,f_4)\}$. We will start at the beginning of Step 2.

2. (a) $P' = \{(f_5,f_1),(f_5,f_2),(f_5,f_3)\}$, all of which have $\deg(t_{f_i,f_j}) = 2$.

   (b) $P = \{(f_4,f_1),(f_4,f_2),(f_4,f_3),(f_5,f_4)\}$

Algorithm 6 A Basic F4 Algorithm

Input: an ideal $I = \langle f_1, \ldots, f_s \rangle$
Output: G, a Gröbner basis for I

1. **let** $G = \{f_1, \ldots, f_s\}$
   $P = \{(f_i, f_j) \mid 1 \leq j < i \leq s\}$
2. **while** $P \neq \emptyset$
   (a) choose $P' \subseteq P$, such that $P' \neq \emptyset$ and t_{f_i,f_j} is minimal for each $(f_i, f_j) \in P'$
   (b) $P = P \setminus P'$
   (c) $L = \left\{ \dfrac{t_{f_i,f_j}}{t_{f_j}} \cdot f_j \mid (f_i, f_j) \in P' \right\}$
   (d) $M = \text{ComputeM}(L, G)$
   (e) $N =$ row reduced echelon form of M
   (f) $N^+ = \{n \in \mathit{rows}(N) \mid \text{LM}(n) \notin \langle \text{LM}(\mathit{rows}(M)) \rangle\}$
   (g) **for** $n \in N^+$ **do**
      i. $g =$ polynomial form of n
      ii. $G = G \cup \{g\}$
      iii. $P = P \cup \{(g, f_i) \mid 1 \leq i < |G|\}$
3. **return** G

(c) $L = \left\{ \dfrac{x^2}{x} \cdot f_5, \dfrac{x^2}{x^2} \cdot f_1, \dfrac{x^2}{x^2} \cdot f_2, \dfrac{xy}{x} \cdot f_5, \dfrac{xy}{xy} \cdot f_3 \right\} = \{xf_4, f_1, f_2, yf_5, f_3\}$

We will take a moment to look at what happens in ComputeM(L, G).

1. $H = \{xf_4, f_1, f_2, yf_5, f_3\}$
   Notice that $Mon(H) = \{x^2, xy, y^2, z^2, 1\}$.
   $done = \{x^2, xy\}$

Loop 1:

2. (a) y^2 is the largest monomial in $Mon(H) \setminus done$
   (b) $done = done \cup \{y^2\}$
   (c) $x^2, xy, z^2, z \nmid y^2$, so we are done with this loop

Loop 2:

2. (a) z^2 is the largest monomial in $Mon(H) \setminus done$

   (b) $done = done \cup \{z^2\}$

   (c) $z^2 \mid z^2$

      i. $f_4 = z^2 - 2$

      ii. $H = H \cup \left\{ \frac{z^2}{z^2} \cdot f_4 \right\}$ We are just adding f_4 to H, and $Mon(H)$ remains unchanged since $z^2, 1$ were already in it.

Loop 3:

2. (a) 1 is the only monomial in $Mon(H) \setminus done$

   (b) $done = done \cup \{1\}$

   (c) $x^2, xy, z^2, z \nmid 1$, so we are done with this loop

   $done = Mon(H)$ so we have finished

3.
$$M = \begin{array}{c} \\ yf_5 \\ xf_5 \\ f_4 \\ f_3 \\ f_2 \\ f_1 \end{array} \begin{array}{c} \begin{array}{ccccc} x^2 & xy & y^2 & z^2 & 1 \end{array} \\ \begin{pmatrix} 0 & 1 & 2 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 & -1 \end{pmatrix} \end{array}$$

Now, we return to our F4 algorithm.

2. (d) $M = \text{ComputeM}(L, G)$

   (e)
$$N = \begin{array}{c} \\ f_2 \\ f_3 \\ yf_5 \\ xf_5 \\ f_4 \\ f_1 \end{array} \begin{array}{c} \begin{array}{ccccc} x^2 & xy & y^2 & z^2 & 1 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & -1 \\ 0 & 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

   (f) $\langle \text{LM}(rows(M)) \rangle = \langle x^2, xy, z^2 \rangle$, so we get $N^+ = \{[0,0,2,0,-1]\}$

   (g) i. $g = 2y^2 - 1$

      ii. $G = G \cup \{g\}$

iii. $P = P \cup \{(g, f_5), (g, f_4), (g, f_3), (g, f_2), (g, f_1)\}$

We will end the example at this point. There are two more loops needed to complete the algorithm; however, neither of them add any new elements to G. This means that the G at the end of our example is the Gröbner basis for I under grevlex order.

3.3.3 A Dynamic F4 Algorithm

Unlike dynamic Buchberger algorithms, it seems that little work has been done into a dynamic F4 algorithm. However, in [18], Perry discusses the advantages of such an algorithm. From the linear algebra perspective, consider the matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Gaussian reduction would take $n-1$ row reductions. Notice that if we use column swapping, we can obtain a matrix in row-echelon form with $\left\lfloor \frac{n}{2} \right\rfloor$ column swaps. In this example matrix 3 versus 2 is not much different in terms of work. However, as the number of rows grows, it would be a substantial change.

In terms of an F4 algorithm, column swapping amounts to reordering the monomials, especially the leading monomials that we are always concerned with. This means that if we allow for *some* column swapping, there is potential for a dynamic F4 algorithm. In the dynamic F4 algorithm described in [18], many of the characteristics used to create a dynamic Buchberger algorithm are still in use. For example, the algorithm passes a heuristic for determining a preferable leading monomial. It is also a restricted algorithm, so it only makes changes to the monomial order if doing so does not change previously chosen leading monomials.

One of the major changes noted by the author is that interreduction between rows is not allowed. This means that once the newest subset of pairs have been added to a matrix, they cannot be used to reduce each other; however, elements that were previously added to the basis can be used. This is an effort to preserve previously chosen leading monomials and, typically, means that the output is not in row-echelon form. However, the matrix would still be able to remove rows that consist only of zeros. The algorithm then chooses one of these nonzero rows and picks a leading monomial for it. After this choice is made, the matrix can be interreduced by the row whose leading monomial is now "set." There is also the option to analyze all of the rows simultaneously to choose an overall preferable leading monomial.

This dynamic F4 algorithm shows promising results. Like dynamic Buchberger algorithms, it generally returns a Gröbner basis of smaller size than its static counterpart. In

many cases, it also computes a smaller basis than the dynamic Buchberger algorithm though not on the same scale as the reduction in size from a static F4 algorithm. For instance, with the Cyclic-8h polynomials, the static Buchberger and F4 algorithms compute Gröbner bases for grevlex order of the same size, 1182; while the dynamic Buchberger algorithm finds a basis almost one-third that size with 415 elements. For this same system, the dynamic F4 algorithm that proceeds with one row at a time returns a basis with 402 elements, and the one that processes all rows simultaneously returns a basis with 403 elements. These are much smaller that the static F4 algorithm and smaller but not significantly so than the dynamic Buchberger algorithm.

Chapter 4

F5 Algorithm

4.1 Background

Recall in Chapters 1 and 3 that we computed many S-polynomials that reduced zero. If our only goal is to find a Gröbner basis, then these zero polynomials add no new information. In that context, it seems like a huge waste of time. In 2002, Jean-Charles Faugére introduced an algorithm called F5 [8]. The algorithm avoids computing S-polynomials for most critical pairs that reduce to zero, making it incredibly useful. The F5 algorithm computes the Gröbner basis incrementally. That is, given a list of polynomials, $F = \{f_1, \ldots, f_m\}$, F5 computes first the Gröbner basis G_1 of $\langle f_1 \rangle$, then the Gröbner basis G_2 of $\langle f_1, f_2 \rangle$. It continues in this fashion, computing each Gröbner basis G_i of the ideal $\langle f_1, \ldots, f_i \rangle$ by building on the previous basis G_{i-1} of $\langle f_1, \ldots, f_{i-1} \rangle$. (It should be noted here that most researchers, including Faugére, denote this in the reverse order, where G_i is the basis of $\langle f_i, \ldots, f_m \rangle$.)

The main difference in the F5 algorithm and other algorithms to compute Gröbner bases, like Buchberger's, is the use of signatures. Each polynomial is given a signature based on how it was computed from the original ideal, $I = \langle f_1, \ldots, f_m \rangle$. These signatures are what allow the algorithm to detect and avoid many reductions to zero.

Definition 4.1.1. Suppose we have a Gröbner basis G. We use $\mathbf{e}_i$ to represent the canonical basis vector with 1 in the i^{th} position. Let $\sigma, \tau \in k[x_1, \ldots, x_n]$ be monomials. We define an ordering $\succ$ on $\sigma\mathbf{e}_i, \tau\mathbf{e}_j$ in the following way.

If one of the following is true:

(i) $i > j$ or

(ii) $i = j$ and $\sigma > \tau$ under the current monomial order,

then

$$\sigma\mathbf{e}_i \succ \tau\mathbf{e}_j.$$

We define a signature in the following way.

Definition 4.1.2. Let $I = \langle f_1, \ldots, f_m \rangle$ be an ideal. The **signature** of a monic polynomial $f \in I$ is $\sigma \mathbf{e}_i$ if $(\sigma, \mathbf{e}_i)$ is the smallest pair such that:

(i) $f = \sum_{j=1}^{m} h_j f_j$ for some $h_j \in k[x_1, \ldots, x_n]$,

(ii) $h_{i+1} = \cdots = h_m = 0$,

(iii) $\mathrm{LM}(h_i) = \sigma$.

Notice that we can view the result of the S-reduction shown in Chapters 1 and 3 as a similar vector dot product

$$f = \mathbf{h} \cdot \mathbf{g} = \sum_{j=1}^{s} h_j g_j, \tag{4.1}$$

where

$$\mathbf{h} = (h_1, \ldots, h_s) \in k[x_1, \ldots, x_n]^s$$

and

$$\mathbf{g} = (g_1, \ldots, g_s) \text{ for } g_j \in G.$$

Definition 4.1.3 (Definition 1 of [7])**.** Let $f \in k[x_1, \ldots, x_n]$, we say that f has a G**-representation** if there exist $h_1, \ldots, h_s \in k[x_1, \ldots, x_n]$ such

$$f = \sum_{j=1}^{s} h_j g_j$$

and for each j, either $h_j = 0$ or $\mathrm{LM}(h_j)\mathrm{LM}(g_j) \leq \mathrm{LM}(f)$. We can also write f in terms of the ideal generators as

$$f = \sum_{j=1}^{m} h_j f_j$$

which we call the S**-representation** of f with respect to G. We will omit "with respect to G" when it is clear from context.

This S-representation allows us to add another criterion, which was proven in [16], to Theorem 1.3.2 to get the following theorem.

Theorem 4.1.1 (Theorem 1 of [6])**.** *Let I be a polynomial ideal. Then the following are equivalent:*

- $G = \{g_1, \ldots, g_t\}$ *is a Gröbner basis of I.*
- *for all pairs $i \neq j, S(g_i, g_j) \underset{G}{\to} 0$.*

- *for all pairs $i \neq j, S(g_i, g_j)$ has an S-representation with respect to G.*

Since we are now using signatures, we need to make a slight change to how we reduce modulo G. In line 2(a) of Algorithm 2, we will now say:

$$\text{if } \mathrm{LM}(g) \mid \mathrm{LM}(f) \text{ for some } g \in G \text{ AND } \mathrm{sig}(f) \succ \mathrm{sig}\left(\frac{t_f}{t_g} \cdot g\right).$$

This means we can only reduce by an element of G if doing so does not change the signature of f. We call this a **signature safe reduction**, and often say it does not cause "signature corruption."

Definition 4.1.4 (Definition 7 of [6] and Defintion 8 of [7])**.** Let $g_i, g_j \in G$ and $S = S(g_i, g_j)$. Suppose that $\sigma \mathbf{e}_v = \mathrm{sig}(S)$, and $\sum_{\lambda=1}^{|G|} h_\lambda g_\lambda$ is a G-representation of S such that for each λ either $h_\lambda = 0$ or the signature of the product $\mathrm{LM}(h_\lambda) g_\lambda$ satisfies

$$\mathrm{LM}(h_\lambda)\,\mathrm{sig}(g_\lambda) \prec \sigma \mathbf{e}_v$$

for all $\lambda \in \{1, \ldots, |G|\}$, except one, say ℓ, where $\mathrm{LM}(h_\ell)\,\mathrm{sig}(g_\ell) = \sigma \mathbf{e}_v$ and $\ell > i, j$. We say

$$S = \sum_{\lambda=1}^{|G|} h_\lambda g_\lambda$$

is a **signature preserving S-representation**. We may omit signature preserving when it is clear from context.

Remark 4.1.1. Notice that having a signature preserving S-representation implies that the pair g_i, g_j also has an S-representation as we can write any g_λ in terms of the generators of I.

We return to Example 1.3.3 to see how signatures work.

Example 4.1.1. Recall

$$I = \langle x^3 - 2xy, x^2y - 2y^3 + x \rangle,$$
$$f_1 = x^3 - 2xy,$$
$$f_2 = x^2y - 2y^3 + x,$$

and $>$ is grevlex order. Then $f_1 = 1 \cdot f_1 + 0 \cdot f_2$, so

$$\mathrm{sig}(f_1) = \mathbf{e}_1.$$

We also have $f_2 = 0 \cdot f_1 + 1 \cdot f_2$, so

$$\mathrm{sig}(f_2) = \mathbf{e}_2.$$

In Equation (1.13), we computed $g_3 = x \cdot f_2 - y \cdot f_1$, so

$$\text{sig}(g_3) = x\mathbf{e}_2.$$

Finally, in Equation (1.14), we computed

$$\begin{aligned} S(g_3, g_2) &= x \cdot g_3 - y^2 \cdot g_2 \\ g_4 &= \frac{1}{2}(x \cdot g_3 - y^2 \cdot g_2 + y \cdot f_2 + \frac{1}{4} f_1) \\ &= \frac{1}{2}\left(x \cdot (x \cdot f_2 - y \cdot f_1) - y^2 \cdot f_2 + y \cdot f_2 + \frac{1}{2} f_1\right) \\ &= \frac{1}{2}(-xy + \frac{1}{2}) \cdot f_1 + \frac{1}{2}(x^2 - y^2 + y) \cdot f_2 \end{aligned}$$

which gives us

$$\text{sig}(g_4) = x^2\mathbf{e}_2.$$

Definition 4.1.5. Let $I = \langle f_1, \ldots, f_m \rangle$. If

$$f = \mathbf{h} \cdot \mathbf{f} = \sum_{i=1}^{m} h_i f_i = 0 \in k[x_1, \ldots, x_n],$$

we call $\mathbf{h}$ a **syzygy** on the polynomials in I.

Example 4.1.2. Let $I, f_1, f_2, <$ be as defined in Example 1.3.2 and let f be the S-reduction of g_1 and g_3 as in Equation (1.11). We have

$$f = (-x^2y + x, x^3 + xy^2) \cdot (f_1, f_2) = 0.$$

Then

$$\mathbf{h} = (-x^2y + x, x^3 + xy^2)$$

is a **syzygy**. Similarly, from the computations leading to Equation (1.12),

$$(-xy + 1, x^2 + y^2) \cdot (f_1, f_2)$$

is a syzygy.

Notice that the second syzygy corresponds to

$$(-f_2, f_1) \cdot (f_1, f_2) = -f_2 f_1 + f_1 f_2$$

which obviously returns 0. For any list of polynomials, $(f_1, \ldots, f_m)$, any syzygy that corresponds to

$$-f_j f_i + f_i f_j$$

with $1 \leq i < j \leq m$ is called a **trivial syzygy**. What if we look at the first syzygy? It corresponds to

$$(-xf_2, xf_1) \cdot (f_1, f_2) = x(-f_2, f_1) \cdot (f_1, f_2) = x(-f_2 f_1 + f_1 f_2).$$

Again, this obviously reduces to 0. The beauty of the F5 algorithm comes from its use of these known syzygies to avoid useless computations like the ones leading to Equations (1.11) and (1.12).

In Section 4.2, we define the F5 algorithm and discuss how it is used to compute a Gröbner basis. Although Faugére presented the algorithm in [8], the paper is often difficult to follow, so we will take our lead from [1], [6], and [7].

4.2 F5 Algorithm

Let $I = \langle f_1, \ldots, f_m \rangle \neq \{0\}$ be a polynomial ideal with f_i monic. We first present the main F5 algorithm and later define the subroutines used therein. The following F5 algorithm is a simplification of Algorithm 1 in [7]. We define each element of G as a tuple (τ, g), where g represents the polynomial and τ is the signature of g. In Algorithm 7, P is used to keep track of critical pairs that have not been computed. At the start of the loop in line 6, P consists only of the generators of I along with their signatures.

Notice that line 7 not only proceeds through the computation in order of ascending signatures but also insures that we are computing the Gröbner basis incrementally. In line 9 of the algorithm, Criterion(τ, G) tests $\tau = \tau' \mathbf{e}_{i+1}$ for divisibility by the leading monomials of the elements of G_i. That is, it checks the following criterion.

Definition 4.2.1 (Definition 15 of [6])**.** A polynomial f satisfies **Faugére's criterion** with respect to G_i if

(i) $\text{sig}(f) = \sigma \mathbf{e}_\ell$ and

(ii) there exists $g \in G_i$ such that

  (a) $\text{sig}(g) = \tau \mathbf{e}_j$ where $j < \ell$ and

  (b) $\text{LM}(g)$ divides σ

Lemma 4.2.1 (Lemma 16 of [6])**.** *If f satisfies Faugére's criterion with respect to G_i, then* $\text{sig}(f)$ *is not the correct signature of f.*

Proof. Let $I = \langle f_1, \ldots, f_m \rangle$ be an ideal. Assume $\text{sig}(f) = \sigma \mathbf{e}_\ell$ and there exists $g \in G_i$ such that $\text{sig}(g) = \tau \mathbf{e}_j$, $j < \ell$, and $\text{LM}(g) \mid \sigma$. By definition of signature, $f = \sum_{k=1}^{\ell} h_k f_k$ with

Algorithm 7 F5 Algorithm

Input: $I = \langle f_1, \ldots, f_m \rangle$ and a monomial order $<$

Output: a signature Gröbner basis $G = \{(\tau_1, g_1), \ldots, (\tau_t, g_t)\}$ for I

let $G = \emptyset$

    $P = \{(e_1, f_1), \ldots, (e_m, f_m)\}$

    $Rule = \emptyset$

while $P \neq \emptyset$ **do**

    $(\tau, g) =$ the element of smallest signature in P

    $P = P \backslash \{(\tau, g)\}$

    if Criterion$(\tau, G) =$ false **and** rewritten$(\tau, g) =$ false **then**

        append $(\tau, |G| + 1)$ to Rule

        $r =$ the signature preserving S-reduction of g by G

        if $r \neq 0$

            $r = \dfrac{1}{\mathrm{LC}(r)} \cdot r$

            if sig_redundant$((\tau, r), G) =$ false

                for $(\sigma, g) \in G$ such that g not sig-redundant **do**

                    if $\dfrac{t_r, t_g}{t_r} \cdot \tau \neq \dfrac{t_r, t_g}{t_g} \cdot \sigma$ **then**

                        $P = P \cup \{(\mathrm{sig}(S(r,g)), S(r,g))\}$

        $G = G \cup \{(\tau, r)\}$

return G

$\mathrm{LM}(h_\ell) = \sigma$ and $g = \sum_{k=1}^{j} H_k f_k$ with $\mathrm{LM}(H_j) = \tau$. By definition of divides, there exists

$t \in \mathbb{M}$ such that $t\,\mathrm{LM}(g) = \sigma$. Then

$$\begin{aligned} f &= f - \sigma f_\ell + \sigma f_\ell \\ &= f - \sigma f_\ell + \sigma f_\ell - t(gf_\ell - gf_\ell) \\ &= f - \sigma f_\ell + (\sigma - t\,\mathrm{LM}(g))f_\ell - t(g - \mathrm{LM}(g))f_\ell + t\sum_{k=1}^{j}(f_\ell H_k)f_k \end{aligned}$$

Since $t\,\mathrm{LM}(g) = \sigma$, we get

$$\begin{aligned} &= f - \sigma f_\ell - t(g - \mathrm{LM}(g))f_\ell + t\sum_{k=1}^{j}(f_\ell H_k)f_k \\ &= \underbrace{\left(\sum_{k=1}^{\ell} h_k f_k - \mathrm{LM}(h_\ell)f_\ell\right)}_{(A)} - \underbrace{t(g - \mathrm{LM}(g))f_\ell}_{(B)} + t\sum_{k=1}^{j}(f_\ell H_k)f_k \end{aligned}$$

Now, in (A) and (B), we have cancellation of $\mathrm{LM}(h_\ell)f_\ell$ and $t\,\mathrm{LM}(g)f_\ell$, respectively. That, along with the fact that $k < \ell$ for all $k \in \{1,\ldots,j\}$ in the last term means that we have not satisfied the requirement in Definition 4.1.2 that $(\sigma, \mathbf{e}_\ell)$ be the *smallest* pair. Therefore, $\sigma\mathbf{e}_\ell$ is not the correct signature of f! There is a smaller signature associated with it, and since we progress through the algorithm in order of ascending signature, we have already considered this signature. □

Also in line 9, rewritten(τ, g) tests if the polynomial was "rewritten" by a prior computation by comparing it to a list of Rules.

Definition 4.2.2 (Adapted from Definition 10 of [6])**. Rule** is a list of rewritings for f if for every $(\tau, \ell) \in \mathit{Rule}$, where $\tau = \mathrm{sig}(g_\ell)$, there exists $g_i, g_j \in G$ such that

(i) τ is the signature of $S(g_i, g_j)$

(ii) $\ell > i, j$, and $S(g_i, g_j)$ was reduced to g_ℓ in line 11 of Algorithm 7.

(iii) there exists (or P is scheduled to compute) a signature preserving S-representation of $S(g_i, g_j)$ such that $h_\ell = 1$.

We add a new tuple to Rule each time we perform an S-reduction (line 11).

Definition 4.2.3. Given signatures $\sigma = \sigma'\mathbf{e}_i$ and $\tau = \tau'\mathbf{e}_j$, we define **signature divisibility** as $\tau \mid \sigma$ if $j = i$ and $\tau' \mid \sigma'$.

Definition 4.2.4. Let $S(g_i, g_j) = tg_i - ug_j, f = tg_i, (\tau, \ell) \in \mathit{Rule}$, and $(\sigma, S(g_i, g_j)) \in P$ without loss of generality $\sigma = \mathrm{sig}(f)$. The **rewritable criterion** tests to see whether an S-reduction for $S(g_i, g_j)$ can be avoided by checking if the following are true:

(i) $\tau \mid \sigma$

(ii) $\ell > i, j$

(iii) if $(\tau', k) \in \mathit{Rule}$ such that $\tau' \mid \sigma$ then $k < \ell$.

If all of these are true, rewritten$(\sigma, S(g_i, g_j))$ returns True, and we say that (σ, f) was rewritten by (τ, g_ℓ).

Lemma 4.2.2 (Lemma 13 of [6])**.** *Let $f = tg_i, (\tau, \ell) \in \mathit{Rule}$, and (σ, f) be rewritten by (τ, ℓ). If the Algorithm 7 terminates with G, then there exist $d, \mathcal{H}_\lambda \in k[x_1, \ldots, x_n]$ with $\lambda \in \{1, \ldots, |G|\} \setminus \{\ell\}$ such that*

$$f = dg_\ell + \sum_{g_\lambda \in G \setminus \{g_\ell\}} \mathcal{H}_\lambda g_\lambda$$

where

(i) for all $g_\lambda \in G \setminus \{g_\ell\}$, if $\mathcal{H}_\lambda \neq 0$ then $\mathrm{sig}(\mathrm{LM}(\mathcal{H}_\lambda) \cdot g_\lambda) \prec t\,\mathrm{sig}(g_i)$ and

(ii) $t\,\mathrm{sig}(g_i)$ is the signature of $d\,\mathrm{sig}(g_\ell)$, meaning $t\,\mathrm{sig}(g_i) = d\,\mathrm{sig}(g_\ell)$.

The proof of this is quite complicated and can be found in [6].

In lines 12 and 13, if we did not compute a syzygy, we make the polynomial monic. The reader will also notice that in line 14, we check whether sig_redundant$((\tau, h), G)$ is true or not. This part of the algorithm is different from Faugére's original F5 algorithm. It was first defined in [7] in the following way.

Definition 4.2.5 (Definition 10 of [7])**.** Let G be a signature Gröbner basis and (τ, f) be an element we are considering adding to G. If there exists $(\sigma, g) \in G$ such that $\sigma \mid \tau$ and $\mathrm{LM}(g) \mid \mathrm{LM}(f)$, then (τ, f) is **signature-redundant** to G, abbreviated **sig-redundant**.

Lemma 4.2.3 (Lemma 13 of [7])**.** *If line 14 of Algorithm 7 finds sig_redundant$((\tau, f), G)$ to be true, the critical pairs associated with (τ, f) already have signature preserving S-representations or will after consideration of the pairs already in P.*

Proof. By Definition 4.2.5, there exists $(\sigma, g) \in G$ such that $\sigma \mid \tau$ and $\mathrm{LM}(g) \mid \mathrm{LM}(f)$. Suppose $\sigma = \sigma' \mathbf{e}_{i+1}$ and $\tau = \tau' \mathbf{e}_{i+1}$. By definition of divides, there exists $u \in \mathbb{M}$ such that

$u\mathrm{LM}(g) = \mathrm{LM}(f)$. If $u\sigma' < \tau'$, Line 11 did not compute a signature preserving S-reduction of $S(f,g)$, because g could reduce f further, a contradiction. Therefore $u\sigma' \geq \tau'$.

Let $t \in \mathbb{M}$ such that $u\sigma' \geq \tau' = t\sigma'$, then $\mathrm{LM}(f) = u\,\mathrm{LM}(g) \geq t\,\mathrm{LM}(g)$. The signature $\mu = \mu'\mathbf{e}_j$ of $s = f - tg$ is smaller that σ, so, by the method of ascending signature, the algorithm has considered S-reductions for μ and smaller signatures. Therefore, (μ, s) has a signature preserving S-representation. We have already generated critical pairs for g since it is in the basis, so for any $(\zeta, g_i) \in G$ there exist $\bar{u}, \bar{t} \in \mathbb{M}$ such that $S(f,g_i) = \bar{u}f - \bar{t}g_i$. By substitution, we have

$$\begin{aligned} S(f,g_i) &= \bar{u}f - \bar{t}g_i \\ &= \bar{u}(s+tg) - \bar{t}g, \end{aligned}$$

whose S-reductions already have signature preserving S-representations or will after the algorithm considers the pairs already in P. □

Finally, we need a way to check if we have computed an actual Gröbner basis, for which we use the following characterization.

Theorem 4.2.4. *(Theorem 9 of [7]) Suppose $G_i = \{g_1, \ldots, g_m\}$. Let $\tau_1, \ldots, \tau_\ell$ be monomials in $\mathbb{M}$ and $p_1, \ldots, p_\ell \in I_{i+1} \setminus \{0\}$ such that*

$$G = \{(\sigma_1\mathbf{e}_{j_1}, g_1), \ldots, (\sigma_m\mathbf{e}_{j_m}, g_m)\} \cup \{(\mathbf{e}_{i+1}, f_{i+1}), (\tau_1\mathbf{e}_{i+1}, p_1), \ldots (\tau_\ell\mathbf{e}_{i+1}, p_\ell)\},$$

$\sigma\mathbf{e}_j = \mathrm{sig}(g)$ for every $(\sigma\mathbf{e}_j, g) \in G$, and for every $(\sigma\mathbf{e}_{i+1}, p), (\tau\mathbf{e}_j, q) \in G$ one of the following holds:

(i) $\left(\frac{t_{p,q}}{t_p} \cdot \sigma\right)\mathbf{e}_{i+1} = \left(\frac{t_{p,q}}{t_q} \cdot \tau\right)\mathbf{e}_j$, *or*

(ii) $\left(\frac{t_{p,q}}{t_p} \cdot \sigma\right)\mathbf{e}_{i+1} \succ \left(\frac{t_{p,q}}{t_q} \cdot \tau\right)\mathbf{e}_j$ *and*
$\left(\frac{t_{p,q}}{t_p} \cdot \sigma\mathbf{e}_{i+1}, S(p,q)\right)$ *has a signature preserving S-representation with respect to G.*

Then $G_{i+1} = \{g \mid (\sigma\mathbf{e}_j, g) \in G \text{ where } j \leq i+1\}$ is a Gröbner basis for I_{i+1} with respect to $<$.

Proof. By Theorem 4.1.1 and since G_i is a Gröbner basis of I_i, we know that for every pair $\hat{p}, \hat{q} \in G_i$, $S(\hat{p}, \hat{q})$ has an S-representation with respect to G_i. To prove that G_{i+1} is a Gröbner basis for I_{i+1}, we need only to show that the pairs $p, q \in G_{i+1}$ where at least $p \notin G_i$ also have an S-representation. By hypothesis, if $\left(\frac{t_{p,q}}{t_p} \cdot \sigma\right)\mathbf{e}_{i+1} \succ \left(\frac{t_{p,q}}{t_q} \cdot \tau\right)\mathbf{e}_j$ and $j \leq i+1$,

$\left(\frac{t_{p,q}}{t_p}\cdot\sigma\mathbf{e}_{i+1}, S(p,q)\right)$ has a signature preserving S-representation with respect to G. This also means the pair p,q has an S-representation with respect to G_{i+1} (see Remark 4.1.1).

Order the remaining S-polynomials by ascending signature and choose the smallest $S(p,q)$ such that $j \leq i+1$, $(\sigma\mathbf{e}_{i+1}, p), (\tau\mathbf{e}_j, q) \in G$, and $\left(\frac{t_{p,q}}{t_p}\cdot\sigma\right)\mathbf{e}_{i+1} = \left(\frac{t_{p,q}}{t_q}\cdot\tau\right)\mathbf{e}_j$. Write

$$S(p,q) = \frac{t_{p,q}}{t_p}\cdot p - \frac{t_{p,q}}{t_q}\cdot q.$$

Then $S(p,q)$ has a signature smaller than $\left(\frac{t_{p,q}}{t_p}\cdot\sigma\right)\mathbf{e}_{i+1}$. Let $h_1,\ldots,h_j \in k[x_1,\ldots,x_n]$ such that

$$S(p,q) = \sum_{k=1}^{j} h_k f_k$$

and $\mathrm{LM}(h_j)\mathbf{e}_j = \mathrm{sig}(S(p,q))$. We see that each $\mathrm{LM}(h_k)\mathbf{e}_k \prec \left(\frac{t_{p,q}}{t_p}\cdot\sigma\right)\mathbf{e}_{i+1}$, since either $k < i+1$ or $k = i+1$ and $\mathrm{LM}(h_k) < \frac{t_{p,q}}{t_p}\cdot\sigma$ by Definition 4.1.2. Since $S(p,q)$ has a smaller signature than the signatures we have canceled, *and* since it was chosen as the smallest S-polynomial with canceling signatures, it must have a signature-preserving S-representation.

We can now proceed inductively: the next smallest $S(p,q)$ with canceling signatures can also be written with a smaller signature, and since all smaller signatures have signature-preserving S-representations (include the canceling pair we just considered), this pair must also have a signature-preserving S-representation.

□

Theorem 4.2.5 (Theorem 14 of [7])**.** *Algorithm 7 terminates correctly.*

Proof. Correctness: The only S-polynomials for which the algorithm does not compute signature preserving S-representations are those thrown out by lines 9, 12, 14, or 16. Lemmas 4.2.1 and 4.2.2 show that those thrown in line 9 have signature preserving S-representations when the algorithm terminates. Line 12 only removes syzygies as in Algorithm 1. Lemma 4.2.3 shows that those thrown out by line 14 already have signature preserving S-representations. Finally, line 16 is one of the hypotheses of Theorem 4.2.4. Therefore, by Theorem 4.2.4, $\hat{G} = \{g \mid g \in G\}$ is a Gröbner basis.

Termination: Let $\mathbb{M}'$ be the set of monomials in $x_1,\ldots,x_{2n}$. By Dickson's Lemma (Corollary 1.3.6 of [14]), we know that $\mathbb{M}'$ is Noetherian, or finitely generated. Any $(\sigma\mathbf{e}_i, g)$ added to G corresponds to an element of $\mathbb{M}'$ in the following way

$$(\sigma, \mathrm{LM}(g)) = \left(\Pi x_i^{\alpha_i}, \Pi x_i^{\beta_i}\right) \mapsto \Pi x_i^{\alpha_i}\Pi x_{n+i}^{\beta_i}.$$

Let J be the $\mathbb{M}'$-submodule generated by these (σ, g). Now, suppose the algorithm adds (σ, g) to G but J does not expand. Then there must exist $(\tau_i, h) \in G$ such that $\tau \mid \sigma$ and $\mathrm{LM}(h) \mid \mathrm{LM}(g)$, because if either of those were not true, J would have expanded. By Definition 4.2.5, $(\sigma \mathbf{e}_i, g)$ is sig-redundant, so line 14 keeps it from generating new pairs. Therefore, each $(\sigma \mathbf{e}_i, g)$ added to G either expands J or the algorithm prevents it from creating new pairs. Since $\mathbb{M}'$ is Noetherian, this process can only repeat finitely many times, meaning the algorithm computes finitely many pairs. Therefore, Algorithm 7 terminates. □

Now that we have defined the F5 algorithm, we will walk through Example 1.3.3 using it.

4.2.1 Example of Gröbner basis computation with the F5 algorithm

Example 4.2.1. As before, let $<$ be grevlex order and $I = \langle x^3 - 2xy, x^2y - 2y^3 + x \rangle$. We will walk through the F5 algorithm.

3: $G = \emptyset$

4: $P = \{(\mathbf{e}_1, f_1), (\mathbf{e}_2, f_2)\}$

5: $Rule = \emptyset$

Loop 1:

6: $P \neq \emptyset$

7: Choose $(\mathbf{e}_1, f_1)$, since $\mathbf{e}_1$ is the smallest signature in P

8: $P = \{(\mathbf{e}_2, f_2)\}$

9: Criterion$(\mathbf{e}_1, G) =$ false **and** rewritten$(\mathbf{e}_1, f_1) =$ false, so we continue.

10: Rule = $\{(\mathbf{e}_1, 1)\}$

11: $r = f_1$ (It cannot be further reduced.)

12: $r \neq 0$

13: r is already monic.

14: G is empty, so we can skip lines 14 - 17.

18: $G = \{(\mathbf{e}_1, x^3 - 2xy)\}$

Loop 2:

7: Choose $(\mathbf{e}_2, f_2)$

8: $P = \emptyset$

9: Criterion$(\mathbf{e}_2, G) =$ false **and** rewritten$(\mathbf{e}_2, f_2) =$ false, so we continue.

10: Rule = $\{(\mathbf{e}_1, 1), (\mathbf{e}_2, 2)\}$

11: $r = f_2$ (Again, it cannot be further reduced.)

12: $r \neq 0$

13: r is already monic.

14: sig_redundant$((\mathbf{e}_2, f_2), G))$ = false

15: G only contains $(\mathbf{e}_1, g_1)$

16: $x \cdot \mathbf{e}_2 \neq y \cdot \mathbf{e}_1$

17: $P = \{(x\mathbf{e}_2, S(r, g_1))\}$

18: $G = G \cup \{(\mathbf{e}_2, x^2y - 2y^3 + x)\}$

Loop 3:

7: Choose $(x\mathbf{e}_2, S(g_2, g_1))$

8: $P = \emptyset$

We need to check rewritten$(x\mathbf{e}_2, S(g_2, g_1))$ against Rule = $\{(\mathbf{e}_1, 1), (\mathbf{e}_2, 2)\}$. Notice that $\mathbf{e}_2 \mid x\mathbf{e}_2$ but $\ell = j = 2$, so it is not rewritten.

9: Criterion$(x\mathbf{e}_2, G) =$ false **and** rewritten$(x\mathbf{e}_2, S(g_2, g_1)) =$ false.

10: Rule = $\{(\mathbf{e}_1, 1), (\mathbf{e}_2, 2), (x\mathbf{e}_2, 3)\}$

11: $r = -2xy^3 + 2xy^2 + x^2$

12: $r \neq 0$

13: $r = xy^3 - xy^2 - \frac{1}{2}x^2$

14: A quick inspection of G shows that r is not sig-redundant.

15: We look at both $(\mathbf{e}_1, g_1)$ and $(\mathbf{e}_2, g_2)$

16: $(\mathbf{e}_1, g_1)$: $x^2 \cdot x\mathbf{e}_2 \neq y^3 \cdot \mathbf{e}_1$

$(\mathbf{e}_2, g_2)$: $x \cdot x\mathbf{e}_2 \neq y^2 \cdot \mathbf{e}_1$

17: $P = \{(x^3\mathbf{e}_2, S(r, g_1)), (x^2\mathbf{e}_2, S(r, g_2))\}$

18: $G = G \cup \{(x\mathbf{e}_2, xy^3 - xy^2 - \frac{1}{2}x)\}$

Loop 4:

7: Choose $(x^2\mathbf{e}_2, S(g_3, g_2))$ since $x^2\mathbf{e}_2 \prec x^3\mathbf{e}_2$.

8: $P = \{(x^3\mathbf{e}_2, S(g_3, g_1))\}$

We need to check $(x^2\mathbf{e}_2, S(g_3, g_2))$ against Rule = $\{(\mathbf{e}_1, 1), (\mathbf{e}_2, 2), (x\mathbf{e}_2, 3)\}$.

We see that $\mathbf{e}_2 \mid x^2\mathbf{e}_2$ but $\ell = i = 2$. Also, $x\mathbf{e}_2 \mid x^2\mathbf{e}_2$ but $\ell = j = 3$.

9: Criterion$(x^2\mathbf{e}_2, G)$ = false **and** rewritten$(x^2\mathbf{e}_2, g)$ = false

10: Rule = $\{(\mathbf{e}_1, 1), (\mathbf{e}_2, 2), (x\mathbf{e}_2, 3), (x^2\mathbf{e}_2, 4)\}$

11: $r = 2y^5 - 2y^4 - xy^2$

12: $r \neq 0$

13: $r = y^5 - y^4 - \frac{1}{2}xy^2$

14: Although, $\mathbf{e}_2$ and $x\mathbf{e}_2$ divide $x^2\mathbf{e}_2$, neither x^3y nor xy^3 divide y^5, so r is not sig-redundant.

15: We look at $(\mathbf{e}_1, g_1)$, $(\mathbf{e}_2, g_2)$, and $(x\mathbf{e}_2, g_3)$.

16: $(\mathbf{e}_1, g_1)$: $x^3 \cdot x^2\mathbf{e}_2 \neq y^5 \cdot \mathbf{e}_1$

$(\mathbf{e}_2, g_2)$: $x^2 \cdot x^2\mathbf{e}_2 \neq y^4 \cdot \mathbf{e}_2$

$(x\mathbf{e}_2, g_3)$: $x \cdot x^2\mathbf{e}_2 \neq y^2 \cdot x\mathbf{e}_2$

17: $P = \{(x^3\mathbf{e}_2, S(g_3, g_1)), (x^5\mathbf{e}_2, S(r, g_1)), (x^4\mathbf{e}_2, S(r, g_2)), (x^3\mathbf{e}_2, S(r, g_3))\}$

18: $G = G \cup \{(x^2\mathbf{e}_2, y^5 - y^4 - \frac{1}{2}xy^2)\}$

For each of the remaining elements in P, we see that Criterion$(\tau, G) =$ true, since $\mathrm{LM}(g_1) = x^3 \mid x^3, x^4, x^5$. This means that we have finished the computation and return

$$G = \{(\mathbf{e}_1, x^3 - 2xy), (\mathbf{e}_2, x^2y - 2y^3 + x), (x\mathbf{e}_2, xy^3 - xy^2 - \frac{1}{2}x), (x^2\mathbf{e}_2, y^5 - y^4 - \frac{1}{2}xy^2)\}.$$

We can check that this is in fact a Gröbner basis using the criteria of Theorem 4.2.4. It is easy to see that any $g_i, g_j \in G$ meet the second criterion, so we have a Gröbner basis.

Moreover, the polynomials in the final basis are exactly those we found in Example 1.3.3, except with this algorithm, we were able to avoid all four of the reductions to zero we had to compute using Buchberger's algorithm.

Chapter 5

Dynamic F5 Algorithm

5.1 Introduction

In Chapter 3, we saw that dynamic algorithms for Gröbner basis computation typically yield a smaller basis than their static counterparts. In Chapter 4, we saw that the F5 algorithm utilizes signatures to avoid useless computations. One might wonder what would happen if we tried to create a dynamic F5 algorithm. Would that lead to a smaller Gröbner basis *and* avoid many useless computations? Another consideration is that, previously, we needed only to ensure that the leading monomials of the basis elements remained unchanged if the order was changed. Now, we will also need to ensure that signatures of basis elements remain unchanged as well. We will address both of these concerns in Section 5.2.

We have designed an algorithm that accomplishes both of these goals. We present the algorithm in Section 4.2 and some computational results in Section 5.3. This algorithm uses a similar approach to previous dynamic algorithms for Gröbner basis computation. In particular, we use the Hilbert heuristic from [19] to choose a preferable leading monomial once an S-reduction has occurred. We use the boundary vector method of [4] to remove incompatible leading monomials from our list of possibilities. We still implement F5 defining procedures, including the rewritable criterion and Faugére's criterion. We also implement the sig-redundant criterion described in [6].

5.2 Dynamic F5 Algorithm

We present the main algorithm first, then follow up with discussion on some of the procedures used. In Algorithm 8, we define the set of critical pairs in essentially the same way as Algorithm 7; however, it is represented differently. In the static algorithm, we showed the critical pairs as (τ, g) where $g = S(g_i, g_j)$ and $\tau = \text{sig}(S(g_i, g_j))$ for the $g_i, g_j \in G$ that we used to find g. In this algorithm, we present a pair as $(\tau, (tg_i, ug_j))$ where $t = \frac{t_{g_i, g_j}}{t_{g_i}}$ and $u = \frac{t_{g_i, g_j}}{t_{g_j}}$. Clearly, $S(g_i, g_j) = tg_i - ug_j$, so we are representing the same elements in a slightly modified format. We will assume without loss of generality that $\tau = \text{sig}(S(g_i, g_j)) = \text{sig}(tg_i)$, and we know that $\tau = \tau' \mathbf{e}_v$ with $\tau' \in \mathbb{M}$ and $v \in \{1, \ldots, m\}$.

Algorithm 8 Dynamic F5 Algorithm

Input: $I = \langle f_1, \ldots, f_m \rangle$

Output: a weight vector ω and a signature Gröbner basis $G = \{(\tau_1, g_1), \ldots, (\tau_t, g_t)\}$ with respect to ω for I

let $G, P' = \emptyset$

    $P = \{(e_1, (f_1, 0)), \ldots, (e_m, (f_m, 0))\}$

    $\omega = (1, \ldots, 1)$

while $P \neq \emptyset$ **do**

    $rule = \emptyset$

    choose $p \in P$ with minimal signature, remove it from P, and append it to P'

    while $P' \neq \emptyset$ **do**

        choose $s = (\tau, (tg_i, ug_j)) \in P'$ with minimal signature and remove it from P'

        if not rewritten(s)

            $r =$ the signature preserving S reduction of s by G

            append $(\tau, |G| + 1)$ to Rule

            $G = G \cup \{(\tau, r)\}$

            if $r \neq 0$:

                $\omega =$update_lp(r, ω)

                if (τ, r) not sig_redundant:

                    $P' = P' \cup \{$generate_pairs$(r, G)\}$

                else:

                    remove $(\tau, |G|)$ from Rule

    minimize(G)

return G, ω

5.2.1 Rewritten

In Line 11, rewritten is the implementation of Definition 4.2.4. For rewritten(s), we iterate through Rule in reverse order, checking to see if there is some $(\sigma' \mathbf{e}_{\hat{v}}, \ell)$ in Rule such that $\sigma' \mid \tau'$ and $v = \hat{v}$. Once we find the first such (σ', ℓ), if $\ell \neq i$, rewritten returns true; otherwise, it returns false. Due to the definition of rewritten, there is no need to continue looking even if the polynomial was not rewritten. We also exit the loop early when we see $\hat{v} < v$, since, by definition of signature division, we cannot satisfy the requirements of rewritten.

5.2.2 S-Reduction

Line 12 uses Algorithm 2 for reduction along with the refinement to prevent signature corruption discussed in Section 4.1. That is, if t is the monomial by which we need to multiply g in order get our desired cancellation of leading monomials and $\text{sig}(tg) \succ \tau$, we do not make the reduction.

In Line 14, you will notice that even if the polynomial r reduces to 0, we append $(\tau, |G|+1)$ to Rule and $(\tau, 0)$ to G as we did in the static algorithm. Suppose we found some (τ, r) with $r = 0$ after reduction and didn't append the signature to Rule, then every polynomial that should have been rewritten by this has to go through the entire reduction process only to again return 0. This would cause our algorithm to waste a large amount of time on useless computations; the one thing we have set out to avoid.

5.2.3 Updating ω

We now take a closer look at what happens if our polynomial r did not reduce to zero. In Line 16, we pass r along with the old weight vector ω to update_lp, which finds a "preferred" leading monomial and updates the weighted order accordingly. Algorithm 9 shows how this is accomplished. In Line 2(a), we use the boundary vector criterion from Theorem 2.3.2 to find the compatible monomials. Line 2(b) uses the Hilbert heuristic from [19] to choose a "preferred" leading monomial and break ties by choosing the smallest monomial under grevlex order.

In Section 2.3, we showed that a weighted order could be represented by a linear program L. In Line 2(c), we update ω by adding additional constraints that ensure $t = \text{LM}(r)$ to this linear program. For example, if we want x_2^2 to be the leading monomial of $r = x_1^2 + x_2^2 + x_2x_3$, we would add $2y_2 \geq y_2 + y_3$ $(x_2^2 > x_2x_3)$ and $2y_2 \geq 2y_1$ $(x_2^2 > x_1^2)$, or equivalently $y_3 - y_2 \leq 0$ and $y_1 - y_2 \leq 0$. In Line 2(d), we check three things. First, we make sure that updating the weight vector does not lead to a linear program that is infeasible. Second, we check that the leading monomials of previous basis elements remain unchanged. The program uses a

Algorithm 9 update_lp(r, ω)

Input: a polynomial r, the current weighted order ω
Output: the updated weight vector ω

1. let $works$ = False
2. **while** not $works$ **do**
   (a) $U = \{m \in \text{Supp}(r) \mid m \text{ is compatible for } r\}$
   (b) **let** $t \in U$ be the preferred leading monomial
   (c) update ω to ω' so that $t = \text{LM}(r)$
   (d) **if** the update fails **or** verify_order(ω', G) = False **or** verify_sig_order($\omega', Rule$) = False **then**
   $U = U \setminus \{t\}$
   (e) **else**
   let $\omega = \omega'$
   $works$ = True
3. **return** ω

list, T, to keep track of the leading monomials of G under ω and checks that the leading monomials of G under ω' are the same. Finally, we check that under the proposed weight ordering, the signatures are in the same order. This is done by checking that the order of Rule is unchanged. We create a copy of Rule, then apply the proposed weight vector to the copy. This allows us to easily compare the ordering of signatures under both the current and proposed orderings. We also have to insure that updating the ordering does not cause any critical pair waiting to be computed to have a smaller signature than those we have already computed. This is done by testing that the maximum signature is Rule is smaller than the minimum signature in S.

An older implementation of this code did not check that previous signatures remained in the same order when the weight vector was updated. We instead attempted to preserve the signatures by insuring that signatures for critical pairs not yet computed were in ascending order. For most systems we investigated, this was not an issue. However, we did encounter systems where it was a problem, specifically Noon4 and Noon5. These two systems ended up not computing a Gröbner basis using the previous implementation. Recall from Theorems 4.2.4 and 4.2.5 that we need the signatures to be in ascending order to guarantee termination. Therefore, we must insure that the previous order of signatures is unchanged, as the algorithm now does.

In one of our early implementations, if we had an infeasible linear program or any previous leading monomial changed, we removed t from U and tried again. We have since added one additional step in an attempt to preserve our preferable leading monomial. If the check against T finds that one or more of the leading monomials has changed, we add new constraints that undo this change. For example, suppose $\mathrm{LM}_{\omega}(g) = x_1x_2$ but $\mathrm{LM}_{\hat{\omega}}(g) = x_0x_2$. We add $y_1y_2 \geq y_0y_2$ to L then check to make sure the system is feasible and that we haven't disrupted any other leading monomials. We only attempt this refinement one time. If we are unable to find a feasible linear system for t that preserves previous leading monomials, we remove t from U and try again. Otherwise, we solve the new linear program, and the solution becomes our updated ω'.

Lemma 5.2.1. *Algorithm 9 terminates correctly.*

Proof. To show that the algorithm terminates, we only need to show that at some point we find *works* = True. Let $t' = \mathrm{LM}(r)$ under the current weighted order ω, then, certainly, $t' \in U$. If we choose t' in Step 2(b), then the updated ω is exactly the current ω. We have added no constraints to the linear program, so we know it has a solution. It is obvious that prior signatures and leading monomials remain unchanged, so *works* = True. We remove elements of U until we find a t that meets our needs, so we either find a t that is "more preferable" than t' with *works* = True, or we end up with t'.

To show correctness, we need to show that $t = \mathrm{LM}(r)$ under the updated ω, but that is exactly what Line 2(c) does. □

5.2.4 Signature Redundant

Line 17 checks if (τ, r) is sig_redundant using Definition 4.2.5. In the static F5 algorithm, we had the freedom to check if the polynomial was signature redundant when we checked if it was equal to 0. We only chose not to in that algorithm because we wanted to make the polynomial monic first. In Algorithm 8, we do not have such freedom. Suppose we checked for signature redundancy before updating ω and found some $(\sigma, g) \in G$ where $\sigma \mid \tau$ and $\mathrm{LM}(g) \mid \mathrm{LM}(r)$. One of two things could go wrong:

- If we choose a new leading monomial for r in update_lp, it is unlikely we will still have $\mathrm{LM}(g) \mid \mathrm{LM}(r)$. This means we no longer have signature redundancy, and we end up not generating critical pairs that are necessary.
- Suppose we skip updating ω once we find a sig-redundant polynomial. It is possible that we have missed out on a "better" order, so we lose some of the benefits of a dynamic algorithm.

5.2.5 Generating Critical Pairs

The only thing left to do is generate critical pairs. This part of the dynamic algorithm is exactly the same as in the static F5 algorithm. For any new pair $(\text{sig}(S(r,g)),(tr,ug))$, if $\text{sig}(tr) \neq \text{sig}(ug)$ and $\text{LM}(g_i) \nmid \text{sig}(S(r,g))$ for any $g_i \in G$, then we add a new critical pair to P'. The second criterion is Faugére's criterion from Definition 4.2.1.

5.2.6 Minimizing the Basis

For each G_i, we use Lemma 1.3.1 to remove any redundant polynomials from the basis before moving on. Line 19 does this by checking whether any leading monomials of the basis elements are divisible by the leading monomial of any other basis element. If there exists $g_j, g_k \in G_i$ such that $\text{LM}(g_j) \mid \text{LM}(g_k)$, we remove g_k from G_i.

5.2.7 Correctness and Termination of Algorithm 8

In Section 5.1, we mentioned that we needed to also ensure that the signatures remain unchanged anytime we update ω. In order to do this, we need a special property of signatures.

Lemma 5.2.2 (Lemma 14 of [6] and Lemma 2 of [1])**.** *Let G_i be a Gröbner basis of $I_i = \langle f_1, \ldots, f_m \rangle$, $f \in G_i$, $t, \sigma \in \mathbb{M}$, and $\text{sig}(f) = \sigma \mathbf{e}_j$ where $j \leq i$. Then*

(i) $\text{sig}(tf) = t\,\text{sig}(f)$ *or*

(ii) there exists a syzygy $\mathbf{h}$ *of I_i such that* $\text{sig}(\mathbf{h} \cdot \mathbf{f}) = t\,\text{sig}(f)$.

Proof. By Definition 4.1.2, there exist $H_1, \ldots, H_m \in k[x_1, \ldots, x_n]$ such that

$$f = \sum_{k=1}^{m} H_k f_k,$$

$H_{j+1} = \cdots = H_m = 0$, and $\text{LM}(H_j) = \sigma$. Then

$$tf = \sum_{k=1}^{j} tH_k f_k.$$

If $\text{sig}(tf) = t\,\text{LM}(H_j) = t\,\text{sig}(f)$, we have (i). Otherwise, $\text{sig}(tf) \prec t\,\text{LM}(H_j)$. Let $v \in \{1, \ldots, m\}$, $\tau \in \mathbb{M}$ such that $\tau \mathbf{e}_v = \text{sig}(tf)$. Again, by Definition 4.1.2, there exist $h'_1, \ldots, h'_m \in k[x_1, \ldots, x_n]$ such that

$$tf = \sum_{k=1}^{m} h'_k f_k,$$

$h'_{v+1} = \cdots = h'_m = 0$, and $\mathrm{LM}(h'_v) = \tau$. Since $\tau \mathbf{e}_v \prec \sigma \mathbf{e}_j$, we know that $v \leq j \leq i$. Let $\mathbf{h} = t\mathbf{H} - \mathbf{h}'$, meaning

$$h_k = \begin{cases} tH_k - h'_k, & 1 \leq k \leq v \\ tH_k, & v < k \leq j \\ 0, & j < k \leq m \end{cases}$$

Obviously, $tf - tf = 0$ which gives us

$$\begin{aligned} 0 &= tf - tf \\ &= \sum_{k=1}^{m} tH_k f_k - \sum_{k=1}^{m} h'_k f_k \\ &= \sum_{k=1}^{m} (tH_k - h'_k) f_k \\ &= \sum_{k=1}^{m} h_k f_k. \end{aligned}$$

We know that $\mathbf{h} \neq 0$, since $tH_j \neq 0$ and $h'_j = 0$ meaning $h_k = tH_j \neq 0$. Therefore, $\mathbf{h}$ is a syzygy of I_i, and, by Definition 4.1.2, $\mathrm{sig}(\mathbf{h} \cdot \mathbf{f}) = t\,\mathrm{LM}(H_k) = t\,\mathrm{sig}(f)$, so we have (ii). □

Now, we can show that our signature remain unchanged upon updating ω.

Lemma 5.2.3. *If $(\tau, g_\ell) \in G$, then $\tau = \mathrm{sig}(g_\ell)$ for any ω returned by Algorithm 9.*

Proof. Suppose $(\tau, g_\ell) \in G$ and ω is a weighted order returned by Algorithm 9. Let $\tau = \tau' \mathbf{e}_k$ for some $k \in \{1, \ldots, m\}$. If $\tau' = 1$, then we are done. Otherwise, $\tau' \mathbf{e}_k = \mathrm{sig}(S(g_i, g_j))$ for some $i, j < \ell$ such that $g_i, g_j \neq 0$. We know by our method of adding critical pairs that $\mathrm{sig}\left(\frac{t_{g_i,g_j}}{t_{g_i}} \cdot g_i\right) \neq \mathrm{sig}\left(\frac{t_{g_i,g_j}}{t_{g_j}} \cdot g_j\right)$. This means that

$$\mathrm{sig}(S(g_i, g_j)) = \mathrm{sig}\left(\frac{t_{g_i,g_j}}{t_{g_i}} \cdot g_i\right)$$

or

$$\mathrm{sig}(S(g_i, g_j)) = \mathrm{sig}\left(\frac{t_{g_i,g_j}}{t_{g_j}} \cdot g_j\right).$$

Without loss of generality, suppose we have $\mathrm{sig}(S(g_i, g_j)) = \mathrm{sig}\left(\frac{t_{g_i,g_j}}{t_{g_i}} \cdot g_i\right)$. Since $g_i \neq 0$ and by Lemma 5.2.2,

$$\begin{aligned} \tau &= \mathrm{sig}(g_\ell) \\ &= \mathrm{sig}(S(g_i, g_j)) \\ &= \mathrm{sig}\left(\frac{t_{g_i,g_j}}{t_{g_i}} \cdot g_i\right) \\ &= \frac{t_{g_i,g_j}}{t_{g_i}} \cdot \mathrm{sig}(g_i). \end{aligned}$$

Now, suppose $\text{sig}(g_i) = \sigma' \mathbf{e}_k$. By the same logic, $\sigma' \mathbf{e}_k = \text{sig}\left(\frac{t_{g_{\hat{i}}, g_{\hat{j}}}}{t_{g_{\hat{i}}}} \cdot g_{\hat{i}}\right)$ for some $g_{\hat{i}}, g_{\hat{j}} \in G$ with $g_{\hat{i}}, g_{\hat{j}} \neq 0$, then

$$\begin{aligned}\tau &= \text{sig}(g_\ell) \\ &= \frac{t_{g_i, g_j}}{t_{g_i}} \cdot \text{sig}\left(\frac{t_{g_{\hat{i}}, g_{\hat{j}}}}{t_{g_{\hat{i}}}} \cdot g_{\hat{i}}\right) \\ &= \frac{t_{g_i, g_j}}{t_{g_i}} \frac{t_{g_{\hat{i}}, g_{\hat{j}}}}{t_{g_{\hat{i}}}} \cdot \text{sig}(g_{\hat{i}}).\end{aligned}$$

We can repeat the same process until we have reached a $g_{\hat{k}}$ that is the S-reduction of the generator f_k, in which case we have

$$\left(\Pi \frac{t_{g_{\hat{i}}, g_{\hat{j}}}}{t_{g_{\hat{i}}}}\right) \cdot \mathbf{e}_k.$$

Notice that every element in the product is the least common multiple of two leading monomials divided by another leading monomial, all of which belong to prior elements of G. We already know that these are unchanged, as we checked them in verify_order. Therefore, τ remains unchanged and $\tau = \text{sig}(g_\ell)$. □

Theorem 5.2.4. *Algorithm 8 terminates correctly in a finite number of steps with a signature Gröbner basis for I under a weighted order ω.*

Proof. By choosing minimal elements from our list of critical pairs, the criteria used to generate critical pairs, and Lemma 5.2.3, we have shown that we proceed by ascending signature. Since no other major elements from Algorithm 7 have changed, Theorem 4.2.5 applies. □

Now that we have shown our algorithm works, let's see how it actually performs.

5.3 Results

In order to check that we have computed a Gröbner basis, we remove the signatures from the output then check whether the polynomials satisfy Theorem 1.3.2, that is, we check that all distinct pairs reduce to zero. The Sage source code can be found in Appendix A. We use object-oriented programming to create classes for signatures, signed polynomials, and critical pairs. The code then follows closely with the pseudo-code seen throughout this work.

	Number of S-polynomials computed			
	Buch	DynB	F5	DynF5
Cyclic-4	12	11	8	6
Cyclic-4 (h)	12	4	8	5
Cyclic-5	113	85	39	134
Cyclic-5 (h)	113	18	39	37
Cyclic-6	352	454	171	316
Cyclic-6 (h)	386	110	171	47
Cyclic-7 (h)	2199	404	1057	653
Eco5	27	22	16	10
Eco5(h)	38	16	23	20
Eco6	68	36	31	28
Eco6(h)	150	134	55	44
Eco8	362	351	118	365
Katsura-5	69	62	42	166
Katsura-5(h)	71	141	42	31
Noon3	20	21	11	12
Noon4	75	107	30	33
Noon4(h)	76	69	34	16
Noon5	268	315	85	276
Trinks	29	8	20	10

Table 5.1: Number of S-polynomials computed using Buchberger's algorithm (Buch), a dynamic Buchberger algorithm (dynB) [19], the F5 algorithm, and the dynamic F5 algorithm (DynF5) described here.

Tables 5.1, 5.2, and 5.3 show the results of computations on some benchmark polynomial systems, which we acquired from the Database of Polynomial Systems [22] and can be found in Appendix B. The results of the dynamic F5 algorithm described in this work are compared with the traditional Buchberger algorithm, the dynamic Buchberger algorithm from [19], and the F5C algorithm from [6], which is an optimization of the traditional F5 algorithm.

Table 5.1 shows the number of S-polynomials computed by each algorithm. The reader may notice that in some cases, the dynamic F5 algorithm computed *more* S-polynomials than the dynamic Buchberger algorithm and, in a few cases, like Eco 8, even more than the traditional Buchberger algorithm. This is due in part to our attempt to avoid signature

corruption, since F5 does not always reduce the leading monomial when Buchberger would. For example, in the dynamic F5 computation of Noon4, we run into the following scenario.

Example 5.3.1. Our current weight vector is $(1,13,19,8)$, and, at Line 10, the algorithm chooses

$$s = (x_0x_2x_3\mathbf{e}_3, (x_3g_{10}, x_1x_2g_4)).$$

This pair was not rewritten, so we compute the S-polynomial to get

$$S(g_{10},g_4) = 3x_0^3x_1x_2x_3 - 14x_0x_1x_2x_3 + x_0x_1x_2 + x_0x_2x_3 + x_0x_1x_3 - 2x_1x_2x_3.$$

When we try to reduce this polynomial by G, we find that $\mathrm{LM}(g_{15}) = x_0x_1x_2$ and $\mathrm{sig}(g_{15}) = x_0x_1^2\mathbf{e}_3$. Clearly, $x_0x_1x_2 \mid x_0^3x_1x_2x_3$, but $\mathrm{sig}(x_0^2x_3(x_0x_1^2\mathbf{e}_3)) = x_0^3x_1^2x_3\mathbf{e}_3$ which gives us

$$\begin{aligned}(1,13,19,8)\cdot(3,2,0,1) &= 37\\ &> 28\\ &= (1,13,19,8)\cdot(1,0,1,1).\end{aligned}$$

Since $x_0^3x_1^2x_3\mathbf{e}_3 \succ x_0x_2x_3\mathbf{e}_3$, we do not make this reduction. The algorithm later chooses the leading monomial to be $x_0^3x_1x_2x_3$. We know from Lemma 1.3.1 that this polynomial is redundant and will be removed in Line 19, but it is added to the current basis to avoid signature corruption. A Buchberger algorithm would have further reduced this polynomial before adding it to the basis. This means an F5 algorithm will have to wait until it computes a later polynomial with the leading monomial obtained by Buchberger at this point to obtain the same critical pairs.

We found that in homogenized systems, like the homogenized Cyclic-n systems, our algorithm computed fewer S-polynomials than the traditional F5 algorithm. Recall from Section 3.1 that the Hilbert function for homogeneous system is invariant, so this is expected. We also know that varying the monomial ordering, therefore varying the Hilbert function, will sometimes cause more S-polynomials, which is what the data reveal. We were, however, able to reduce the number of S-polynomials computed in a few of these systems by more than half. For example, Cyclic-7h computes only 653 S-polynomials in dynamic F5 but computes 1057 S-polynomials using traditional F5. Both of these are a major reduction from traditional Buchberger, which computes 2199 S-polynomials. The algorithm also performed well with Noon3, Noon4, and Trinks which are all non-homogeneous systems. In the first two, the dynamic F5 algorithm computed fewer S-polynomials than both the Buchberger and the dynamic Buchberger algorithms. However, it computed more than the traditional

	Number of zero polynomials computed			
	Buch	DynB	F5	DynF5
Cyclic-4	5	5	1	1
Cyclic-4 (h)	5	1	1	1
Cyclic-5	75	51	0	0
Cyclic-5 (h)	75	8	0	0
Cyclic-6	254	307	9	9
Cyclic-6 (h)	288	73	9	9
Cyclic-7 (h)	1756	298	44	90
Eco5	15	7	0	0
Eco5(h)	22	6	4	7
Eco6	43	13	0	0
Eco6(h)	105	95	17	19
Eco8	270	63	0	0
Katsura-5	47	42	0	0
Katsura-5(h)	48	105	0	0
Noon3	9	9	0	0
Noon4	47	67	0	0
Noon4(h)	47	44	0	0
Noon5	195	215	0	2
Trinks	16	1	0	0

Table 5.2: Number of useless S-reductions to zero computed using Buchberger's algorithm (Buch), a dynamic Buchberger algorithm (dynB) [19], the F5 algorithm, and the dynamic F5 algorithm (DynF5) described here.

F5 algorithm. For Trinks, our algorithm computed fewer S-polynomials than the traditional Buchberger and F5 algorithms and only 2 more than dynamic Buchberger.

Table 5.2 shows the number of zero polynomials computed by each of these algorithms. As was expected with an F5 style algorithm, we were able to avoid the majority of zero reductions computed with both traditional and dynamic Buchberger algorithms. In fact, with the exception of Cyclic-7(h), Eco5(h), Eco6(h), and Noon5, we computed exactly the same number of zero polynomials as the traditional F5 algorithm. From [8], we know that the traditional F5 algorithm only guarantees that we will not compute any zero polynomials for a "regular" system. A regular system is one in which all of the syzygies are trivial. This, however, depends on the monomial ordering. Therefore, varying the ordering may vary the

	Final basis size $\|G\|$			
	Buch	DynB	F5	DynF5
Cyclic-4	7	7	7	5
Cyclic-4 (h)	7	4	7	4
Cyclic-5	20	13	20	17
Cyclic-5 (h)	38	11	38	22
Cyclic-6	44	21	44	19
Cyclic-6 (h)	98	38	98	28
Cyclic-7 (h)	443	107	443	110
Eco5	11	6	11	9
Eco5(h)	15	9	15	12
Eco6	18	10	18	11
Eco6(h)	44	39	28	19
Eco8	59	12	59	12
Katsura-5	22	20	22	11
Katsura-5(h)	22	36	22	24
Noon3	11	8	11	11
Noon4	28	21	28	26
Noon4(h)	28	25	28	15
Noon5	72	53	72	53
Trinks	13	8	13	9

Table 5.3: Size of the Gröbner basis computing using Buchberger's algorithm (Buch), a dynamic Buchberger algorithm (dynB) [19], the F5 algorithm, and the dynamic F5 algorithm (DynF5) described here.

number of zero polynomials computed. This helps to explain why we compute more zero polynomials than the traditional F5 algorithm in these four systems.

Table 5.3 compares the final sizes of the Gröbner bases computed. We remind the reader that the algorithm computes a minimal Gröbner basis, so the final basis size represents the number of S-polynomials computed minus not only the number of zero polynomials computed but also the elements of G whose leading monomials were divisible by another leading monomial of the basis. With the exception of Noon3 and Katsura-5(h), the dynamic F5 algorithm consistently returns a smaller basis than traditional F5, as we expected from a dynamic approach. We were happy to find that dynamic F5 also outperformed dynamic Buchberger in many cases. For example, Cyclic-6h had a final basis size of 28 polynomials

using our dynamic F5 algorithm, while the dynamic Buchberger algorithm had a final basis with 38 polynomials. For this same system, both the traditional Buchberger and the traditional F5 algorithms returned a Gröbner basis with 98 polynomials. In many cases where our algorithm did not do better than dynamic Buchberger, it was not far behind. For instance, our dynamic F5 algorithm returned a basis of 110 polynomials for Cyclic-7h compared to 107 polynomials using dynamic Buchberger. Even better, our basis is less than one-fourth of the size of the bases computed by the traditional Buchberger and traditional F5 algorithms, both having 443 polynomials.

Overall, the dynamic F5 algorithm outperforms its static counterpart. Although, we occasionally compute more zero polynomials than traditional F5, we still reap the benefits of avoiding many of the reductions to zero performed by Buchberger style algorithms. We were also generally able to compute fewer S-polynomials and still achieve smaller basis sizes than both traditional Buchberger and traditional F5. In these ways, the algorithm performs exactly as one would expect. It avoids reductions to zero as much as possible while also minimizing the number of polynomials in the final basis.

5.4 Future Work

Table 5.4 shows the timings for running our dynamic F5 algorithm compared to a traditional F5 algorithm. The times were obtained by running the code in Appendix A in an instance of CoCalc at https://bagheera.usm.edu/. This page runs Sage version 8.9 [21] on a 24-core Intel Xeon X5690 @3.47 GHz with 64 GB memory using Ubuntu 7.4.0 with the 4.15.0-74-generic Linux kernel. The times for the static algorithm were obtained after removing any lines of code that make the algorithm dynamic, such as updating the linear program. All times are an average of the times obtained by three full computations. We see that in the larger homogeneous Cyclic-n systems, the dynamic version ran faster than this static version. This is due to there being fewer S-polynomials to compute when working with the final generator. The dynamic algorithm chooses h^n to be the leading monomial for f_n in any Cyclic-n(h) system, because it has the best Hilbert data. Since h^n is relatively prime to *any* other leading monomial in the basis, there are no additional critical pairs to add at this point, so the algorithm terminates.

For any other system, we see that our dynamic F5 algorithm comes with a high computational cost. This is due in part to algorithm's need to preserve signature order. As mentioned in Section 5.2.3, we sometimes do not choose the leading monomial with the best Hilbert data, because it would corrupt the order of the signatures. This means that we choose one of the "runner ups." From Section 3.1, we know that choosing the leading monomial with the

	Time in seconds	
	F5	DynF5
Cyclic-4	0.02	0.08
Cyclic-4 (h)	0.02	0.09
Cyclic-5	0.71	9.60
Cyclic-5 (h)	0.78	1.44
Cyclic-6	17.70	82.32
Cyclic-6 (h)	19.28	4.56
Cyclic-7 (h)	2030.85	1034.88
Eco5	0.08	0.25
Eco5(h)	0.15	0.93
Eco6	0.32	1.14
Eco6(h)	0.94	4.84
Eco8	17.13	232.74
Eco8(h)	51.09	961.28
Katsura-5	1.17	38.16
Katsura-5(h)	1.17	2.23
Noon3	0.07	0.18
Noon4	0.55	1.33
Noon4(h)	0.55	0.77
Noon5	5.03	174.76
Trinks	0.11	0.35

Table 5.4: Computational time using the traditional F5 algorithm and the dynamic F5 algorithm (DynF5) described here.

best Hilbert data gets us closer to a Gröbner basis, so removing that option causes us to have to do more work to achieve a Gröbner basis. However, the largest cost comes from solving the linear program associated with our leading monomials. It would be worth investigating methods for reducing the size of the program in order to reduce the cost of solving it.

Remark 5.4.1. After approval of this thesis, we discovered a bug wherein the signatures were sometimes not reevaluated correctly, despite an explicit command to do so. This error occurs in sort_S, where the command to coerce a signature term to the new ring is not obeyed. This appears to be a bug in Sage. We know of a workaround; however, it was too late to fix with the attention necessary, so we do not add data for some bases that we could have added.

Appendix A

SOURCE CODE

```
#these two definitions are used to check if
#we have actually found a Groebner basis
def reduce_by_nosig(f, G):
    while any(g.lm().divides(f.lm()) for g in G):
        for g in G:
            if g.lm().divides(f.lm()):
                t = f.lm().quo_rem(g.lm())[0]*f.monomial_coefficient(f.lm())
                f -= t*g
                if f == 0: return f
    raise ValueError('Not a GB')

def check_basis(T, G):
    G = [ g.poly() for g in G ]
    reductions = []
    for i in range(len(G)):
        for j in range(i + 1, len(G)):
            tij = lcm(T[i], T[j])
            ui = tij.quo_rem(T[i])[0]
            uj = tij.quo_rem(T[j])[0]
            s = ui*G[i] - uj*G[j]
            reductions.append(reduce_by_nosig(s, G))
    return reductions

#create a signature
class PolynomialSignature(list):
        #initialize to signature [0], a monomial, and
        #signature[1], a positive integer
    def __init__(self, signature):
        verbose('polysig', level = 3)
        verbose((signature, type(signature)), level = 3)
        self._term = signature[0]
        self._index = signature[1]
    def __getitem__(self, key):
        if key == 0:
            return self._term
        elif key == 1:
            return self._index
        else:
```

```
            raise ValueError("Signature␣Key␣must␣be␣0␣or␣1")
        #returns a representation "sigma*e_i" where
        #sigma is signature[0] and i is signature[1]
        def __repr__(self):
            return "%s*e%s"%(self._term, self._index)
        #check for equality
        def __eq__(self, other):
            return (self._term == other._term and self._index == other._index)
        def __ne__(self, other):
            return not self == other
        def term(self):
            return self._term
        def index(self):
            return self._index
        def max_sig(self, other):
            if self._index > other._index:
                return self
            elif self._index == other._index:
                if self._term > other._term:
                    return self
                else:
                    return other
            else:
                return other
        #multiply the signature by a monomial. This only multiplies
        #sigma by the monomial
        def mult_monomial(self, monomial):
            return PolynomialSignature((monomial*self._term, self._index))
        #returns True if and only if adding t*g to self would
        #corrupt the signature
        def sig_corrupt(self, g, t):
            #if the index of the signature we are trying to
            #reduce by is larger, return True for signature
            #corruption
            if g._index > self._index: return True
            #if the indeces are the same, but sigma of the
            #index we are trying to reduce by is larger,
            #return True for signature corruption
            if g._index == self._index:
                if g.mult_monomial(t)._term >= self._term:
                    return True
            return False

#initialize the polynomial and its signature
#creates (sig, poly)
class SignedPolynomial(list):
        #initialize to SignedPolynomial[0], a Polynomial Signature,
```

```
        #and to SignedPolynomial[1], a polynomial
    def __init__(self, signedpoly):
        self._sig = PolynomialSignature(signedpoly[0])
        self._poly = signedpoly[1]
    def __eq__(self, other):
        return (self._sig == other._sig and self._poly == other._poly)
    def __ne__(self, other):
        return not self == other
    #return the leading monomial
    def lt(self):
        return self._poly.lm()*self._poly.lc()
    #returns the representation
    #(sigma*e_i, poly), where sigma*e_i is the
    #PolynomialSignature of poly
    def __repr__(self):
        return "(%s, %s)"%(self._sig, self._poly)
    #return the PolynomialSignature
    def sig(self):
        verbose('sig', level = 3)
        return self._sig
    #return the polynomial
    def poly(self):
        return self._poly
    #multiply the polynomial and its signature by monomial
    def multiply(self, monomial):
        sig = self._sig.mult_monomial(monomial)
        poly = monomial*self._poly
        return SignedPolynomial((sig, poly))
#return the signed S-polynomial of self and poly
def generate_s_poly(self, poly):
        lcm_of_pair = lcm(self.lt(), poly.lt())
        t = lcm_of_pair.quo_rem(self.lt())[0]
        u = lcm_of_pair.quo_rem(poly.lt())[0]
        s_poly = t*self._poly - u*poly._poly
        #check for signature corruption
        if (self._sig.max_sig(poly.sig())).is_equal(self.sig()):
            sig = self._sig
            return SignedPolynomial((sig, s_poly))
        else:
            raise ValueError("Signature corruption.")
    #return the polynomial reduced by self*mon
    #we only perform this after checking for signature corruption
    def reduce_by(self, poly, mon):
        f = self.poly()
        g = poly.poly()
        t = mon.quo_rem(g.lm())[0]*f.monomial_coefficient(mon)
        f -= t*g
```

```
            self._poly = f
            return self

#a class for critical pairs of signed polynomials
class Pair(list):
            #initialize to Pair[0], a PolynomialSignature, Pair[1], the
            #first SignedPolynomial in the critical pair, and to Pair[2],
            #the second SignedPolynomial in the critical pair
        def __init__(self, pair):
            self._sig = PolynomialSignature(pair[0])
            self._poly1 = pair[1]
            self._poly2 = pair[2]
        #represent the critical pair as "(sigma*e_i, f, g)", where
        #sigma*e_i is the PolynomialSignature of the critical pair,
        #f is the first polynomial, and g is the second
        def __repr__(self):
            return "(%s, %s, %s)"%(self._sig, self._poly1, self._poly2)
        #return the signature of the critical pair
        def sig(self):
            verbose('sig', level = 3)
            return self._sig
        #return the first polynomial
        def poly1(self):
            return self._poly1
        #return the second polynomial
        def poly2(self):
            return self._poly2
        def __eq__(self, other):
            return (self._sig == other._sig and self._poly1[0] == other._poly1[0] \
                    and self._poly1[1] == other._poly1[1] and self._poly2[0] == \
                    other._poly2[0] and self._poly2[1] == other._poly2[1] )
        def __ne__(self, other):
            return not self == other

#initialize a polynomial ring and return the ring
def initialize_ring(vars, ring):
        if type(vars) == Integer:
            n = vars
            R = PolynomialRing(ring, 'x', n)
        else:
            R = PolynomialRing(ring, vars)
        R.inject_variables()
        return R

#initialize the ideal and return the ring R, the ideal I,
#the sorted generators F, the ring generators X, the weight
#vector lp, and Y, a list of variables for the linear program
```

```
def initialize_ideal(polys, homogenize = False):
    R = polys[0].parent()
    if (homogenize):
        I = R.ideal(polys).homogenize()
        R = I.ring()
        R.inject_variables()
    else:
        I = R.ideal(polys)
    F = list(I.gens())
    F.sort(key = lambda f: f.lm())
    for i in range(len(F)):
        F[i] = SignedPolynomial(((R(1), i), F[i]))
        #F keeps up with the signatures and their corresponding polynomials
        #here, we are just declaring the original signatures f_1 -> (1*e_1, f_1), etc.
    X = I.ring().gens()
    lp, Y = initialize_lp(X, homogenize)
    return R, I, F, X, lp, Y

#initialize the linear program to a vector of ones
#return this weight vector lp and Y, a list of variables for the
#linear program
#to be used in solving the linear program
def initialize_lp(X, homogenize = False):
    lp = MixedIntegerLinearProgram(solver='ppl', maximization=False)
    Y = [ lp[i] for i in range(len(X)) ]
    for y in Y:
        lp.set_integer(y)
        lp.add_constraint(y >= 1)
    lp.set_objective(sum(y for y in Y))
    lp.solve()
    return lp, Y

#update lp based on new lm and return the new weight vector
def update_lp(f, lp, Y, T, X, R, G, S, trace, homogenize = False):
    i = 0
    poly = f
    works = False
    bad_t = False
    lp_temp = copy(lp)
    lp_temp.solve()
    pp_old = 1
    while not works:
        lp_new = copy(lp_temp)
        if bad_t == True:
            U = [ u for u in U if u[0] != t ]
        else: U = compatible_pps(poly, lp_temp, Y)
        #if there are no monomials to compare, exit the loop
```

```
            if len(U) == 1: return lp
            #otherwise, use the hilbert heuristic to find the
            #'best' lm
            t = preferred_pp(T, [u[0] for u in U], R)
            a = t.exponents(as_ETuples=False)[0]
            new_expr = sum(a[i]*Y[i] for i in range(len(a)))
            #make it the leading monomial
            for u in [ v[0] for v in U ]:
                if u != t:
                    b = u.exponents(as_ETuples=False)[0]
                    lp_new.add_constraint(new_expr >= sum(b[i]*Y[i] for \
                                    i in range(len(b))) + 1)
            try:
                lp_new.solve()
            except:
                #if updating the lp creates an infeasible system,
                #remove t from the options
                bad_t = True
                continue
            #insure that the signatures remained in the same order AND
            #the prior leading monomials remain unchanged
            if verify_signature_order(R, lp_temp, lp_new, Y, S, trace):
                T_temp = copy(T)
                T_temp.append(t)
                works, lp_temp = verify_order(T_temp, G, R, lp_temp, lp_new, Y, X)
                pp_old = t
                bad_t = False
            else:
                bad_t = True
                continue
        lp_temp.solve()
        return lp_temp

#return a matrix order with weight vector v and
#grevlex to break ties
def matrix_order(v, homogenize = False):
    if homogenize:
        vlen = len(v) - 1
    else:
        vlen = len(v)
    A = matrix(ZZ, vlen, vlen)
    for i in range(vlen):
        A[0,i] = v[i]
    for i in range(1,vlen):
        for j in range(vlen-i):
            A[i,j] = 1
    return TermOrder(A)
```

```
#returns the monomials of f that are compatible for f
#lp tells us our current weight vector and Y is
#a list of variables for the linear program
def compatible_pps(f, lp, Y):
    result = [(f.lm(),lp.get_values(Y))]
    TOs = list(ray.vector() for ray in lp.polyhedron().rays())
    # test every monomial of f
    for t in f.monomials()[1:]:
        b = vector(t.exponents()[0]) # exponent vector
        for v in TOs:
            success = True # innocent until proven guilty
            # test against every other monomial in f
            for u in result:
                if v * b <= v * vector(u[0].exponents()[0]):
                    # give up on this ray; try another
                    success = False
                    break
            if success:
                # we'll add both t and a "certificate" that t can lead
                result.append((t,v))
                # no point in remaining in the ray loop, since we've found a
                #winner for t
                break
    return result

#apply the new monomial order and return the current basis under the
#new weight ordering G, the updated ring R2, and the updated generators F
def redefine_vals(lp, Y, R, G, F):
    TO = matrix_order(lp.get_values(Y))
    R2 = R.change_ring(order=TO)
    #reorder polys in G according to new weight vector
    G = [ SignedPolynomial(((R2(g.sig().term()), g.sig().index()), \
            R2(g.poly()))) for g in G ]
    #reorder polys in F according to new weight vector
    F = [ SignedPolynomial(((R2(f.sig().term()), f.sig().index()), \
            R2(f.poly()))) for f in F ]
    if G[-1].poly() != 0:
        #insure that the lc is 1
        if G[-1].poly().coefficient(G[-1].poly().lm()) != 1:
            G[-1] = SignedPolynomial(((G[-1].sig().term(), G[-1].sig().index()), \
            G[-1].poly().quo_rem(G[-1].poly().coefficient(G[-1].poly().lm()))[0]))
    return G, R2, F

# return the leading monomial that is "best" using the
#hilbert heuristic. T is old basis's lms, U is list of compatible
#monomials to test
```

```
def preferred_pp(T, U, R):
    u0 = U[0]
    V = T + [u0]
    #create ideals with prior lms and the new options
    hp = R.ideal(V).hilbert_polynomial(algorithm='singular')
    hn = R.ideal(V).hilbert_numerator(algorithm='singular')
    result = u0
    for u in U[1:]:
        V = T + [u]
        hp2 = R.ideal(V).hilbert_polynomial(algorithm='singular')
        hn2 = R.ideal(V).hilbert_numerator(algorithm='singular')
        if hilbert_cmp((result,hp,hn),(u,hp2,hn2)) > 0:
            result, hp, hn = u, hp2, hn2
    return result

#return a negative integer if H1 is preferable to H2,
#a positve integer if H2 is preferable, and 0 if H1 and
#H2 are the same. H1 and H2 represent the Hilbert numerator
#and Hilbert polynomial for two choices of leading monomials
def hilbert_cmp(H1, H2):
    hpdiff = H1[1] - H2[1]
    #if the hilbert polynomial is smallest, choose as
    #preferred lm
    if hpdiff != 0:
        result = hpdiff.leading_coefficient()
    #otherwise compare the hilbert numerators
    else:
        hndiff = (H1[2] - H2[2]).coefficients(sparse=False)
        hndiff.reverse()
        while len(hndiff) > 0 and hndiff[0] == 0:
            hndiff = hndiff[1:]
        if len(hndiff) == 0:
                #if the Hilbert data is the same, choose the
                #monomial that is smaller under the monomial order
                #-1 will choose the monomial associated with H1,
                #1 will choose the monomial associated with H2,
                #0 means they have the same weighted degree and
                #defaults to the monomial associated with H1
            if H1[0] < H2[0]: result = -1
            elif H1[0] > H2[0]: result = 1
            else: result = 0
        else:
        #choose the monomial associated with the smallest
        #trailing term of the Hilbert numerator
            result = hndiff[0]
    return result
```

```
#return a list of "critical pairs" for the generators
def generate_first_pairs(F):
    #create pairs associated with zero for the initial polynomials
    #so we can add f_1, f_2, etc to the basis without an S−poly
    P = []
    S = []
    for i in range(len(F)):
        P.append(Pair((F[i].sig(), (1, i), (0, 0))))
    S.append(P[0])
    P.remove(P[0])
    return P, S

#return a sorted list of the new critical pairs generated by
#adding the most recent element to the basis
def generate_pairs(G, S, R2, lp, Y, T, X, homogenize):
    g = G[−1]
    lm_g = g.poly().lm()
    for i in range(len(G) − 1):
        f = G[i]
        if f.poly() == 0: continue
        least = lcm(lm_g, f.poly().lm())
        t = least.quo_rem(lm_g)[0]
        u = least.quo_rem(f.poly().lm())[0]
        #if they have the same sig, move on
        if f.multiply(u).sig() == g.multiply(t).sig():
            continue
        #assign the max signature to the pair
        sig = f.multiply(u).sig().max_sig(g.multiply(t).sig())
        if not syzygy_check(G, f.multiply(u).sig(), g.multiply(t).sig()):
            if sig == f.multiply(u).sig():
                S.insert(0, Pair((sig, (u, i), (t, len(G) − 1))))
            else:
                S.insert(0, Pair((sig, (t, len(G) − 1), (u, i))))
    return sort_S(S, R2, lp, Y, T, X, G)

#return a sorted list of critical pairs
def sort_S(S, R, lp, Y, T, X, G):
    if len(S) > 1:
        S.sort(key = lambda s:(R(s.sig().term()).degree(), s.sig().term()))
    return S, lp

#compute the s−poly and a signature safe reduction for pair, then
#adds it to the basis G
#returns the number of s−polynomials computed so far (spoly),
#the number of zero polys computed (zero), a list of critical pairs that
#have already been processed (generated), a boolean for whether or not
#we have computed a zero poly (zero_poly), the current basis (G),
```

```
#the updated linear program (lp), and the updated list of rewriting
#rules (rule)
def create_poly(spoly, zero, generated, G, F, R, T, X, pair, lp, Y, \
                    rule, S, trace, homogenize = False):
    #pair is (sig, (multiple of e_i, e_i #)
    #[Ex. (1, 0) for 1*e_0], (multiple of e_j, e_j #))
    zero_poly = False
    generated.append(pair)
    spoly += 1
    rule.append((pair.sig(), len(G)))
    if pair.poly2()[0] == 0: # <-- if it is one of the original polynomials
        if pair.poly1()[1] != 0:
            s = reduce_all_clo(F[pair.poly1()[1]], G, R)
        else: s = F[pair.poly1()[1]]
        t = s.poly();
        #t gives the polynomial associated with e_i
    else:
        #compute the s-poly and reduce by elements with smaller
        #signature
        s = G[pair.poly1()[1]].poly()*(pair.poly1()[0]) - \
            G[pair.poly2()[1]].poly()*(pair.poly2()[0])
        F.append(SignedPolynomial((pair.sig(), R(s))))
        s = reduce_all_clo(F[-1], G, R)
        t = s.poly()
    G.append(s)
    if t == 0:
        zero_poly = True
        zero += 1
    else:
        lp = update_lp(t, lp, Y, T, X, R, G, S, trace, homogenize)
    return spoly, zero, generated, zero_poly, G, lp, rule

#returns True iff sig1 or sig2 is a syzygy signature
def syzygy_check(G, sig1, sig2):
    if check_gs(G, R, sig1) or check_gs(G, R, sig2): return True
    return False

#returns True if there is a g in the basis whose signature has
#a lower index than sig and whose leading monomial divides sig
def check_gs(G, R, sig):
    if any(g.poly() != 0 and R.monomial_divides(g.poly().lm(), \
            sig.term()) for g in G if g.sig().index() < sig.index()):
        return True
    return False

#returns True iff there is an r in rule such that the index of r
#equal the index of p, the signature of r divides the signature of p
```

```
#and the polynomial associated with r is not associated with p
#rule should be a list of tuples of the form (sig, poly #), where sig is
#a signature and poly # is the index of a poly in the basis
#p is pair, R is the ring
def rewritten(rule, p, R):
    i = len(rule)−1
    while i > −1 and rule[i][0].index() == p.sig().index():
        if R.monomial_divides(R(rule[i][0].term()), p.sig().term()):
            if rule[i][1] == p.poly1()[1] or rule[i][1] == p.poly2()[1]:
                return False
            else:
                return True
        i = i − 1
    return False

#returns a list of signature redundant SignedPolynomials(sig_red)
#and True iff there exists (tau, g) in the basis such
#that tau divides sig(f) and lm(g) divides lm(f)
def sig_redundant(sig_red, G, f):
    if len(G) == 1: return sig_red, False
    if f.poly() == 0:
        return sig_red, False
    if any(g.poly() != 0 and R.monomial_divides(g.sig().term(), f.sig().term())\
           and R.monomial_divides(g.poly().lm(), f.poly().lm()) for g in G):
        sig_red.append(f.poly())
        return sig_red, True
    return sig_red, False

#returns the signature safe reduction of f by G according to a modified
#version of the division algorithm found in Cox, Little, O'Shea
def reduce_all_clo(f, G, R):
    #division algorithm
    r = 0
    while f.poly() != 0:
        reduced = False
        if any(g.poly() != 0 and R.monomial_divides(g.poly().lm(), f.poly().lm()) and not \
                f.sig().sig_corrupt(g.sig(), f.poly().lm().quo_rem(g.poly().lm())[0])\
                for g in G):
            for g in G:
                if g.poly() == 0: continue
                if R.monomial_divides(g.poly().lm(), f.poly().lm()):
                    t = f.poly().lm().quo_rem(g.poly().lm())[0]
                    if not f.sig().sig_corrupt(g.sig(), t):
                        f.reduce_by(g, f.poly().lm())
                        reduced = True
                        break
        if not reduced:
```

```
                r += f.poly().lm()*(f.poly().coefficient(f.poly().lm()))
                f._poly -= f.poly().lm()*(f.poly().coefficient(f.poly().lm()))
        f._poly = r
        return f

#returns the signature safe reduction of f by the basis elements
def reduced_basis(f, G, R):
        s = SignedPolynomial((f.sig(), f.poly()))
        r = f.poly().lm()
        s._poly -= r
        while s.poly() != 0:
                reduced = False
                if any(R.monomial_divides(g.poly().lm(), s.poly().lm()) and not \
                        s.sig().sig_corrupt(g.sig(), s.poly().lm().quo_rem(g.poly().lm())[0])\
                        for g in G):
                        for g in G:
                                if R.monomial_divides(g.poly().lm(), s.poly().lm()):
                                        t = s.poly().lm().quo_rem(g.poly().lm())[0]
                                        if not s.sig().sig_corrupt(g.sig(), t):
                                                s.reduce_by(g, s.poly().lm())
                                                reduced = True
                                                break
                if not reduced:
                        r += s.poly().lm()*(s.poly().coefficient(s.poly().lm()))
                        s._poly -= s.poly().lm()*(s.poly().coefficient(s.poly().lm()))
        s._poly = r
        return s

#returns True iff lm(f) is not divisible by another leading monomial in
#G
def remove_divisible_lts(f, G, R):
        #pass a polynomial to check (f), remove f from G, then test for divisibility
        lm = f.poly().lm()
        for g in G:
                if g.poly() == 0: continue
                if f != g:
                        if R.monomial_divides(g.poly().lm(), lm):
                                return True
        return False

#returns True and the updated linear program (lp_temp)
#if the leading monomials of the basis elements are
#unchanged under lp_temp
#returns False and the prior linear program (lp) if updating
#the linear program changes the lms in the basis
def verify_order(T, G, R, lp, lp_temp, Y, X):
        TO = matrix_order(lp_temp.get_values(Y))
```

```
    R2 = R.change_ring(order = TO)
    H = [ g.poly() for g in G ]
    for g in G:
        if g.poly() == 0: H.remove(g.poly())
    T_new = [i for i in T if i != 0]
    H_new = [ R2(h).lm() for h in H ]
    if T_new == H_new: return True, lp_temp
    #if the monomial order changed, try adding constraint to the linear
    #program to preserve the old order
    for j in range(len(T_new) - 1):
        if T_new[j] != H_new[j]:
            lp.add_constraint(sum(T_new[j].degree(X[i], std_grading = True)*Y[i]\
                    for i in range(len(X))) >= sum(H_new[j].degree(X[i], \
                    std_grading = True)*Y[i] for i in range(len(X))) + 1)
    return False, lp

#returns True iff the prior signatures remain in the same
#order under the new linear program AND there are no signatures
#waiting to be computed that have a smaller signature than
#a basis element
def verify_signature_order(R, lp_old, lp_new, Y, S, trace):
    TO = matrix_order(lp_new.get_values(Y))
    R2 = R.change_ring(order = TO)
    trace1 = [ R2(r) for r in trace ]
    trace2 = sorted(trace1)
    #check the prior signature order against their order under the
    #new monomial order
    if trace1 != trace2:
        return False
    #also check that the new monomial order doesn't cause smaller
    #signatures waiting to be computed
    if len(S) > 0 and R2(trace[-1]) > R2(S[0].sig().term()):
        return False
    return True

#returns a reduced, minimal Groebner basis by removing zero polynomials,
#elements whose leading monomial is divisible by another basis lm,
#and reduces the remaining elements by other basis elements
def clean_g(G, sig_red, R):
    #get rid of polys with same leading term and reduce each poly
    G_new = []
    for g in G:
        if g.poly() != 0 and not (g.poly() in sig_red or remove_divisible_lts(g, G, R)):
            G_new.append(g)
    for g in G_new:
        g = reduced_basis(g, G_new, R)
    return G_new
```

```
#returns a signature Groebner basis (G), the corresponding
#weight vector (w), and a list of leading monomials of basis
#elements (T)
#needs as input a list of generators (polys)
#if you want the system to be homogenized, input
#homogenize = True
def f5_dynamic(polys, homogenize = False):
    R, I, F, X, lp, Y = initialize_ideal(polys, homogenize)
    #keep up with the number of spolys
    spoly = 0
    #keep up with signature redundant and zeros
    sig_red = []
    num_red = 0
    zero = 0
    #T keeps up with leading monomials
    T = [];
    #G keeps track of the GB
    G = [];
    #rule keeps up with rewrite rules
    rule = [];
    trace = []
    generated = []
    P, S = generate_first_pairs(F)
    pair = S[0]
    S.remove(pair)
    spoly, zero, generated, zero_poly, G, lp, rule = create_poly(spoly, zero, generated,\
                            G, F, R, T, X, pair, lp, Y, rule, S, trace, homogenize)
    G, R2, F = redefine_vals(lp, Y, R, G, F);
    T = [ g.poly().lm() for g in G ];
    S, lp = generate_pairs(G, S, R2, lp, Y, T, X, homogenize);
    G, R2, F = redefine_vals(lp, Y, R, G, F);
    deg = R2(pair.sig().term()).degree()
    while len(P) > 0:
        generated = []
        rule = []
        trace = []
        S.append(P[0])
        P.remove(P[0])
        print 'working␣on␣', S[0]
        while len(S) > 0:
            S_empty = False
            pair = S[0]
            S.remove(pair)
            trace.append(pair.sig().term())
            if pair in generated:
                continue
```

```
            if not rewritten(rule, pair, R2):
                spoly, zero, generated, zero_poly, G, lp, rule= create_poly(spoly,\
                  zero, generated, G, F, R2, T, X, pair, lp, Y, rule, S, trace, homogenize)
                if len(G)%20 == 0: print 'Size of G: ', len(G)
                G, R2, F = redefine_vals(lp, Y, R, G, F);
                sig_red, redundant = sig_redundant(sig_red, G[:-1], G[-1])
                if redundant:
                    rule.pop(-1)
                T = [ g.poly().lm() for g in G ];
                if not redundant and not zero_poly:
                    S, lp = generate_pairs(G, S, R2, lp, Y, T, X, homogenize)
                    G, R2, F = redefine_vals(lp, Y, R, G, F);
        G = clean_g(G, sig_red, R2)
        print 'current size of G: ', len(G)
        print 'current lp:', [ lp.get_values(y) for y in Y ]
        T = [ g.poly().lm() for g in G ];
        num_red += len(sig_red)
        sig_red = []
        print 's-polynomials computed:', spoly
        print 'signature redundant:', num_red
        print 'zero polynomials:', zero
num_red += len(sig_red)
sig_red = []
print 's-polynomials computed:', spoly
print 'signature redundant:', num_red
print 'zero polynomials:', zero
w = [ lp.get_values(y) for y in Y ]
G = clean_g(G, sig_red, R2)
print 'size of basis:', len(G)
return G, w, [ g.poly().lm() for g in G ]
```

Appendix B

POLYNOMIAL SYSTEMS

All computations were made using a ground field of characteristic 43.

B.1 Cyclic-4

$$
\begin{aligned}
f_0 &= x_0 + x_1 + x_2 + x_3 \\
f_1 &= x_0x_1 + x_1x_2 + x_2x_3 + x_0x_3 \\
f_2 &= x_0x_1x_2 + x_1x_2x_3 + x_0x_2x_3 + x_0x_1x_3 \\
f_3 &= x_0x_1x_2x_3 - 1 \\
(f_3 &= x_0x_1x_2x_3 - h^4 \quad \text{Cyclic-4(h)})
\end{aligned}
$$

B.2 Cyclic-5

$$
\begin{aligned}
f_0 &= x_0 + x_1 + x_2 + x_3 + x_4 \\
f_1 &= x_0x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_0x_4 \\
f_2 &= x_0x_1x_2 + x_1x_2x_3 + x_2x_3x_4 + x_0x_3x_4 + x_0x_1x_4 \\
f_3 &= x_0x_1x_2x_3 + x_1x_2x_3x_4 + x_0x_2x_3x_4 + x_0x_1x_3x_4 + x_0x_1x_2x_4 \\
f_4 &= x_0x_1x_2x_3x_4 - 1 \\
(f_4 &= x_0x_1x_2x_3x_4 - h^5 \quad \text{Cyclic-5(h)})
\end{aligned}
$$

B.3 Cyclic-6

$$
\begin{aligned}
f_0 &= x_0 + x_1 + x_2 + x_3 + x_4 + x_5 \\
f_1 &= x_0x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_0x_5 \\
f_2 &= x_0x_1x_2 + x_1x_2x_3 + x_2x_3x_4 + x_3x_4x_5 + x_0x_4x_5 + x_0x_1x_5 \\
f_3 &= x_0x_1x_2x_3 + x_1x_2x_3x_4 + x_2x_3x_4x_5 + x_0x_3x_4x_5 + x_0x_1x_4x_5 + x_0x_1x_2x_5 \\
f_4 &= x_0x_1x_2x_3x_4 + x_1x_2x_3x_4x_5 + x_0x_2x_3x_4x_5 + x_0x_1x_3x_4x_5 + x_0x_1x_2x_4x_5 + x_0x_1x_2x_3x_5 \\
f_5 &= x_0x_1x_2x_3x_4x_5 - 1 \\
(f_5 &= x_0x_1x_2x_3x_4x_5 - h^6 \qquad \text{Cyclic-6(h)})
\end{aligned}
$$

B.4 Cyclic-7

$$
\begin{aligned}
f_0 &= x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \\
f_1 &= x_0x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_6 + x_0x_6 \\
f_2 &= x_0x_1x_2 + x_1x_2x_3 + x_2x_3x_4 + x_3x_4x_5 + x_4x_5x_6 + x_0x_5x_6 + x_0x_1x_6 \\
f_3 &= x_0x_1x_2x_3 + x_1x_2x_3x_4 + x_2x_3x_4x_5 + x_3x_4x_5x_6 + x_0x_4x_5x_6 + x_0x_1x_5x_6 + x_0x_1x_2x_6 \\
f_4 &= x_0x_1x_2x_3x_4 + x_1x_2x_3x_4x_5 + x_2x_3x_4x_5x_6 + x_0x_3x_4x_5x_6 + x_0x_1x_4x_5x_6 + x_0x_1x_2x_5x_6 + \\
&\quad x_0x_1x_2x_3x_6 \\
f_5 &= x_0x_1x_2x_3x_4x_5 + x_1x_2x_3x_4x_5x_6 + x_0x_2x_3x_4x_5x_6 + x_0x_1x_3x_4x_5x_6 + x_0x_1x_2x_4x_5x_6 + \\
&\quad x_0x_1x_2x_3x_5x_6 + x_0x_1x_2x_3x_4x_6 \\
f_6 &= x_0x_1x_2x_3x_4x_5x_6 - 1 \\
(f_6 &= x_0x_1x_2x_3x_4x_5x_6 - h^7 \qquad \text{Cyclic-7(h)})
\end{aligned}
$$

B.5 Eco5

$$
\begin{aligned}
f_0 &= x_0 + x_1 + x_2 + x_3 + 1 \\
f_1 &= x_3 x_4 - 4 \\
f_2 &= (x_0 x_1 + x_1 x_2 + x_2 x_3 + x_0) x_4 - 1 \\
f_3 &= (x_0 x_2 + x_1 x_3 + x_1) x_4 - 2 \\
f_4 &= (x_0 x_3 + x_2) x_4 - 3
\end{aligned}
$$

B.6 Eco5(h)

$$
\begin{aligned}
f_0 &= x_0 + x_1 + x_2 + x_3 + h \\
f_1 &= x_3 x_4 - 4h^2 \\
f_2 &= (x_0 x_1 + x_1 x_2 + x_2 x_3 + x_0 h) x_4 - h^3 \\
f_3 &= (x_0 x_2 + x_1 x_3 + x_1 h) x_4 - 2h^3 \\
f_4 &= (x_0 x_3 + x_2 h) x_4 - 3h^3
\end{aligned}
$$

B.7 Eco6

$$
\begin{aligned}
f_0 &= x_0 + x_1 + x_2 + x_3 + x_4 + 1 \\
f_1 &= x_4 x_5 - 5 \\
f_2 &= (x_0 x_1 + x_1 x_2 + x_2 x_3 + x_3 x_4 + x_0) x_5 - 1 \\
f_3 &= (x_0 x_2 + x_1 x_3 + x_2 x_4 + x_1) x_5 - 2 \\
f_4 &= (x_0 x_3 + x_1 x_4 + x_2) x_5 - 3 \\
f_5 &= (x_0 x_4 + x_3) x_5 - 4
\end{aligned}
$$

B.8 Eco6(h)

$$
\begin{aligned}
f_0 &= x_0 + x_1 + x_2 + x_3 + x_4 + h \\
f_1 &= x_4 x_5 - 5h^2 \\
f_2 &= (x_0 x_1 + x_1 x_2 + x_2 x_3 + x_3 x_4 + x_0 h) x_5 - h^3 \\
f_3 &= (x_0 x_2 + x_1 x_3 + x_2 x_4 + x_1 h) x_5 - 2h^3 \\
f_4 &= (x_0 x_3 + x_1 x_4 + x_2 h) x_5 - 3h^3 \\
f_5 &= (x_0 x_4 + x_3 h) x_5 - 4h^3
\end{aligned}
$$

B.9 Eco8

$$
\begin{aligned}
f_0 &= x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + 1 \\
f_1 &= x_6 x_7 - 7 \\
f_2 &= (x_0 x_1 + x_1 x_2 + x_2 x_3 + x_3 x_4 + x_4 x_5 + x_5 x_6 + x_0) x_7 - 1 \\
f_3 &= (x_0 x_2 + x_1 x_3 + x_2 x_4 + x_3 x_5 + x_4 x_6 + x_1) x_7 - 2 \\
f_4 &= (x_0 x_3 + x_1 x_4 + x_2 x_5 + x_3 x_6 + x_2) x_7 - 3 \\
f_5 &= (x_0 x_4 + x_1 x_5 + x_2 x_6 + x_3) x_7 - 4 \\
f_6 &= (x_0 x_5 + x_1 x_6 + x_4) x_7 - 5 \\
f_7 &= (x_0 x_6 + x_5) x_7 - 6
\end{aligned}
$$

B.10 Eco8(h)

$$
\begin{aligned}
f_0 &= x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + h \\
f_1 &= x_6 x_7 - 7h^2 \\
f_2 &= (x_0 x_1 + x_1 x_2 + x_2 x_3 + x_3 x_4 + x_4 x_5 + x_5 x_6 + x_0 h) x_7 - h^3 \\
f_3 &= (x_0 x_2 + x_1 x_3 + x_2 x_4 + x_3 x_5 + x_4 x_6 + x_1 h) x_7 - 2h^3 \\
f_4 &= (x_0 x_3 + x_1 x_4 + x_2 x_5 + x_3 x_6 + x_2 h) x_7 - 3h^3 \\
f_5 &= (x_0 x_4 + x_1 x_5 + x_2 x_6 + x_3 h) x_7 - 4h^3 \\
f_6 &= (x_0 x_5 + x_1 x_6 + x_4 h) x_7 - 5h^3 \\
f_7 &= (x_0 x_6 + x_5 h) x_7 - 6h^3
\end{aligned}
$$

B.11 Katsura-5

$$
\begin{aligned}
f_0 &= 2x + 2y + 2z + 2t + 2u + v - 1 \\
f_1 &= t^2 + 2xv + 2yv + 2zv - y \\
f_2 &= 2xt + 2yu + 2tu + 2zv - z \\
f_3 &= 2xz + 2yt + 2zu + u^2 + 2tv - t \\
f_4 &= xy + yz + 2zt + 2tu + 2uv - u \\
f_5 &= 2x^2 + 2y^2 + 2z^2 + 2t^2 + 2u^2 + v^2 - v
\end{aligned}
$$

B.12 Noon3

$$
\begin{aligned}
f_0 &= 10x_0x_1^2 + 10x_0x_2^2 - 11x_0 + 10 \\
f_1 &= 10x_1x_0^2 + 10x_1x_2^2 - 11x_1 + 10 \\
f_2 &= 10x_2x_0^2 + 10x_2x_1^2 - 11x_2 + 10
\end{aligned}
$$

B.13 Noon4

$$
\begin{aligned}
f_0 &= 10x_0x_1^2 + 10x_0x_2^2 + 10x_0x_3^2 - 11x_0 + 10 \\
f_1 &= 10x_1x_0^2 + 10x_1x_2^2 + 10x_1x_3^2 - 11x_1 + 10 \\
f_2 &= 10x_2x_0^2 + 10x_2x_1^2 + 10x_2x_3^2 - 11x_2 + 10 \\
f_3 &= 10x_3x_0^2 + 10x_3x_1^2 + 10x_3x_2^2 - 11x_3 + 10
\end{aligned}
$$

B.14 Noon4(h)

$$
\begin{aligned}
f_0 &= 10x_0x_1^2 + 10x_0x_2^2 + 10x_0x_3^2 - 11x_0h^2 + 10h^3 \\
f_1 &= 10x_1x_0^2 + 10x_1x_2^2 + 10x_1x_3^2 - 11x_1h^2 + 10h^3 \\
f_2 &= 10x_2x_0^2 + 10x_2x_1^2 + 10x_2x_3^2 - 11x_2h^2 + 10h^3 \\
f_3 &= 10x_3x_0^2 + 10x_3x_1^2 + 10x_3x_2^2 - 11x_3h^2 + 10h^3
\end{aligned}
$$

B.15 Noon5

$$
\begin{aligned}
f_0 &= 10x_0x_1^2 + 10x_0x_2^2 + 10x_0x_3^2 + 10x_0x_4^2 - 11x_0 + 10 \\
f_1 &= 10x_1x_0^2 + 10x_1x_2^2 + 10x_1x_3^2 + 10x_1x_4^2 - 11x_1 + 10 \\
f_2 &= 10x_2x_0^2 + 10x_2x_1^2 + 10x_2x_3^2 + 10x_2x_4^2 - 11x_2 + 10 \\
f_3 &= 10x_3x_0^2 + 10x_3x_1^2 + 10x_3x_2^2 + 10x_3x_4^2 - 11x_3 + 10 \\
f_4 &= 10x_4x_0^2 + 10x_4x_1^2 + 10x_4x_2^2 + 10x_4x_3^2 - 11x_4 + 10
\end{aligned}
$$

B.16 Trinks

$$
\begin{aligned}
f_0 &= 45y + 35u - 165v - 36 \\
f_1 &= 35y + 25z + 40t - 27u \\
f_2 &= 25yu - 165v^2 + 15x - 18z + 30t \\
f_3 &= 15yz + 20tu - 9x \\
f_4 &= -11v^3 + xy + 2zt \\
f_5 &= -11uv + 3v^2 + 99x
\end{aligned}
$$

BIBLIOGRAPHY

[1] Alberto Arri and John Perry. The F5 criterion revised. *Journal of Symbolic Computation*, 46(9):1017 – 1029, 2011.

[2] B. Buchberger. *An Algorithm for Finding the Basis Elements in the Residue Class Ring Modulo a Zero Dimensional Polynomial Ideal*. PhD thesis, 3 2006.

[3] Massimo Caboara. A dynamic algorithm for Gröbner basis computation. In *Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation*, ISSAC '93, pages 275–283, New York, NY, USA, 1993. ACM.

[4] Massimo Caboara and John Perry. Reducing the size and number of linear programs in a dynamic Gröbner basis algorithm. *Applicable Algebra in Engineering, Communication and Computing*, 25(1):99–117, Apr 2014.

[5] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer Publishing Company, Incorporated, 4th edition, 2015.

[6] Christian Eder and John Perry. F5C: A variant of Faugère's F5 algorithm with reduced Gröbner bases. *Journal of Symbolic Computation*, 45(12):1442 – 1458, 2010. MEGA'2009.

[7] Christian Eder and John Edward Perry. Signature-based algorithms to compute Gröbner bases. *Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation - ISSAC '11*, 2011.

[8] Jean Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, page 75–83, New York, NY, USA, 2002. Association for Computing Machinery.

[9] Jean-Charles Faugére. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139(1):61 – 88, 1999.

[10] Alessandro Giovini, Teo Mora, Gianfranco Niesi, Lorenzo Robbiano, and Carlo Traverso. "One Sugar Cube, Please" or selection strategies in the Buchberger algorithm. In *Proceedings of the ISSAC'91, ACM Press*, pages 5–4, 1991.

[11] Oleg Golubitsky. Converging term order sequences and the dynamic Buchberger algorithm. 2007. Unpublished preprint.

[12] Peter Gritzmann and Bernd Sturmfels. Minkowski Addition of Polytopes: Computational Complexity and Applications to Gröbner Bases. *SIAM J. Disc. Math*, 6(2):246–269, 1993.

[13] Martin Kreuzer and Lorenzo Robbiano. *Computational Commutative Algebra 2*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2005.

[14] Martin Kreuzer and Lorenzo Robbiano. *Computational Commutative Algebra 1*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2008.

[15] G. Langeloh. Unrestricted dynamic Gröbner basis algorithms. Master's thesis, Universidade Federal do Rio Grande do Sul Instituto de Informática Programa de Pós-Graduação em Computação, 3 2019.

[16] D. Lazard. Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations. In J. A. van Hulzen, editor, *Computer Algebra*, pages 146–156, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg.

[17] Teo Mora and Lorenzo Robbiano. The gröbner fan of an ideal. *Journal of Symbolic Computation*, 6(2):183 – 208, 1988.

[18] John Perry. A dynamic F4 algorithm to compute Gröbner bases. Preprint received through private communication; software available at `github.com/johnperry-math/DynGB.git`.

[19] John Perry. Exploring the dynamic Buchberger algorithm. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '17, pages 365–372, New York, NY, USA, 2017. ACM.

[20] J.H. Silverman. *A Friendly Introduction to Number Theory*. Pearson Prentice Hall, 2012.

[21] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.9)*, 2020. `https://www.sagemath.org`.

[22] Jan Verschelde. The database of polynomial systems, 2020. http://homepages.math.uic.edu/ jan/demo.html.