2021

Maximums of Total Betti Numbers in Hilbert Families

Jay White
University of Kentucky, jaywhitemath@gmail.com
Digital Object Identifier: https://doi.org/10.13023/etd.2021.187

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Jay White, Student

Dr. Uwe Nagel, Major Professor

Dr. Benjamin Braun, Director of Graduate Studies

Maximums of Total Betti Numbers in Hilbert Families

DISSERTATION

A dissertation submitted in partial
fulfillment of the requirements for
the degree of Doctor of Philosophy
in the College of Arts and Sciences
at the University of Kentucky

By
Jay White
Lexington, Kentucky

Director: Dr. Uwe Nagel, Professor of Mathematics
Lexington, Kentucky
2021

ABSTRACT OF DISSERTATION

Maximums of Total Betti Numbers in Hilbert Families

Fix a family of ideals in a polynomial ring and consider the problem of finding a single ideal in the family that has Betti numbers that are greater than or equal to the Betti numbers of every ideal in the family. Or decide if this special ideal even exists. Bigatti, Hulett, and Pardue showed that if we take the ideals with a fixed Hilbert function, there is such an ideal: the lexsegment ideal. Caviglia and Murai proved that if we take the saturated ideals with a fixed Hilbert polynomial, there is also such an ideal. We present a generalization of these two situations, an algorithm for determining the existence of these special ideals and finding them when they do exist, and some cases where we guarantee existence.

KEYWORDS: Betti numbers, Hilbert function, commutative algebra, free resolutions, lexicographic order, polynomial ring

Jay White

May 16, 2021

Maximums of Total Betti Numbers in Hilbert Families

By
Jay White

Dr. Uwe Nagel
Director of Dissertation

Dr. Benjamin Braun
Director of Graduate Studies

May 16, 2021
Date

To my wife, Samantha.

ACKNOWLEDGMENTS

There are many people that deserve credit and that I would like to thank for the endless support and help they have provided me.

I want to thank my advisor, Dr. Uwe Nagel, for his patience and encouragement, his willingness to work with me, and the many things that he has taught me throughout the last several years.

I also want to give credit to the many other faculty members and graduate students with whom I have worked, taught, and laughed. In particular, I want to thank those that worked in (and near) POT 702.

Thank you to my wife, family, friends, and especially my family at Immanuel Baptist Church, for being a constant encouragement to me throughout every part of my life.

Finally, and most importantly, I want to acknowledge Jesus Christ as my Creator, Savior, and King. To God be all of the glory.

TABLE OF CONTENTS

Acknowledgments iii

List of Tables v

List of Figures vi

Chapter 1 Introduction 1
1.1 Background 1
1.2 Main Question 2

Chapter 2 Counting in the Lexicographic Order 6
2.1 The Lexicographic Order 6
2.2 Intervals and Binomial Coefficients 7
2.3 Macaulay Representations 12
2.4 Basic Inequalities 17
2.5 A More Complex Inequality 19
2.6 The $m_{\geqslant}$-vector 26

Chapter 3 Maximum Betti Numbers 30
3.1 Lexsegment Ideals 30
3.2 The 0-Depth Case 33
3.3 Reduction of Variables 35
3.4 The Positive Depth Case 38

Chapter 4 The 4 variable Case 39
4.1 A Ratio of m's 39
4.2 Maximizing m_2 and β 46

Chapter 5 Algorithm for Computing Maximal Betti Numbers 56
5.1 A Recursion 56
5.2 Algorithms 60

Chapter 6 Examples and Counterexamples 63
6.1 Macaulay2 Examples 63
6.2 Counterexamples 65

Appendices 72
Appendix: MaxBettiNumbers Source Code 72

Bibliography 104

Vita 105

LIST OF TABLES

1.1 Table indicating when a D-depth family of ideals in a polynomial ring with $n+1$ variables is guaranteed to either contain an ideal with maximum Betti numbers or have unbounded Betti numbers. 5

LIST OF FIGURES

2.1 The set $\mathrm{Mon}[R]_{\leqslant 3}$ when $R = K[x_0, \ldots, x_2]$. 6
2.2 Monomials and intervals near $x_0^2 x_1 x_2^3$. 14

4.1 Visualization of Lemma 4.2.2 47
4.2 Visualization of L_1 in Lemma 4.2.3. 49

6.1 Polynomial vs Runtime for Different Algorithms (Log Plot) 66
6.2 The ideals A, B, C in Example 6.2.2. 67
6.3 The ideals A and B from Example 6.2.4. 69
6.4 The ideals A and B from Example 6.2.5. 71

Chapter 1 Introduction

1.1 Background

Free resolutions are of great interest in algebra and form the foundation for the study of homological algebra. In the 1890's Hilbert introduced the concept of a free resolution of a module, M, in order to derive properties of M.

Definition 1.1.1. A *free resolution* of a module, M, is an exact sequence of modules

$$\cdots \to F_1 \to F_0 \to M \to 0$$

where the F_i's are free modules.

Betti numbers are numerical information of resolutions, and along with resolutions, are the subject of much current research. These numbers can be difficult to compute, and it is even unknown which values they can take on. Often, we wish to at least find bounds on these Betti numbers. As we will discuss in Section 3.2, Bigatti [2], Hulett [9], and Pardue [11] showed that there is an ideal, called the lexsegment ideal, which has maximum graded Betti numbers over all ideals with a given Hilbert function. In Section 3.4, we will see that Caviglia and Murai [4] extended this by constructing an ideal that has maximum total Betti numbers over all saturated ideals with a given Hilbert polynomial. Both of these situations constrain the Hilbert function in some way. In Section 1.2, we will look at a more general way of constraining the Hilbert function, and we will go on to explore maximums for Betti numbers in the resulting, more general, families of ideals.

We will look at homogeneous ideals in the polynomial ring $S = K[x_0, \ldots, x_n]$. The two titular invariants of any ideal are its Hilbert function and its Betti numbers. If I is a monomial ideal, its Hilbert function in degree d equals the number of degree-d monomials in S that are not in I.

Definition 1.1.2. Given a homogeneous ideal $I \subset S$, we define the *Hilbert function of* S/I, denoted $h_{S/I}$, to be the function giving the dimension of the graded components of S/I. That is, for any integer d,

$$h_{S/I}(d) = \dim_K [S/I]_d.$$

Note that although $h_{S/I}$ is actually the Hilbert function of S/I, we can abuse notation and simply refer to it as the Hilbert function of I.

For our purposes, it will be sufficient to consider only a certain type of monomial ideal whose Betti numbers are determined by a simple formula due to Eliahou and Kervaire [6] (see Theorem 3.1.8.) The Betti numbers are determined using a special free resolution of I.

Definition 1.1.3. Given a homogeneous ideal $I \subset S$, a *graded free resolution of* I is a graded exact sequence of S-modules of the form

$$0 \to F_m \to \cdots \to F_0 \to I \to 0,$$

where the homomorphisms are graded and have degree 0 and each F_i is a free S-module. Such a finite resolution does exist by Hilbert's syzygy theorem [8].

A *minimal free resolution of* I is a graded free resolution such that the number of generators of each F_q is simultaneously minimal. That such a resolution exists is non-obvious. For a proof that it does, we refer to, e.g., Section 7 of [12]. We denote by $\beta_q^S(I)$ the number of generators of F_q in a minimal free resolution of I. These $\beta_q^S(I)$ values are called the *total Betti numbers* of the ideal I.

We can also define finer invariants analogously for *graded minimal free resolutions* of I. Each free module F_q can be uniquely written as $F_q = \bigoplus_d S^{\beta_{q,d}}(-d)$. The integers $\beta_{q,d}^S(I) := \beta_{q,d}$ are called the *graded Betti numbers* of I.

Definition 1.1.4. Given an ideal $I \subset S$, the *depth* of S/I is the length of the longest regular sequence $(\ell_1, \ldots, \ell_r)$ in S/I. We denote this by $\operatorname{depth}(S/I) := r$. Furthermore, we say that I is *saturated* if it has positive depth.

As with the Hilbert function, we will sometimes abuse notation and refer to $\operatorname{depth}(S/I)$ as the depth of I. Depth is closely related to free resolutions by the Auslander-Buchsbaum formula [1], which says the length of the resolution in Definition 1.1.3 must be $m = n + 1 - \operatorname{depth}(S/I)$.

1.2 Main Question

We now discuss the main question we are trying to answer. It is simple to compute the Hilbert function of an ideal when given the graded Betti numbers of that ideal. This map from Betti numbers to Hilbert functions is not bijective, but we wish to know information about preimages of this map. This is an important goal in the study of resolutions, and we will focus on studying maximums in these preimages. To achieve this, we will define two important concepts. This first definition is simply the definition of common concepts in the element-wise (product) partial order on Betti numbers.

Definition 1.2.1. Let $\mathcal{I}$ be a family of ideals in S. We say an ideal, $J \in \mathcal{I}$, has *maximum total Betti numbers in* $\mathcal{I}$ if, for each $I \in \mathcal{I}$,

$$\beta_q(J) \geqslant \beta_q(I) \text{ for every } q.$$

We say that J has *maximal total Betti numbers in* $\mathcal{I}$ if, for each $I \in \mathcal{I}$, either

$$\beta_q(J) = \beta_q(I) \text{ for every } q$$

or

$$\beta_q(J) > \beta_q(I) \text{ for some } q.$$

We say that $\mathcal{I}$ has *unbounded total Betti numbers* if there is no $N \in \mathbb{N}$ such that, for each $I \in \mathcal{I}$,

$$N > \beta_q(I) \text{ for every } q.$$

Finally, we say that ideals $I, J \in \mathcal{I}$ have *comparable total Betti numbers* if either

$$\beta_q(I) \geqslant \beta_q(J) \text{ for every } q$$

or

$$\beta_q(J) \geqslant \beta_q(I) \text{ for every } q.$$

We also define *maximum/maximal/comparable graded Betti numbers* analogously.

Definition 1.2.2. Let $\mathcal{I}$ be a family of ideals in S. We say that $\mathcal{I}$ is a *Hilbert family of ideals in* S if, for every pair of ideals $I, L \subset S$,

$$\big(I \in \mathcal{I} \text{ and } h_{S/L} = h_{S/I}\big) \Rightarrow L \in \mathcal{I}.$$

Furthermore, We say that $\mathcal{J} \subset \mathcal{I}$ is a *D-depth Hilbert family of ideals in* S if $\mathcal{J}$ is the subset of $\mathcal{I}$ containing exactly the ideals whose quotient has depth at least D.

In other words, a Hilbert family is the family of all ideals satisfying some constraint on the Hilbert function and a D-depth family has an additional constraint of a lower bound on the depth.

In general, a Hilbert family of ideals may not contain an ideal with maximum total Betti numbers. However, in certain cases, such ideals do exist. This leads to a plethora of questions we can ask. Some of these questions have been answered by others, some we will answer in later chapters, and some are still left open. Here is a sampling of such questions:

- When does a Hilbert family of ideals in S contain an ideal with maximum Betti numbers in that family?
- What ideals have maximal Betti numbers in the family?
- What if we look at a depth D family?
- What conditions can we put on the Hilbert function and still guarantee the existence of an ideal with maximum Betti numbers?

We will attack these problems through the remainder of the chapters. We will now give a brief overview of the structure of these chapters.

In Chapter 2, we will discuss properties of a particular order on the monomials in S: the lexicographic order. In particular, we will establish several inequalities and results about the combinatorics of the lexicographic order. The primary result is Proposition 2.6.6, which eventually leads to Proposition 3.3.5. The latter result allows us, in certain situations, to replace our consideration of ideals with simpler sets of monomials.

Chapter 3 begins with an overview of some known answers to the questions above about Hilbert families. Then we prove Lemma 3.3.2 which allows us to work with D-depth Hilbert families. After discussing some results about D-depth Hilbert families, we will arrive at Proposition 3.3.5, which will be pivotal for later results.

Chapter 4 is devoted to establishing one of our main results, Theorem 4.0.1. It shows that ideals with maximum Betti numbers do exist in a 1-depth family in a polynomial ring with less than 5 variables. On the other hand, Chapter 6 gives some counter-examples with a higher numbers of variables and depths, showing that Theorem 4.0.1 cannot be completely extended. See Table 1.1 for a summary of results on when an ideal with maximum Betti numbers exists.

In addition to theoretical results, we may wish to compute these maximum Betti numbers. To that end, we will describe an algorithm in Chapter 5 which is implemented in the Macaulay2 package "MaxBettiNumbers". This algorithm, along with modifications, is able to find saturated ideals with maximal Betti numbers in the 1-depth Hilbert family where bounds are set on the Hilbert function. As a result, the algorithm can give sharp upper bounds for the Betti numbers, and also determine when such bounds are simultaneously attainable.

Table 1.1: Table indicating when a D-depth family of ideals in a polynomial ring with $n+1$ variables is guaranteed to either contain an ideal with maximum Betti numbers or have unbounded Betti numbers.

<table>
<tr><th></th><th>Fixed Hilbert Function</th><th>Fixed Hilbert Polynomial</th><th>Bounds on Hilbert Function</th><th>Any Hilbert Family</th></tr>
<tr><td>$n \leqslant D+1$</td><td colspan="4">✓
3.3.4</td></tr>
<tr><td>$D=0$
$n \geqslant 2$</td><td rowspan="4"></td><td>✓
3.2.2</td><td>✓
3.2.3</td><td rowspan="4"></td></tr>
<tr><td>$D=1$
$n=3$</td><td rowspan="2">✓
3.4.2</td><td>✓
4.0.1</td></tr>
<tr><td>$D=1$
$n \geqslant 4$</td><td>✗
6.2.4, 6.2.5</td></tr>
<tr><td>$D \geqslant 2$
$n \geqslant D+2$</td><td colspan="2">✗
6.2.2, 6.2.3</td></tr>
<tr><td colspan="2">✓
3.2.1</td><td colspan="2">✗
6.2.1</td></tr>
</table>

Chapter 2 Counting in the Lexicographic Order

In this chapter, we will discuss the lexicographic order. The primary focus will be to count monomials that are in this order, with most attention on the max index of each monomial. While there are some other results that will be helpful, the main takeaway from this chapter is Proposition 2.6.6.

Section 2.1 will give some basic definitions. Sections 2.2 and 2.3 will explore a connection between intervals and Macaulay representations, culminating in Proposition 2.3.6. Sections 2.4 and 2.5 will use Proposition 2.3.6 to prove some inequalities, the most important being Proposition 2.5.6, which will lead to the proof of Proposition 2.6.6 in Section 2.6.

2.1 The Lexicographic Order

Throughout this chapter, we will be defining and developing a special ordering on the monomials in the polynomial ring $R = K[x_0, \ldots, x_n]$. We use the notation $a = x_0^{a_0} \cdots x_n^{a_n}$ when referring to monomials in R. We will also use $\operatorname{Mon} R$ to denote the set of monomials in R and $\operatorname{Mon}[R]_d$ to denote the set of degree d monomials in R.

Definition 2.1.1. Given monomials $a = x_0^{a_0} \cdots x_n^{a_n}$ and $b = x_0^{b_0} \cdots x_n^{b_n}$ in R, we will say that $a > b$ if $a_i > b_i$, where i is the smallest value such that $a_i \neq b_i$. This defines the *lexicographic* (or *"lex"*) order on monomials.

Note: with this definition, we have that $x_0 > x_1 > \cdots > x_n$.

Remark 2.1.2. The lexicographic order is a monomial order, which means that for all monomials a, b, and c, (1) $a \geqslant 1$ and (2) $a > b \Rightarrow ca > cb$.

Example 2.1.3. If $n = 2$, the set $\operatorname{Mon}[R]_3$ is ordered as follows:

$$x_0^3 > x_0^2x_1 > x_0^2x_2 > x_0x_1^2 > x_0x_1x_2 > x_0x_2^2 > x_1^3 > x_1^2x_2 > x_1x_2^2 > x_2^3.$$

To show the interaction between degrees, we can draw a triangular shape for the set $\operatorname{Mon}[R]_{\leqslant 3}$, as shown in Figure 2.1.

x_0^3	$x_0^2x_1$	$x_0^2x_2$	$x_0x_1^2$	$x_0x_1x_2$	$x_0x_2^2$	x_1^3	$x_1^2x_2$	$x_1x_2^2$	x_2^3
		x_0^2		x_0x_1	x_0x_2		x_1^2	x_1x_2	x_2^2
					x_0			x_1	x_2
									1

Figure 2.1: The set $\operatorname{Mon}[R]_{\leqslant 3}$ when $R = K[x_0, \ldots, x_2]$.

Definition 2.1.4. Given $a, b \in \mathrm{Mon}[R]_d$, we will define the following *interval* notation:

$$[a, b] := \{c \in \mathrm{Mon}[R]_d \mid a \geqslant c \geqslant b\},$$
$$(a, b) := \{c \in \mathrm{Mon}[R]_d \mid a > c > b\}.$$

We define half-open intervals as could be expected. Also, for any $c \in \mathrm{Mon}\, R$, we write

$$c[a, b] := [ca, cb].$$

Note: this notation is the reverse of standard interval notation because the larger monomial is written first. This is so that the order matches the way it would be written on a page (e.g. Example 2.1.3, where $\mathrm{Mon}[R]_3 = [x_0^3, x_2^3]$).

Very often, the most important information about a monomial is the last variable that appears in it – the variable with highest index. This index is encoded by the maxi function.

Definition 2.1.5. Given $a \in \mathrm{Mon}[R]_d$, the *max index* of $a \neq 1$ is

$$\mathrm{maxi}\, a := \max\{i \mid a_i > 0\}$$

where $\mathrm{maxi}\, 1 = 0$. Setting $m = \mathrm{maxi}\, a$ means that for $a \neq 1$ we can write

$$a = x_0^{a_0} \cdots x_m^{a_m}$$

where $a_m > 0$.

2.2 Intervals and Binomial Coefficients

The goal of these next two sections is to explore and apply the relationship between the cardinality of lexicographic intervals and Macaulay representations (Definition and Lemma 2.3.1). This relationship is motivated by Lemmas 2.2.3 and 2.2.4, and the the main result is Proposition 2.3.6 from which some useful inequalities easily flow.

Remark 2.2.1. Throughout this document, we are assuming that if $n < k$, then $\binom{n}{k} = 0$. In particular, we are using the assumption that $\binom{-1}{0} = \binom{-1}{1} = 0$.

Lemma 2.2.2. *Given any power of a variable $x_j^d \in R$, the number of monomials in $\mathrm{Mon}[R]_d$ that are less than or equal to it is,*

$$\#[x_j^d, x_n^d] = \binom{n-j+d}{n-j} = \binom{n-j+d}{d}.$$

Proof. Because the variables $x_0, \ldots, x_{j-1}$ cannot appear in a monomial less than x_j^d, the desired quantity is the same as $\dim_K [(x_j, \ldots, x_n)]_d$, which can be interpreted as the problem of placing d indistinguishable items into $n - j + 1$ bins. □

Next, we will decompose the two intervals $[x_0^d, a)$ and $[a, x_n^d]$ into sub-intervals that allow us to count their monomials using binomial coefficients.

Lemma 2.2.3. *Given any monomial $a = x_0^{a_0} \cdots x_n^{a_n} \in \mathrm{Mon}[R]_d$, including $a = 1$,*

$$\#[x_0^d, a) = \sum_{k=1}^{n} \binom{A_k}{k}$$

where $A_k = k + a_{n-k+1} + \cdots + a_n - 1$. In particular, $A_n > \cdots > A_1 \geqslant 0$.

Proof. If $a = 1$, then $A_k = k - 1$, and the sum is 0, as it should be. Otherwise, we set $m = \operatorname{maxi} a$ and $b_i := a_{i+1} + \cdots + a_m > 0$, as $a_m > 0$, for $0 \leqslant i \leqslant m$. There are two things to note.

1. First, note that when $i = 0$,

$$x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} x_i^{a_i + b_i} = x_0^d.$$

2. Second, for $0 \leqslant i \leqslant m - 1$, we have that

$$x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} x_i^{a_i + 1} x_n^{b_i - 1}$$

is the monomial immediately before/larger than

$$x_0^{a_0} \cdots x_{(i+1)-1}^{a_{(i+1)-1}} x_{i+1}^{a_{i+1} + b_{i+1}} = x_0^{a_0} \cdots x_i^{a_i} x_{i+1}^{b_i}$$

which equals a when $i = m - 1$.

This gives us the following disjoint union of intervals:

$$\begin{aligned}
[x_0^d, a) &= \bigcup_{i=0}^{m-1} [x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} x_i^{a_i + b_i}, x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} x_i^{a_i + 1} x_n^{b_i - 1}] \\
&= \bigcup_{i=0}^{m-1} x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} x_i^{a_i + 1} [x_i^{b_i - 1}, x_n^{b_i - 1}].
\end{aligned}$$

We can now use Lemma 2.2.2 to find a formula.

$$\begin{aligned}
\#[x_0^d, a) &= \sum_{i=0}^{m-1} \#[x_i^{b_i - 1}, x_n^{b_i - 1}] \\
&= \sum_{i=0}^{m-1} \binom{n - i + b_i - 1}{n - i}
\end{aligned}$$

Because $b_i = 0$ for $i \geqslant m$, we can change the upper bound without effect.

$$\#[x_0^d, a) = \sum_{i=0}^{n-1} \binom{n - i + b_i - 1}{n - i}$$

The substitution $k = n - i$ gives the desired result:

$$\begin{aligned}\#[x_0^d, a) &= \sum_{k=1}^{n} \binom{k + b_{n-k} - 1}{k} \\ &= \sum_{k=1}^{n} \binom{A_k}{k}.\end{aligned}$$

□

The next formula may seem somewhat unnecessary since we could simply use the fact that $\#[a, x_n^d] = \binom{n+d}{d} - \#[x_0^d, a)$. While this is true, and is actually the point of Corollary 2.3.8, our goal here is to establish the relationship between intervals and Macaulay representations. The important thing is that we have this form where $A_d > \cdots > A_1 \geqslant 0$.

Lemma 2.2.4. *Consider $a = x_0^{a_0} \cdots x_n^{a_n} \in \operatorname{Mon}[R]_d$ with $m \geqslant \operatorname{maxi} a$. Then,*

$$\#[a, x_n^d] = \sum_{k=a_m}^{d} \binom{A_k}{k}$$

where

$$A_k = \begin{cases} k + n - m & \text{if } k = a_m \\ k + n - 1 - i & \text{if } a_{i+1} + \cdots + a_m < k \leqslant a_i + \cdots + a_m \text{ for some } 0 \leqslant i < m. \end{cases}$$

In particular, $A_d > \cdots > A_{a_m} \geqslant 0$. Written explicitly, this sum is

$$\#[a, x_n^d] = \binom{a_m + n - m}{a_m} + \sum_{i=0}^{m-1} \sum_{k=a_{i+1}+\cdots+a_m+1}^{a_i+\cdots+a_m} \binom{k + n - 1 - i}{k}.$$

Proof. We set $b_i := a_{i+1} + \cdots + a_m$ for $0 \leqslant i \leqslant m$. There are three things to note.

1. First, we have that for $i = 0$,

$$x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} x_n^{b_i + a_i} = x_n^d.$$

2. Second, for $1 \leqslant i \leqslant m$ where there is some $\tau < i$ with $a_\tau > 0$, we let τ be the largest such τ. Then, we have that

$$x_0^{a_0} \cdots x_{\tau-1}^{a_{\tau-1}} x_i^{a_\tau - 1} x_{\tau+1}^{b_\tau + 1}$$

is the monomial immediately after/smaller than

$$x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} x_n^{b_i}.$$

3. Finally, for $0 \leqslant i \leqslant m - 1$ and $1 < j \leqslant a_i$, we have that

$$x_i^{a_i - j} x_{i+1}^{b_i + j}$$

is the monomial immediately after/smaller than

$$x_i^{a_i - (j-1)} x_n^{b_i + (j-1)} = x_i^{a_i - j + 1} x_n^{b_i + j - 1}.$$

This gives us the following disjoint union of intervals:

$$\begin{aligned}
[a, x_n^d] &= [x_0^{a_0} \cdots x_{m-1}^{a_{m-1}} x_m^{a_m}, x_0^{a_0} \cdots x_{m-1}^{a_{m-1}} x_n^{b_m+a_m}] \\
&\quad \cup \bigcup_{\substack{i=0 \\ a_i>0}}^{m-1} [x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} x_i^{a_i-1} x_{i+1}^{b_i+1}, x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} x_n^{b_i+a_i}] \\
&= x_0^{a_0} \cdots x_{m-1}^{a_{m-1}} [x_m^{a_m}, x_n^{b_m+a_m}] \cup \bigcup_{\substack{i=0 \\ a_i>0}}^{m-1} x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} [x_i^{a_i-1} x_{i+1}^{b_i+1}, x_n^{b_i+a_i}] \\
&= x_0^{a_0} \cdots x_{m-1}^{a_{m-1}} [x_m^{a_m}, x_n^{a_m}] \cup \bigcup_{\substack{i=0 \\ a_i>0}}^{m-1} x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} \bigcup_{j=1}^{a_i} [x_i^{a_i-j} x_{i+1}^{b_i+j}, x_i^{a_i-j} x_n^{b_i+j}] \\
&= x_0^{a_0} \cdots x_{m-1}^{a_{m-1}} [x_m^{a_m}, x_n^{a_m}] \cup \bigcup_{\substack{i=0 \\ a_i>0}}^{m-1} x_0^{a_0} \cdots x_{i-1}^{a_{i-1}} \bigcup_{j=1}^{a_i} x_i^{a_i-j} [x_{i+1}^{b_i+j}, x_n^{b_i+j}].
\end{aligned}$$

We can now use Lemma 2.2.2 to find a formula.

$$\begin{aligned}
\#[a, x_n^d] &= \#[x_m^{a_m}, x_n^{a_m}] + \sum_{i=0}^{m-1} \sum_{j=1}^{a_i} \#[x_{i+1}^{b_i+j}, x_n^{b_i+j}] \\
&= \binom{n-m+a_m}{a_m} + \sum_{i=0}^{m-1} \sum_{j=1}^{a_i} \binom{n-(i+1)+b_i+j}{b_i+j}.
\end{aligned}$$

Using the substitution $k = b_i + j$ gives the desired result:

$$\begin{aligned}
\#[a, x_n^d] &= \binom{n-m+a_m}{a_m} + \sum_{i=0}^{m-1} \sum_{k=b_i+1}^{b_i+a_i} \binom{k+n-1-i}{k} \\
&= \sum_{k=a_m}^{d} \binom{A_k}{k}.
\end{aligned}$$

Finally, suppose $d \geqslant k_1 > k_2 > a_m$ where

$$\begin{aligned}
a_{i_1+1} + \cdots + a_m < k_1 \leqslant a_{i_1} + \cdots + a_m \\
a_{i_2+1} + \cdots + a_m < k_2 \leqslant a_{i_2} + \cdots + a_m
\end{aligned}$$

for some $0 \leqslant i_1, i_2 < m$. This gives us that $a_{i_2+1} + \cdots + a_m < a_{i_1} + \cdots + a_m$. So, $i_2 + 1 > i_1$ and thus, $i_2 \geqslant i_1$. Hence, $A_{k_1} = k_1 + n - 1 - i_1 > k_2 + n - 1 - i_2 = A_{k_2}$. Additionally, $A_{a_m+1} \geqslant (a_m + 1) + n - 1 - (m - 1) > a_m + n - m = A_{a_m}$. Thus, we have that $A_d > \cdots > A_{a_m} \geqslant 0$. $\square$

Example 2.2.5. Consider $n = 3$ and $a = x_0^2x_1x_2^3 \in K[x_0, \ldots, x_3]$. So, $d = 6$ and $\operatorname{maxi} a = 2$. Lemma 2.2.3 gives that

$$\begin{aligned}\#[x_0^6, x_0^2x_1x_2^3) &= \binom{1+0-1}{1} + \binom{2+3-1}{2} + \binom{3+4-1}{3} \\ &= \binom{0}{1} + \binom{4}{2} + \binom{6}{3} \\ &= 26\end{aligned}$$

with the decomposition being

$$[x_0^6, x_0^2x_1x_2^3) = [x_0^6, x_0^3x_3^3] \cup [x_0^2x_1^4, x_0^2x_1^2x_3^2].$$

Taking $m = 2$, we can use Lemma 2.2.4 to get

$$\begin{aligned}\#[x_0^2x_1x_2^3, x_3^6] &= \binom{1-1}{1} + \binom{2-1}{2} + \binom{3-2+3}{3} \\ &\quad + \binom{4+3-1-1}{4} \\ &\quad + \binom{5+3-1-0}{5} + \binom{6+3-1-0}{6} \\ &= \binom{0}{1} + \binom{1}{2} + \binom{4}{3} + \binom{5}{4} + \binom{7}{5} + \binom{8}{6} \\ &= 58\end{aligned}$$

with the decomposition being,

$$[x_0^2x_1x_2^3, x_3^6] = [x_0^2x_1x_2^3, x_0^2x_1x_3^3] \cup [x_0^2x_2^4, x_0^2x_3^4] \cup \big([x_0x_1^5, x_0x_3^5] \cup [x_1^6, x_3^6]\big).$$

Or, taking $m = 3$, we can use Lemma 2.2.4 to get

$$\begin{aligned}\#[x_0^2x_1x_2^3, x_3^6] &= \binom{3-3+0}{0} \\ &\quad + \binom{1+3-1-2}{1} + \binom{2+3-1-2}{2} + \binom{3+3-1-2}{3} \\ &\quad + \binom{4+3-1-1}{4} \\ &\quad + \binom{5+3-1-0}{5} + \binom{6+3-1-0}{6} \\ &= \binom{0}{0} + \binom{1}{1} + \binom{2}{2} + \binom{3}{3} + \binom{5}{4} + \binom{7}{5} + \binom{8}{6} \\ &= 58\end{aligned}$$

with the decomposition being,

$$\begin{aligned}[x_0^2x_1x_2^3, x_3^6] = \{x_0^2x_1x_2^3\} &\cup \big(\{x_0^2x_1x_2^2x_3\} \cup \{x_0^2x_1x_2x_3^2\} \cup \{x_0^2x_1x_3^3\}\big) \\ &\cup \big([x_0^2x_2^4, x_0^2x_3^4]\big) \cup \big([x_0x_1^5, x_0x_3^5] \cup [x_1^6, x_3^6]\big).\end{aligned}$$

2.3 Macaulay Representations

The counts obtained in Lemmas 2.2.3 and 2.2.4 are examples of, and motivate, this next definition.

Definition and Lemma 2.3.1. Given a positive integer, d, and a nonnegative integer, A, we can uniquely write A in the form

$$A = \sum_{i=1}^{d} \binom{A_i}{i} = \binom{A_d}{d} + \binom{A_{d-1}}{d-1} + \cdots + \binom{A_1}{1}$$

where $A_d > A_{d-1} > \cdots > A_1 \geqslant 0$. This is called the *dth Macaulay representation* of A. Alternatively, we can uniquely write

$$A = \sum_{i=k}^{d} \binom{A_i}{i} = \binom{A_d}{d} + \binom{A_{d-1}}{d-1} + \cdots + \binom{A_k}{k}$$

where $A_d > A_{d-1} > \cdots > A_k \geqslant k$.

Note: these A_i values are the same in both sums; we simply have that $A_i = i - 1$ for $i < k$. This means that the last $k - 1$ binomials equal 0. As a result, these sums are interchangeable.

Furthermore, we define the following functions based off of the Macaulay representation.

$$A^{\langle d \rangle} := \sum_{i=1}^{d} \binom{A_i + 1}{i + 1}$$

$$A^{-\langle d \rangle} := \sum_{i=1}^{d} \binom{A_i - 1}{i - 1}$$

$$A_{\langle d \rangle} := \sum_{i=1}^{d} \binom{A_i - 1}{i} = A - A^{-\langle d \rangle}.$$

Proof. This is an well-established result. See, e.g., Lemma 4.2.6 in [3] for a proof. □

The notation $A^{\langle d \rangle}$ and $A_{\langle d \rangle}$ are common in existing literature, so we are using the same notation for the sake of compatibility.

Remark 2.3.2. There are several important points to make here:

- Recall from Remark 2.2.1 that we are assuming that $\binom{-1}{0} = \binom{-1}{1} = 0$.
- As a result, because the zero terms in the representation will still be zero after one of these three functions are applied, all three functions can simply be applied to the alternative, truncated version of the representation.

- However, the functions $A^{-\langle d\rangle}$ and $A_{\langle d\rangle}$ do not typically give a Macaulay representation while $A^{\langle d\rangle}$ gives a Macaulay representation that is missing an omissible $\binom{0}{1}$ term. This shows up in the following corollary.

Corollary 2.3.3. *Given positive d and nonnegative A, we have $\left[A^{\langle d\rangle}\right]^{-\langle d+1\rangle} = A$. However, $\left[A^{-\langle d\rangle}\right]^{\langle d-1\rangle}$ does not always equal A.*

Proof. The 2nd Macaulay representation of 2 is $2 = \binom{2}{2}+\binom{1}{1}$. So, $2^{-\langle 2\rangle} = \binom{1}{1}+\binom{0}{0} = 2$. The 1st Macaulay representation of 2 is $2 = \binom{2}{1}$. So, $2^{\langle 1\rangle} = \binom{3}{2} = 3$. This is an example of the second statement because $\left[2^{-\langle 2\rangle}\right]^{\langle 2-1\rangle} \neq 2$.

We have $A = \sum_{i=1}^{d} \binom{A_i}{i}$. So,

$$\begin{aligned}
\left[A^{\langle d\rangle}\right]^{-\langle d+1\rangle} &= \left[\sum_{i=1}^{d} \binom{A_i+1}{i+1}\right]^{-\langle d+1\rangle} \\
&= \left[\sum_{j=2}^{d+1} \binom{A_{j-1}+1}{j}\right]^{-\langle d+1\rangle} \\
&= \sum_{j=2}^{d+1} \binom{A_{j-1}}{j-1} \\
&= \sum_{i=1}^{d} \binom{A_i}{i} \\
&= A.
\end{aligned}$$

□

For the main motivation of this section, Proposition 2.3.6, we need a definition.

Definition 2.3.4. Setting $m = \operatorname{maxi} a$ for a monomial $a \neq 1$ means that we can write

$$a = x_0^{a_0} \cdots x_m^{a_m}$$

where $a_m > 0$. We then define

$$\begin{aligned}
\breve{a} &:= x_0^{a_0} \cdots x_m^{a_m-1} = a/x_m \\
\widehat{a} &:= x_0^{a_0} \cdots x_m^{a_m+1} = a x_m
\end{aligned}$$

and also define $\widehat{1} := x_0$.

This is best interpreted with a pictorial view of monomials.

Example 2.3.5. If we take $a = x_0^2 x_1 x_2^3$, we have $\widehat{a} = x_0^2 x_1 x_2^4$ and $\breve{a} = x_0^2 x_1 x_2^2$.

Looking at Figure 2.2, we see that $\breve{a}$ is the monomial "below" a. $\widehat{a}$ is a bit more tricky, since there is more than one monomial that has a as the monomial below it: both $x_0^2 x_1 x_2^4$ and $x_0^2 x_1 x_2^3 x_3$ are "above" a. We define $\widehat{a}$ to be the leftmost one.

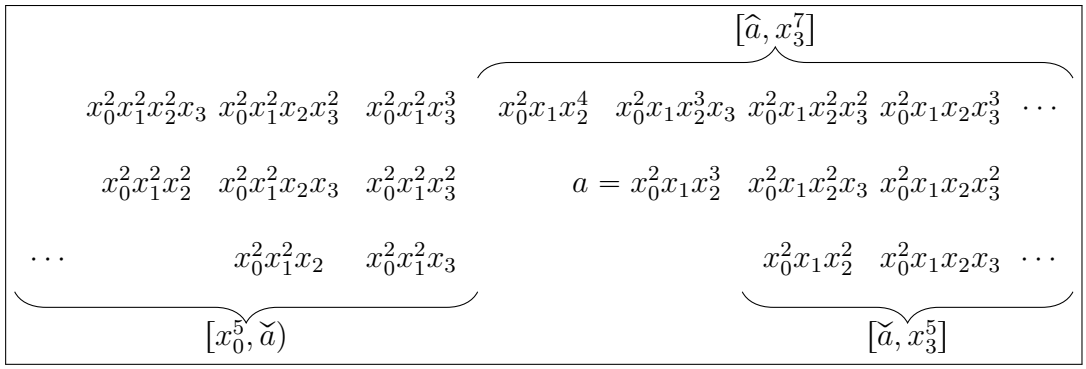

Figure 2.2: Monomials and intervals near $x_0^2x_1x_2^3$.

As a result, $[\widehat{a}, ax_n]$ is the set of monomials with degree $\deg a + 1$ that are a multiple of a but not a multiple of anything larger, and $\breve{a}$ is the largest factor of a with degree $\deg a - 1$.

Proposition 2.3.6. *Consider $a \in \mathrm{Mon}[R]_d$. Then, we have*

$$\#[\widehat{a}, x_n^{d+1}] = \#[a, x_n^d]^{\langle d \rangle} \tag{2.1}$$

and if $d \geqslant 1$,

$$\#[x_0^{d-1}, \breve{a}) = \#[x_0^d, a)_{\langle n \rangle}, \tag{2.2}$$

$$\#[\breve{a}, x_n^{d-1}] = \#[a, x_n^d]^{-\langle d \rangle}. \tag{2.3}$$

Proof. Let $m = \operatorname{maxi} a$, and we have

$$\begin{aligned} a &= x_0^{a_0} \cdots x_m^{a_m} \\ \breve{a} &= x_0^{a_0} \cdots x_m^{a_m - 1} \\ \widehat{a} &= x_0^{a_0} \cdots x_m^{a_m + 1}. \end{aligned}$$

First, we will show Equation (2.2) by applying of Lemma 2.2.4.

$$\begin{aligned} \#[x_0^d, a)_{\langle n \rangle} &= \left[\sum_{k=1}^{n} \binom{k + a_{n-k+1} + \cdots + a_n - 1}{k} \right]_{\langle n \rangle} \\ &= \sum_{k=1}^{n} \binom{k + a_{n-k+1} + \cdots + a_n - 1 - 1}{k} \\ &= \sum_{k=1}^{n} \binom{k + a_{n-k+1} + \cdots + (a_m - 1) + \cdots + a_n - 1}{k} \\ &= \#[x_0^{d-1}, \breve{a}). \end{aligned}$$

Next, we can show Equation (2.1) by applying Lemma 2.2.4:

$$\begin{aligned}
\#[a, x_n^d]^{\langle d \rangle} &= \left[\binom{a_m + n - m}{a_m} + \sum_{i=0}^{m-1} \sum_{k=a_{i+1}+\cdots+a_m+1}^{a_i+\cdots+a_m} \binom{k+n-1-i}{k} \right]^{\langle d \rangle} \\
&= \binom{a_m + n - m + 1}{a_m + 1} + \sum_{i=0}^{m-1} \sum_{k=a_{i+1}+\cdots+a_m+1}^{a_i+\cdots+a_m} \binom{k+n-1-i+1}{k+1}.
\end{aligned}$$

The substitution $j = k + 1$ and Lemma 2.2.4 give,

$$\begin{aligned}
\#[a, x_n^d]^{\langle d \rangle} &= \binom{(a_m + 1) + n - m}{a_m + 1} + \sum_{i=0}^{m-1} \sum_{j=a_{i+1}+\cdots+(a_m+1)+1}^{a_i+\cdots+(a_m+1)} \binom{j+n-1-i}{j} \\
&= \#[\widehat{a}, x_n^{d+1}].
\end{aligned}$$

Doing the exact same thing for Equation (2.3) gives us that

$$\begin{aligned}
\#[a, x_n^d]^{-\langle d \rangle} &= \binom{(a_m - 1) + n - m}{a_m - 1} + \sum_{i=0}^{m-1} \sum_{j=a_{i+1}+\cdots+(a_m-1)+1}^{a_i+\cdots+(a_m-1)} \binom{j+n-1-i}{j} \\
&= \#[\breve{a}, x_n^{d-1}].
\end{aligned}$$

□

Example 2.3.7. Using the values from Example 2.2.5, we can compute the lengths of the intervals shown in Figure 2.2:

$$\begin{aligned}
\#[x_0^5, x_0^2 x_1 x_2^2) &= \left[\binom{0}{1} + \binom{4}{2} + \binom{6}{3} \right]_{\langle 3 \rangle} \\
&= \binom{-1}{1} + \binom{3}{2} + \binom{5}{3} \\
&= 13 \\
\#[x_0^2 x_1 x_2^2, x_3^5] &= \left[\binom{0}{1} + \binom{1}{2} + \binom{4}{3} + \binom{5}{4} + \binom{7}{5} + \binom{8}{6} \right]^{-\langle 6 \rangle} \\
&= \binom{-1}{0} + \binom{0}{1} + \binom{3}{2} + \binom{4}{3} + \binom{6}{4} + \binom{7}{5} \\
&= 43 \\
\#[x_0^2 x_1 x_2^4, x_3^7] &= \left[\binom{0}{1} + \binom{1}{2} + \binom{4}{3} + \binom{5}{4} + \binom{7}{5} + \binom{8}{6} \right]^{\langle 6 \rangle} \\
&= \binom{1}{2} + \binom{2}{3} + \binom{5}{4} + \binom{6}{5} + \binom{8}{6} + \binom{9}{7} \\
&= 75
\end{aligned}$$

In terms of Figure 2.2, Proposition 2.3.6 states that $A^{-\langle d \rangle}$ represents the length of the interval beneath the rightmost, length A, degree d interval. Also, $B_{\langle n \rangle}$ represents the length of the interval beneath the leftmost, length B, degree d interval.

Corollary 2.3.8 states that if we know that $A + B$ is the total number of monomials in degree d, then $A^{-\langle d\rangle} + B_{\langle n\rangle}$ must also be the total number of monomials in degree $d-1$. While this is the basic idea, it is stated in a more usable form.

Corollary 2.3.8. *Suppose $\alpha \geqslant d > 0$ and $\binom{\alpha}{d} \geqslant A \geqslant 0$. Then, we have*

$$A^{-\langle d\rangle} = A - \binom{\alpha-1}{d} + \left[\binom{\alpha}{d} - A\right]^{-\langle \alpha-d\rangle}, \tag{2.4}$$

$$A^{-\langle \alpha-d\rangle} = A - \binom{\alpha-1}{d-1} + \left[\binom{\alpha}{d} - A\right]^{-\langle d\rangle}. \tag{2.5}$$

Proof. Let $n = \alpha - d$. Because $\binom{n+d}{d} \geqslant A \geqslant 0$, there is some monomial $a \in \operatorname{Mon}[R]_d$ such that $\#[a, x_n^d] = A$. Thus, we have

$$\#[x_0^d, a) = \#[x_0^d, x_n^d] - \#[a, x_n^d] = \binom{\alpha}{d} - A.$$

Applying the identity $b_{\langle n\rangle} = b - b^{-\langle n\rangle}$, along with Proposition 2.3.6, gives

$$\begin{aligned}
\binom{\alpha}{d} - A - \left[\binom{\alpha}{d} - A\right]^{-\langle n\rangle} + A^{-\langle d\rangle} &= \left[\binom{\alpha}{d} - A\right]_{\langle n\rangle} + A^{-\langle d\rangle} \\
&= \#[x_0^d, a)_{\langle n\rangle} + \#[a, x_n^d]^{-\langle d\rangle} \\
&= \#[x_0^{d-1}, \breve{a}) + \#[\breve{a}, x_n^{d-1}] \\
&= \#[x_0^{d-1}, x_n^{d-1}] \\
&= \binom{\alpha-1}{d-1},
\end{aligned}$$

which is equivalent to Equation (2.4) after rearranging. Setting $B = \#[x_0^d, a)$ and writing everything in terms of B gives us Equation (2.5):

$$\begin{aligned}
\binom{\alpha-1}{d-1} &= B_{\langle n\rangle} + \left[\binom{\alpha}{d} - B\right]^{-\langle d\rangle} \\
&= B - B^{-\langle \alpha-d\rangle} + \left[\binom{\alpha}{d} - B\right]^{-\langle d\rangle}.
\end{aligned}$$

□

Corollary 2.3.9. *The functions $A \mapsto A^{\langle d\rangle}$, $A \mapsto A^{-\langle d\rangle}$, and $A \mapsto A_{\langle d\rangle}$ are monotone increasing.*

Proof. If $a, b \in \operatorname{Mon} R_d$ are monomials with $a \geqslant b$, then $\breve{a} \geqslant \breve{b}$ and $\widehat{a} \geqslant \widehat{b}$. Hence, the claim follows directly from the interval interpretation of Proposition 2.3.6. □

2.4 Basic Inequalities

We will now turn to establishing some inequalities we use later on. Macaulay representations play a key role in the characterization of Hilbert functions through the bound given in Theorem 2.4.1. We can use this inequality to find a sort of Triangle Inequality (Lemma 2.4.3). We will also find a Squeeze Theorem (Lemma 2.4.4) and ultimately prove another useful inequality in Proposition 2.5.6, which is the key to Proposition 2.6.6.

Macaulay [10] showed in 1927 that the Hilbert function must satisfy the following bound. This inequality gives an upper bound for the Hilbert function in a higher degree.

Theorem 2.4.1 (Macaulay's bound)**.** *Let h be a Hilbert function of some quotient of S. Then,*

$$h(d)^{\langle d\rangle} \geqslant h(d+1)$$

for any $d \in \mathbb{N}_0$.

Proof. For a proof, we refer to Theorem 4.2.10 in [3]. □

We can also invert this bound, giving a lower bound for the Hilbert function in a lower degree.

Corollary 2.4.2. *Let h be a Hilbert function of an ideal. Then,*

$$h(d)^{-\langle d\rangle} \leqslant h(d-1).$$

Proof. Because $h(d) \leqslant h(d-1)^{\langle d-1\rangle}$ by Theorem 2.4.1, Corollaries 2.3.3 and 2.3.9 yield

$$h(d)^{-\langle d\rangle} \leqslant \left[h(d-1)^{\langle d-1\rangle}\right]^{-\langle d\rangle} = h(d-1).$$ □

A consequence of these inequalities is a Triangle Inequality. The gist of this proof is that we construct an ideal with Hilbert function $A+B$ in degree d, and $A + B^{-\langle d\rangle}$ in degree $d-1$. Then we conclude by Corollary 2.4.2.

Lemma 2.4.3 (Triangle Inequality for Macaulay Representations)**.** *If A, B, and d are non-negative integers, then*

$$[A+B]^{-\langle d\rangle} \leqslant A^{-\langle d\rangle} + B^{-\langle d\rangle},$$
$$[A+B]_{\langle d\rangle} \geqslant A_{\langle d\rangle} + B_{\langle d\rangle}.$$

Proof. Fix d, and let n, m be large enough so that $A \leqslant \#\operatorname{Mon}[R]_d$ and $B \leqslant \#\operatorname{Mon}[S]_d$ where $R = K[x_0, \ldots, x_n]$ and $S = K[y_0, \ldots, y_m]$. Let $a \in \operatorname{Mon}[R]_d$ and $b \in \operatorname{Mon}[S]_d$ be such that $A = \#[a, x_n^d]$ and $B = \#[b, x_m^d]$. Setting $t = \operatorname{maxi} a$ gives

$$a = x_0^{a_0} \cdots x_t^{a_t} \qquad\qquad \breve{a} = x_0^{a_0} \cdots x_t^{a_t - 1}.$$

Consider any monomial $e \in R$ with $e > \breve{a}$ and any variable x_i. We wish to show that $ex_i > a$. By the definition of the lexicographic order, there is some $k \leqslant t$ with $e_k > a_k$ and $e_j = a_j$ for $j < k$. Now, if $i > t \geqslant k$, then

$$ex_i = x_0^{e_0} \cdots x_k^{e_k} \cdots x_i^{e_i+1} \cdots x_n^{e_n} > x_0^{a_0} \cdots x_k^{a_k} \cdots x_t^{a_t} = a$$

and if $i \leqslant t$, then because the lexicographic order is a monomial order (Remark 2.1.2),

$$ex_i > \breve{a}x_i \geqslant \breve{a}x_t = a.$$

In either case, we have that $ex_i > a$ if $e > \breve{a}$. So, there is no monomial in $[a, x_n^d]$ that is a multiple of a monomial in $[x_0^{d-1}, \breve{a})$.

Thus, we have that quotient by the ideal, $I \subset R$, generated by $[x_0^d, a) \cup [x_0^{d-1}, \breve{a})$ has Hilbert function such that

$$h_{R/I}(d) = \#[a, x_n^d] = A,$$

and, by Proposition 2.3.6,

$$h_{R/I}(d-1) = \#[\breve{a}, x_n^{s-1}] = A^{-\langle d \rangle}.$$

Similarly, for the ideal, J, generated by $[x_0^d, b) \cup [x_0^{d-1}, \breve{b})$, we have

$$h_{S/J}(d) = B$$

and

$$h_{S/J}(d-1) = B^{-\langle d \rangle}.$$

Now, let $T = K[x_0, \ldots, x_m, y_0, \ldots, y_n]$ and

$$H = \langle x_i y_j \rangle + IT + JT.$$

Thus,

$$\begin{aligned} h_{T/H}(d) &= h_{R/I}(d) + h_{S/J}(d) \\ &= A + B \end{aligned}$$

and

$$\begin{aligned} h_{T/H}(d-1) &= h_{R/I}(d-1) + h_{S/J}(d-1) \\ &= A^{-\langle d \rangle} + B^{-\langle d \rangle}. \end{aligned}$$

By Corollary 2.4.2, we obtain the first desired result. The second follows from the fact that $A = A^{-\langle d \rangle} + A_{\langle d \rangle}$ which follows from Definition and Lemma 2.3.1. □

Often, if we can estimate an integer, we may wish to know what the first few terms of its Macaulay representation are. The following Squeeze Theorem gives us such a result, and is used heavily in proving some of the following results.

Lemma 2.4.4 (Squeeze Theorem for Macaulay Representations)**.** *Consider the dth Macaulay representations of nonnegative integers A, B, and C.*

$$A = \binom{A_d}{d} + \binom{A_{d-1}}{d-1} + \cdots + \binom{A_1}{1}$$
$$B = \binom{B_d}{d} + \binom{B_{d-1}}{d-1} + \cdots + \binom{B_1}{1}$$
$$C = \binom{C_d}{d} + \binom{C_{d-1}}{d-1} + \cdots + \binom{C_1}{1}$$

We have the following

1. *Let* $k = \max\{i \mid A_i \neq B_i\}$. *Then* $A > B$ *if and only if* $A_k > B_k$.
2. *Let* $j \leqslant d$. *If* $A \leqslant B \leqslant C$ *and* $A_i = C_i$ *for* $i > j$, *then*
   a) $A_i = B_i = C_i$ *for* $i > j$,
   b) $A_j \leqslant B_j \leqslant C_j$, *and*
   c) *if* $B \lneq \binom{C_d}{d} + \cdots + \binom{C_j}{j}$, *then* $B_j \lneq C_j$.

Proof. The first statement is simply saying that the Macaulay representation respects the lexicographic order. See, e.g., Lemma 4.2.7 in [3] for a proof of this fact.

The second statement generalizes this by saying that if the first few terms are the same, then the next term must respect the same order, with a possible strict upper bound. This follows immediately from the first statement. □

2.5 A More Complex Inequality

The last inequality that we will need is Proposition 2.5.6. The proof is very complex, and we will break it down into smaller pieces. Lemmas 2.5.3 to 2.5.5 give us the result in two different cases while Equations (2.6) to (2.8) give helpful simplifications for these lemmas.

The main difficulty involved in this proof is the subtraction of Macaulay representations. This is accomplished by focusing on dividing the representations into a group of terms that match, and a group of terms that do not match. The leading terms will be the ones that match, while the trailing terms will be the ones that do not match. When the numbers are then subtracted, the leading terms will all cancel. As a result, we often only will need to focus on the trailing part of a representation. We will introduce some notation to simplify the expressions we will get.

Definition 2.5.1. Consider the dth Macaulay representation of a positive integer A.

$$A = \binom{A_d}{d} + \binom{A_{d-1}}{d-1} + \cdots + \binom{A_1}{1}.$$

Define, for an integer k with $1 \leqslant k \leqslant d$,

$$\begin{aligned} A/(k,d) &:= \binom{A_k}{k} + \binom{A_{k-1}}{k-1} + \cdots + \binom{A_1}{1} \\ A*(k,d) &:= \binom{A_k+1}{k+1} + \binom{A_{k-1}+1}{k} + \cdots + \binom{A_1+1}{2} \\ &= [A/(k,d)]^{\langle k \rangle}. \end{aligned}$$

Because we will consider d to be fixed for the remainder of the section, we will simply write A/k and $A*k$.

We will need one of following identities for each of Lemmas 2.5.3 to 2.5.5. Each of these identities essentially breaks a Macaulay representation into a leading and a trailing part that will cancel later on.

Lemma 2.5.2. *Consider the dth Macaulay representation of A.*

$$A = \binom{A_d}{d} + \binom{A_{d-1}}{d-1} + \cdots + \binom{A_1}{1}.$$

Then, the following two equations hold for $k \leqslant d$,

$$A^{-\langle d \rangle} - [A/k]^{-\langle k \rangle} = [A - A/k]^{-\langle d \rangle} \tag{2.6}$$

$$\begin{aligned} A - [A/k]^{-\langle k \rangle} = &\left[A^{\langle d \rangle} - A*k + \binom{A_k}{k+1} \right]^{-\langle d+1 \rangle} \\ &+ \left[A*k - \binom{A_k}{k+1} - A/k \right]^{-\langle k \rangle}. \end{aligned} \tag{2.7}$$

Also, for $k < d$,

$$\begin{aligned} A - [A/(k+1)]^{-\langle k+1 \rangle} = &\left[A^{\langle d \rangle} - A*k \right]^{-\langle d+1 \rangle} \\ &+ [A*k - A/k]^{-\langle k+1 \rangle} - \binom{A_{k+1}-1}{k}. \end{aligned} \tag{2.8}$$

Proof. Equation (2.6) is easy. Both sides are simply

$$\binom{A_d - 1}{d-1} + \binom{A_{d-1}-1}{d-2} + \cdots + \binom{A_{k+1}-1}{k}.$$

Equation (2.7) is more work. We will first compute the terms of the right hand side:

$$\begin{aligned} A^{\langle d \rangle} - A*k + \binom{A_k}{k+1} &= \binom{A_d+1}{d+1} + \cdots + \binom{A_{k+1}+1}{k+2} + \binom{A_k}{k+1} \\ A*k - A/k - \binom{A_k}{k+1} &= \binom{A_k}{k+1} + \cdots + \binom{A_1}{2} - \binom{A_k}{k+1} \\ &= \binom{A_{k-1}}{k} + \cdots + \binom{A_1}{2}. \end{aligned}$$

So, we get

$$\begin{aligned}
&A - [A/k]^{-\langle k\rangle} \\
&\quad = \binom{A_d}{d} + \cdots + \binom{A_{k+1}}{k+1} + \binom{A_k}{k} \quad + \binom{A_{k-1}}{k-1} \quad + \cdots + \binom{A_1}{1} \\
&\qquad\qquad - \binom{A_k - 1}{k-1} - \binom{A_{k-1} - 1}{k-2} - \cdots - \binom{A_1 - 1}{0} \\
&\quad = \binom{A_d}{d} + \cdots + \binom{A_{k+1}}{k+1} + \binom{A_k - 1}{k} + \binom{A_{k-1} - 1}{k-1} + \cdots + \binom{A_1 - 1}{1} \\
&\quad = \left[A^{\langle d\rangle} - A * k + \binom{A_k}{k+1}\right]^{-\langle d+1\rangle} \quad + \left[A * k - \binom{A_k}{k+1} - A/k\right]^{-\langle k\rangle}.
\end{aligned}$$

Finally, we show Equation (2.8).

$$\begin{aligned}
&A - [A/(k+1)]^{-\langle k+1\rangle} \\
&\quad = \binom{A_d}{d} + \cdots \quad + \binom{A_{k+1}}{k+1} \quad + \binom{A_k}{k} \quad + \cdots + \binom{A_1}{1} \\
&\qquad\qquad - \binom{A_{k+1} - 1}{k} - \binom{A_k - 1}{k-1} - \cdots - \binom{A_1 - 1}{0} \\
&\quad = \binom{A_d}{d} + \cdots \quad + \binom{A_{k+1}}{k+1} \\
&\qquad\qquad - \binom{A_{k+1} - 1}{k} + \binom{A_k - 1}{k} + \cdots + \binom{A_1 - 1}{1} \\
&\quad = \left[A^{\langle d\rangle} - A * k\right]^{-\langle d+1\rangle} \\
&\qquad\qquad - \binom{A_{k+1} - 1}{k} + [A * k - A/k]^{-\langle k+1\rangle}.
\end{aligned}$$

□

Lemmas 2.5.3 to 2.5.5 all have very similar structure and give formulas that drive the proof of Proposition 2.5.6. In each case, we use the Squeeze Theorem to get exact values for the leading part of a difference. This leading part can then be isolated from the trailing part. Lemma 2.5.3 gives a formula for $[A - C]^{-\langle d\rangle}$ while Lemmas 2.5.4 and 2.5.5 deal with two cases for $\left[A^{\langle d\rangle} - C\right]^{-\langle d+1\rangle}$.

Lemma 2.5.3. *Suppose* $1 \leqslant k \leqslant d$, *and* $0 \leqslant C \leqslant A/k$. *Then,*

$$[A - C]^{-\langle d\rangle} = A^{-\langle d\rangle} - [A/k]^{-\langle k\rangle} + [A/k - C]^{-\langle k\rangle}.$$

Proof. Consider the dth Macaulay representation of $A - C$,

$$A - C = \binom{B_d}{d} + \cdots + \binom{B_1}{1}.$$

We can find part of $A - C$ by realizing that $A \geqslant A - C \geqslant A - A/k$ and that

$$\begin{aligned} A &= \binom{A_d}{d} + \cdots + \binom{A_{k+1}}{k+1} + \cdots + \binom{A_1}{1}, \\ A - C &= \binom{B_d}{d} + \cdots + \binom{B_{k+1}}{k+1} + \cdots + \binom{B_1}{1}, \\ A - A/k &= \binom{A_d}{d} + \cdots + \binom{A_{k+1}}{k+1}. \end{aligned}$$

By the squeeze theorem (Lemma 2.4.4), we have $B_i = A_i$ for $i > k$. So,

$$\begin{aligned} A - C &= \binom{A_d}{d} + \cdots + \binom{A_{k+1}}{k+1} + \binom{B_k}{k} + \cdots + \binom{B_1}{1} \\ &= A - A/k \qquad\qquad + \binom{B_k}{k} + \cdots + \binom{B_1}{1}. \end{aligned}$$

So, by Equation (2.6) of Lemma 2.5.2,

$$\begin{aligned} [A - C]^{-\langle d \rangle} &= [A - A/k]^{-\langle d \rangle} \qquad + \left[\binom{B_k}{k} + \cdots + \binom{B_1}{1} \right]^{-\langle k \rangle} \\ &= A^{-\langle d \rangle} - [A/k]^{-\langle k \rangle} \qquad + [A/k - C]^{-\langle k \rangle}. \end{aligned}$$

□

Lemma 2.5.4. *Suppose $1 \leqslant k \leqslant d$ and $A * (k-1) < C \leqslant A/k$. Then,*

$$\left[A^{\langle d \rangle} - C\right]^{-\langle d+1 \rangle} \leqslant A - [A/k]^{-\langle k \rangle} + [A/k - C]^{-\langle k \rangle}.$$

Proof. Consider the $(d+1)$th Macaulay representation of $A^{\langle d \rangle} - C$,

$$A^{\langle d \rangle} - C = \binom{B_{d+1}}{d+1} + \cdots + \binom{B_1}{1}.$$

We can compute part of $A^{\langle d \rangle} - C$ as before,

$$\begin{aligned} A^{\langle d \rangle} - A * (k-1) &= \binom{A_d + 1}{d+1} + \cdots + \binom{A_{k+1} + 1}{k+2} + \binom{A_k + 1}{k+1} \\ A^{\langle d \rangle} - C &= \binom{B_{d+1}}{d+1} + \cdots + \binom{B_{k+2}}{k+2} + \binom{B_{k+1}}{k+1} + \cdots + \binom{B_1}{1}, \\ A^{\langle d \rangle} - A/k &= \binom{A_d + 1}{d+1} + \cdots + \binom{A_{k+1} + 1}{k+2} + \binom{A_k}{k+1} + \cdots + \binom{A_1}{2}. \end{aligned}$$

By the squeeze theorem (Lemma 2.4.4), we have $B_{i+1} = A_i + 1$ for $i > k$ and $B_{k+1} = A_k$. So,

$$\begin{aligned} A^{\langle d \rangle} - C &= \binom{A_d + 1}{d+1} + \cdots + \binom{A_{k+1} + 1}{k+2} + \binom{A_k}{k+1} + \binom{B_k}{k} + \cdots + \binom{B_1}{1} \\ &= A^{\langle d \rangle} - A * k + \binom{A_k}{k+1} \qquad\qquad + \binom{B_k}{k} + \cdots + \binom{B_1}{1}. \end{aligned}$$

So,

$$\begin{aligned}\left[A^{\langle d\rangle}-C\right]^{-\langle d+1\rangle} &= \left[A^{\langle d\rangle}-A*k+\binom{A_k}{k+1}\right]^{-\langle d+1\rangle}+\left[\binom{B_k}{k}+\cdots+\binom{B_1}{1}\right]^{-\langle k\rangle}\\ &= \left[A^{\langle d\rangle}-A*k+\binom{A_k}{k+1}\right]^{-\langle d+1\rangle}+\left[A*k-\binom{A_k}{k+1}-C\right]^{-\langle k\rangle}\\ &= \left[A^{\langle d\rangle}-A*k+\binom{A_k}{k+1}\right]^{-\langle d+1\rangle}\\ &\quad+\left[A*k-\binom{A_k}{k+1}-A/k+A/k-C\right]^{-\langle k\rangle}.\end{aligned}$$

Because $A*k-A/k=\binom{A_k}{k+1}+\cdots+\binom{A_1}{2}\geqslant 0$, the triangle inequality (Lemma 2.4.3) applied to the second summand gives

$$\begin{aligned}\left[A^{\langle d\rangle}-C\right]^{-\langle d+1\rangle} &\leqslant \left[A^{\langle d\rangle}-A*k+\binom{A_k}{k+1}\right]^{-\langle d+1\rangle}\\ &\quad+\left[A*k-\binom{A_k}{k+1}-A/k\right]^{-\langle k\rangle}+[A/k-C]^{-\langle k\rangle}.\end{aligned}$$

And finally, we use Equation (2.7) of Lemma 2.5.2 to get the desired result.

$$\left[A^{\langle d\rangle}-C\right]^{-\langle d+1\rangle}\leqslant A-[A/k]^{-\langle k\rangle}+[A/k-C]^{-\langle k\rangle}.$$ □

Lemma 2.5.5. *Suppose $1\leqslant k<d$ and $A/k\leqslant C\leqslant A*k$ then,*

$$\left[A^{\langle d\rangle}-C\right]^{-\langle d+1\rangle}\leqslant A-[A/(k+1)]^{-\langle k+1\rangle}+[A/(k+1)-C]^{-\langle k+1\rangle}.$$

Proof. Consider the $(d+1)$th Macaulay representation of $A^{\langle d\rangle}-C$.

$$A^{\langle d\rangle}-C=\binom{B_{d+1}}{d+1}+\cdots+\binom{B_1}{1}.$$

We can compute part of $A^{\langle d\rangle}-C$ as we have done before.

$$\begin{aligned}A^{\langle d\rangle}-A/k &= \binom{A_d+1}{d+1}+\cdots+\binom{A_{k+1}+1}{k+2}+\binom{A_k}{k+1}+\cdots+\binom{A_1}{2}\\ A^{\langle d\rangle}-C &= \binom{B_{d+1}}{d+1}+\cdots+\binom{B_{k+2}}{k+2}+\binom{B_{k+1}}{k+1}+\cdots+\binom{B_1}{1}\\ A^{\langle d\rangle}-A*k &= \binom{A_d+1}{d+1}+\cdots+\binom{A_{k+1}+1}{k+2}.\end{aligned}$$

This time, by the squeeze theorem (Lemma 2.4.4), we get that $B_{i+1}=A_i+1$ for $i>k$. So,

$$\begin{aligned}A^{\langle d\rangle}-C &= \binom{A_d+1}{d+1}+\cdots+\binom{A_{k+1}+1}{k+2}+\binom{B_{k+1}}{k+1}+\cdots+\binom{B_1}{1}\\ &= A^{\langle d\rangle}-A*k+\binom{B_{k+1}}{k+1}+\cdots+\binom{B_1}{1}.\end{aligned}$$

This gives us that $\binom{B_{k+1}}{k+1} + \cdots + \binom{B_1}{1} = A * k - C$. So,

$$\left[A^{\langle d \rangle} - C\right]^{-\langle d+1 \rangle} = \left[A^{\langle d \rangle} - A * k\right]^{-\langle d+1 \rangle} + \left[A * k - C\right]^{-\langle k+1 \rangle}. \tag{2.9}$$

Note: We will use Corollary 2.3.8 three times. In each case, we will use $\alpha = A_{k+1}$, $d = k + 1$, and in each case some $B \leqslant A * k - A/k$. The corollary will apply because $A_{k+1} > A_k$ and thus,

$$A * k - A/k = \binom{A_k}{k+1} + \cdots + \binom{A_1}{2} \leqslant \binom{A_{k+1}}{k+1}.$$

We will focus on the final term of Equation (2.9) here:

$$\left[A * k - C\right]^{-\langle k+1 \rangle} = \left[(A * k - A/k) - (C - A/k)\right]^{-\langle k+1 \rangle}.$$

We use Equation (2.4) to get

$$\begin{aligned}\left[A * k - C\right]^{-\langle k+1 \rangle} = {} & (A * k - A/k) - (C - A/k) - \binom{A_{k+1} - 1}{k+1} \\ & + \left[\binom{A_{k+1}}{k+1} - (A * k - A/k) + (C - A/k)\right]^{-\langle A_{k+1} - k - 1 \rangle}.\end{aligned}$$

The triangle inequality (Lemma 2.4.3) applied to the second summand gives

$$\begin{aligned}\left[A * k - C\right]^{-\langle k+1 \rangle} \leqslant {} & (A * k - A/k) - (C - A/k) - \binom{A_{k+1} - 1}{k+1} \\ & + \left[\binom{A_{k+1}}{k+1} - (A * k - A/k)\right]^{-\langle A_{k+1} - k - 1 \rangle} \\ & + \left[C - A/k\right]^{-\langle A_{k+1} - k - 1 \rangle}.\end{aligned}$$

We now use Equations (2.4) and (2.5) to convert back

$$\begin{aligned}
\Big[A * k - C\Big]^{-\langle k+1\rangle} &= (A * k - A/k) - (C - A/k) - \binom{A_{k+1} - 1}{k+1} \\
&\quad + \Big[A * k - A/k\Big]^{-\langle k+1\rangle} - (A * k - A/k) + \binom{A_{k+1} - 1}{k+1} \\
&\quad + (C - A/k) - \binom{A_{k+1} - 1}{k} + \left[\binom{A_{k+1}}{k+1} - C + A/k\right]^{-\langle k+1\rangle} \\
&= \Big[A * k - A/k\Big]^{-\langle k+1\rangle} + \left[\binom{A_{k+1}}{k+1} - C + A/k\right]^{-\langle k+1\rangle} \\
&\quad - \binom{A_{k+1} - 1}{k} \\
&= \Big[A * k - A/k\Big]^{-\langle k+1\rangle} + \Big[A/(k+1) - C\Big]^{-\langle k+1\rangle} \\
&\quad - \binom{A_{k+1} - 1}{k}.
\end{aligned}$$

Substituting this into Equation (2.9) and applying Equation (2.8) of Lemma 2.5.2 gives the desired result. □

Finally, we arrive at the main result for this section, which is a combination of Lemmas 2.5.3 to 2.5.5. An interpretation and motivation of this inequality in terms of lexicographic intervals can be found in Proposition 2.6.6.

Proposition 2.5.6. *If $0 \leqslant C \leqslant A$ then, for any integer $d \geqslant 0$,*

$$A - \big[A^{\langle d\rangle} - C\big]^{-\langle d+1\rangle} \geqslant A^{-\langle d\rangle} - [A - C]^{-\langle d\rangle}.$$

Proof. Consider the dth Macaulay representation of A,

$$A = \binom{A_d}{d} + \cdots + \binom{A_j}{j}$$

with $A_j \geqslant j \geqslant 1$. Because $A_{i+1} \geqslant A_i + 1$ and $\binom{A_i+1}{i+1} \geqslant \binom{A_i}{i}$ for all i, $A/k \geqslant A * (k-1)$ and $A * k \geqslant A/k$ giving us that

$$A = A/d \geqslant A * (d-1) \geqslant \cdots \geqslant A/j \geqslant A * (j-1) = 0.$$

So, we have two cases. Either $A/k \geqslant C > A*(k-1)$, $A*k \geqslant C \geqslant A/k$ for appropriate k, or $C = 0$. Lemmas 2.5.3 and 2.5.4 give us the result in the first case, Lemmas 2.5.3 and 2.5.5 give us the result in the second case, and the $C = 0$ case is trivial. □

Remark 2.5.7. It is straightforward to verify that this bound is sharp when $C = A/k$ for some k.

2.6 The $m_{\geqslant}$-vector

This next definition plays a huge role in computing and maximizing Betti numbers. The notation used here may differ from existing literature where m_i is often used in place of three different concepts. We will primarily focus on $m_{\geqslant i}$, but will define all three for the sake of clarity and symmetry.

The idea of this definition is to create a tally of the max index of a set of monomials. m_i counts the number of monomials in a set with max index i, $m_{\geqslant i}$ counts the number of monomials in a set with max index at least i, and $m_{\leqslant i}$ counts the number of monomials in a set with max index at most i.

Definition 2.6.1. Recall from Definition 2.1.5 that $\operatorname{maxi} a$ is the variable with highest index that appears in a. Let Q be a set of monomials in $R = K[x_0, \ldots, x_n]$. We write

$$\begin{aligned} m_i(Q) &:= \#\{a \in Q \mid \operatorname{maxi} a = i\} \\ m_{\leqslant i}(Q) &:= \#\{a \in Q \mid \operatorname{maxi} a \leqslant i\} \\ m_{\geqslant i}(Q) &:= \#\{a \in Q \mid \operatorname{maxi} a \geqslant i\} \end{aligned}$$

Note: we will occasionally refer to the tuple $(m_0(Q), \ldots, m_n(Q))$ as the m-vector of Q and the tuple $(m_{\geqslant n}(Q), \ldots, m_{\geqslant n}(Q))$ as the $m_{\geqslant}$-vector of Q.

Lemma 2.6.2. *Given monomials $a, b \in [x_0^d, x_n^d]$ such that $a \geqslant b$, one has $\widehat{a} \geqslant \widehat{b}$ and if $b \neq 1$, $\breve{a} \geqslant \breve{b}$.*

Proof. This follows from Definitions 2.1.1 and 2.3.4. □

Lemma 2.6.3. *Let $a \neq 1$ be a monomial of $R = K[x_0, \ldots, x_n]$ with degree d. Then,*

$$\begin{aligned} m_n[a, x_n^d] &= \#[\breve{a}, x_n^{d-1}], \text{ and} \\ m_n[x_n^d, a) &= \#[x_n^{d-1}, \breve{a}). \end{aligned}$$

Proof. We will prove the first equation. The second has an analogous proof. Let $m = \operatorname{maxi} a$. Given $b \in [\breve{a}, x_n^{d-1}]$, we have that $\breve{a} \geqslant b$ which implies that $a = \breve{a}x_m \geqslant \breve{a}x_n \geqslant bx_n$ (see Remark 2.1.2.) So, $bx_n \in [a, x_n^d]$.

Thus, it suffices to show that the map $b \mapsto bx_n$ gives a bijection from $[\breve{a}, x_n^{d-1}]$ to the monomials in $[a, x_n^d]$ with max index n. The map is clearly injective.

Given $bx_n \in [a, x_n^d]$, we have that $a \geqslant bx_n$ which implies that $\breve{a} \geqslant \widetilde{bx_n} = b$ via Lemma 2.6.2. So, $b \in [\breve{a}, x_n^{d-1}]$ and the map is surjective. □

Corollary 2.6.4. *Let $a \neq 1$ be a monomial of $R = K[x_0, \ldots, x_n]$ with degree d. Then,*

$$\begin{aligned} m_n[a, x_n^d] &= \#[a, x_n^d]^{-\langle d \rangle} & m_{\leqslant n-1}[a, x_n^d] &= \#[a, x_n^d]_{\langle d \rangle} \\ m_n[x_0^d, a) &= \#[x_0^d, a)_{\langle n \rangle} & m_{\leqslant n-1}[x_0^d, a) &= \#[x_0^d, a)^{-\langle n \rangle} \end{aligned}$$

Proof. The first and third equations are a direct consequence of Proposition 2.3.6 and Lemma 2.6.3. The other equations are implied by $A = A^{-\langle j \rangle} + A_{\langle j \rangle}$. □

Lemma 2.6.5. *Let a, b be monomials of $R = K[x_0, \ldots, x_n]$ both with degree d. Then, there exists a monomial $\widetilde{b} \in R$ of degree $d+1$ such that $\#[a,b] = \#[\widehat{a}, \widetilde{b}]$. Furthermore,*

$$m_n[\widehat{a}, \widetilde{b}] \geqslant m_n[a,b].$$

Proof. This result comes from applying Proposition 2.3.6 and Lemma 2.6.3 to Proposition 2.5.6. If $d = 0$, the result is trivial. So, we can safely assume that a and b are not 1.

First, we note that $\widetilde{b}$ must exist because the map $r \mapsto \widehat{r}$ is an injection from $[a,b]$ to $[\widehat{a}, x_n^{d+1}]$.

Now, we set $A = \#[a, x_n^d]$ and $C = \#[a,b]$. By Proposition 2.3.6 and Corollary 2.6.4 we have

$$\begin{aligned}
m_n[a, x_n^d] &= \#[a, x_n^d]^{-\langle d \rangle} \\
&= [A]^{-\langle d \rangle} \\
m_n[\widehat{a}, x_n^{d+1}] &= \#[\widehat{a}, x_n^{d+1}]^{-\langle d+1 \rangle} \\
&= \left[A^{\langle d \rangle}\right]^{-\langle d+1 \rangle} = A \\
m_n(b, x_n^d] &= \#(b, x_n^d]^{-\langle d \rangle} \\
&= [A - C]^{-\langle d \rangle} \\
m_n(\widetilde{b}, x_n^{d+1}] &= \#(\widetilde{b}, x_n^{d+1}]^{-\langle d+1 \rangle} \\
&= \left[\#[\widehat{a}, x_n^{d+1}] - \#[\widehat{a}, \widetilde{b}]\right]^{-\langle d+1 \rangle} \\
&= \left[A^{\langle d \rangle} - C\right]^{-\langle d+1 \rangle}.
\end{aligned}$$

Substituting these equations into Proposition 2.5.6 gives us

$$\begin{aligned}
m_n[\widehat{a}, \widetilde{b}] &= m_n[\widehat{a}, x_n^{d+1}] - m_n(\widetilde{b}, x_n^{d+1}] \\
&\geqslant m_n[a, x_n^d] - m_n(b, x_n^d] \\
&= m_n[a,b].
\end{aligned}$$

□

Proposition 2.6.6. *Let a, b be monomials of $R = K[x_0, \ldots, x_n]$ both with degree d and let $m = \operatorname{maxi} a$. Consider any monomial $\widetilde{a} \in [\widehat{a}, a x_n] = [a x_m, a x_n]$. Then, there exists a monomial $\widetilde{b} \in R$ of degree $d+1$ such that $\#[a,b] = \#[\widetilde{a}, \widetilde{b}]$. Furthermore,*

$$m_{\geqslant i}[\widetilde{a}, \widetilde{b}] \geqslant m_{\geqslant i}[a,b]$$

for all $0 \leqslant i \leqslant n$.

Proof. If $d = 0$, the result is trivial. So, assume that a and b are not 1. Also, we note that $\widetilde{b}$ must exist because the map $r \mapsto r x_n$ is an injection from $[a,b]$ to $[\widetilde{a}, x_n^{d+1}]$.

We will use induction on both n and $\#[a,b]$. When $n = 0$ or $\#[a,b] \leqslant 1$, the result is trivial. We will assume the claim is true for all smaller values of n and $\#[a,b] \geqslant 2$. For clarity, we will explicitly state where the induction is used.

Remark about notation: to keep things organized, we use a, b, etc. to indicate monomials of degree d; $\widetilde{a}$, $\widetilde{b}$, etc. to indicate monomials of degree $d+1$; and a', $\widetilde{a}'$, etc. to indicate monomials in the ring R', which we will define.

There are three cases to consider.

(1) Consider the case $i = n$. Let $\widetilde{c} \in [\widehat{a}, \widetilde{b}]$ be such that $\#[\widehat{a}, \widetilde{c}] = \#[\widetilde{a}, \widetilde{b}]$. Notice that $m_n[\widehat{a}, \widetilde{a}) = 0$. Since $[\widetilde{a}, \widetilde{b}] = [\widetilde{a}, \widetilde{c}] \cup (\widetilde{c}, \widetilde{b}]$, one gets

$$\begin{aligned} m_{\geqslant n}[\widetilde{a}, \widetilde{b}] &= m_n[\widetilde{a}, \widetilde{b}] \\ &= m_n[\widetilde{a}, \widetilde{c}] + m_n(\widetilde{c}, \widetilde{b}] \\ &= m_n[\widehat{a}, \widetilde{a}) + m_n[\widetilde{a}, \widetilde{c}] + m_n(\widetilde{c}, \widetilde{b}] \\ &\geqslant m_n[\widehat{a}, \widetilde{c}] \end{aligned}$$

By choice of $\widetilde{b}$, we have $\#[a, b] = \#[\widetilde{a}, \widetilde{b}]$, and so $\#[a, b] = \#[\widehat{a}, \widetilde{c}]$. Hence Lemma 2.6.5 give $m_n[\widehat{a}, \widetilde{c}] \geqslant m_n[a, b]$. Combined with the previous estimate, we obtain

$$m_{\geqslant n}[\widetilde{a}, \widetilde{b}] \geqslant m_n[a, b] = m_{\geqslant n}[a, b].$$

(2) Consider the case where $\widetilde{a} = ax_n$. Let $c = \max(a, b]$, which exists because $\#[a, b] \geqslant 2$. So,

$$\begin{aligned} [a, b] &= \{a\} \cup [c, b] \\ [\widetilde{a}, \widetilde{b}] &= \{\widetilde{a}\} \cup [\widehat{c}, \widetilde{b}] \end{aligned}$$

because, by choice of c, the monomial $\widehat{c}$ follows ax_n immediately in the lexicographic order of degree $d+1$ monomials. Thus, $\#[c, b] = \#[\widehat{c}, \widetilde{b}]$. Using induction on $\#[a, b]$, we have that $m_{\geqslant i}[\widehat{c}, \widetilde{b}] \geqslant m_{\geqslant i}[c, b]$. Hence, we obtain

$$\begin{aligned} m_{\geqslant i}[\widetilde{a}, \widetilde{b}] &= m_{\geqslant i}\{\widetilde{a}\} + m_{\geqslant i}[\widehat{c}, \widetilde{b}] \\ &= 1 + m_{\geqslant i}[\widehat{c}, \widetilde{b}] \\ &\geqslant m_{\geqslant i}\{a\} + m_{\geqslant i}[c, b] \\ &= m_{\geqslant i}[a, b]. \end{aligned}$$

(3) Lastly, we will consider the case where $i < n$ and $\widetilde{a} \neq ax_n$. Let $R' := K[x_0, \ldots, x_{n-1}]$. Consider the injective ring homomorphism $\iota : R' \to R$ where $x_i \mapsto x_i$. Taking the preimage ι^{-1} of a set of monomials simply "removes" all monomials with max index n. As a result, for $A \subset \mathrm{Mon}(R)$, we have $m_{\geqslant k}(A) = m_n(A) + m_{\geqslant k}(\iota^{-1}(A))$. Specifically, when $k = 0$, we have that $\#A = m_n(A) + \#\iota^{-1}(A)$. We have

$$\begin{aligned} \#\iota^{-1}[a, b] &= \#[a, b] - m_n[a, b] \\ &\geqslant \#[\widetilde{a}, \widetilde{b}] - m_n[\widetilde{a}, \widetilde{b}] \\ &= \#\iota^{-1}[\widetilde{a}, \widetilde{b}], \end{aligned} \tag{2.10}$$

where the inequality follows from Case #1.

Because $\widetilde{a} \neq ax_n$, $n > \operatorname{maxi} \widetilde{a} \geqslant \operatorname{maxi} a$. So, neither $\widetilde{a}$ nor a are removed by ι^{-1}. Thus, there exist $a', \widetilde{a}' \in R'$ such that $\iota(a') = a$ and $\iota(\widetilde{a}') = \widetilde{a}$. In essence, a and a' are the same monomials, just in different rings. (The same can be said of $\widetilde{a}$ and $\widetilde{a}'$.) It follows that $\widetilde{a}' \in [\widehat{a'}, a'x_{n-1}]$.

Let $b' := \min \iota^{-1}[a, b]$ and $\widetilde{c}' := \min \iota^{-1}[\widetilde{a}, \widetilde{b}]$. Notice that ι^{-1} takes an interval in R to an interval in R'. Thus,

$$\iota^{-1}[a, b] = [a', b'] \tag{2.11}$$

$$\iota^{-1}[\widetilde{a}, \widetilde{b}] = [\widetilde{a}', \widetilde{c}']. \tag{2.12}$$

From Inequality (2.10), we see that $\#[a', b'] \geqslant \#[\widetilde{a}', \widetilde{c}']$. By induction on n, there exists some monomial $\widetilde{\delta} \in R'$ such that $\#[\widetilde{a}', \widetilde{\delta}] = \#[a', b'] \geqslant \#[\widetilde{a}', \widetilde{c}']$ (which gives $\widetilde{c}' \geqslant \widetilde{\delta}$) and

$$m_{\geqslant i}[\widetilde{a}', \widetilde{\delta}] \geqslant m_{\geqslant i}[a', b']. \tag{2.13}$$

We now get:

$$m_{\geqslant i}[\widetilde{a}, \widetilde{b}] = m_n[\widetilde{a}, \widetilde{b}] + m_{\geqslant i}\iota^{-1}[\widetilde{a}, \widetilde{b}],$$

by Equation (2.12),

$$m_{\geqslant i}[\widetilde{a}, \widetilde{b}] = m_n[\widetilde{a}, \widetilde{b}] + m_{\geqslant i}[\widetilde{a}', \widetilde{c}'],$$

because $\widetilde{c}' \geqslant \widetilde{\delta}$,

$$m_{\geqslant i}[\widetilde{a}, \widetilde{b}] = m_n[\widetilde{a}, \widetilde{b}] + m_{\geqslant i}[\widetilde{a}', \widetilde{\delta}] - m_{\geqslant i}(\widetilde{c}', \widetilde{\delta}],$$

from Inequality (2.13),

$$m_{\geqslant i}[\widetilde{a}, \widetilde{b}] \geqslant m_n[\widetilde{a}, \widetilde{b}] + m_{\geqslant i}[a', b'] - \#(\widetilde{c}', \widetilde{\delta}],$$

using Equation (2.11),

$$m_{\geqslant i}[\widetilde{a}, \widetilde{b}] \geqslant m_n[\widetilde{a}, \widetilde{b}] + m_{\geqslant i}\iota^{-1}[a, b] - (\#[\widetilde{a}', \widetilde{\delta}] - \#[\widetilde{a}', \widetilde{c}']),$$

by choice of $\widetilde{\delta}$ and Equation (2.12),

$$m_{\geqslant i}[\widetilde{a}, \widetilde{b}] \geqslant m_n[\widetilde{a}, \widetilde{b}] + m_{\geqslant i}\iota^{-1}[a, b] - \#[a', b'] + \#\iota^{-1}[\widetilde{a}, \widetilde{b}],$$

and using Equation (2.11) again and rearranging finally gives,

$$\begin{aligned} m_{\geqslant i}[\widetilde{a}, \widetilde{b}] &\geqslant (m_n[\widetilde{a}, \widetilde{b}] + \#\iota^{-1}[\widetilde{a}, \widetilde{b}]) - \#\iota^{-1}[a, b] + m_{\geqslant i}\iota^{-1}[a, b] \\ &= \#[\widetilde{a}, \widetilde{b}] - \#\iota^{-1}[a, b] + m_{\geqslant i}\iota^{-1}[a, b] \\ &= \#[a, b] - \#\iota^{-1}[a, b] + m_{\geqslant i}\iota^{-1}[a, b] \\ &= m_n[a, b] + m_{\geqslant i}\iota^{-1}[a, b] \\ &= m_{\geqslant i}[a, b]. \end{aligned}$$

□

Chapter 3 Maximum Betti Numbers

We now have enough machinery in place to answer some of the questions of Section 1.2. We will see that the ideals with largest Betti numbers are lexsegment ideals. We then explore 0-depth Hilbert families in Section 3.2, and find answers to some basic questions about them. Finally, we will look at D-depth Hilbert families and reduce questions on them to 0-depth Hilbert families with most complex conditions, and this will make them much easier to attack.

3.1 Lexsegment Ideals

Definition 3.1.1.

a. An ideal $I \subset S = K[x_0, \ldots, x_n]$ is a *lexsegment ideal* if for every pair of monomials $a, b \in S$ with the same degree,

$$(a \in I \text{ and } b \geqslant a) \Rightarrow b \in I.$$

In other words, for each d, there is some a such that $\operatorname{Mon} I_d = [x_0^d, a]$.

b. Analogously, if P is a set of monomials, a set $Q \subset P$ is a *lexsegment subset* of P if for every pair of monomials $a, b \in P$ with the same degree,

$$(a \in Q \text{ and } b \geqslant a) \Rightarrow b \in Q.$$

In other words, for each d, there is some a such that $\operatorname{Mon} Q_d = [x_0^d, a] - P$.

c. For lexsegment ideals $I \subset J$, we define

$$I - J := \operatorname{Mon} I - \operatorname{Mon} J.$$

Remark 3.1.2. Note that the set of monomials of a lexsegment ideal must form a lexsegment subset of $\operatorname{Mon} S$, and in fact, this is a characterization of lexsegment ideals. However, not every lexsegment subset of $\operatorname{Mon} S$ is the set of monomials of some lexsegment ideal.

Lemma 3.1.3. *Let $\mathcal{I}$ be a, possibly infinite, family of lexsegment ideals. Then, $\bigcap_{I \in \mathcal{I}}$ and $\sum_{I \in \mathcal{I}}$ are lexsegment ideals.*

Proof. Let $a, b \in S$ be monomials with the same degree.

Suppose $a \in \bigcap_{I \in \mathcal{I}}$ and $b \geqslant a$. So, for all $I \in \mathcal{I}$, $a \in I$ and thus, $b \in I$. So, $b \in \bigcap_{I \in \mathcal{I}}$.

Suppose $a \in \sum_{I \in \mathcal{I}}$ and $b \geqslant a$. Because the ideals in $\mathcal{I}$ are monomial ideals, there is some $I \in \mathcal{I}$ such that $a \in I$ and thus, $b \in I$. So, $b \in \sum_{I \in \mathcal{I}}$.

Thus, $\bigcap_{I \in \mathcal{I}}$ and $\sum_{I \in \mathcal{I}}$ are lexsegment ideals. □

One essential fact about lexsegment ideals is that there is a one-to-one correspondence between Hilbert functions and lexsegment ideals. This is the essence of Macaulay's theorem [10] in Theorem 2.4.1.

Theorem 3.1.4 (Macaulay)**.** *Let h be the Hilbert function of some quotient of S. Then, there exists a unique lexsegment ideal with that Hilbert function.*

Because this connection is so ubiquitous, we will introduce some notation.

Definition 3.1.5. Consider any ideal $I \subset S$, with Hilbert function $h = h_{S/I}$. We will let both L_h^S and I^{lex} denote the unique lexsegment ideal with Hilbert function h.

Because these lexsegment ideals will be so useful, we wish to have formulas for their Betti numbers. The next few results will accomplish this.

Recall that $\breve{a} = a/x_m$ for a monomial a with $m = \operatorname{maxi} a$.

Lemma 3.1.6. *The set of minimal generators of a proper lexsegment ideal $I \subsetneq S$, is*

$$G(I) = \{a \in \operatorname{Mon} I \mid \breve{a} \notin I\}.$$

Proof. Consider $a \in G(I)$. Thus, we have that $a \in \operatorname{Mon} I$ and a is not a proper multiple of any monomial in I. However, a is a proper multiple of $\breve{a}$. So, $\breve{a} \notin I$.

Consider $a \in \operatorname{Mon} I$ such that $\breve{a} \notin I$. Setting $m = \operatorname{maxi} a$, we have $a = x_0^{a_0} \cdots x_m^{a_m}$. Consider any b where a is a multiple of $b = x_0^{b_0} \cdots x_m^{b_m}$. So, $a_i \geqslant b_i$ for all $i \leqslant m$. Setting $e = \deg \breve{a} - \deg b$, we get that $\breve{a} \geqslant b x_m^e$. Thus, $b x_m^e \notin I$ and so, $b \notin I$. Thus, a is not a multiple of any monomial in I. So, $a \in G(I)$. □

Recall that m-vectors are defined in Definition 2.1.5.

Lemma 3.1.7. *Consider lexsegment ideals $I \subset J \subset S$ where $J - I$ is finite. Then, for every i,*

$$m_i G(I) - m_i G(J) = m_{\leqslant i-1}(J - I)$$

Proof. Consider any $1 \neq b \in \operatorname{Mon} J$. If $\breve{b} \in I$ then $b \in I$, which shows that

$$J - I = \{b \in J - I \mid \breve{b} \notin I\}.$$

Additionally, because $\{b \in \operatorname{Mon} S \mid \breve{b} = a\} = [\widehat{a}, a x_n]$ and $\breve{b} \in J - I$ implies $b \in J$, one gets

$$\{b \in \operatorname{Mon} J \mid \breve{b} \in J - I\} = \{b \in \operatorname{Mon} S \mid \breve{b} \in J - I\} = \{b \in [\widehat{a}, a x_n] \mid a \in J - I\}.$$

Using these two equations along with Lemma 3.1.6, we can write the following equation, where all unions are disjoint:

$$\begin{aligned} G(I) \cup (J - I) &= \{b \in \operatorname{Mon} I \mid \breve{b} \notin I\} \cup \{b \in J - I \mid \breve{b} \notin I\} \\ &= \{b \in \operatorname{Mon} J \mid \breve{b} \notin I\} \\ &= \{b \in \operatorname{Mon} J \mid \breve{b} \notin J\} \cup \{b \in \operatorname{Mon} J \mid \breve{b} \in J - I\} \\ &= G(J) \cup \{b \in [\widehat{a}, a x_n] \mid a \in J - I\} \\ &= G(J) \cup \bigcup_{a \in J - I} [\widehat{a}, a x_n] \end{aligned}$$

Observe that $\bigcup_{a\in J-I}[\widehat{a}, ax_n]$ is indeed a disjoint union since $a > b$ implies $ax_n > \widehat{b}$. We now compute m_i:

$$\begin{aligned} m_iG(I) + m_i(J-I) &= m_iG(J) + \sum_{a\in J-I} m_i[\widehat{a}, ax_n] \\ &= m_iG(J) + \sum_{a\in J-I} \begin{cases} 1 & \text{if } \operatorname{maxi} a \leqslant i \\ 0 & \text{else} \end{cases} \\ &= m_iG(J) + \#\{a \in J-I \mid \operatorname{maxi} a \leqslant i\} \\ &= m_iG(J) + m_{\leqslant i}(J-I). \end{aligned}$$

Rearranging gives the desired result. □

Now, we arrive at a formula for the Betti numbers. In 1990, Eliahou and Kervaire [6] constructed a minimal resolution for what are called stable ideals. We omit the definition. Here, it is sufficient to note that every lexsegment ideal is stable. From this resolution, we also get a simple formula where we only need to know $m_iG(I)$ to compute the Betti numbers. This is exactly what we need. Recall from Definition 1.1.3 that $\beta^S_{i,j}(I)$ denotes a graded Betti number of I.

Theorem 3.1.8 (Eliahou-Kervaire)**.** *For a lexsegment ideal $I \subset S$, we have the following formula for the graded Betti numbers*

$$\beta^S_{q,q+j}(I) = \sum_{a\in[G(I)]_j} \binom{\operatorname{maxi}(a)}{q},$$

which gives a formula for the total Betti numbers,

$$\beta^S_q(I) = \sum_{a\in G(I)} \binom{\operatorname{maxi}(a)}{q}.$$

Corollary 3.1.9. *Consider lexsegment ideals $I \subset J \subset S$ where $J - I$ is finite. Then one has, for each q,*

$$\begin{aligned} \beta^S_q(J) &= \beta^S_q(I) - \sum_{i=q}^{n} m_{\leqslant i-1}(J-I)\binom{i}{q} \\ &= \beta^S_q(I) - \#(J-I)\binom{n+1}{q+1} + \sum_{i=q}^{n} m_{\geqslant i}(J-I)\binom{i}{q} \end{aligned}$$

In particular, $\beta^S_q(J) \leqslant \beta^S_q(I)$.

Proof. The first equality follows from Lemma 3.1.7 and Theorem 3.1.8. The second is a result of the fact that $m_{\leqslant i-1}(J-I)+m_{\geqslant i}(J-I) = \#(J-I)$ and $\sum_{i=q}^{n}\binom{i}{q} = \binom{n+1}{q+1}$. □

3.2 The 0-Depth Case

The first, most natural question to ask is: what is the homogeneous ideal with a given Hilbert function that has largest Betti numbers? The answer is that the unique lexsegment ideal is. This was shown in 1993 by Bigatti [2] and Hulett [9] independently using combinatorial methods if K has characteristic zero, and the result was then proved in 1996 for nonzero characteristic by Pardue [11] using a non-combinatorial proof. Recall that any ideal of the polynomial ring S is assumed to be homogeneous.

Theorem 3.2.1 (Bigatti-Hulett-Pardue)**.** *Let h be the Hilbert function of some quotient of S. Then, L_h^S has maximum graded Betti numbers in the Hilbert family,*

$$\mathcal{I} := \left\{ \text{ideal } I \subset S \;\middle|\; h_{S/I} = h \right\}.$$

In other words, $\beta_{q,q+j}(I^{\text{lex}}) \geqslant \beta_{q,q+j}(I)$ for all ideals $I \subset S$ and integers q,j.

This result works nicely for the graded Betti numbers. However, when we compare ideals with different Hilbert functions, the graded Betti numbers are not necessarily comparable. Instead we will compare the total Betti numbers. First, instead of restricting our family of ideals by specifying the entire Hilbert function, we only specifying what the Hilbert function will be eventually, i.e. we specify the Hilbert polynomial. We get the following result.

Proposition 3.2.2. *Let p be the Hilbert polynomial of some graded quotient of S and $p \neq p_S$. Then, the Hilbert family,*

$$\mathcal{I} := \left\{ \text{ideal } I \subset S \;\middle|\; p_{S/I} = p \right\},$$

has unbounded total Betti numbers.

Proof. Consider $I \in \mathcal{I}$, and let a be any monomial in I^{lex}. Set $J = [I^{\text{lex}}]_{>d}$ where $d = \deg a$. Note that J is a lexsegment ideal with $p_{S/J} = p$. Because I^{lex} is a lexsegment ideal and $x_0^d \geqslant a$, $x_0^d \in I^{\text{lex}} - J$ and we get that $m_{\leqslant n-1}(I^{\text{lex}} - J) > 0$. By Corollary 3.1.9 and Theorem 3.2.1, we have that $\beta_q(J) > \beta_q(I^{\text{lex}}) \geqslant \beta_q(I)$. □

Next, we can ask what will happen if we put bounds on the Hilbert function. This case is still fairly simple, but we find that whether or not we have a maximum will depend on our bounds.

Proposition 3.2.3. *Let $F, G : \mathbb{N}_0 \to \mathbb{N}_0$ be numeric functions, and consider the Hilbert family,*

$$\mathcal{I} := \left\{ \text{ideal } I \subset S \;\middle|\; F(d) \geqslant h_{S/I}(d) \geqslant G(d) \text{ for all } d \right\}.$$

Assume $\mathcal{I}$ is non-trivial, and let L and J be the intersection and sum, respectively, of all lexsegment ideals in $\mathcal{I}$. There are three cases:

1. *Suppose that $F(d) = G(d)$ for $d \gg 0$.*
   Then, L is in $\mathcal{I}$ and has maximum total Betti numbers in $\mathcal{I}$.

2. *Suppose* $m_{\leqslant n-1}(J-L)$ *is finite. Choose any* N *such that* $N > \deg a$ *for all* $a \in J-L$ *with* $\operatorname{maxi} a \leqslant n-1$.
Then, $L + [J]_{>N}$ *has maximum total Betti numbers in* $\mathcal{I}$.

3. *Otherwise,* $\mathcal{I}$ *has unbounded total Betti numbers.*

Note: we cannot say that $L = L_F^S$ because F may not be a valid Hilbert function. Instead, L is the lexsegment ideal of the degree-wise maximum valid Hilbert function satisfying those bounds. Analogously, J is the lexsegment ideal of the degree-wise minimum valid Hilbert function satisfying those bounds.

Proof. First, by Lemma 3.1.3 we can see that I and J must be lexsegment ideals. Note that, by the definition of L, for each d, there is some lexsegment ideal $I \in \mathcal{I}$ such that $[L]_d = [I]_d$ and thus, $h_{S/L}(d) = h_{S/I}(d)$. So, $F(d) \geqslant h_{S/L}(d) \geqslant G(d)$ for all d, and thus $L \in \mathcal{I}$. Similarly, $J \in \mathcal{I}$. Additionally, given any lexsegment ideal, I, such that $L \subset I \subset J$, we have that

$$F(d) \geqslant h_{S/L}(d) \geqslant h_{S/I}(d) \geqslant h_{S/J}(d) \geqslant G(d)$$

for all d giving that $I \in \mathcal{I}$. In part, this means that, in the second case, $L + [J]_{>N} \in \mathcal{I}$ because $L \subset L + [J]_{>N} \subset J$.

Case 1: Suppose $F(d) = G(d)$ for $d \gg 0$. Because $F(d) \geqslant h_{S/L}(d) \geqslant h_{S/J} \geqslant G(d)$ for all d, we have that $h_{S/L}(d) = h_{S/J}(d)$ for $d \gg 0$ which implies finiteness of $J-L$ and that $m_{\leqslant n-1}(J-L)$ is finite, reducing to the second case. In particular, by choosing $N > \deg a$ for all $a \in J-L$, we get $L + [J]_{>N} = L$.

Case 2: Suppose that $m_{\leqslant n-1}(J-L)$ is finite, and consider any $I \in \mathcal{I}$. Because $I^{\text{lex}} \in \mathcal{I}$, $L \subset I^{\text{lex}} \subset J$. Let $I' = I^{\text{lex}} + [J]_{>N}$ which gives $I' - I^{\text{lex}} \subset [J]_{>N} - L$ and thus, $m_{\leqslant n-1}(I' - I^{\text{lex}}) = 0$ because, by choice of N, we have $m_{\leqslant n-1}([J]_{>N} - L) = 0$.

We will now define a map $\varphi : G(I^{\text{lex}}) \to G(I')$. We will show that it is well-defined, injective, and preserves maxi. Given $a = x_0^{a_0}x_1^{a_1}x_2^{a_2} \in G(I^{\text{lex}}) \subset I'$, let $\varphi(a) := x_0^{a_0} \cdots x_{n-1}^{a_{n-1}} x_n^e$ where $e \leqslant a_n$ is the smallest value such that $\varphi(a) \in I'$.

Case a) Suppose $\varphi(a) = a$ and $\operatorname{maxi} a = n$. So, $\operatorname{maxi} \varphi(a) = \operatorname{maxi} a = n$. By choice of e, $\widetilde{\varphi(a)} \notin I'$.

Case b) Suppose $\varphi(a) = a$ and $\operatorname{maxi} a < n$. So, $\operatorname{maxi}(\breve{a}) < n$. By Lemma 3.1.6, $\breve{a} \notin I^{\text{lex}}$ and because $m_{\leqslant n-1}([J]_{>N} - L) = 0$, $\widetilde{\varphi(a)} = \breve{a} \notin I'$.

Case c) Suppose $\varphi(a) \neq a$. Because a is a proper multiple of $\varphi(a)$ and $\widetilde{a \in G(I^{\text{lex}})}$, $\varphi(a) \notin I^{\text{lex}}$. So, $\operatorname{maxi} \varphi(a) = n = \operatorname{maxi} a$, $e > 0$, and thus by choice of e, $\widetilde{\varphi(a)} \notin I'$.

In all cases, $\breve{a} \notin I'$ so, by Lemma 3.1.6, $\varphi(a) \in G(I')$. Additionally, $\operatorname{maxi} \varphi(a) = \operatorname{maxi} a$.

Finally, if $\varphi(a) = \varphi(b)$ then either a is a multiple of b or b is a multiple of a. If both a and b are in $G(I^{\text{lex}})$, this means that $a = b$.

So, φ is an injective map from $G(I^{\text{lex}})$ to $G(I')$ that preserves maxi. By Theorem 3.1.8, we have that $\beta_q(I') \geqslant \beta_q(I^{\text{lex}})$.

Furthermore, because $I' - (L + [J]_{>N})$ is finite, Corollary 3.1.9 and Theorem 3.2.1 give that

$$\beta_q(L + [J]_{>N}) \geqslant \beta_q(I') \geqslant \beta_q(I^{\text{lex}}) \geqslant \beta_q(I)$$

Thus, $L + [J]_{>N}$ has maximum total Betti numbers in $\mathcal{I}$.

Case 3: Consider $I \in \mathcal{I}$, and thus $L \subset I^{\text{lex}} \subset J$. Suppose $J - I^{\text{lex}}$ is finite and thus, $m_{\leqslant n-1}(I^{\text{lex}} - L)$ is not. Let a be any monomial in $I^{\text{lex}} - L$ such that $\operatorname{maxi}(a) \leqslant n-1$ and set $I' = L + [I^{\text{lex}}]_{>d}$ where $d = \deg a$. Thus, $I' - I^{\text{lex}}$ is finite and $m_{\leqslant n-1}(I^{\text{lex}} - I') > 0$. So, by Corollary 3.1.9 and Theorem 3.2.1, we have that $\beta_q(I') > \beta_q(I^{\text{lex}}) \geqslant \beta_q(I)$. Also, because $L \subset I' \subset J$, $I' \in \mathcal{I}$.

Suppose, however, that $J - I^{\text{lex}}$ is infinite. Let a be any monomial in $J - I^{\text{lex}}$ such that $\deg a \geqslant \deg b$ for all $b \in G(I^{\text{lex}})$. Let $I' = I^{\text{lex}} + [J]_{>d}$ where $d = \deg a$. Thus, $G(I^{\text{lex}}) \subsetneq G(I')$. So, by Theorems 3.1.8 and 3.2.1, $\beta_q(I') > \beta_q(I^{\text{lex}}) \geqslant \beta_q(I)$. Also, because $L \subset I' \subset J$, $I' \in \mathcal{I}$. In either situation, given any $I \in \mathcal{I}$, we have produced an ideal with strictly larger Betti numbers than those of I. So, the total Betti numbers are unbounded.

□

3.3 Reduction of Variables

We now describe a way to maximize the Betti numbers when looking at ideals of any depth. We first look at what happens when we restrict our family of ideals to have nonzero depth, and later, restrict even further. As in Propositions 3.2.2 and 3.2.3, we can use Theorem 3.2.1 to reduce the problem from a question of looking for maximums from all ideals to simply looking at maximums from just lexsegment ideals. However, because not all ideals with the same Hilbert function have the same depth, we will need to modify this strategy to work when restricting the depth in some way.

First, we will introduce some notation used to "remove" variables.

Definition 3.3.1. Consider a polynomial ring $S = K[x_0, \ldots, x_n]$. We define $S^{(k)} = K[x_0, \ldots, x_{n-k}]$ to be the polynomial ring in $k \leqslant n$ fewer variables.

Given a numeric function h, we define Δh by $\Delta h(d) = h(d) - h(d-1)$ for all d. We define $\Delta^{-1} h$ by $\Delta^{-1} h(d) = \sum_{i=0}^{d} h(i)$ for all d, and we define $\Delta^k h$ as repeated use of these operations.

Finally, if $\mathcal{I}$ is a D-depth Hilbert family of ideals in S, we define $\mathcal{I}^{(k)}$ to be the following, $(D-k)$-depth Hilbert family of ideals in $S^{(k)}$:

$$\begin{aligned}\mathcal{I}^{(k)} &:= \{\text{ideal } J \subset S^{(k)} \mid \operatorname{depth} J \geqslant D-k \text{ and } h_{S^{(k)}/J} = \Delta^k h_{S/I} \text{ for some } I \in \mathcal{I}\} \\ &= \{\text{ideal } J \subset S^{(k)} \mid \operatorname{depth} J \geqslant D-k \text{ and } \Delta^{-k} h_{S^{(k)}/J} = h_{S/I} \text{ for some } I \in \mathcal{I}\}.\end{aligned}$$

Lemma 3.3.2. *Let $\mathcal{I}$ be a D-depth Hilbert family of ideals in $S = K[x_0, \ldots, x_n]$ and let $k \leqslant D$. Then,*

- *For every $I \in \mathcal{I}$, there is some $J \in \mathcal{I}^{(k)}$ such that*

$$\beta^{S^{(k)}}_{q,q+j}(J) = \beta^{S}_{q,q+j}(I)$$

- *For every $J \in \mathcal{I}^{(k)}$, one has $JS \in \mathcal{I}$ and*

$$\beta^{S}_{q,q+j}(JS) = \beta^{S^{(k)}}_{q,q+j}(J)$$

Thus, if J has maximum/maximal total/graded Betti numbers in $\mathcal{I}^{(k)}$, JS has maximum/maximal total/graded Betti numbers in $\mathcal{I}$.

Proof. We will prove this if $k = 1$. Higher k is achieved by induction, noting that $\left(S^{(k-1)}\right)^{(1)} = S^{(k)}$, $\left(\mathcal{I}^{(k-1)}\right)^{(1)} = \mathcal{I}^{(k)}$, and $JS^{(1)}S = JS$.

Let $R = S^{(1)}$ and consider $I \in \mathcal{I}$. Note that a Hilbert function does not change under an extension of the base field. Hence, by passing to the algebraic closure in necessary, we may assume that K is an infinite field. Since $\operatorname{depth}(S/I) \geqslant 1$, there is some $\ell \in [S]_1$ such that $I : \ell = I$. So, there is an isomorphism $\varphi : S/(\ell S) \to R$. Setting $J = \varphi((I + \ell S)/(\ell S))$ gives us that $R/J \cong \frac{S/(\ell S)}{(I+\ell S)/(\ell S)} \cong S/(I + \ell S)$ and $\operatorname{depth}(R/J) = \operatorname{depth}(S/I) - 1$. The exact sequence

$$0 \to (S/I)(-1) \xrightarrow{\cdot \ell} S/I \to S/(I + \ell S) \cong R/J \to 0$$

gives $h_{R/J} = \Delta h_{S/I}$. Thus, $J \in \mathcal{I}^{(1)}$. By applying Proposition 1.1.5 from [3], we get that

$$\beta^S_{q,q+j}(I) = \beta^R_{q,q+j}(J),$$

which is the first equality. To show the second part of the claim, we note that $R/J \cong \frac{S/(\ell S)}{(JS+\ell S)/(\ell S)}$. We can replace I with JS in the first part of the proof to get the desired result. □

Corollary 3.3.3. *Let $\mathcal{I}$ be a D-depth family of ideals in $S = K[x_0, \ldots, x_n]$. Then, there exists an ideal, I, with maximum total Betti numbers in $\mathcal{I}$ if and only if there exists a lexsegment ideal, L, with maximum total Betti numbers in $\mathcal{I}^{(D)}$. Furthermore, $I = LS$ is one such ideal. The analogous statements with maximal and/or graded Betti numbers also holds.*

Proof. By Lemma 3.3.2, it is sufficient to search for maximums in $\mathcal{I}^{(D)}$, which is a 0-depth family. By Theorem 3.2.1, lexsegment ideals have larger Betti numbers than any others. So, we only need to search for lexsegment ideals in $\mathcal{I}^{(D)}$. □

Corollary 3.3.4. *Let $\mathcal{I}$ be a D-depth family of ideals in the polynomial ring $S = K[x_0, \ldots, x_{D+1}]$. Then, either $\mathcal{I}$ is empty, has unbounded total Betti numbers, or contains an ideal with maximum total Betti numbers in $\mathcal{I}$.*

Proof. Note, $S^{(D)} = K[x_0, \ldots, x_1]$. Any lexsegment ideal $L \subset S^{(D)}$ must contain exactly one generator of the form x_0^d with the remaining generators being of the form $x_0^{e_0}x_1^{e_1}$. As a result, by Theorem 3.1.8, we have that $m_0(G(L)) = 1$ and $m_1(G(L)) = \#G(L) - 1$. So, $\beta_0(L) = \#G(L)$ and $\beta_1(L) = \#G(L) - 1$. As a result, every pair of lexsegment ideal in $\mathcal{I}^{(D)}$ have comparable total Betti numbers. This gives us exactly the desired result. □

The above result shows that it suffices to consider only lexsegment ideals in an appropriate polynomial ring, which greatly simplifies our work. Sometimes is can be helpful to ignore the ideal structure and simply look at sets. This is the main point of the following result and the culmination of Chapter 2.

Proposition 3.3.5. *Consider $S = K[x_0, \dots, x_n]$. Given lexsegment ideals $I \subset J \subset S$ such that $J - I$ is finite, let Q be a lexsegment subset of $J - I$. Then, there exists a lexsegment ideal L with $I \subset L \subset J$ such that*

$$m_{\geqslant i}(L - I) \geqslant m_{\geqslant i}(Q)$$

for all i and

$$\#[L - I]_{\leqslant d} \leqslant \#[Q]_{\leqslant d}$$

for all d.

Proof. Let $\bar{Q} := Q \cup \operatorname{Mon} I$. Note that $\bar{Q}$ is a lexsegment subset of S.

Consider the set of monomials in $\bar{Q}$ that generate a monomial not in $\bar{Q}$:

$$N := \left\{ a \in \bar{Q} \;\middle|\; ax_n \notin \bar{Q} \right\} \subset Q$$

In other words, N is the set of monomials that show that $\bar{Q}$ is not the set of monomials of an ideal. If $N = \varnothing$, we let L be the ideal generated by $\bar{Q}$ and we get $Q = L - I$. So the claimed inequalities are equalities, and we are done.

Otherwise, there is some D such that $[N]_D \neq \varnothing$. Let $a = \max[N]_D$ and consider and $c \in \bar{Q} \cap [a, x_n^D]$. Since $c \leqslant a$, we obtain $cx_n > ax_n$ and because $\bar{Q}$ is a lexsegment subset of R and $ax_n \notin \bar{Q}$, we get $cx_n \notin \bar{Q}$. Thus, $c \in N$ and so, $[N]_D = \bar{Q} \cap [a, x_n^D]$ is the intersection of two intervals and can be written as $[N]_D = [a, b]$ for some b. Set $j = \min\{k \mid ax_k \notin \bar{Q}\}$.

Suppose $[\bar{Q}]_{D+1}$ is nonempty. This means $a \neq x_0^D$. Let $a^+ := \min[x_0^D, a) \notin [N]_D$. Since $a^+ > a$ and $a \in \bar{Q}$, we know $a^+ \in \bar{Q}$. Because $a^+ \notin N$, the definition of N gives that $a^+ x_n \in \bar{Q}$. By choice of j, $ax_j \notin \bar{Q}$, giving $a^+ x_n > ax_j$. Then, because $a^+ x_n$ is the monomial immediately larger than $ax_{\operatorname{maxi}(a)}$, $j \geqslant \operatorname{maxi} a$.

Suppose $[\bar{Q}]_{D+1}$ is empty. So, $a = x_0^D$ and $j = 0$, which gives $j \geqslant \operatorname{maxi} a$ as well.

By Proposition 2.6.6, there is some monomial $\tilde{b}$ of degree $D+1$ such that $\#[a, b] = \#[ax_j, \tilde{b}]$. Define

$$Q' := \Big(Q \cup [ax_j, \tilde{b}] \Big) - [a, b].$$

Because $b \in J$ and $\tilde{b} \geqslant bx_n$, $\tilde{b} \in J$. Because $ax_j \notin \bar{Q}$ by choice of j, $ax_j \notin I$. So, $[ax_j, \tilde{b}]$ is a subset of $J - I$ and thus, Q' is as well. By Proposition 2.6.6, $m_{\geqslant i}(Q') \geqslant m_{\geqslant i}(Q)$ for all i. Also, $\#[Q']_{\leqslant d} \leqslant \#[Q]_{\leqslant d}$ for all d.

Note that $\sum_{a \in Q'} \deg a > \sum_{a \in Q} \deg a$. Because the first sum is bounded by the sum $\sum_{a \in J - I} \deg a$, we can repeat this process only finitely many times until eventually, we have some lexsegment subset $Q'' \subset J - I$ where $N = \varnothing$, $m_{\geqslant i}(Q'') \geqslant m_{\geqslant i}(Q)$, and $\#[Q'']_{\leqslant d} \leqslant \#[Q]_{\leqslant d}$. Setting $L = I + \langle Q'' \rangle$ gives $Q'' = L - I$ from which the desired result follows. □

This results allows us to replace searching for maximums over ideals with searching for maximums over sets. For the sets that give these maximums, the inequalities become equalities because the sets are able to be realized as the set $L - I$ of some ideal L.

3.4 The Positive Depth Case

We now generalize Theorem 3.2.1 to D-depth Hilbert families by applying the reduction of Lemma 3.3.2 with $k = D$ and then using Theorem 3.2.1 on the resulting 0-depth Hilbert family.

Proposition 3.4.1. *Let h be the Hilbert function of some quotient of S, and consider the D-depth Hilbert family,*

$$\mathcal{I} := \left\{ \text{ideal } I \subset S \;\middle|\; h_{S/I} = h \text{ and } \operatorname{depth} I \geqslant D \right\}.$$

If $f = \Delta^D h$ is a valid Hilbert function, SL_f^R has maximum graded Betti numbers in $\mathcal{I}$, where $R = S^{(D)}$. If f is not a valid Hilbert function, $\mathcal{I} = \varnothing$.

Proof. This follows be combining Theorem 3.2.1 and Corollary 3.3.3. □

The natural next question is to ask what may happen if we only specify what the Hilbert function will be eventually, in other words, if we fix the Hilbert polynomial. Proposition 3.2.2 handles the 0-depth case and gives unbounded Betti numbers. Caviglia and Murai [4] proved the following result, which answers this question for the 1-depth case.

Theorem 3.4.2 (Caviglia-Murai)**.** *Let p be the Hilbert polynomial of some quotient of S, and consider the* 1*-depth Hilbert family,*

$$\mathcal{I} := \left\{ \text{saturated ideal } I \subset S \;\middle|\; p_{S/I} = p \right\}.$$

There exists a J with maximum total Betti numbers in $\mathcal{I}$.

If we try to extend this result to higher depths, it fails. For the D-depth case, where $D > 1$, we do not have such existence. We will present two counterexamples in Examples 6.2.2 and 6.2.3.

Another question is how to extend the result of Proposition 3.2.3. In other words, what happens when we look at a D-depth Hilbert family of ideals with certain bounds on the Hilbert function? For $D > 1$, Examples 6.2.2 and 6.2.3 will give us families with no maximums. So, we will only consider 1-depth Hilbert families.

Question 3.4.3. When does the 1-depth family of ideals,

$$\mathcal{I} = \left\{ \text{saturated ideal } I \subset S \;\middle|\; \begin{array}{l} F(d) \geqslant h_{S/I}(d) \geqslant G(d), \\ f(d) \geqslant \Delta h_{S/I}(d) \geqslant g(d) \end{array} \text{ for all } d \right\},$$

contain an ideal with maximum Betti numbers?

We will, in part, answer this question over the next two chapters. Our approach used tools developed above. Corollary 3.3.3 allows us to look at only lexsegment ideals when answering this question and Proposition 3.3.5, at times, allows us to look at only lexsegment subsets when answering this question.

Chapter 4 The 4 variable Case

Throughout this chapter, we will assume that $S = K[x_0, \dots, x_3]$ and $R = S^{(1)} = K[x_0, \dots, x_2]$. The goal of this chapter is to answer one case of the question at the end of last chapter. We will prove the following result.

Theorem 4.0.1. *Let $S = K[x_0, \dots, x_3]$, let f and g be numeric functions, let p be the Hilbert polynomial of some quotient of S, and consider the 1-depth Hilbert family,*

$$\mathcal{I} := \left\{ \text{saturated ideal } I \subset S \;\middle|\; p_{S/I} = p \text{ and } f(d) \geqslant \Delta h_{S/I}(d) \geqslant g(d) \text{ for all } d \right\}.$$

If $\mathcal{I}$ is nonempty, then there exists a J with maximum total Betti numbers in $\mathcal{I}$.

We will need several lemmas before we can prove this at the end of Section 4.2.

4.1 A Ratio of m's

In this section we consider monomials in $R = K[x_0, x_1, x_2]$. We will see that we need to collect monomials in R that maximize m_2. However, it is difficult to collect them all at once, so we wish to have some way to know if a given subset has a proportionally large m_2. To that end, we will focus on the ratio $\frac{m_2}{m_{\leqslant 1}}$. There are three key lemmas that give inequalities involving this ratio. However, this ratio is no defined if $m_{\leqslant 1} = 0$. As a result, although the following results are motivated by this ratio, it does not show up explicitly. Nevertheless, to more easily understand the motivation behind the proofs, it can be helpful to divide by $m_{\leqslant 1}$ when relevant.

It is straightforward to get an exact formula for the m-vector of the interval $[x_0^d, a] \subset R$. Note: we won't use the formulas for $[x_0^d, a)$ from Chapter 2. This is good because those formulas end up give more complex expressions.

Lemma 4.1.1. *For $x_0^{a_0} x_1^{a_1} x_2^{a_2} = a \in \mathrm{Mon}[R]_d$, one has*

$$\begin{aligned} m_{\leqslant 1}[x_0^d, a] &= a_1 + a_2 + 1 \\ m_2[x_0^d, a] &= \frac{(a_1 + a_2)(a_1 + a_2 + 1)}{2} - a_1 \end{aligned}$$

Proof. First note that

$$\begin{aligned} [x_0^d, a] &= [x_0^{a_0+a_1+a_2}, x_0^{a_0} x_1^{a_1} x_2^{a_2}] \\ &= [x_0^{a_0+a_1+a_2}, x_0^{a_0} x_2^{a_1+a_2}] - (x_0^{a_0} x_1^{a_1} x_2^{a_2}, x_0^{a_0} x_2^{a_1+a_2}] \\ &= \left(\bigcup_{d=0}^{a_1+a_2} [x_0^{a_0+d} x_1^{a_1+a_2-d}, x_0^{a_0+d} x_2^{a_1+a_2-d}] \right) - (x_0^{a_0} x_1^{a_1} x_2^{a_2}, x_0^{a_0} x_2^{a_1+a_2}] \\ &= \left(\bigcup_{d=0}^{a_1+a_2} x_0^{a_0+d} [x_1^{a_1+a_2-d}, x_2^{a_1+a_2-d}] \right) - x_0^{a_0} (x_1^{a_1} x_2^{a_2}, x_2^{a_1+a_2}] \end{aligned}$$

Now, $(x_1^{a_1}x_2^{a_2}, x_2^{a_1+a_2}]$ has cardinality a_1 and only contains monomials with max index 2. The interval $[x_1^{a_1+a_2-d}, x_2^{a_1+a_2-d}]$ has cardinality $a_1 + a_2 - d + 1$ and the only monomial in it whose max index is at most 1 is $x_1^{a_1+a_2-d}$. So,

$$m_{\leqslant 1}[x_0^d, a] = \left(\sum_{d=0}^{a_1+a_2} 1\right) - 0 = a_1 + a_2 + 1$$
$$m_2[x_0^d, a] = \left(\sum_{d=0}^{a_1+a_2} a_1 + a_2 - d\right) - a_1 = \frac{(a_1 + a_2)(a_1 + a_2 + 1)}{2} - a_1 \qquad \square$$

Corollary 4.1.2. *For $a \in \mathrm{Mon}[R]_d$, one has*

$$m_2[x_0^{d+1}, ax_2] = m_2[x_0^d, a] + m_{\leqslant 1}[x_0^d, a], \tag{4.1}$$
$$m_{\leqslant 1}[x_0^{d+1}, ax_2] = m_{\leqslant 1}[x_0^d, a] + 1 \tag{4.2}$$

and

$$m_{\leqslant 1}[x_0^d, a] \cdot \frac{m_{\leqslant 1}[x_0^d, a] - 1}{2} \geqslant m_2[x_0^d, a] \tag{4.3}$$

Proof. This follows by a straightforward computation using Lemma 4.1.1. $\square$

Inequality (4.3) gives an upper bound for $\frac{m_2}{m_{\leqslant 1}}$ ratio on the interval $[x_0^d, a]$. We will also need another upper bound in a more general case. There are two subcases, which we will deal with independently in Lemmas 4.1.3 and 4.1.4. In some sense, Lemma 4.1.3 is a generalization of Proposition 2.6.6 that will not extend to more variables.

Lemma 4.1.3. *For $a = x_0^{a_0}x_1^{a_1}x_2^{a_2} \in [R]_d$ where $a_2 + 1 \geqslant a_1$ and $b \in [x_0^{d+e}, ax_2^e]$ with $e \geqslant 0$, we have*

$$m_{\leqslant 1}[x_0^d, a] \cdot m_2[b, ax_2^e] \geqslant m_2[x_0^d, a] \cdot m_{\leqslant 1}[b, ax_2^e]$$

Proof. Suppose $b \neq x_0^{d+e}$. Setting $c = \min[x_0^{d+e}, b)$, we get

$$[x_0^{d+e}, c] \cup [b, ax_2^e] = [x_0^{d+e}, ax_2^e]$$

We write $c = x_0^{c_0}x_1^{c_1}x_2^{c_2}$ and let $\delta := c_0 - a_0$. Because $c \geqslant ax_2^e$, we have $\delta \geqslant 0$ and

$$\delta > 0 \qquad \text{or} \qquad c_1 \geqslant a_1 \tag{$\star$}$$

Because $c_0 + c_1 + c_2 = a_0 + a_1 + a_2 + e$, we obtain

$$\delta - e = (a_1 + a_2) - (c_1 + c_2)$$

We use Lemma 4.1.1 to get the following intermediate values,

$$\begin{aligned}
N_1 &:= 2m_2[x_0^{d+e}, ax_2^e] \cdot m_{\leqslant 1}[x_0^d, a] \\
&= (a_1 + a_2 + e)(a_1 + a_2 + e + 1)(a_1 + a_2 + 1) - 2a_1(a_1 + a_2 + 1) \\
N_2 &:= 2m_2[x_0^d, a] \cdot m_{\leqslant 1}[x_0^{d+e}, ax_2^e] \\
&= (a_1 + a_2)(a_1 + a_2 + 1)(a_1 + a_2 + e + 1) - 2a_1(a_1 + a_2 + e + 1) \\
N_3 &:= 2m_2[x_0^d, a] \cdot m_{\leqslant 1}[x_0^{d+e}, c] \\
&= (a_1 + a_2)(a_1 + a_2 + 1)(c_1 + c_2 + 1) - 2a_1(c_1 + c_2 + 1) \\
N_4 &:= 2m_2[x_0^{d+e}, c] \cdot m_{\leqslant 1}[x_0^d, a] \\
&= (c_1 + c_2)(c_1 + c_2 + 1)(a_1 + a_2 + 1) - 2c_1(a_1 + a_2 + 1)
\end{aligned}$$

This gives us that

$$\begin{aligned}
N_1 - N_2 &= e(a_1 + a_2 + e + 1)(a_1 + a_2 + 1) + 2a_1e \\
N_3 - N_4 &= (\delta - e)(c_1 + c_2 + 1)(a_1 + a_2 + 1) \\
&\qquad - 2a_1(c_2 + 1) + 2c_1(a_2 + 1) \\
N_1 - N_2 + N_3 - N_4 &= \delta(c_1 + c_2 + 1)(a_1 + a_2 + 1) - 2a_1(c_2 + 1) + 2c_1(a_2 + 1) \\
&\qquad + e(a_1 + a_2 - c_1 - c_2 + e)(a_1 + a_2 + 1) + 2a_1e \\
&= \delta(c_1 + c_2 + 1)(a_1 + a_2 + 1) - 2a_1(c_2 + 1) + 2c_1(a_2 + 1) \\
&\qquad + e\delta(a_1 + a_2 + 1) + 2a_1e
\end{aligned}$$

Because all variables are nonnegative and $a_2 + 1 \geqslant a_1$, we can bound this below,

$$\begin{aligned}
N_1 - N_2 + N_3 - N_4 &\geqslant \delta(c_2 + 1)(a_1 + a_1) - 2a_1(c_2 + 1) + 2c_1(a_2 + 1) + 2a_1e \\
&= 2a_1(\delta - 1)(c_2 + 1) + 2c_1(a_2 + 1) + 2a_1e
\end{aligned}$$

If $\delta > 0$, this is nonnegative. And if $\delta = 0$, we get

$$\begin{aligned}
N_1 - N_2 + N_3 - N_4 &\geqslant -2a_1(c_2 + 1) + 2c_1(a_2 + 1) + 2a_1(c_1 + c_2 - a_1 - a_2) \\
&= 2(a_1 + a_2 + 1)(c_1 - a_1)
\end{aligned}$$

which is also nonnegative by $(\star)$. Note that

$$\frac{N_1 - N_4}{2} = m_{\leqslant 1}[x_0^d, a] \cdot m_2(c, ax_2^e], \qquad \frac{N_3 - N_2}{2} = -m_{\leqslant 1}(c, ax_2^e] \cdot m_2[x_0^d, a],$$

and $(c, ax_2^e] = [b, ax_2^e]$ by the choice of c. Thus,

$$m_2[b, ax_2^e]m_{\leqslant 1}[x_0^d, a] - m_2[x_0^d, a]m_{\leqslant 1}[b, ax_2^e] = \frac{N_1 - N_2 + N_3 - N_4}{2} \geqslant 0.$$

Suppose $b = x_0^{d+e}$. Then,

$$m_2[b, ax_2^e]m_{\leqslant 1}[x_0^d, a] - m_2[x_0^d, a]m_{\leqslant 1}[b, ax_2^e] = \frac{N_1 - N_2}{2} \geqslant 0.$$

In either case, we have established the desired result. □

Lemma 4.1.4. *For $a, b \in [R]_d$ and $r \geqslant 0$ with $a \geqslant b$, we write $b = x_0^{b_0} x_1^{b_1} x_2^{b_2}$. Then, if $b_1 - 1 \geqslant r$ and $b_2 \geqslant r$, we have,*

$$m_2[a, b] \geqslant r \cdot m_{\leqslant 1}[a, b]$$

Proof. Let us write $a = x_0^{a_0} x_1^{a_1} x_2^{a_2}$. Note that $a_0 \geqslant b_0$ because $a \geqslant b$. There are three cases depending on a_0, b_0, and a_2.

Case 1: Suppose $a_0 = b_0$ and $a_2 > 0$. Then, $a \in (x_0^{b_0} x_1^{b_1+b_2}, b] = x_0^{b_0} x_1^{b_1} (x_1^{b_2}, x_2^{b_2}]$. Thus, $m_{\leqslant 1}[a, b] = 0$ and the desired inequality holds.

Case 2: Suppose $a_2 = 0$. Thus, $b_0 + b_1 + b_2 - a_0 = a_1$ and $[a, b]$ is a disjoint union as follows:

$$\begin{aligned}
[a, b] &= \left(\bigcup_{i=1}^{a_0 - b_0} [x_0^{b_0+i} x_1^{b_1+b_2-i}, x_0^{b_0+i} x_2^{b_1+b_2-i}] \right) \cup [x_0^{b_0} x_1^{b_1+b_2}, x_0^{b_0} x_1^{b_1} x_2^{b_2}] \\
&= \left(\bigcup_{i=1}^{a_0 - b_0} x_0^{b_0+i} [x_1^{b_1+b_2-i}, x_2^{b_1+b_2-i}] \right) \cup x_0^{b_0} x_1^{b_1} [x_1^{b_2}, x_2^{b_2}].
\end{aligned}$$

Using this to compute $m_2[a, b]$ and $m_{\leqslant 1}[a, b]$ gives us $m_{\leqslant 1}[a, b] = a_0 - b_0 + 1$ and

$$\begin{aligned}
m_2[a, b] &= \left(\sum_{i=1}^{a_0 - b_0} b_1 + b_2 - i \right) + b_2 \\
&= (a_0 - b_0) \frac{(b_1 + b_2 - 1) + (b_1 + b_2 - (a_0 - b_0))}{2} + b_2 \\
&= (a_0 - b_0) \frac{(b_1 + b_2 - 1) + (a_1 + a_2)}{2} + b_2 \\
&\geqslant (a_0 - b_0) \frac{b_1 + b_2 - 1}{2} + b_2 \\
&\geqslant (a_0 - b_0) \frac{r + r}{2} + r \\
&= r \cdot ((a_0 - b_0) + 1) \\
&= r \cdot m_{\leqslant 1}[a, b].
\end{aligned}$$

Case 3: Suppose $a_0 > b_0$ and $a_2 > 0$. Thus, $[a, b]$ is a disjoint union as follows:

$$\begin{aligned}
[a, b] &= [x_0^{a_0} x_1^{a_1} x_2^{a_2}, x_0^{a_0} x_2^{a_1+a_2}] \cup \left(\bigcup_{i=1}^{a_0 - b_0 - 1} [x_0^{b_0+i} x_1^{b_1+b_2-i}, x_0^{b_0+i} x_2^{b_1+b_2-i}] \right) \\
&\quad \cup [x_0^{b_0} x_1^{b_1+b_2}, x_0^{b_0} x_1^{b_1} x_2^{b_2}] \\
&= x_0^{a_0} [x_1^{a_1} x_2^{a_2}, x_2^{a_1+a_2}] \cup \left(\bigcup_{i=1}^{a_0 - b_0 - 1} x_0^{b_0+i} [x_1^{b_1+b_2-i}, x_2^{b_1+b_2-i}] \right) \cup x_0^{b_0} x_1^{b_1} [x_1^{b_2}, x_2^{b_2}]
\end{aligned}$$

Using this to compute $m_2[a, b]$ and $m_{\leqslant 1}[a, b]$ gives us

$$m_{\leqslant 1}[a, b] = 0 + (a_0 - b_0 - 1) + 1 = a_0 - b_0$$

and

$$\begin{aligned}
m_2[a,b] &= (a_1+1) + \left(\sum_{i=1}^{a_0-b_0-1} b_1+b_2-i\right) + b_2 \\
&= a_1 + (a_0-b_0-1)\,\frac{(b_1+b_2-1)+(b_1+b_2-(a_0-b_0-1))}{2} + b_2 \\
&= a_1 + (a_0-b_0-1)\,\frac{(b_1+b_2-1)+(a_1+a_2+1)}{2} + b_2 \\
&\geqslant (a_0-b_0-1)\,\frac{b_1+b_2-1}{2} + b_2 \\
&\geqslant (a_0-b_0-1)\,\frac{r+r}{2} + r \\
&= r\cdot(a_0-b_0) \\
&= r\cdot m_{\leqslant 1}[a,b].
\end{aligned}$$

□

Combining the last two results, we can obtain a lower bound for the $\frac{m_2}{m_{\leqslant 1}}$ ratio of the interval $[b, ax_2^e]$.

Corollary 4.1.5. *For $a = x_0^{a_0}x_1^{a_1}x_2^{a_2} \in [R]_d$ and $b \in [x_0^{d+e}, ax_2^e]$ with $e \geqslant 0$, we have*

$$m_{\leqslant 1}[x_0^d, a]\cdot m_2[b, ax_2^e] \geqslant m_2[x_0^d, a]\cdot m_{\leqslant 1}[b, ax_2^e]$$

or

$$m_2[b, ax_2^e] \geqslant a_2\cdot m_{\leqslant 1}[b, ax_2^e]$$

Proof. If $a_2+1 \geqslant a_1$, we use Lemma 4.1.3, giving the first inequality. If $a_1-1 \geqslant a_2$ we apply Lemma 4.1.4 to the interval $[b, ax_2^e]$, giving the second inequality. □

We will now work toward the final, key inequality, Lemma 4.1.8, which gives us a bound on the $\frac{m_2}{m_{\leqslant 1}}$ ratio for subsets of Q with $m_0(Q) = 0$.

The following defines a concept of the "best" subset of Q, in the sense of maximizing this ratio, when fixing $m_{\leqslant 1}$.

Definition 4.1.6. Consider a set $Q \subset \operatorname{Mon} R$.
Define $r_Q : \{i \in \mathbb{N}_0 \mid 0 \leqslant i \leqslant m_{\leqslant 1}(Q)\} \to \mathbb{N}_0$ by

$$r_Q(i) := \max\{m_2(P) \mid P \text{ is a lexsegment subset of } Q \text{ and } m_{\leqslant 1}(P) = i\}.$$

Proof of well-definition. By setting $P_0 = \varnothing$ and $P_{j+1} = P_j \cup \{\max(Q - P_j)\}$ we get a sequence of lexsegment subsets of Q,

$$\varnothing = P_0 \subset P_1 \subset \cdots \subset P_t = Q$$

such that $\#P_j = j$. As a result, $m_{\leqslant 1}(P_{j+1}) - m_{\leqslant 1}(P_j)$ is at most 1. So, $m_{\leqslant 1}(P_j) = i$ for some j. □

Lemma 4.1.7. *Suppose $q_1 \geqslant \cdots \geqslant q_t$ is a decreasing sequence of integers. For $1 \leqslant i \leqslant t$, one has*

$$(t-i)\sum_{j=1}^{t} q_j \geqslant t \sum_{j=i+1}^{t} q_j$$

Proof. The arithmetic mean of the last $t-i$ elements, $q_{i+1}, \ldots, q_t$, is at most the arithmetic mean of the entire sequence. The result follows from this fact. □

Lemma 4.1.8. *Consider $Q = L - I$ for lexsegment ideals $I \subset L \subset R$. Suppose $r_Q(1) < a_1 + a_2$ for each $x_0^{a_0}x_1^{a_1}x_2^{a_2} = a \in Q$. Then*

$$r_Q(i) \cdot m_{\leqslant 1}(Q) \geqslant i \cdot m_2(Q)$$

for any nonnegative integer $i \leqslant m_{\leqslant 1}(Q)$.

Remark: If i and $m_{\leqslant 1}(Q)$ are positive, this is equivalent

$$\frac{r_Q(i)}{i} \geqslant \frac{m_2(Q)}{m_{\leqslant 1}(Q)},$$

where equality is clear if $i = m_{\leqslant 1}(Q)$. A stronger statement is that $\frac{r_Q(i)}{i}$ is a decreasing function of i. This follows by using a slightly stronger version of Lemma 4.1.7. We omit the argument because we do not need this result.

Proof. We set $Q_d = \{a \in Q \mid \deg a = d\}$ and begin by showing that $m_{\leqslant 1}(Q_d) \leqslant 1$ for all d. For the sake of contradiction, assume that $m_{\leqslant 1}(Q_d) > 1$ for some d. Let

$$a = \max\{b \in Q_d \mid \operatorname{maxi}(b) \leqslant 1\} = x_0^{a_0}x_1^{a_1}$$

where $a_0 + a_1 = d$. Since $m_{\leqslant 1}(Q_d) > 1$ and $Q = L - I$, the monomial $x_0^{a_0-1}x_1^{a_1+1}$ must also be in Q_d. Set $b = x_0^{a_0}x_2^{a_1}$. We see that $b \in [x_0^{a_0}x_1^{a_1}, x_0^{a_0-1}x_1^{a_1+1}] \subset Q_d$.

We will now find an interval similar to $[a, b]$ in the highest possible degree. Let

$$\begin{aligned} e &= \max\{f \mid ax_1^f \in Q\} \\ a^{\uparrow} &= ax_1^e = x_0^{a_0}x_1^{a_1+e} \in Q_{d+e} \\ b^{\uparrow} &= bx_2^e = x_0^{a_0}x_2^{a_1+e}. \end{aligned}$$

Since $b \in Q \subset L$, we have that $b^{\uparrow} \in L$. Because $a^{\uparrow} \in Q$ gives $a^{\uparrow} \notin I$, $a^{\uparrow} > b^{\uparrow}$ gives $b^{\uparrow} \notin I$ and thus, $b^{\uparrow} \in Q$. Let $T \subset L$ be the lexsegment ideal generated by $[x_0^{d+e}, b^{\uparrow}]$. We now wish to show that $m_2(T - I) \leqslant r_Q(1)$. (Note: $a^{\uparrow} \in T - I$.)

Consider any $c \in T - I \subset Q$ such that $\operatorname{maxi}(c) \leqslant 1$. So, $c = x_0^{c_0}x_1^{c_1}$. Because T is generated by elements greater than or equal to $b^{\uparrow}$, we must have $c \geqslant b^{\uparrow}x_2^f$ for $f = \deg c - \deg b^{\uparrow} \geqslant 0$. That is, $x_0^{c_0}x_1^{c_1} \geqslant x_0^{a_0}x_2^{a_1+e+f}$, and so $c_0 \geqslant a_0$. By definition of e, $a^{\uparrow}x_1^f = ax_1^{e+f} \notin Q = L - I$ if $f > 0$. However, $a^{\uparrow} \in L$ and thus, $a^{\uparrow}x_1^f \in I$. Because $c \notin I$, $c \leqslant a^{\uparrow}x_1^f$. Thus, $c_0 \leqslant a_0$. So, $c_0 = a_0$. Thus, $ax_1^{e+f} \in [x_0^{d+e+f}, b^{\uparrow}x_2^f]$ Because $c = x_0^{c_0}x_1^{c_1} = x_0^{a_0}x_1^{c_1}$, we have $c = ax_1^{e+f}$. By definition of e, it follows that $e + f \leqslant e$,

a contradiction of $f > 0$. So, $f = 0$. Thus, $c = a^{\uparrow}$. So, $m_{\leqslant 1}(T - I) = 1$ and thus, $m_2(T - I) \leqslant r_Q(1)$ by definition.

Additionally, $[a^{\uparrow}, b^{\uparrow}] \subset (T - I)$ and $m_2[a^{\uparrow}, b^{\uparrow}] = m_2[x^{a_1+e}, x_2^{a_1+e}] = a_1 + e$. So, $a_1 + e \leqslant m_2(T - I) \leqslant r_Q(1)$. But, $x_0^{a_0} x_1^{a_1} \in Q$, which means $r_Q(1) < a_1$ by hypothesis. This is a contradiction. Thus, we have that

$$m_{\leqslant 1}(Q_d) \leqslant 1$$

for all d.

So, in each degree of Q, we have at most one monomial with max index less than 2, and the rest have max index 2. We care mostly about the set of degrees with exactly one monomial with max index less than 2. This set of degrees is

$$D := \{d \mid m_{\leqslant 1}(Q_d) = 1\}$$

For $d \in D$, we define Q_d^- to be the set of monomials in Q_d less than this single monomial with max index less than 2. In other words,

$$Q_d^- := [q_d, x_2^d] \cap Q_d$$

where $q_d \in Q_d$ is the unique monomial in Q_d with $\operatorname{maxi} q_d \leqslant 1$. Order the set of integers $D = \{d_1, \ldots, d_t\}$ such that

$$\#Q_{d_1}^- \geqslant \cdots \geqslant \#Q_{d_t}^- \tag{$*$}$$

where, $t = \#D = m_{\leqslant 1}(Q)$. Finally, for $0 \leqslant i \leqslant t$, define

$$P_i := Q - \bigcup_{j=i+1}^{t} Q_{d_j}^-$$

Because $Q_{d_j}^-$ is a set of the smallest elements in Q_{d_j}, P_i is a lexsegment subset of Q. Additionally, because $m_{\leqslant 1}(Q_{d_j}^-) = 1$ and $m_{\leqslant 1}(Q) = t$, $m_{\leqslant 1}(P_i) = i$.

We claim that each P_i is the largest lexsegment subset of Q with $m_{\leqslant 1}(P_i) = i$. This is true because any lexsegment subset P' of Q such that $m_{\leqslant 1}(P') = i$ must have $t - i$ monomials with max index less than 2 removed from Q, which means that $P' \subset Q - \bigcup_{d \in E} Q_d^-$ for some $E \subset D$ with $\#E = t - i$. From this, $(*)$ gives $\#P_i \geqslant \#P'$, proving the claim. Thus, $m_2(P_i) = \#P_i - m_{\leqslant 1}(P_i) \geqslant \#P' - m_{\leqslant 1}(P') = m_2(P')$.

So,

$$r_Q(i) = m_2(P_i) = m_2(Q) - \sum_{j=i+1}^{t} (\#Q_{d_j}^- - 1)$$

Because $\#Q_{d_j} - 1$ is a decreasing sequence, Lemma 4.1.7 gives

$$\begin{aligned} m_1(Q) \cdot r_Q(i) &= t \cdot m_2(P_i) \\ &= t \cdot m_2(Q) - t \sum_{j=i+1}^{t} (\#Q^-_{d_j} - 1) \\ &\geqslant t \cdot m_2(Q) - (t-i) \sum_{j=1}^{t} (\#Q^-_{d_j} - 1) \\ &\geqslant t \cdot m_2(Q) - (t-i) m_2(Q) \\ &= i \cdot m_2(Q). \end{aligned}$$

□

4.2 Maximizing m_2 and β

There are three results we will prove. Each has a fewer conditions in the hypothesis than the previous, and a stronger conclusion. Lemma 4.2.2 is the weakest and is the driving force behind Lemma 4.2.3. Lemma 4.2.3 is tedious, but removes the technical conditions on Lemma 4.2.2. Finally, Lemma 4.2.6 is the main result of the section and is essentially an induction argument for which Lemma 4.2.3 is the base case.

We will be looking at the following family of ideals, which is closely linked with Theorem 4.0.1.

Definition 4.2.1. Given a lexsegment ideal $I \subset R$ and integer c, define the following family of lexsegment ideals

$$\mathcal{U}(I, c) := \{\text{lexsegment ideal } L \subset R \mid I \subset L \text{ and } \#(L - I) = c\}.$$

Often, I and c will be clear, and we will simply write $\mathcal{U}$. Additionally, we will use L^{set} as shorthand for the set of monomials in L that are not in I. In other words, $L^{\text{set}} := L - I$.

Consider ideals $L_1, L_2 \in \mathcal{I}(I, c)$. The general idea of the next result is to "replace" part of L_1 with part of L_2. This results in a "better" ideal than we started with. This is a technical lemma, and its only purpose is to establish Lemma 4.2.3. We have written it as a separate lemma simply because it is used three times in Lemma 4.2.3. It would be a good idea to read the first step of the proof of Lemma 4.2.3 first to get a motivation for this lemma. See Figure 4.1 for a visualization of this next statement/proof. Recall that the function r_Q is introduced in Definition 4.1.6.

Lemma 4.2.2. *Consider $L_1, L_2 \in \mathcal{U}(I, c)$ such that $L_1 \cap L_2 = I$ and a nonempty set, $X = L_1 - Y$, where Y is a lexsegment ideal satisfying $I \subset Y \subset L_1$. Suppose that*

(i) $m_{\leqslant 1}(X) \leqslant m_{\leqslant 1}(L_2^{\text{set}})$ *and*

(ii) $m_2(X) \leqslant r_{L_2^{\text{set}}}(m_{\leqslant 1}(X))$.

Then, there is some $L_3 \in \mathcal{U}(I, c)$ such that

- $L_3 \subset L_1 + L_2$,
- $L_3 \neq L_1$, *and*
- $m_2(L_3^{\mathrm{set}}) \geqslant m_2(L_1^{\mathrm{set}})$,

where $L_i^{\mathrm{set}} = L_i - I$.

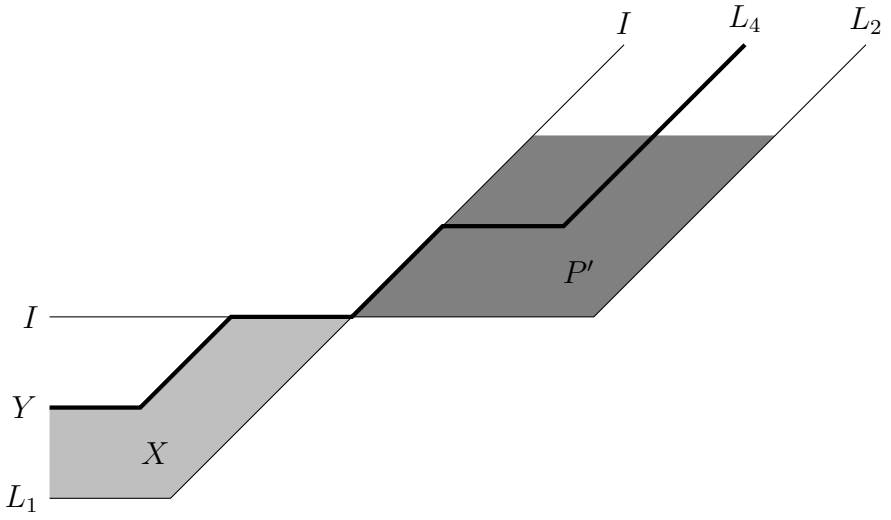

Figure 4.1: Visualization of Lemma 4.2.2

Proof. Consider P, a lexsegment subset of L_2^{set}, such that $m_2(P) = r_{L_2^{\mathrm{set}}}(m_{\leqslant 1}(X))$ and $m_{\leqslant 1}(P) = m_{\leqslant 1}(X)$. (Such a P must exist by Definition 4.1.6.) Thus, $m_2(P) \geqslant m_2(X)$ and

$$\#X = m_2(X) + m_{\leqslant 1}(X) \leqslant m_2(P) + m_{\leqslant 1}(P) = \#P.$$

So, there exists some lexsegment subset of L_2^{set}, $P' \subset P$, with $\#P' = \#X$.

Since $L_2^{\mathrm{set}} = L_2 - I$ is finite, by Proposition 3.3.5, there is some lexsegment ideal, $L_4 \in \mathcal{U}(I, \#X)$, such that $L_4 \subset L_2$ and $m_2(L_4^{\mathrm{set}}) \geqslant m_2(P')$. In particular, $\#L_4^{\mathrm{set}} = \#X > 0$.

Let $L_3 = Y + L_4$. Since $Y^{\mathrm{set}} \cap L_4^{\mathrm{set}} \subset L_1^{\mathrm{set}} \cap L_2^{\mathrm{set}}$ is empty by assumption, $L_3^{\mathrm{set}} = Y^{\mathrm{set}} \cup L_4^{\mathrm{set}}$ is a disjoint union and thus,

$$\#L_3^{\mathrm{set}} = \#Y^{\mathrm{set}} + \#L_4^{\mathrm{set}} = \#Y^{\mathrm{set}} + \#X = \#(Y - I) + \#(L_1 - Y) = c.$$

So, $L_3 \in \mathcal{U}(I, c)$. Because $L_4^{\mathrm{set}} \subset L_2^{\mathrm{set}}$ is nonempty and disjoint with L_1^{set}, we conclude that $L_3 \neq L_1$. Finally, using $m_2(L_4^{\mathrm{set}}) \geqslant m_2(P')$ and $m_2(P) \geqslant m_2(X)$, we get

$$\begin{aligned} m_2(L_3^{\mathrm{set}}) &\geqslant m_2(Y^{\mathrm{set}}) + m_2(P') \\ &= m_2(L_1^{\mathrm{set}}) - m_2(X) + m_2(P) - m_2(P - P') \\ &\geqslant m_2(L_1^{\mathrm{set}}). \end{aligned}$$

□

Finally, we will prove the following lemma, from which Theorem 4.0.1 will follow easily.

Lemma 4.2.3. *Consider $L_1, L_2 \in \mathcal{U}(I, c)$ such that $L_1 \cap L_2 = I$. Suppose that*

(i) $m_2(L_1^{\text{set}}) = m_2(L_2^{\text{set}}) + 1$ *and*

(ii) $m_1(L_1^{\text{set}}) + 2 < m_1(L_2^{\text{set}})$.

Then, there is some $L_3 \in \mathcal{U}(I, c)$ such that

- $L_3 \subset L_1 + L_2$,
- $L_3 \neq L_1$, *and*
- $m_2(L_3^{\text{set}}) \geqslant m_2(L_1^{\text{set}})$.

where $L_i^{\text{set}} = L_i - I$.

Proof. There are two phases to this proof. The first step is to break L_1^{set} into parts on which we can use Lemma 4.2.2 in 3 different cases. The second step is to show that no other cases exist.

Step 1: Define the following:

$$\begin{aligned} d_0 &:= \min\{\deg a \mid a \in L_1^{\text{set}}\} \\ f &:= \min[L_1^{\text{set}}]_{d_0} = x_0^{f_0} x_1^{f_1} x_2^{f_2} \\ U_i &:= [x_0^{d_0+i}, f x_2^i] \cap L_1^{\text{set}} \\ V &:= L_1^{\text{set}} - \bigcup_{i \geqslant 0} U_i \\ W &:= [x_0^{f_0} x_1^{f_1+f_2}, f]. \end{aligned}$$

Note: because L_1^{set} is $L_1 - I$, the monomials in each degree form an interval. Thus U_i is the intersection of two intervals and is an interval itself. Furthermore, because $f x_2^i \in L_1$, we can write $U_i = [b, f x_2^i]$ for some b.

We can be even more specific for U_0 and U_1. Since $\#L_1^{\text{set}} = \#L_2^{\text{set}}$, assumptions (i) and (ii) imply,

$$m_0(L_1^{\text{set}}) = m_0(L_2^{\text{set}}) + m_1(L_2^{\text{set}}) - m_1(L_1^{\text{set}}) + m_2(L_2^{\text{set}}) - m_2(L_1^{\text{set}}) > m_0(L_2^{\text{set}}) + 1$$

So, $m_0(L_1^{\text{set}}) \geqslant 2$. This gives that $x_0^{d_0}, x_0^{d_0+1} \notin I$ and $x_0^{d_0}, x_0^{d_0+1} \in L_1^{\text{set}}$. Consequently,

$$\begin{aligned} U_0 &= [x_0^{d_0}, f] \\ U_1 &= [x_0^{d_0+1}, f x_2] \\ m_{\leqslant 1}(U_0) &> 0. \end{aligned} \tag{4.4}$$

We can visualize L_1^{set} as the disjoint union of V and the U_i's by drawing the largest monomials on the left, and the largest degrees on top. Note that W is a non-empty subset of U_0. This is shown in Figure 4.2.

Notice that $\#U_0 \geqslant \#W > 0$, but $\#V$ could be 0. There are 3 cases to deal with:

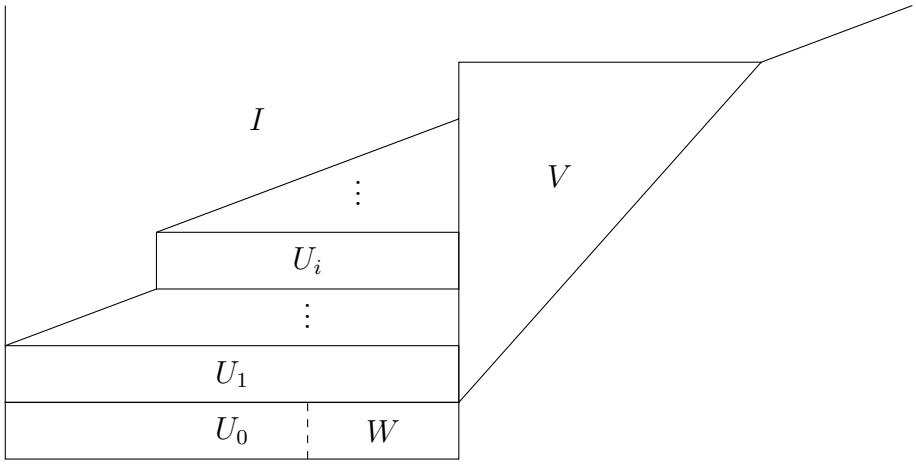

Figure 4.2: Visualization of L_1 in Lemma 4.2.3.

- $r_{L_2^{\text{set}}}(m_{\leqslant 1}(W)) \geqslant m_2(W)$, that is, $r_{L_2^{\text{set}}}(1) \geqslant f_2$
- $r_{L_2^{\text{set}}}(m_{\leqslant 1}(U_0)) \geqslant m_2(U_0)$
- $r_{L_2^{\text{set}}}(m_{\leqslant 1}(V)) \geqslant m_2(V)$ and $\#V > 0$

These are all handled by Lemma 4.2.2 where X equals W, U_0, and V respectively. Note that $m_{\leqslant 1}(X) \leqslant m_{\leqslant 1}(L_1^{\text{set}}) = m_{\leqslant 1}(L_2^{\text{set}}) - 1$, where equality follows from assumption (i), as needed in each case. Lemma 4.2.2 gives us exactly the result we desire.

Step 2: What remains is to show that there are no other cases. So, we will assume the following three conditions and arrive at a contradiction.

$$f_2 > r_{L_2^{\text{set}}}(1) \tag{4.5}$$

$$m_2(U_0) \geqslant r_{L_2^{\text{set}}}(m_{\leqslant 1}(U_0)) + 1 \tag{4.6}$$

$$m_2(V) \geqslant r_{L_2^{\text{set}}}(m_{\leqslant 1}(V)) + 1 \qquad \text{or} \qquad \#V = 0. \tag{4.7}$$

To reach the contradiction, we will work towards finding lower bounds for the quantities $m_2(V) \cdot m_{\leqslant 1}(L_2^{\text{set}})$ and $m_2(U_i) \cdot m_{\leqslant 1}(L_2^{\text{set}})$. We will then show that these are incompatible bounds.

We will want to be able to use Lemma 4.1.8 to find these lower bounds. Since L_1^{set} and L_2^{set} are disjoint and $x_0^{d_0} \in L_1^{\text{set}}$, we get $x_0^{d_0} \notin L_2^{\text{set}}$ and thus, L_2^{set} contains only monomials with degree larger than d_0.

Consider any $a = x_0^{a_0} x_1^{a_1} x_2^{a_2} \in L_2^{\text{set}}$. So, $a \notin L_1$ and $\deg a > d_0$. This means that $f x_2^{\deg a - d_0}$ has the same degree as a. Because $f x_2^{\deg a - d_0} \in L_1$, $f x_2^{\deg a - d_0} > a$. So, $f_0 \geqslant a_0$ and

$$\begin{aligned} a_1 + a_2 &= \deg a - a_0 \\ &\geqslant \deg\left(f x_2^{\deg a - d_0}\right) - f_0 \\ &\geqslant \deg f - f_0 \\ &\geqslant f_2. \end{aligned}$$

By Inequality (4.7),

$$a_1 + a_2 > r_{L_2^{\text{set}}}(1)$$

As a result, we can use Lemma 4.1.8 on L_2^{set} for the rest of this proof.

- Assuming $\#V > 0$, we can apply Inequality (4.7) and Lemma 4.1.8 to get

$$\begin{aligned} m_2(V) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &\geqslant r_{L_2^{\text{set}}}(m_{\leqslant 1}(V)) \cdot m_{\leqslant 1}(L_2^{\text{set}}) + m_{\leqslant 1}(L_2^{\text{set}}) \\ &\geqslant m_{\leqslant 1}(V) \cdot m_2(L_2^{\text{set}}) \end{aligned} \tag{4.8}$$

  If, however, $\#V = 0$, both sides of the inequality are zero, so this still holds true.

- From Inequality (4.6) and Lemma 4.1.8 we get

$$\begin{aligned} m_2(U_0) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &\geqslant r_{L_2^{\text{set}}}(m_{\leqslant 1}(U_0)) \cdot m_{\leqslant 1}(L_2^{\text{set}}) + m_{\leqslant 1}(L_2^{\text{set}}) \\ &\geqslant m_{\leqslant 1}(U_0) \cdot m_2(L_2^{\text{set}}) + m_{\leqslant 1}(L_2^{\text{set}}) \end{aligned} \tag{4.9}$$

- Inequality (4.3) of Corollary 4.1.2 shows $m_{\leqslant 1}(U_0)\left(m_{\leqslant 1}(U_0) - 1\right)/2 \geqslant m_2(U_0)$. This, along with Inequalities (4.4) and (4.9) gives

$$\begin{aligned} m_{\leqslant 1}(U_0) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &= m_{\leqslant 1}(L_2^{\text{set}}) + \frac{2m_{\leqslant 1}(L_2^{\text{set}})}{m_{\leqslant 1}(U_0)} \cdot \frac{m_{\leqslant 1}(U_0)\left(m_{\leqslant 1}(U_0) - 1\right)}{2} \\ &\geqslant m_{\leqslant 1}(L_2^{\text{set}}) + \frac{2m_{\leqslant 1}(L_2^{\text{set}})}{m_{\leqslant 1}(U_0)} \cdot m_2(U_0) \\ &\geqslant m_{\leqslant 1}(L_2^{\text{set}}) + \frac{2}{m_{\leqslant 1}(U_0)} \cdot m_{\leqslant 1}(U_0) \cdot m_2(L_2^{\text{set}}) \\ &= 2m_2(L_2^{\text{set}}) + m_{\leqslant 1}(L_2^{\text{set}}) \end{aligned} \tag{4.10}$$

- Equations (4.1) and (4.2) from Corollary 4.1.2 give $m_2(U_1) = m_2(U_0) + m_{\leqslant 1}(U_0)$ and $m_{\leqslant 1}(U_1) = m_{\leqslant 1}(U_1) = m_{\leqslant 1}(U_0) + 1$. Combined with Inequalities (4.9) and (4.10), we get

$$\begin{aligned} m_2(U_1) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &= m_2(U_0) \cdot m_{\leqslant 1}(L_2^{\text{set}}) + m_{\leqslant 1}(U_0) \cdot m_{\leqslant 1}(L_2^{\text{set}}) \\ &\geqslant m_{\leqslant 1}(U_0) \cdot m_2(L_2^{\text{set}}) + m_{\leqslant 1}(L_2^{\text{set}}) + 2m_2(L_2^{\text{set}}) + m_{\leqslant 1}(L_2^{\text{set}}) \\ &= (m_{\leqslant 1}(U_0) + 1) \cdot m_2(L_2^{\text{set}}) + m_2(L_2^{\text{set}}) + 2m_{\leqslant 1}(L_2^{\text{set}}) \end{aligned}$$

  Note that, by assumption (ii), $m_{\leqslant 1}(L_2^{\text{set}})$ is nonzero. This results in the strict inequality,

$$\begin{aligned} m_2(U_1) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &> (m_{\leqslant 1}(U_0) + 1) \cdot m_2(L_2^{\text{set}}) + m_2(L_2^{\text{set}}) \\ &= m_{\leqslant 1}(U_1) \cdot m_2(L_2^{\text{set}}) + m_2(L_2^{\text{set}}) \end{aligned} \tag{4.11}$$

- From Inequality (4.5) and Lemma 4.1.8 we get

$$f_2 \cdot m_{\leqslant 1}(L_2^{\text{set}}) > r_{L_2^{\text{set}}}(1) \cdot m_{\leqslant 1}(L_2^{\text{set}}) \geqslant m_2(L_2^{\text{set}}) \tag{4.12}$$

- Recall that each U_i can be written as the interval $[b, fx_2^i]$ for some b. We apply Corollary 4.1.5 to U_0 and U_1 to get that, for $i > 1$,

  $$m_2(U_i) \geqslant f_2 \cdot m_{\leqslant 1}(U_i) \qquad \text{or} \qquad m_{\leqslant 1}(U_0) \cdot m_2(U_i) \geqslant m_{\leqslant 1}(U_i) \cdot m_2(U_0)$$

  In the first case, Inequality (4.12) gives us that

  $$\begin{aligned} m_2(U_i) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &\geqslant f_2 \cdot m_{\leqslant 1}(U_i) \cdot m_{\leqslant 1}(L_2^{\text{set}}) \\ &\geqslant m_{\leqslant 1}(U_i) \cdot m_2(L_2^{\text{set}}), \end{aligned}$$

  and in the second case, Inequalities (4.4) and (4.9) give the same:

  $$\begin{aligned} m_2(U_i) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &= \frac{m_{\leqslant 1}(U_0) \cdot m_2(U_i) \cdot m_{\leqslant 1}(L_2^{\text{set}})}{m_{\leqslant 1}(U_0)} \\ &\geqslant \frac{m_{\leqslant 1}(U_i) \cdot m_2(U_0) \cdot m_{\leqslant 1}(L_2^{\text{set}})}{m_{\leqslant 1}(U_0)} \\ &\geqslant \frac{m_{\leqslant 1}(U_i) \cdot m_{\leqslant 1}(U_0) \cdot m_2(L_2^{\text{set}})}{m_{\leqslant 1}(U_0)} \\ &= m_{\leqslant 1}(U_i) \cdot m_2(L_2^{\text{set}}). \end{aligned} \tag{4.13}$$

In summary, our bounds are

$$\begin{aligned} m_2(V) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &\geqslant m_{\leqslant 1}(V) \cdot m_2(L_2^{\text{set}}) && \text{(4.8 restated)} \\ m_2(U_0) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &\geqslant m_{\leqslant 1}(U_0) \cdot m_2(L_2^{\text{set}}) + m_{\leqslant 1}(L_2^{\text{set}}) && \text{(4.9 restated)} \\ m_2(U_1) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &> m_{\leqslant 1}(U_1) \cdot m_2(L_2^{\text{set}}) + m_2(L_2^{\text{set}}) && \text{(4.11 restated)} \\ m_2(U_i) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &\geqslant m_{\leqslant 1}(U_i) \cdot m_2(L_2^{\text{set}}) && \text{(4.13 restated)} \end{aligned}$$

Summing these together gives us that

$$\begin{aligned} m_2(L_1^{\text{set}}) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &= \left(m_2(V) + m_2(U_0) + m_2(U_1) + \sum_{i>1} m_2(U_i) \right) \cdot m_{\leqslant 1}(L_2^{\text{set}}) \\ &> \left(m_{\leqslant 1}(V) + m_{\leqslant 1}(U_0) + m_{\leqslant 1}(U_1) + \sum_{i>1} m_{\leqslant 1}(U_i) \right) \cdot m_2(L_2^{\text{set}}) \\ &\quad + m_2(L_2^{\text{set}}) + m_{\leqslant 1}(L_2^{\text{set}}) \\ &= \big(m_{\leqslant 1}(L_1^{\text{set}}) + 1\big) \cdot m_2(L_2^{\text{set}}) + m_{\leqslant 1}(L_2^{\text{set}}). \end{aligned}$$

By assumption (i),

$$\begin{aligned} m_2(L_1^{\text{set}}) \cdot m_{\leqslant 1}(L_2^{\text{set}}) &> m_{\leqslant 1}(L_2^{\text{set}}) \cdot m_2(L_2^{\text{set}}) + m_{\leqslant 1}(L_2^{\text{set}}) \\ &= m_{\leqslant 1}(L_2^{\text{set}}) \cdot (m_2(L_2^{\text{set}}) + 1), \end{aligned}$$

and by assumption (i) again,

$$m_2(L_1^{\text{set}}) \cdot m_{\leqslant 1}(L_2^{\text{set}}) > m_{\leqslant 1}(L_2^{\text{set}}) \cdot m_2(L_1^{\text{set}}).$$

This is a contradiction. So, there are only 3 cases discussed above. □

Lemma 4.2.4. *Consider $L_1, L_2 \subset \mathcal{U}(I, c)$. Then, $\beta_q(L_1) \geqslant \beta_q(L_2)$ for all q if and only if both of the following hold:*

- $m_2(L_1^{\text{set}}) \geqslant m_2(L_2^{\text{set}})$ *and*
- $2m_2(L_1^{\text{set}}) + m_1(L_1^{\text{set}}) \geqslant 2m_2(L_2^{\text{set}}) + m_1(L_2^{\text{set}})$.

where $L_i^{\text{set}} = L_i - I$.

Furthermore, if $m_2(L_1^{\text{set}}) = m_2(L_2^{\text{set}})$, L_1 and L_2 have comparable total Betti numbers and if L_1 and L_2 have comparable total Betti numbers where $m_2(L_1^{\text{set}}) > m_2(L_2^{\text{set}})$, then $\beta_q(L_1) \geqslant \beta_q(L_2)$ for all q.

Proof. From Corollary 3.1.9 we have that

$$\begin{aligned}
\beta_2(L_i) &= \beta_2(I) - c + m_2(L_i^{\text{set}}) \\
\beta_1(L_i) &= \beta_1(I) - 3c + 2m_2(L_i^{\text{set}}) + m_{\geqslant 1}(L_i^{\text{set}}) \\
&= \beta_1(I) - 3c + 3m_2(L_i^{\text{set}}) + m_1(L_i^{\text{set}}) \\
\beta_0(L_i) &= \beta_0(I) - 3c + m_2(L_i^{\text{set}}) + m_{\geqslant 1}(L_i^{\text{set}}) + m_{\geqslant 0}(L_i^{\text{set}}) \\
&= \beta_0(I) - 2c + 2m_2(L_i^{\text{set}}) + m_1(L_i^{\text{set}})
\end{aligned}$$

Because $\#L_1 = c = \#L_2$, we have the following three conditions:

1. $\beta_2(L_1) \geqslant \beta_2(L_2) \Leftrightarrow m_2(L_1^{\text{set}}) \geqslant m_2(L_2^{\text{set}})$
2. $\beta_1(L_1) \geqslant \beta_1(L_2) \Leftrightarrow 3m_2(L_1^{\text{set}}) + m_1(L_1^{\text{set}}) \geqslant 3m_2(L_2^{\text{set}}) + m_1(L_2^{\text{set}})$
3. $\beta_0(L_1) \geqslant \beta_0(L_2) \Leftrightarrow 2m_2(L_1^{\text{set}}) + m_1(L_1^{\text{set}}) \geqslant 2m_2(L_2^{\text{set}}) + m_1(L_2^{\text{set}})$

The first and third imply the second. So, we are done. The additional statements also follow quickly from these three conditions. □

Lemma 4.2.5. *Let A, B, C be finite sets such that $A \cap B$ is empty, $\#A = \#C$ and $A \neq C \subset A \cup B$. Then, $\#(C \triangle B) < \#(A \triangle B)$ where $\triangle$ represents the symmetric difference.*

Proof. Because $\#A = \#C$ and $A \neq C$, there is some $a \in A$ such that $a \notin C$ and $a \notin B$. So, $a \notin C \cup B$. Thus,

$$C \triangle B \subseteq C \cup B \subsetneq A \cup B = A \triangle B.$$ □

Lemma 4.2.6. *Given lexsegment ideals $I \subset L_1, L_2 \subset R$ where $\#(L_1 - I) = \#(L_2 - I)$ is finite, there is a third lexsegment ideal, $L_3 \subset R$, such that $\#(L_3 - I) = \#(L_1 - I)$, $I \subset L_3$,*

- $L_3 \subset L_1 + L_2$, *and*
- $\beta_q(L_3) \geqslant \beta_q(L_i)$ *for all q and i.*

Remark: Recall the notation of Definition 4.2.1. We can rephrase the lemma to state that for $L_1, L_2 \in \mathcal{U}(I,c)$, we can find $L_3 \in \mathcal{U}(I,c)$ such that $L_3 \subset L_1 + L_2$ and $\beta_q(L_3) \geqslant \beta_q(L_i)$. This is what we will prove.

Proof. Set $c = \#L_1^{\text{set}} = \#L_2^{\text{set}}$. We use induction on the size of the symmetric difference $\#(L_1^{\text{set}} \triangle L_2^{\text{set}})$. When $\#(L_1^{\text{set}} \triangle L_2^{\text{set}}) = 0$, $L_1 = L_2$ and by setting $L_3 = L_1$, the statement is trivial.

It is enough to show the claim if $I = L_1 \cap L_2$. So, we can assume that $I = L_1 \cap L_2$. As a result we will assume that $L_1^{\text{set}} \cap L_2^{\text{set}}$ is empty.

Case 0: Suppose L_1 and L_2 have comparable Betti numbers. We can set L_3 to be the one with larger total Betti numbers, and we are done.

Case 1: Suppose $m_2(L_1^{\text{set}}) = m_2(L_2^{\text{set}})$. By Lemma 4.2.4, the Betti numbers of L_1 and L_2 are comparable and this reduces to Case 0.

Case 2: Suppose $m_2(L_1^{\text{set}}) = m_2(L_2^{\text{set}}) + 1$ and that L_1 and L_2 have incomparable total Betti numbers.

The idea will be to find L_5 such that the total Betti numbers of L_5 are not less than or equal to the total Betti numbers of L_1. Then, by induction, we will find L_4 such that the total Betti numbers of L_4 are greater than or equal to the total Betti numbers of L_5 and the total Betti numbers of L_5 are strictly greater than the total Betti numbers of L_2. Finally, by induction we can find L_3 such that the total Betti numbers of L_3 are greater than or equal to the total Betti numbers of L_1 and the total Betti numbers of L_4.

By Lemma 4.2.4, $m_1(L_1^{\text{set}}) + 2 < m_1(L_2^{\text{set}})$. Applying Lemma 4.2.3 to L_1, L_2, and $L_1 \cap L_2$ gives $L_5 \in \mathcal{U}(I,c)$ such that $L_1 \neq L_5 \subset L_1 + L_2$ and

$$m_2(L_5^{\text{set}}) \geqslant m_2(L_1^{\text{set}}).$$

By Lemma 4.2.5, $\#(L_5^{\text{set}} \triangle L_2^{\text{set}})$ is less than $\#(L_1^{\text{set}} \triangle L_2^{\text{set}})$ and induction gives some $L_4 \in \mathcal{U}(I,c)$ such that $L_4 \subset L_5 + L_2$ and

$$\beta_q(L_4) \geqslant \beta_q(L_5) \qquad \text{and} \qquad \beta_q(L_4) \geqslant \beta_q(L_2)$$

for all q. By Lemma 4.2.4,

$$m_2(L_4^{\text{set}}) \geqslant m_2(L_5^{\text{set}}) \geqslant m_2(L_1^{\text{set}}) = m_2(L_2^{\text{set}}) + 1.$$

So, $L_2 \neq L_4 \subset L_1 + L_2$. By Lemma 4.2.5, $\#(L_1^{\text{set}} \triangle L_4^{\text{set}})$ is less than $\#(L_1^{\text{set}} \triangle L_2^{\text{set}})$ and induction gives some $L_3 \in \mathcal{U}(I,c)$ such that $L_3 \subset L_1 + L_4 \subset L_1 + L_2$ and

$$\beta_q(L_3) \geqslant \beta_q(L_1) \qquad \text{and} \qquad \beta_q(L_3) \geqslant \beta_q(L_4) \geqslant \beta_q(L_2)$$

for all q, and we are done.

Case 3: Suppose $m_2(L_1^{\text{set}}) \geqslant m_2(L_2^{\text{set}}) + 2$.

The idea is that we can create a sequence of ideals between L_1 and L_2 that slowly increment m_2. We can then apply Case 2 to each step. This argument is similar to, but simpler than, the one in Case 2.

Let $M_0, \ldots, M_t \in \mathcal{U}(I,c)$ be a sequence of ideals such that $M_i \subset L_1 + L_2$, $M_0 = L_1$, $M_t = L_2$, and M_i^{set} and M_{i+1}^{set} differ by exactly 1 monomial. Thus, $m_2(M_i^{\text{set}})$ and $m_2(M_{i+1}^{\text{set}})$ differ by at most 1. So, for some M_j,

$$m_2(L_1^{\text{set}}) > m_2(M_j^{\text{set}}) > m_2(L_2^{\text{set}})$$

Therefore, $L_1 \neq M_j \subset L_1 \cup L_2$. By Lemma 4.2.5, $\#(M_j^{\text{set}} \triangle L_2^{\text{set}})$ is less than $\#(L_1^{\text{set}} \triangle L_2^{\text{set}})$ and induction gives some $L_4 \in \mathcal{U}(I,c)$ such that $L_4 \subset M_j \cup L_2$ and

$$\beta_q(L_4) \geqslant \beta_q(M_j) \qquad \text{and} \qquad \beta_q(L_4) \geqslant \beta_q(L_2)$$

for all q. By Lemma 4.2.4 and the choice of M_j, it follows that

$$m_2(L_4^{\text{set}}) \geqslant m_2(M_j^{\text{set}}) > m_2(L_2^{\text{set}}).$$

So, $L_2 \neq L_4 \subset L_1 \cup L_2$. By Lemma 4.2.5, $\#(L_1^{\text{set}} \triangle L_4^{\text{set}})$ is less than $\#(L_1^{\text{set}} \triangle L_2^{\text{set}})$ and induction gives some $L_3 \in \mathcal{U}(I,c)$ such that $L_3 \subset L_1 \cup L_4 \subset L_1 \cup L_2$ and

$$\beta_q(L_3) \geqslant \beta_q(L_1) \qquad \text{and} \qquad \beta_q(L_3) \geqslant \beta_q(L_4) \geqslant \beta_q(L_2)$$

for all q, as desired. □

We are now ready for the proof of Theorem 4.0.1. Recall that we wish to show that

$$\mathcal{I} := \left\{\text{saturated ideal } I \subset S \;\middle|\; p_{S/I} = p \text{ and } f(d) \geqslant \Delta h_{S/I}(d) \geqslant g(d) \text{ for all } d\right\}$$

contains an ideal with maximum total Betti numbers.

Proof of Theorem 4.0.1. Gotzmann's regularity bound [7] gives an upper bound for the regularity of a saturated ideal with a given Hilbert polynomial. For a proof, see e.g., Theorem 4.3.2 in [3]. Because there are finitely many possible values for the Hilbert function in each degree below this bound and the Hilbert function must match the Hilbert polynomial at or above this bound, there are only finitely many Hilbert functions satisfying the constraints. Thus, there are finitely many Hilbert functions of ideals in $\mathcal{I}$. Because there are finitely many Hilbert functions of ideals in $\mathcal{I}$, there are finitely many Hilbert functions of ideals in $\mathcal{I}^{(1)}$. Thus,

$$\begin{aligned}\mathcal{L} &:= \left\{\text{lexsegment ideal } L \subset R \;\middle|\; L \in \mathcal{I}^{(1)}\right\} \\ &= \left\{\text{lexsegment ideal } L \subset R \;\middle|\; \begin{array}{c} \Delta^{-1} h_{R/L}(d) = p(d) \text{ for } d \gg 0 \\ f(d) \geqslant h_{R/L}(d) \geqslant g(d) \text{ for all } d \end{array}\right\}\end{aligned}$$

is a finite family. By Corollary 3.3.3, we need only to find an ideal with maximum total Betti numbers in $\mathcal{L}$.

We assume that $\mathcal{I}$ and thus $\mathcal{L}$ is nonempty. Suppose, for contradiction, that there does not exist an ideal with maximum total Betti numbers. So, there exist two lexsegment ideals $L_1, L_2 \in \mathcal{L}$ with maximal and incomparable total Betti numbers. Set $I = L_1 \cap L_2$.

Recall that $\Delta^{-1}h(d) = \sum_{i=0}^{d} h(i)$ and thus, for a lexsegment ideal, $L \subset R$, $\#\operatorname{Mon}[L]_{\leqslant d} = \Delta^{-1}h_{R/L}(d)$.

Because there is some integer D such that $\Delta^{-1}h_{R/L_1}(d) = \Delta^{-1}h_{R/L_2}(d)$ for $d \geqslant D$, we have that $h_{R/L_1}(d) = h_{R/L_2}(d) = h_{R/I}(d)$ for $d > D$. Thus,

$$\#(L_1 - I) = \Delta^{-1}h_{R/L_1}(d) - \Delta^{-1}h_{R/I}(d) = \#(L_2 - I)$$

is finite for all $d > D$. By Lemma 4.2.6, there exists a lexsegment $L_3 \in R$ such that $L_1 \cap L_2 \subset L_3 \subset L_1 + L_2$, $\#(L_3 - I) = \#(L_1 - I)$, and $\beta_q(L_3) \geqslant \beta(L_i)$ for all q and i.

Thus,

$$f(d) \geqslant \max\{h_{R/L_1}(d), h_{R/L_2}(d)\} \geqslant h_{R/L_3}(d) \geqslant \min\{h_{R/L_1}(d), h_{R/L_2}(d)\} \geqslant g(d)$$

for all d and

$$\Delta^{-1}h_{R/L_3}(d) - \Delta^{-1}h_{R/I}(d) = \Delta^{-1}h_{R/L_1}(d) - \Delta^{-1}h_{R/I}(d)$$

for $d > D$ giving that $\Delta^{-1}h_{R/L_3}(d) = p(d)$ for $d \geqslant D$. So, $L_3 \in \mathcal{L}$ and has total Betti numbers that are at least as big as those of L_1 and L_2. This is a contradiction since L_1 and L_2 were assumed to have maximal but incomparable total Betti numbers.

Thus, there exists an ideal with maximum total Betti numbers in $\mathcal{I}$. □

Chapter 5 Algorithm for Computing Maximal Betti Numbers

So far, we have discussed the existence of ideals with maximum total Betti numbers in a Hilbert family. However, one may wish to actually compute bounds on the total Betti numbers or find ideals with maximal total Betti numbers.

The naive method of doing this is to enumerate all Hilbert functions that satisfy the conditions of a Hilbert family, $\mathcal{I}$, and compute the total Betti numbers for the lexsegment ideals in $\mathcal{I}^{(1)}$, using Corollary 3.3.3. This can be very computationally expensive because there can be a large number of Hilbert functions. When the Hilbert family is defined by bounds on the Hilbert function, as in Question 3.4.3, we describe an alternative, incremental approach of increasing a "cutoff" degree and storing the "best" ideals at each degree. As a result, we do not need to enumerate all Hilbert functions, leading to a significant improvement in the computational complexity.

5.1 A Recursion

The foundation of the algorithm is the group of recursions in Lemma 5.1.3 and Corollary 5.1.5. These recursions are an efficient way to compute the structures in Definitions 5.1.1 and 5.1.4 which, in turn, allow us to achieve the desired result via Lemma 5.1.6.

This first structure is analogous to the collection of Hilbert functions satisfying given bounds, and as a result, to a Hilbert family.

Definition 5.1.1. Consider the numeric functions F, G, f, g such that $F(d) = G(d)$ for $d \geqslant D$ where D is some integer. Define the following sets of ordered tuples. Let

$$\underset{F,G,f,g}{\mathcal{H}}(d) := \left\{ (\ell_0, \ldots, \ell_d) \in \mathbb{N}_0^{d+1} \;\middle|\; \begin{array}{c} F(i) \geqslant \sum_{k=0}^{i} \ell_k \geqslant G(i) \\ f(i) \geqslant \ell_i \geqslant g(i) \end{array} \text{ for all } 0 \leqslant i \leqslant d \right\}$$

$$\underset{F,G,f,g}{\mathcal{H}}(d, c) := \left\{ (\ell_0, \ldots, \ell_d) \in \underset{F,G,f,g}{\mathcal{H}}(d) \;\middle|\; \sum_{i=0}^{d} \ell_i = c \right\}$$

$$\underset{F,G,f,g}{\mathcal{H}}(d, c, k) := \left\{ (\ell_0, \ldots, \ell_d) \in \underset{F,G,f,g}{\mathcal{H}}(d, c) \;\middle|\; \begin{array}{l} \ell_{i-1} \geqslant \ell_i{}^{-\langle i \rangle} \text{ for all } 0 < i \leqslant d \\ \ell_d \geqslant k \text{ if } 0 \leqslant d \end{array} \right\}$$

Definition 5.1.2. Given a set of ordered d-tuples, $M \subset \mathbb{N}_0^d$, and a nonzero integer, $j \in \mathbb{N}_0$, let

$$M \times j := \{ (\ell_0, \ldots, \ell_d, j) \mid (\ell_0, \ldots, \ell_d) \in M \}$$

denote the set of tuples where j is appended to each tuple in M.

Lemma 5.1.3. *Consider the numeric functions F, G, f, g such that $F(d) = G(d)$ for $d \geqslant D$ where D is some integer. Then, we have the following two recursions for $\mathcal{H}$,*

where we drop the subscripts F, G, f, g to simplify notation.

$$\mathcal{H}(d,c) = \bigcup_{j=g(d)}^{f(d)} \mathcal{H}(d-1, c-j) \times j$$
$$\mathcal{H}(d,c,k) = \left(\mathcal{H}(d-1, c-k, k^{-\langle d\rangle}) \times k\right) \cup \mathcal{H}(d,c,k+1)$$

for all d, c, and k. Additionally, $\mathcal{H}(-1,0,k) = \mathcal{H}(-1,0) = \{()\}$ for all k, and $\mathcal{H}(-1,c,k) = \mathcal{H}(-1,c) = \varnothing$ for all k and all $c \neq 0$.

Proof. The first equality is true because, by Definition 5.1.2, any $(\ell_0, \ldots, \ell_d) \in \mathcal{H}(d,c)$ satisfies $g(d) \leqslant \ell_d \leqslant f(d)$ and $\ell_d \leqslant c$.

The second claim is a consequence of the fact that $(\ell_0, \ldots, \ell_d) \in \mathcal{H}(d,c,k)$ implies $\ell_d \geqslant k$ and $\ell_{d-1} \geqslant {\ell_d}^{-\langle d\rangle} \geqslant k^{-\langle d\rangle}$. □

This second structure is analogous to the collection of total Betti numbers in a Hilbert family.

Definition 5.1.4. Given some degree d and $\ell \in \mathcal{N}_0$, let $[x_0^d, a_{\ell,d}] \subset \mathrm{Mon}[R]_d$ be the lexicographically largest interval with length ℓ. In other words, $a_{\ell,d}$ is a monomial such that $\#[x_0^d, a_{\ell,d}] = \ell$. Define the following function for $0 \leqslant q \leqslant n$:

$$V_q[d,\ell] := \sum_{a \in [x_0^d, a_{\ell,d}]} \left(\binom{n+1}{q+1} - \binom{\operatorname{maxi} a}{q+1} \right)$$
$$\underset{F,G,f,g}{maxV_q}(d,c) := \max\left\{ \sum_{i=0}^{d} V_q[i,\ell_i] \;\middle|\; (\ell_0,\ldots,\ell_d) \in \underset{F,G,f,g}{\mathcal{H}}(d,c) \right\}$$
$$\underset{F,G,f,g}{maxV_q}(d,c,k) := \max\left\{ \sum_{i=0}^{d} V_q[i,\ell_i] \;\middle|\; (\ell_0,\ldots,\ell_d) \in \underset{F,G,f,g}{\mathcal{H}}(d,c,k) \right\}$$

where we define $maxV_q$ to be 0 if $\mathcal{H}$ is empty.

Corollary 5.1.5. *Consider the numeric functions F, G, f, g such that $F(d) = G(d)$ for $d \geqslant D$ where D is some integer. Then, we have the following two recursions for $maxV_q$:*

$$maxV_q(d,c) = \max\left\{ maxV_q(d-1, c-j) + V_q[d,j] \mid f(d) \geqslant j \geqslant g(d) \right\}$$
$$maxV_q(d,c,k) = \max\left\{ maxV_q(d-1, c-k, k^{-\langle d\rangle}) + V_q[d,k], maxV_q(d,c,k+1) \right\}$$

for all d, c, and k. Additionally, $maxV_q(-1,c,k) = maxV_q(-1,c) = 0$ for all c and all k.

Proof. This is an immediate consequence of Lemma 5.1.3 and Definition 5.1.4. □

We now can bring these concepts together to get a result that is the foundation to the algorithms we will describe.

Lemma 5.1.6. *Consider the numeric functions F, G, f, g such that $F(d) = G(d)$ for $d \geqslant D$ where D is some integer. Consider also the following 1-depth Hilbert family of ideals,*

$$\mathcal{I} = \left\{ \text{saturated ideal } I \subset S \;\middle|\; \begin{array}{l} F(d) \geqslant h_{S/I}(d) \geqslant G(d), \\ f(d) \geqslant \Delta h_{S/I}(d) \geqslant g(d) \end{array} \text{ for all } d \right\}.$$

Suppose $\mathcal{I}$ is nonempty and let J be the sum of all lexsegment ideals in $\mathcal{I}^{(1)}$. Then, for any integer $q \geqslant 0$, one has,

$$\max\left\{ \beta_q^S(I) \;\middle|\; I \in \mathcal{I} \right\} = \underset{F,G,f,g}{maxV_q}(D, G(D), 0) + \left(\beta_q^S(JS) - \sum_{d=0}^{D} V_q[d, h_{S^{(1)}/J}(d)] \right).$$

Furthermore, if $I \in \mathcal{I}$ is an ideal that achieves this maximum β_q, then the corresponding tuple $(\Delta h_{S/I}(0), \ldots, \Delta h_{S/I}(D)) \in \mathcal{H}$ achieves this $maxV_q$ value.
Finally, if we remove the upper bound, F, by specifying that $F(d) = h_S(d)$ for $d < D$, we have that

$$\underset{F,G,f,g}{maxV_q}(D, G(D), 0) = \underset{F,G,f,g}{maxV_q}(D, G(D)).$$

Remark: as previously discussed, see e.g. Theorem 3.1.4, there is a bijection between Hilbert functions and lexsegment ideals. In this case, because the Hilbert function is fixed in degrees higher than D, the lexsegment ideals we are searching are also in bijection with the Hilbert function truncated up to degree D. This is precisely what the tuples in Definition 5.1.1 describe. In this proof, we will move somewhat fluidly between these three concepts.

Proof. Note that the conditions on $\mathcal{H}(D, G(D), 0)$ are identical to the constraints on the Hilbert function of an ideal in $\mathcal{I}^{(1)}$. So, $\mathcal{H}(D, G(D), 0)$ is the collection of all Hilbert functions that we need to search.

Consider any lexsegment ideal $L \in \mathcal{I}^{(1)}$. Because $F(d) = G(d)$ for $d \geqslant D$, $h_{S^{(1)}/J}(d) = h_{S^{(1)}/L}(d)$ for all $d > D$. So, $[J]_{>D} = [L]_{>D}$ and thus, $J - L = [J - L]_{\leqslant D}$ is finite.

By Corollary 3.1.9, we have

$$\begin{aligned}
\beta_q^{S^{(1)}}(L) &= \beta_q^{S^{(1)}}(J) + \#[J-L]_{\leqslant D}\binom{n+1}{q+1} - \sum_{i=q}^{n} m_{\geqslant i}[J-L]_{\leqslant D}\binom{i}{q} \\
&= \beta_q^{S^{(1)}}(J) + \#[J]_{\leqslant D}\binom{n+1}{q+1} - \sum_{i=q}^{n} m_{\geqslant i}[J]_{\leqslant D}\binom{i}{q} \\
&\quad - \#[L]_{\leqslant D}\binom{n+1}{q+1} + \sum_{i=q}^{n} m_{\geqslant i}[L]_{\leqslant D}\binom{i}{q} \\
&= \beta_q^{S^{(1)}}(J) - \sum_{d=0}^{D}\sum_{a\in[J]_d}\left(\binom{n+1}{q+1} - \binom{\operatorname{maxi} a}{q+1}\right) \\
&\quad + \sum_{d=0}^{D}\sum_{a\in[L]_d}\left(\binom{n+1}{q+1} - \binom{\operatorname{maxi} a}{q+1}\right) \\
&= \beta_q^{S^{(1)}}(J) - \sum_{d=0}^{D} V_q[d, h_{S^{(1)}/J}(d)] + \sum_{d=0}^{D} V_q[d, h_{S^{(1)}/L}(d)]
\end{aligned}$$

By Corollary 3.3.3,

$$\max\left\{\beta_q^S(I)\,\middle|\, I\in\mathcal{I}\right\} = \max\left\{\beta_q^{S^{(1)}}(L)\,\middle|\, \text{lexsegment ideal } L\in\mathcal{I}^{(1)}\right\}.$$

Combining this with the previous equation gives the desired maximum. Furthermore, by Corollary 3.3.3, for an ideal that attains a maximum, there is some LS that shares the same Hilbert function. The corresponding tuple is $(h_{S^{(1)}/L}(0),\dots,h_{S^{(1)}/L}(D))$. Because $h_{S^{(1)}/L} = \Delta h_{S/LS}$, this is the desired tuple.

Given a tuple $(\ell_0,\dots,\ell_d) \in \mathcal{H}(D,G(D))$, by Proposition 3.3.5, there is some $(\ell'_0,\dots,\ell'_d)\in\mathbb{N}_0^{d+1}$ such that

$$\begin{gathered}
f(i) \geqslant \ell'_i \geqslant g(i) \text{ for all } 0\leqslant i\leqslant D, \\
l_{i-1} \geqslant {\ell_i}^{-\langle i\rangle} \text{ for all } 0<i\leqslant D, \\
\sum_{k=0}^{i}\ell_k \geqslant G(i) \text{ for all } 0\leqslant i\leqslant D,
\end{gathered}$$

and

$$\sum_{i=0}^{D} V_q[i,\ell'_i] \geqslant \sum_{i=0}^{D} V_q[i,\ell_i] \text{ for all } q.$$

If $F(d) = h_S(d)$ for $d<D$, $(\ell'_0,\dots,\ell'_d)\in\mathcal{H}(D,G(D),0)$ and thus,

$$\max_{F,G,f,g} V_q(D,G(D),0) = \max_{F,G,f,g} V_q(D,G(D)). \qquad \square$$

5.2 Algorithms

Because of Lemma 5.1.6, to find maximums for β_q, we simply need to find $maxV_q$. We can achieve this by using the recursion of Corollary 5.1.5. This is implemented in the method `maxBettiNumbers`.

If we put no upper bound on $h_{S/I}$, i.e. when $F(d) = h_S(d)$ for $d < D$, Algorithm 5.1 is an algorithm that can be used to find the maximum qth total Betti number. This is the algorithm used when the method `maxBettiNumbers` is given the option `Algorithm => "Simplified"`. This is also the default option for the method if no upper bound is specified.

If we have any other upper bound $h_{S/I}$, we can instead use Algorithm 5.2. Even though the set we are searching is smaller, this algorithm is slower because we must store the ideals with maximal total Betti numbers for each triple (d, c, k), while in Algorithm 5.1 we only need to store the ideals with maximal total Betti numbers for each pair (d, c). Algorithm 5.2 is the algorithm used when the method `maxBettiNumbers` has the option `Algorithm => "Complete"`. If not specified, this algorithm will be run by default when an upper bound for $h_{S/I}$ is given.

Both Algorithms 5.1 and 5.2 will give the maximum value of β_q. We can easily modify these algorithms to run on all possible q simultaneously, which would give upper bounds for the total Betti numbers. This is implemented in the package with the option `ResultsCount => "None"`, which is the default.

There are two possible ways to find Hilbert functions of ideals with maximal total Betti numbers. The most straightforward way is to find the functions that maximize $\sum_q \beta_q$, the sum of the total Betti numbers. All functions that maximize $\sum_q \beta_q$ necessarily have maximal total Betti numbers. We can further modify Algorithms 5.1 and 5.2 by making $maxHF$ track this sum. The downside of this method is that not all functions with maximal total Betti numbers are discovered. This modification is implemented in the package with the option `ResultsCount => "One"` or `ResultsCount => "AllMaxBettiSum"`.

The second way involves using the $(n + 1)$-tuple $(V_0, \ldots, V_n)$ in the place of V_q. These tuples form a partially ordered set with maximal elements and possibly no maximum. The algorithms remain essentially the same, however, more care is needed when comparing V_{new} and V_{best} as well as when taking the union of HF_{new} and HF_{best}. This method, while slower, allows us to find all ideals with maximal total Betti numbers. This modification is implemented in the package with the option `ResultsCount => "All"`.

Algorithm 5.1: Algorithm to maximize β_q with no upper bound for $h_{S/I}$.

Initialize $maxV(-1, 0) \leftarrow 0$;
Initialize $maxHF(-1, 0) \leftarrow \{()\}$;
for d **from** 0 **to** D **do**
  for j **from** $g(d)$ **to** $f(d)$ **do** Precompute $V_q[d, j]$;
  for c **from** $G(d)$ **to** $F(d)$ **do**
    Set $V_{best} \leftarrow 0$;
    Set $HF_{best} \leftarrow \varnothing$;
    for j **from** $g(d)$ **to** $f(d)$ **do**
      Compute $V_{new} \leftarrow maxV(d-1, c-j) + V_q[d, j]$;
      Compute $HF_{new} \leftarrow maxHF(d-1, c-j) \times j$;
      if $V_{best} < V_{new}$ **then**
        Set $V_{best} \leftarrow V_{new}$;
        Set $HF_{best} \leftarrow HF_{new}$;
      else if $V_{best} = V_{new}$ **then**
        Set $HF_{best} \leftarrow HF_{new} \cup HF_{best}$;
    end
    Set $maxV(d, c) \leftarrow V_{best}$;
    Set $maxHF(d, c) \leftarrow HF_{best}$;
  end
end
return $maxV(D, G(D))$, $maxHF(D, G(D))$

Algorithm 5.2: Algorithm to maximize β_q with any $h_{S/I}$.

Initialize $maxV(-1, 0, 0) \leftarrow 0$;
Initialize $maxHF(-1, 0, 0) \leftarrow \{()\}$;
for d **from** 0 **to** D **do**
  for j **from** $g(d)$ **to** $f(d)$ **do** Precompute $V_q[d, j]$;
  for c **from** $G(d)$ **to** $F(d)$ **do**
    Set $V_{best} \leftarrow 0$;
    Set $HF_{best} \leftarrow \varnothing$;
    for j **from** $f(d)$ **to** $g(d)$ **by** -1 **do**
      Compute $V_{new} \leftarrow maxV(d-1, c-j, j^{-\langle d \rangle}) + V_q[d, j]$;
      Compute $HF_{new} \leftarrow maxHF(d-1, c-j, j^{-\langle d \rangle}) \times j$;
      if $V_{best} < V_{new}$ **then**
        Set $V_{best} \leftarrow V_{new}$;
        Set $HF_{best} \leftarrow HF_{new}$;
      else if $V_{best} = V_{new}$ **then**
        Set $HF_{best} \leftarrow HF_{new} \cup HF_{best}$;
      end
      Set $maxV(d, c, j) \leftarrow V_{best}$;
      Set $maxHF(d, c, j) \leftarrow HF_{best}$;
    end
  end
end
return $maxV(D, G(D), 0)$, $maxHF(D, G(D), 0)$

In this chapter, we will give several examples illustrating the results in some of the other chapters. In particular, Section 6.1 gives examples of the Macaulay2 implementation of the algorithms from Chapter 5. Section 6.2 gives several counterexamples to possible extensions of the some of the results in Chapter 3.

6.1 Macaulay2 Examples

We will consider an example where S is the polynomial ring in 5 variables ($n = 3$). This example has only maximal total Betti numbers, and not maximum total Betti numbers. Also, Algorithms 5.1 and 5.2 give different results. Both of these situations are somewhat unusual, but give an illuminating example. For this example, we will choose the following constraints:

$$\begin{array}{lll} h_{S/I}(6) = 41 & \Delta h_{S/I}(3) \geqslant 8 & \Delta h_{S/I}(5) \geqslant 5 \\ h_{S/I}(d) = 49 \text{ for } d \gg 0 & \Delta h_{S/I}(4) \geqslant 8 & \Delta h_{S/I}(6) \geqslant 5 \end{array}$$

Since we will be using these constraints in several examples, we will first define a few variables to reduce repetition. By leaving certain degrees empty, we request the default, trivial constraints on those degrees.

```
i1 : N = 5;
i2 : g = HilbertDifferenceLowerBound => {,,,8,8,5,5};
i3 : G = HilbertFunctionLowerBound => {,,,,,,41};
i4 : F = HilbertFunctionUpperBound => {,,,,,,41};
i5 : p = HilbertPolynomial => 49;
```

We can use `maxBettiNumbers` to find that $(23, 54, 47, 14)$ is the upper bound for, $(\beta_0(I), \ldots, \beta_3(I))$, the vector of total Betti numbers of any saturated ideal, I, satisfying these constraints. Additionally, the maximum for the sum of the total Betti numbers is 137. Because $23 + 54 + 47 + 14 = 138$, there is no single ideal with total Betti numbers of $(23, 54, 47, 14)$. This is indicated by the value of `isRealizable`.

```
i6 : maxBettiNumbers(N,p,g,G,F)
o6 = MaxBetti{BettiUpperBound => {23, 54, 47, 14}}
              isRealizable => false
              MaximumBettiSum => 137
```

If we want the Hilbert function of an ideal with maximal total Betti numbers, we can pass `ResultsCount=>"One"` as an option. This gives an ideal that realizes the maximum for the sum of the total Betti numbers.

```
i7 : maxBettiNumbers(N,p,g,G,F, ResultsCount=>"One")
o7 = MaxBetti{BettiUpperBound => {23, 54, 47, 14}                        }
              HilbertFunctions => {{1, 5, 11, 21, 30, 36, 41, 46, 49, 49}}
              isRealizable => false
              MaximumBettiSum => 137
```

If we want the Hilbert function of all ideals that realize the maximum sum of the total Betti numbers, we can pass `ResultsCount=>"AllMaxBettiSum"` as an option.

```
i8 : maxBettiNumbers(N,p,g,G,F, ResultsCount=>"AllMaxBettiSum")
o8 = MaxBetti{BettiUpperBound => {23, 54, 47, 14}                                    }
              HilbertFunctions => {{1, 5, 11, 21, 30, 36, 41, 42, 43, 44, 45, 46, 47, 48, 49, 49}}
                                  {{1, 5, 11, 21, 30, 36, 41, 43, 44, 45, 46, 47, 48, 49, 49}   }
                                   ------------------------14 others-------------------------
                                  {{1, 5, 11, 21, 30, 36, 41, 45, 49, 49}                      }
                                  {{1, 5, 11, 21, 30, 36, 41, 46, 49, 49}                      }
              isRealizable => false
              MaximumBettiSum => 137
```

Finally, if we want the Hilbert function of all ideals that have maximal total Betti numbers, we can pass `ResultsCount=>"All"` as an option. In this case, the maximal total Betti numbers are $(23, 54, 45, 13)$ and $(22, 54, 47, 14)$.

```
i9 : maxBettiNumbers(N,p,g,G,F, ResultsCount=>"All")
o9 = MaxBetti{BettiUpperBound => {23, 54, 47, 14}                                    }
              HilbertFunctions => {{1, 5, 15, 23, 31, 36, 41, 42, 43, 44, 45, 46, 47, 48, 49, 49}}
                                  {{1, 5, 11, 21, 30, 36, 41, 42, 43, 44, 45, 46, 47, 48, 49, 49}}
                                   ------------------------32 others-------------------------
                                  {{1, 5, 15, 23, 31, 36, 41, 46, 49, 49}                      }
                                  {{1, 5, 11, 21, 30, 36, 41, 46, 49, 49}                      }
              isRealizable => false
              MaximalBettiNumbers => {{23, 54, 45, 13}}
                                     {{22, 54, 47, 14}}
              MaximumBettiSum => 137
```

The package `MaxBettiNumbers` also provides methods that can be useful for dealing with Hilbert functions, notably `almostLexBetti` and `almostLexIdeal`, which give, respectively, the Betti table and saturated ideal with largest graded Betti numbers associated with a given Hilbert function. These match, as expected, with `MaximalBettiNumbers` above.

```
i10 : almostLexBetti(N, last o9.HilbertFunctions)
            0  1  2  3  4
o10 = total: 1 22 54 47 14
          0: 1  .  .  .  .
          1: .  4  6  4  1
          2: .  .  .  .  .
          3: .  6 14 11  3
          4: .  5 14 13  4
          5: .  2  5  4  1
          6: .  .  .  .  .
          7: .  2  6  6  2
          8: .  3  9  9  3
i11 : almostLexIdeal(QQ[x_1..x_N], last o9.HilbertFunctions)
             2                   4   3    3    2 2   2      2 2    4
o11 = ideal (x , x x , x x , x x , x , x x , x x , x x , x x x , x x , x x ,
             1   1 2   1 3   1 4   2   2 3   2 4   2 3   2 3 4   2 4   2 3
        3      2 2      3    4   6   5    4 4   3 5   2 7    8   9
     x x x , x x x , x x x , x x , x , x x , x x , x x , x x , x x , x )
      2 3 4   2 3 4   2 3 4   2 4   3   3 4   3 4   3 4   3 4   3 4   4
```

Because we are setting an upper bound of $41 \geqslant h_{S/I}(6)$, we cannot use the algorithm described in Algorithm 5.1, but must use the algorithm described in Algorithm 5.2. The method `maxBettiNumbers` automatically chooses the appropriate algorithm, but we can force it to use a different one if we wish. In this case, if we specify `Algorithm=>"Simplified"`, we get an upper bound that is, necessarily, larger. Additionally, we are given a Hilbert function that appears to be valid, but is not the Hilbert function of any saturated ideal.

```
i12 : maxBettiNumbers(N,p,g,G,F, Algorithm=>"Simplified", ResultsCount=>"One")
o12 = MaxBetti{BettiUpperBound => {24, 57, 50, 15}                    }
               HilbertFunctions => {{1, 5, 11, 21, 30, 36, 41, 49, 49}}
               isRealizable => false
               MaximumBettiSum => 145
```

We can compare the speed of Algorithm 5.1 and Algorithm 5.2 with an example of fixing the Hilbert polynomial to be $3d^2 - 6d + 175$ in a ring with 6 variables. Because there is no upper bound for $h_{S/I}$, both algorithms give valid results, which we do not display.

```
i13 : N = 6;
i14 : QQ[i]; p = HilbertPolynomial => 3*i^2-6*i+175;
i16 : time maxBettiNumbers(N, p, Algorithm=>"Simplified", ResultsCount=>"None");
     -- used 5.82732 seconds
i17 : time maxBettiNumbers(N, p, Algorithm=>"Simplified", ResultsCount=>"All");
     -- used 7.40666 seconds
i18 : time maxBettiNumbers(N, p, Algorithm=>"Complete", ResultsCount=>"None");
     -- used 21.4503 seconds
i19 : time maxBettiNumbers(N, p, Algorithm=>"Complete", ResultsCount=>"All");
     -- used 28.4778 seconds
```

One alternative to these algorithms is a function such as `stronglyStableIdeals` in the package `StronglyStableIdeals`. This method generates all strongly stable ideals with a specified Hilbert polynomial. It is known that strongly stable ideals have the largest graded Betti numbers. We could then select the ideals that satisfy the constraints, compute the total Betti numbers, and then find those with maximal total Betti numbers. We will use `stronglyStableIdeals` as a benchmark for comparing speeds with existing methods. Although `stronglyStableIdeals` only generates the ideals, we are not adding to its timing the additional time it would take to compute total Betti numbers and find maximums. As a result the actual speed differences are even more pronounced. We will time examples with a constant Hilbert polynomial.

```
i20 : loadPackage "StronglyStableIdeals";
i21 : benchmark("maxBettiNumbers(5, HilbertPolynomial => 25)")
o21 = .0894562386249997
i22 : benchmark("stronglyStableIdeals(25, 5)")
o22 = 61.856958304
```

To further illustrate this difference, Figure 6.1 displays a plot of the runtime for the constant Hilbert polynomials below 100.

6.2 Counterexamples

This first example gives a simple Hilbert family that contains only two lexsegment ideals that have incomparable Betti numbers. As a result, this family does not have maximum or unbounded total Betti numbers.

Example 6.2.1. Let $S = K[x_0, \ldots, x_2]$. Consider the functions $h_1, h_2 : \mathbb{N} \to \mathbb{Z}$ defined by

$$h_1(d) = \begin{cases} 1 & \text{if } d = 0 \\ 6 & \text{if } d = 2 \\ 3 & \text{otherwise} \end{cases} \qquad h_2(d) = \begin{cases} d+1 & \text{if } d \leqslant 3 \\ 0 & \text{otherwise} \end{cases}$$

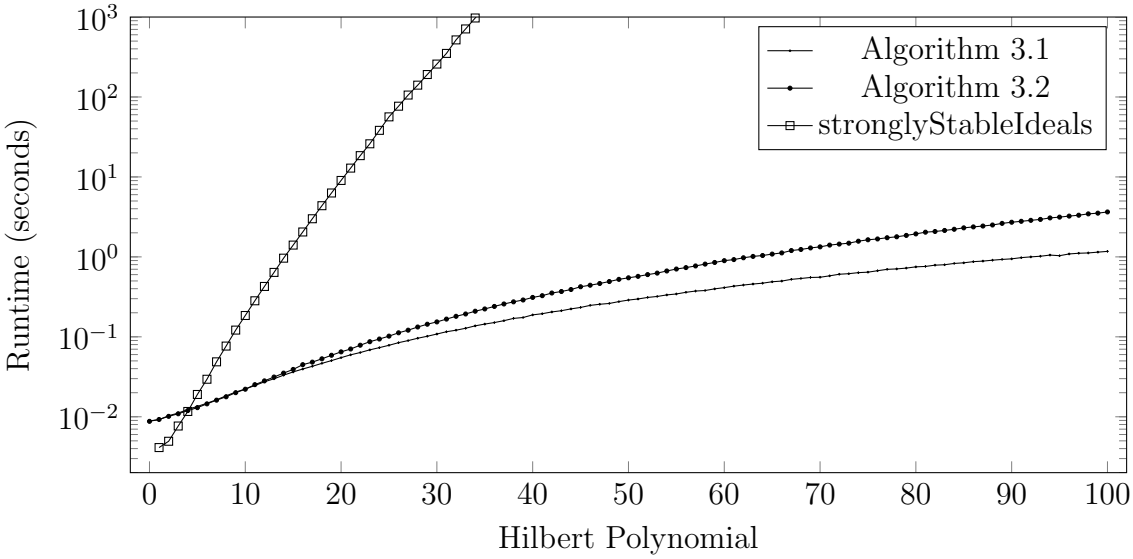

Figure 6.1: Polynomial vs Runtime for Different Algorithms (Log Plot)

and the following Hilbert family of ideals in S,

$$\mathcal{I} = \left\{\text{ideal } I \subset S \;\middle|\; h_{S/I} = h_1 \text{ or } h_{S/I} = h_2 \right\}.$$

The two lexsegment ideals in $\mathcal{I}$ are

$$L_{h_1} = x_0\langle x_0, x_1, x_2\rangle^2 + x_1^3 \qquad\qquad L_{h_2} = \langle x_0, x_1, x_2\rangle^4 + x_0$$

and their total Betti numbers are

$$\beta(L_{h_1}) = (7, 9, 3) \qquad\qquad \beta(L_{h_2}) = (6, 9, 4)$$

As a result L_{h_1} and L_{h_2} have maximal total Betti numbers in $\mathcal{I}$, and there is no ideal with maximum total Betti numbers in $\mathcal{I}$.

This example can be extended to the following D-depth Hilbert family in the ring with $D + 3$ variables, $R = K[x_0, \ldots, x_{D+2}]$.

$$\mathcal{J} = \left\{\text{ideal } I \subset R \;\middle|\; \Delta^D h_{S/I} = h_1 \text{ or } \Delta^D h_{S/I} = h_2 \right\}$$

This result follows from use of Corollary 3.3.3 and noticing that $\mathcal{J}^{(D)} = \mathcal{I}$.

This next example is an example of how Theorem 3.4.2 cannot be extended to a 2-depth Hilbert family.

Example 6.2.2. Let $S = K[x_0, \ldots, x_4]$. Consider the polynomial

$$p(d) = \frac{7}{2}d^2 + \frac{11}{2}d - 120$$

and the following 2-depth Hilbert family of ideals in S,

$$\mathcal{I} = \left\{\text{ideal } I \subset S \;\middle|\; \operatorname{depth} I \geqslant 2 \text{ and } p_{S/I} = p \right\}$$

Note, $S^{(2)} = K[x_0, x_1, x_2]$. The only three lexsegment ideals in $\mathcal{I}^{(2)}$ are

$$\begin{aligned}
A &= \left\langle x_0,\, x_1^{12},\, x_1^{11}x_2,\, x_1^{10}x_2^2,\, x_1^9x_2^3,\, x_1^8x_2^4,\, x_1^7x_2^6 \right\rangle,\\
B &= \left\langle x_0^3,\, x_0^2x_1,\, x_0^2x_2,\, x_0x_1^2,\, x_0x_1x_2,\, x_0x_2^2,\, x_1^8,\, x_1^7x_2^{12} \right\rangle,\\
C &= \left\langle x_0^2,\, x_0x_1,\, x_0x_2,\, x_1^{10},\, x_1^9x_2^2,\, x_1^8x_2^4,\, x_1^7x_2^9 \right\rangle.
\end{aligned}$$

Their total Betti numbers are

$$\begin{aligned}
\beta(A) &= (7, 11, 5),\\
\beta(B) &= (8, 11, 4),\\
\beta(C) &= (7, 10, 4).
\end{aligned}$$

As a result, A and B have maximal total Betti numbers in $\mathcal{I}$, and there is no ideal with maximum total Betti numbers in $\mathcal{I}$.

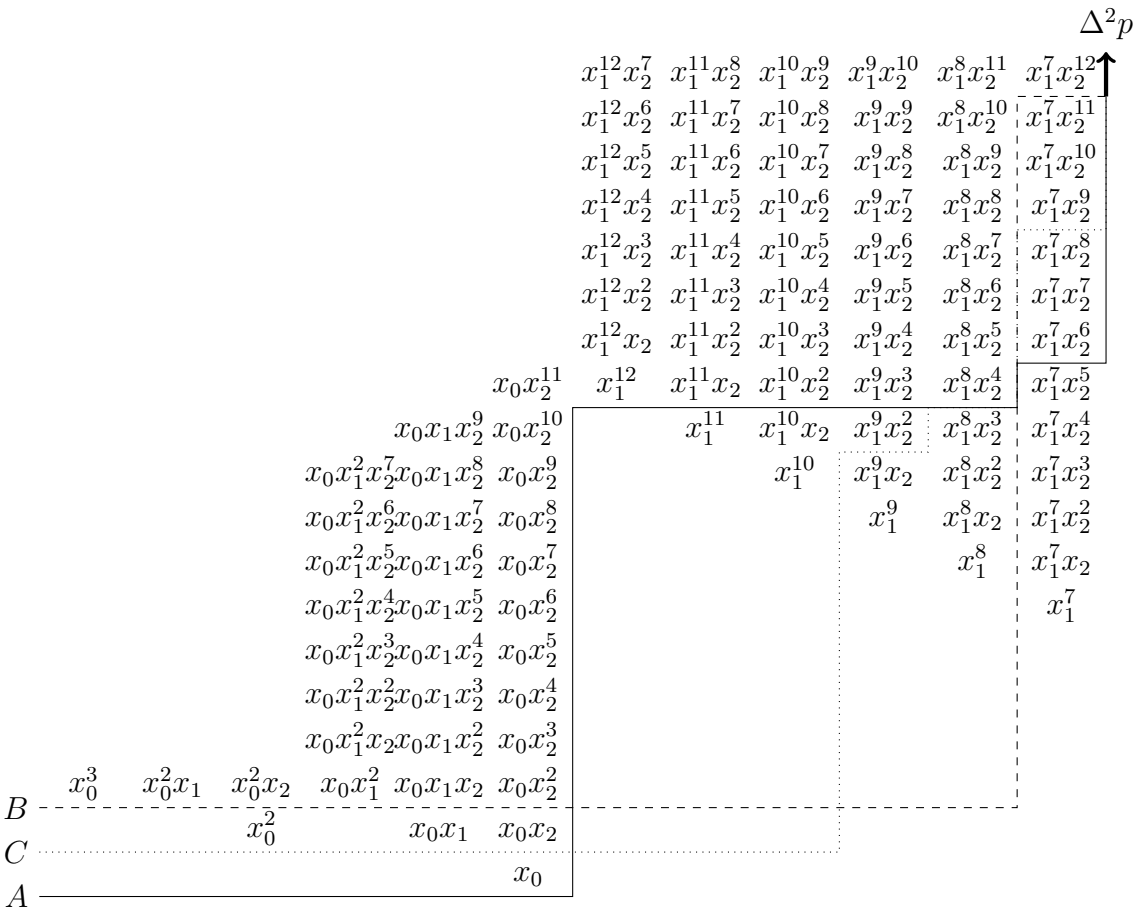

Figure 6.2: The ideals A, B, C in Example 6.2.2.

Example 6.2.3 is an example of how Theorem 3.4.2 cannot be extended to a 3-depth Hilbert family. While it is probable that an example could be constructed for any depth, we do not have an explicit construction.

Example 6.2.3. Let $S = [x_0, \ldots, x_5]$. Consider the polynomial

$$p(d) = 38d^2 - 690d + 5083$$

and the following 3-depth Hilbert family of ideals in S,

$$\mathcal{I} = \left\{\text{ideal } I \subset S \;\middle|\; \operatorname{depth} I \geqslant 3 \text{ and } p_{S/I} = p\right\}$$

Note, $S^{(3)} = K[x_0, x_1, x_2]$. The only two lexsegment ideals in $\mathcal{I}^{(3)}$ are

$$A = \left\langle x_0,\, x_1^8,\, x_1^7x_2,\, x_1^6x_2^2,\, x_1^5x_2^3,\, x_1^4x_2^5,\, x_1^3x_2^6,\, x_1^2x_2^8,\, x_1x_2^{22},\, x_2^{29}\right\rangle,$$
$$B = \left\langle x_0^3,\, x_0^2x_1,\, x_0^2x_2,\, x_0x_1^2,\, x_0x_1x_2,\, x_0x_2^3,\, x_1^4,\, x_1^3x_2^9,\, x_1^2x_2^{11},\, x_1x_2^{23},\, x_2^{28}\right\rangle.$$

Their total Betti numbers are

$$\beta(A) = (10, 17, 8),$$
$$\beta(B) = (11, 17, 7).$$

As a result, A and B have maximal total Betti numbers in $\mathcal{I}$, and there is no ideal with maximum total Betti numbers in $\mathcal{I}$.

Example 6.2.4 is an example of how Theorem 4.0.1 cannot be extended to a polynomial ring of more than 4 variables. In this example, we set a lower bound on Δh.

Example 6.2.4. Let $S = K[x_0, \ldots, x_4]$. Consider the polynomial

$$p(d) = 5d + 11$$

and the function g where $g(3) = 8$, $g(4) = 8$, and $g(d) = \Delta h_S(d)$ otherwise. Consider also the following 1-depth Hilbert family of ideals in S,

$$\mathcal{I} = \left\{\text{saturated ideal } I \subset S \;\middle|\; p_{S/I} = p \text{ and } \Delta h_{S/I}(d) \geqslant g(d) \text{ for all } d\right\}.$$

Note, $S^{(1)} = K[x_0, \ldots, x_3]$. Let $\mathfrak{m}_i = \langle x_0, \ldots, x_i\rangle$ and

$$A = x_0\mathfrak{m}_3^2 + x_1^2(\mathfrak{m}_2 + \mathfrak{m}_3^2) + x_1(\mathfrak{m}_2^3 + \mathfrak{m}_3^4) + x_2^5$$
$$B = x_0\mathfrak{m}_3 + x_1^2\mathfrak{m}_3^2 + x_1\mathfrak{m}_3^4 + x_2^5x_3.$$

Both A and B have maximal total Betti numbers in $\mathcal{I}$ with

$$\beta(A) = (18, 39, 30, 8)$$
$$\beta(B) = (17, 39, 32, 9)$$

Thus, neither has maximum total Betti numbers in $\mathcal{I}$. See Figure 6.3 for a visual representation of these ideals and constraints.

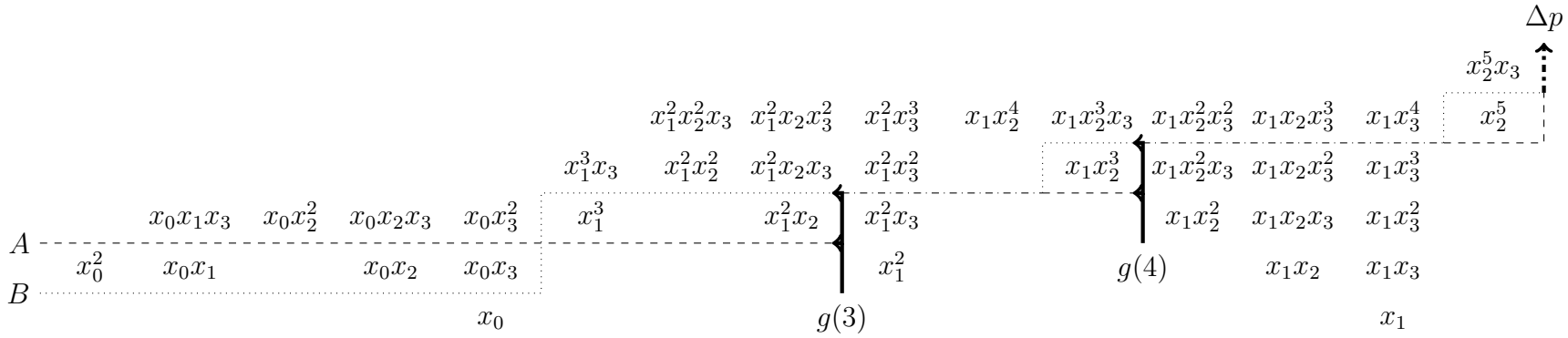

Figure 6.3: The ideals A and B from Example 6.2.4.

This next example is another example of how Theorem 4.0.1 cannot be extended to a polynomial ring of more than 4 variables. In this example, we set a upper bound on Δh. Notice that this example uses 10 variables instead of only 5. While there is no proof that this is the fewest number of variables where an upper bound fails to give maximum total Betti numbers, it is probable that the fewest number of variables is greater than 5. This could result in a partial extension of Theorem 4.0.1.

Example 6.2.5. Let $S = K[x_0, \ldots, x_9]$. Consider the polynomial

$$p(d) = \frac{1}{5040}\,d^7 + \frac{1}{180}\,d^6 + \frac{29}{360}\,d^5 + \frac{37}{72}\,d^4 + \frac{1327}{720}\,d^3 + \frac{893}{360}\,d^2 + \frac{323}{105}\,d + 35$$

and the function f where $f(2) = 40$, $f(3) = 116$, $f(5) = 586$, and $f(d) = \Delta h_S(d)$ otherwise. Consider also the following 1-depth Hilbert family of ideals in S,

$$\mathcal{I} = \left\{\text{saturated ideal } I \subset S \;\middle|\; p_{S/I} = p \text{ and } f(d) \geqslant \Delta h_{S/I}(d) \text{ for all } d\right\}.$$

Note, $S^{(1)} = K[x_0, \ldots, x_8]$. Let $\mathfrak{m}_i = \langle x_0, \ldots, x_i\rangle + \langle x_0, \ldots, x_8\rangle^2$ and

$$\begin{aligned} A &= x_0\mathfrak{m}_8 + x_1^2\mathfrak{m}_8 + x_1x_2\mathfrak{m}_8\mathfrak{m}_2 + x_1x_3^2\mathfrak{m}_8 + x_1x_3^2x_4\mathfrak{m}_8 + x_1x_3^2x_5\mathfrak{m}_8 \\ B &= x_0\mathfrak{m}_4 + x_1^2\mathfrak{m}_4 + x_1x_2\mathfrak{m}_8\mathfrak{m}_8 + x_1x_3^2\mathfrak{m}_4 + x_1x_3^2x_4\mathfrak{m}_3 + x_1x_3^2x_5\mathfrak{m}_4. \end{aligned}$$

Both A and B have maximal total Betti numbers in $\mathcal{I}$ with

$$\begin{aligned} \beta(A) &= (95, 574, 1623, 2724, 2933, 2061, 919, 237, 27) \\ \beta(B) &= (94, 575, 1646, 2791, 3027, 2136, 954, 246, 28) \end{aligned}$$

Thus, neither has maximum total Betti numbers in $\mathcal{I}$. See Figure 6.4 for a visual representation of these ideals and constraints.

Figure 6.4: The ideals A and B from Example 6.2.5.

Appendices

Appendix: MaxBettiNumbers Source Code

```
-- -*- coding: utf-8 -*-
newPackage( "MaxBettiNumbers",
  Headline =>
  "Methods to find Maximum Betti numbers given bounds on the Hilbert function",
  Version => "0.9",
  Date => "July 14, 2020",
  Authors => { { Name => "Jay White", Email => "jay.white@uky.edu" } },
  DebuggingMode => true,
  HomePage => "https://github.com/JayWhite2357/maxbetti"
);

-----------------------------------------------------------------------------
--- exports
-----------------------------------------------------------------------------

--- exports and options for the main method of the package, maxBettiNumbers
export { "maxBettiNumbers" };
export { "HilbertFunctionLowerBound", "HilbertDifferenceLowerBound" };
export { "HilbertFunctionUpperBound", "HilbertDifferenceUpperBound" };
export { "HilbertPolynomial", "ResultsCount" };
--export { "Algorithm" };

--- exports for the type, MaxBetti, that is returned by maxBettiNumbers
export { "MaxBetti" };
export { "isRealizable", "BettiUpperBound", "MaximumBettiSum" };
export { "HilbertFunctions", "MaximalBettiNumbers" };

--- exports and options for the auxillary methods of the package
export { "lexBetti", "almostLexBetti" };
export { "lexsegmentIdeal", "almostLexIdeal" };
export { "AsTally" };

-----------------------------------------------------------------------------
--- end exports
-----------------------------------------------------------------------------



-----------------------------------------------------------------------------
--- types
-----------------------------------------------------------------------------

MaxBetti = new Type of HashTable

-----------------------------------------------------------------------------
--- end types
-----------------------------------------------------------------------------



-----------------------------------------------------------------------------
--- Functions to deal with the Macaulay Representation of a Number
-----------------------------------------------------------------------------

--- This is the macaulay representation of a. From this, it is easy to read off
---   the max index of a monomial as well as the Macaulay upper and lower
---   bounds.
--- The output, rep, is a sequcne of length d
--- a = binomial(rep#0, 1) + binomial(d + rep#d, d + 1)
--- Note: this is a different output from most versions of this method.
---   the reason is that this is more compact and efficient.
--- The idea is that we want to easily increment and decrement a without
```

```
---   recomputing the representation each time.

--- Incrementing rep: (increasing a)
---   Find the last element equal to rep#0.
---   Increase that element by 1 and set all preceding elements to 0
--- Decrementing rep: (decreasing a)
---   Find the first nonzero element
---      Reduce it by 1
---      Set all preceding elements to that element's new value
--- Reading off monomial:
---   The first nonzero element indicates the max index.
---   Add 1 to every other nonzero element.
---   Set all zero elements to equal the first nonzero element.
---   Take n+1-e for each element e. This is the index of the variable.
---   The product of all the variables is the ath last monomial,
---      where the 1st last is the power of the last variable.


macaulayRepresentation = ( a, d ) -> (
  v := if d <= 1 then a else 0;
  if d > 1 then (
    while a > binomial( v - 1 + d, d ) do (
      v = v + 1
    )
  );
  reverse for i in reverse( 0 .. d - 1 ) list (
    if a === 0 then 0 else if i === 0 then a else (
      a = a - binomial( v + i, i + 1 );
      while a < 0 do (
        v = v - 1;
        a = a + binomial( v + i, i );
      );
      v
    )
  )
)

macaulayAboveBound = ( a, d ) -> (
  if ( a === infinity or ( d === 0 and a > 0 ) ) then infinity else (
    r := macaulayRepresentation( a, d );
    sum for i to #r-1 list (
      binomial( r#i + i + 1, i + 2 )
    )
  )
);

macaulayBelowBound = ( a, d ) -> (
  if ( a <= 0 or d <= 0 ) then 0 else if d === 1 then 1 else (
    r := macaulayRepresentation( a, d );
    sum for i to #r-1 list (
      if r#i === 0 then 0 else (
        binomial( r#i + i - 1, i )
      )
    )
  )
);

decrementRep = ( rep, firstNonzeroIndex, firstNonzeroValue ) -> (
  rep = join(
    firstNonzeroIndex + 1 : firstNonzeroValue - 1,
    drop( rep, firstNonzeroIndex + 1 )
  );
  if firstNonzeroValue === 1 then (
    firstNonzeroIndex = firstNonzeroIndex + 1;
```

```
    if rep#?firstNonzeroIndex then (
      firstNonzeroValue = rep#firstNonzeroIndex;
    )
  ) else (
    firstNonzeroIndex = 0;
    firstNonzeroValue = firstNonzeroValue - 1;
  );
  ( rep, firstNonzeroIndex, firstNonzeroValue )
);

incrementRep = ( rep, lastrep0Index ) -> (
  rep = join(
    toList( lastrep0Index : 0 ),
    { rep#0 + 1 },
    drop( rep, lastrep0Index + 1 )
  );
  if lastrep0Index === 0 then (
    while rep#?( lastrep0Index + 1 ) and
      rep#( lastrep0Index + 1 ) === rep#0 do
      lastrep0Index = lastrep0Index + 1;
  ) else lastrep0Index = lastrep0Index - 1;
  ( rep, lastrep0Index )
)

-------------------------------------------------------------------------------
--- End Macaulay Representation Methods
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--- Beginning of functions to Sanitize and Optimize the inputs ----------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
--- functions to sanitize the input bounds
-------------------------------------------------------------------------------

--- This function sanitizes g(d) so that it is a valid hilbert function.
--- This relies on the fact that h(d) is must be at least
--- macaulayAboveBound(g(d-1), d-1)
cleanUpperBound = f -> (
  bound := 1;
  for d to #f - 1 list (
    if f#d =!= null then (
      bound = min( f#d, bound )
    );
    bound
  ) do (
    bound = macaulayAboveBound( bound, d )
  )
);

--- This function sanitizes f(d) so that it is a valid hilbert function.
--- This relies on the fact that h(d) is must be at most
--- macaulayBelowBound(g(d+1), d+1)
cleanLowerBound = f -> if #f == 0 then f else (
  bound := 0;
  reverse for d in reverse( 0 .. #f - 1 ) list (
    if f#d =!= null then (
      bound = max( f#d, bound )
    );
    bound
  ) do (
```

```
    bound = macaulayBelowBound( bound, d )
  )
);

-------------------------------------------------------------------------------
--- end sanitize bounds
-------------------------------------------------------------------------------

--- function for padding lists with null
padList = ( l, f ) -> join( f, toList( l - #f : null ) );

-------------------------------------------------------------------------------
--- functions to sanitize the polynomial input
-------------------------------------------------------------------------------

--- This function gives the minimum degree at which that every saturated ideal
--- with hilbert polynomial p is guarenteed to have hilbert function match
--- hilbert polynomial
minPolyBoundDegree = p -> (
  n := first degree p;
  i := ( ring p )_0;
  lC := 0;
  while n > 0 do (
    lC = leadCoefficient p;
    p = p - binomial( n + i, n + 1 ) + binomial( n + i - n! * lC, n + 1 );
    n = first degree p;
  );
  d := sub( p, ZZ );
  if lC > d then error "Invalid Hilbert Polynomial.";
  d
);

--- This function sets G,F,g,f to be the appropriate values to ensure that all
--- ideals in the family have the specified hilbert polynomial
--- if no polynomial is specified, nothing is done.
cleanPolynomial = ( G, F, g, f, p, d ) -> (
  if p =!= null then (
    i := ( ring p )_0;
    pd := sub( sub( p, i => d ), ZZ );
    pd' := sub( sub( p, i => d - 1 ), ZZ );
    deltapd := pd - pd';
    G = padList( d + 1, G );
    F = padList( d + 1, F );
    g = padList( d + 1, g );
    f = padList( d + 1, f );
    if G_d === null or G#d < pd then G = replace( d, pd, G );
    if F_d === null or F#d > pd then F = replace( d, pd, F );
    if g_d === null or g#d < deltapd then g = replace( d, deltapd, g );
    if f_d === null or f#d > deltapd then f = replace( d, deltapd, f );
  );
  ( G, F, g, f )
);

-------------------------------------------------------------------------------
--- end sanitize polynomial
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
--- functions to optimize the bounds
-------------------------------------------------------------------------------

--- This function makes an optimizations of g(d) based on the following:
--- The smallest that h(d) could be is G(d)-F(d-1)
--- Usage: g = optimizeLowerBound ( G, F, g );
optimizeLowerBound = ( G, F, g ) -> (
```

```
  gprime := G - prepend( 0, drop( F, -1 ) );
  max \ transpose { g, gprime }
);

--- This function makes an optimizations of f(d) based on the following:
--- The largest that h(d) could be is G(d)-G(d-1)
--- Usage: f = optimizeLowerBound ( G, F, f );
optimizeUpperBound = ( G, F, f ) -> (
  fprime := F - prepend( 0, drop( G, -1 ) );
  min \ transpose { f, fprime }
);

--- This function makes two optimizations on G(d) based on the following:
--- The smallest that H(d) could be is G(d) + g(d)
--- The smallest that H(d) could be is G(d+1) - f(d)
optimizeAccumulatedLowerBound = ( G, g, f ) -> (
  cumulativeSum := 0;
  G = for d to #G - 1 list (
    cumulativeSum = max( G#d, cumulativeSum + g#d )
  );
  bound := 0;
  reverse for d in reverse( 0 .. #G - 1 ) list (
    bound = max( G#d, bound )
  ) do (
    bound = bound - f#d
  )
);

--- This function makes two optimizations on F(d) based on the following:
--- The largest that H(d) could be is F(d) + f(d)
--- The largest that H(d) could be is F(d+1) - g(d)
optimizeAccumulatedUpperBound = ( F, g, f ) -> (
  cumulativeSum := 0;
  F = for d to #F - 1 list (
    cumulativeSum = min( F#d, cumulativeSum + f#d )
  );
  bound := infinity;
  reverse for d in reverse( 0 .. #F - 1 ) list (
    bound = min( F#d, bound )
  ) do (
    bound = bound - g#d
  )
);

-------------------------------------------------------------------------------
--- end optimize bounds
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
--- main function to sanitize (and optimize) the inputs
-------------------------------------------------------------------------------

sanitizeInputs = ( G, F, g, f, p, n ) -> (
  --- Sanitize the polynomial input by converting it to bounds
  D := if p === null then 0 else minPolyBoundDegree( p );
  D = max( D, #G-1, #F-1, #g-1, #f-1 );
  ( G, F, g, f ) = cleanPolynomial( G, F, g, f, p, D );
  --- End Sanitize polynomial

  --- Sanitize the input length so that they include at least degree 1
  l := max( #G, #F, #g, #f, 2 );
  ( G, F, g, f ) = ( G, F, g, f ) / padList_l;
  --- End Sanitize input length
```

```
  --- Sanitize the degree 1 upper difference bound
  if ( f#1 === null or f#1 > n + 1 ) then (
    f = replace( 1, n + 1, f )
  );
  --- End Sanitize degree 1

  --- Sanitize the inputs so that they are Hilbert functions
  F = cleanUpperBound F;
  f = cleanUpperBound f;
  G = cleanLowerBound G;
  g = cleanLowerBound g;
  --- End sanitize functions

  --- Check validity of inputs
  valid := min( min( F - G ), min( f - g ) ) >= 0;
  --- End check validity

  --- Optimize the bounds. This drastically improves the run time.
  --- This is done by repeated application of the 4 optimization Functions.
  --- Once G,F,g,f no longer change, we stop and re-sanitize the values
  --- We repeat this process until G,F,g,f are stable.
  --- Throughout this process we check that it is always a possible scenario.
  prevGFgf := null;
  while valid and prevGFgf =!= ( G, F, g, f ) do (
    while valid and prevGFgf =!= ( G, F, g, f ) do (
      prevGFgf = ( G, F, g, f );
      f = optimizeUpperBound( G, F, f );
      g = optimizeLowerBound( G, F, g );
      F = optimizeAccumulatedUpperBound( F, g, f );
      G = optimizeAccumulatedLowerBound( G, g, f );
      valid = min( min( F - G ), min( f - g ) ) >= 0;
    );
    if valid then (
      F = cleanUpperBound F;
      f = cleanUpperBound f;
      G = cleanLowerBound G;
      g = cleanLowerBound g;
      valid = min( min( F - G ), min( f - g ) ) >= 0;
    )
  );
  --- End Optimize bounds

  --- Throw an error if bounds don't make sense
  if not valid then (
    error concatenate (
      "The given inputs are invalid or give impossible constraints:\nG = ",
      toString G, "\nF = ", toString F, "\ng = ", toString g, "\nf = ",
      toString f
    );
  );
  if last G =!= last F then (
    error concatenate (
      "The given inputs don't specify constraints that eventually fix the ",
      "Hilbert function:\nG = ", toString G, "\nF = ", toString F, "\ng = ",
      toString g, "\nf = ", toString f );
  );
  --- End throw error

  ( G, F, g, f )
);

-------------------------------------------------------------------------------
--- end main sanitize function
-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--- End Sanitize and Optimize -------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
--- Functions to build V and lowerBound for use in the algorithms.
-------------------------------------------------------------------------------

--- We precompute the V_q values as well as the Macaulay lower bound.
---   Doing this ahead of time saves a ton of repitition.
--- We are setting V#d#(j - g#d) = V_q[d, j]
--- We are setting lowerBound#d#(j - g#d) = [j]_<d>
--- Although we could compute these directly, the following is a more efficient
---   way of computing them iteratively.
BuildVLowerBound = ( g, f, n ) -> (
  --- Make a list of all possible vectors. This way we don't have to compute
  ---   binomials repeatedly
  Vi := for i to n list (
    --- The next line takes the sum of the for loop and appends it to the end;
    ---   there might be a neater way to do this, but I can't find one.
    ( v -> append( v, sum v ) ) for q to n list (
      binomial( n + 1, q + 1 ) - binomial( i + 1, q + 1 )
    )
  );
  --- This is one of our outputs that we will populate.
  V := for d to #g - 1 list new MutableList from g#d .. f#d;
  --- This is the other output that we will populate.
  lowerBound := for d to #g - 1 list new MutableList from g#d .. f#d;
  for d to #g - 1 do (
    VAccumulated := Vi#n; -- This is simply the zero vector to start.
    --- begin initialize the macaulay representation
    rep := macaulayRepresentation( g#d, d );
    --- initialize the lastrep0Index to be the last index in rep that equals
    ---   rep#0.
    lastrep0Index := 0;
    while rep#?( lastrep0Index + 1 ) and
      rep#( lastrep0Index + 1 ) === rep#0 do
      lastrep0Index = lastrep0Index + 1;
    --- end initialize lastrep0Index
    nextLowerBound := macaulayBelowBound( g#d, d );
    nextrep0 := 0;
    for k to f#d - g#d do (
      --- the index of the monomial is n - nextrep0.
      VAccumulated = VAccumulated + Vi#( n - nextrep0 );
      V#d#k = VAccumulated;
      lowerBound#d#k = nextLowerBound;
      if #rep === 0 then continue;
      nextrep0 = rep#0;
      if nextrep0 === 0 then nextLowerBound = nextLowerBound + 1;
      ( rep, lastrep0Index ) = incrementRep( rep, lastrep0Index );
    )
  );
  ( V, lowerBound )
)

-------------------------------------------------------------------------------
--- End functions for building V and lowerBound
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--- Begining of the driver methods of the Algorithms themselves ---------------
-------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------

-----------------------------------------------------------------------------
--- Simplified Method returning no hilbert Functions
-----------------------------------------------------------------------------

--- This is an implementation of the algorithm described in the paper
--- Note: the portions of the pseudocode in algorithm 3.1 of above are
---   written in comments beginning with --**
--- Notationally: putting a ' (prime) on a variable indicates that it is the
---   value of the variable in the previous degree.

--- Instead of using a dictionary with (d, c) keys, we opt to only use a list,
---    the dictionaries are lists of the theoretical (d, c) keys.
---    the "dictionary" in each degree is a list of all the keys, where the
---    0th entry is the (d, G#d key). Note, to access a key with a d-1 degree,
---    we simply access the previous version of that dictionary, which is why we
---    have to keep track of that.
--- To make sense of the more compact way of storing these values, we will use
---    the shorthand b to represend c-g and i to represent j-G.


SimplifiedNone = ( G, F, g, f, V, lowerBound ) -> (
  --**We initialize the base case by creating a dictionary maxVDict'
  --**  containing (-1, 0) => 0
  --- V#0#0 is the zero vector
  maxVDict' := { V#0#0 };
  --- These are the correct values in degree -1.
  G' := 0;
  F' := 0;
  maxj' := { 0 };
  --**For each value of d from 0 to D do:
  for d to #G - 1 do (
    --- maxj#c is the max j value that we can get if we respect the Macaulay
    ---   bound. Although it is not necessary, we can quit looping early by
    ---   respecting this bound.
    maxj := new MutableList from G#d .. F#d;
    --**For each value of c from G(d) to F(d) do:
    --- maxVDict' get's assigned only once we have looped through all c, so
    ---   there is no need to create a maxVDict just to reassign maxVDict'
    ---   notationally, it is confusing, but it works well this way.
    maxVDict' = for c from G#d to F#d list (
      --- max \ transpose does elementwise maximization on a bunch of lists
      max \ transpose(
        --**For each value of j from g(d) to f (d) do:
        --- The situations where j + F' < c or j + G' > c are impossible
        for j from max( g#d, c - F' ) to min( f#d, c - G' )
        --- We can ignore the situations where we violate the lower bound.
        ---   This is done by comparing the Macaulay lower bound with the
        ---   maximum j in the previous degree.
        when maxj'#( c - j - G' ) >= lowerBound#d#( j - g#d )
        list (
          maxj#( c - G#d ) = j;
          --**Compute V0 = maxVDict(d', c') + Vq[d,j].
          --- This next line is the potential maximum for this iteration
          maxVDict'#( c - j - G' ) + V#d#( j - g#d )
        )
      )
      --**Add the entry (d,c)=>maxV to the dictionary maxVDict
    );
    --- Update all of the values so that they are correct for the next degree
    G' = G#d;
    F' = F#d;
    maxj' = maxj;
  );
```

```
  --**Return the value maxVDict(D, G(D), g(D))
  maxVDict'#0
);

-----------------------------------------------------------------------------
--- End Simplified None
-----------------------------------------------------------------------------

-----------------------------------------------------------------------------
--- Simplified Method returning all hilbert functions with max betti sum
-----------------------------------------------------------------------------

--- This method is similar in structure to SimplifiedNone
--- The functions that give the maximum sum of the Vq's is returned in a
---   "raveled" format. raveledHFs contains a list of each degree, which is a
---   list of the value of the functions in that degree that give the maximum
---   sum of V. This "raveled" result can be unraveled with the
---   UnravelSimplified methods.
--- Note, the last element of the V vectors is the sum of the Vq's, which is why
---   we use that element for tracking HF.

SimplifiedSome = ( G, F, g, f, V, lowerBound ) -> (
  maxVDict' := { V#0#0 };
  G' := 0;
  F' := 0;
  maxj' := { 0 };
  --- Each degree of this for loop returns all the j values in that degree with
  ---   max V.
  raveledHFs := for d to #G - 1 list (
    maxj := new MutableList from G#d .. F#d;
    maxHFDict := new MutableList from G#d .. F#d;
    maxVDict' = for c from G#d to F#d list (
      --- Note, we need to track the maxSum so that we can collect the j values
      maxSum := 0;
      maxHF := { };
      --- However, we can still utilize max \ transpose to maximize the vectors
      maxV := max \ transpose (
        for j from max( g#d, c - F' ) to min( f#d, c - G' )
        when maxj'#( c - j - G' ) >= lowerBound#d#( j - g#d ) list (
          maxj#( c - G#d ) = j;
          V0 := maxVDict'#( c - j - G' ) + V#d#( j - g#d );
          if last V0 === maxSum then (
            maxHF = append( maxHF, j );
          ) else if last V0 > maxSum then (
            maxHF = { j };
            maxSum = last V0;
          );
          V0
        )
      );
      maxHFDict#( c - G#d ) = maxHF;
      maxV
    );
    G' = G#d;
    F' = F#d;
    maxj' = maxj;
    --- This is the list of all j values that gave us max V
    toList maxHFDict
  );
  ( maxVDict'#0, raveledHFs )
);

-----------------------------------------------------------------------------
--- End Simplified Some
-----------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
--- Simplified Method returning all hilbert Functions
-------------------------------------------------------------------------------

SimplifiedAll = ( G, F, g, f, V, lowerBound ) -> (
  maxVDict' := { { V#0#0 } };
  G' := 0;
  F' := 0;
  maxj' := { 0 };
  raveledHFs := for d to #G - 1 list (
    maxj := new MutableList from G#d .. F#d;
    maxHFDict := new MutableList from G#d .. F#d;
    --- Instead of being a vectors, maxVDict#c is a list of vectors.
    maxVDict' = for c from G#d to F#d list (
      --- maxVHF will be the collection of all V and HF that are maximal.
      ---   The keys of the table are the maximal values of V.
      ---   The values of each key are the j's that give that value of V.
      maxVHF := new MutableHashTable;
      for j from max( g#d, c - F' ) to min( f#d, c - G' )
      when maxj'#( c - j - G' ) >= lowerBound#d#( j - g#d )
      do (
        maxj#( c - G#d ) = j;
        --- Instead of being a vectors, maxVDict'#c' is a list of vectors.
        for maxV' in maxVDict'#( c - j - G' ) do (
          V0 := maxV' + V#d#( j - g#d );
          if maxVHF#?V0 then (
            --- In the case where V0 is already in maxVHF, add j to it's key.
            maxVHF#V0 = append( maxVHF#V0, j );
          ) else if false =!= ( --- If max Vdiff <= 0 below is never true...
            for V1 in keys maxVHF do (
              Vdiff := V0 - V1;
              --- If V0 is greater than V1, remove V1.
              if min Vdiff >= 0 then remove( maxVHF, V1 )
              --- If V0 is less than V1, break and do nothing.
              else if max Vdiff <= 0 then break false
            )
          ) then (
            --- In the case where V0 is not less than any V1, add it to maxVHF.
            maxVHF#V0 = { j };
          )
        )
      );
      --- We want to make maxHFDict simply a list of possible j values.
      maxHFDict#( c - G#d ) = unique flatten values maxVHF;
      --- maxVDict#c is a list the vectors stored in the keys of maxVHF
      keys maxVHF
    );
    G' = G#d;
    F' = F#d;
    maxj' = maxj;
    toList maxHFDict
  );
  ( maxVDict'#0, raveledHFs )
);

-------------------------------------------------------------------------------
--- End Simplified All
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
--- Complete Method returning no hilbert Functions
-------------------------------------------------------------------------------

--- Notationally: putting a ' (prime) on a variable indicates that it is the
---   value of the variable in the previous degree.
```

```
--- Also: i is shorthand for j-g and b is shorthand for c-G, as indicated above
---   the method SimplifiedNone

--- The "dictionary" entry is thus
---   Dict#b#i instead of (d,c,j) and
---   Dict'#b'#i' instead of (d-1,c',j')

CompleteNone = ( G, F, g, f, V, lowerBound ) -> (
  maxVDict' := { { V#0#0 } };
  G' := 0;
  g' := 0;
  for d to #G - 1 do (
    --- Each iteration of the following is a list of the maxV values for each j.
    maxVDict' = for c from G#d to F#d list (
      maxV := null;
      --- We have to traverse the list in reverse order.
      reverse for j in reverse( g#d .. min( f#d, c - G' ) ) list (
        b' := c - j - G';
        i := j - g#d;
        i' := max( lowerBound#d#i - g', 0 );
        if maxVDict'#?b' and maxVDict'#b'#?i' then (
          V0 := maxVDict'#b'#i' + V#d#i;
          if maxV === null then (
            maxV = V0
          ) else (
            Vdiff := V0 - maxV;
            if min Vdiff >= 0 then (
              maxV = V0
            ) else if max Vdiff > 0 then (
              maxV = max \ transpose{ maxV, V0 }
            )
          )
        );
        --- If this value of j is impossible, we simply won't save it.
        ---   This won't mess up indexing because it can only happen in the
        ---   beginnin iterations, and we reverse the list after.
        if maxV === null then continue;
        --- The main difference from SimplifiedNone is that we need to save this
        ---   value for each j so that we can use it in the next degree
        ---   as a result, the loops cannot be written as compactly.
        maxV
      )
      --- We return a list of the maxV values for each j.
    );
    G' = G#d;
    g' = g#d;
  );
  maxVDict'#0#0
);

-----------------------------------------------------------------------------
--- End Complete None
-----------------------------------------------------------------------------

-----------------------------------------------------------------------------
--- Complete Method returning all hilbert functions with max betti sum
-----------------------------------------------------------------------------

--- This is a combination of CompleteNone and SimplifiedSome.
--- The primary difference is that the "raveled" result in each degree must be
---   a list where each entry corresponds to a c value, which in turn is a list
---   of the possible j values that give maxV for that c and d. So, it is a
---   list of lists of lists. This "raveled" result can be unraveled with the
---   UnravelComplete methods.

CompleteSome = ( G, F, g, f, V, lowerBound ) -> (
```

```
  maxVDict' := { { V#0#0 } };
  G' := 0;
  g' := 0;
  raveledHFs := for d to #G - 1 list (
    maxHFDict := new MutableList from G#d .. F#d;
    maxVDict' = for c from G#d to F#d list (
      maxV := null;
      maxHF := { };
      --- In this case, we need to collect both the maxV and the maxHF.
      ---   The easiest way is to do it at the same time, and then just split it
      ---   up later.
      maxVHFList := reverse for j in reverse( g#d .. min( f#d, c - G' ) ) list (
        b' := c - j - G';
        i := j - g#d;
        i' := max( lowerBound#d#i - g', 0 );
        --- We need to check that this is actually a valid value of j that has
        ---   any valid functions in the previous degree
        if maxVDict'#?b' and maxVDict'#b'#?i' then (
          V0 := maxVDict'#b'#i' + V#d#i;
          if maxV === null then (
            maxHF = { j };
            maxV = V0;
          ) else (
            if last V0 === last maxV then (
              maxHF = append( maxHF, j );
            ) else if last V0 > last maxV then (
              maxHF = { j };
            );
            Vdiff := V0 - maxV;
            if min Vdiff >= 0 then (
              maxV = V0
            ) else if max Vdiff > 0 then (
              maxV = max \ transpose{ maxV, V0 }
            )
          );
        );
        if maxV === null then continue;
        ( maxV, maxHF )
      );
      --- Here we just split up the list so that maxV and maxHF are separate.
      maxHFDict#( c - G#d ) = maxVHFList / last;
      maxVHFList / first
    );
    G' = G#d;
    g' = g#d;
    toList maxHFDict
  );
  ( maxVDict'#0#0, raveledHFs )
);

-------------------------------------------------------------------------------
--- End Complete Some
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
--- Complete Method returning all hilbert Functions
-------------------------------------------------------------------------------

--- This is by far the most complex version. However, there are no new ideas,
---   it is simply a combination of the techniques in CompleteSome and
---   SimplifiedAll.

CompleteAll = ( G, F, g, f, V, lowerBound ) -> (
  maxVDict' := { { { V#0#0 } } };
  G' := 0;
```

```
  g' := 0;
  raveledHFs := for d to #G - 1 list (
    maxHFDict := new MutableList from G#d .. F#d;
    maxVDict' = for c from G#d to F#d list (
      maxVHF := new MutableHashTable;
      maxVHFList := reverse for j in reverse( g#d .. min( f#d, c - G' ) ) list (
        b' := c - j - G';
        i := j - g#d;
        i' := max( lowerBound#d#i - g', 0 );
        if maxVDict'#?b' and maxVDict'#b'#?i' then (
          for maxV' in maxVDict'#b'#i' do (
            V0 := maxV' + V#d#i;
            if maxVHF#?V0 then (
              maxVHF#V0 = append( maxVHF#V0, j );
            ) else if false =!= (
              for V1 in keys maxVHF do (
                Vdiff := V0 - V1;
                if min Vdiff >= 0 then remove( maxVHF, V1 )
                else if max Vdiff <= 0 then break false
              )
            ) then (
              maxVHF#V0 = { j }
            )
          )
        );
        if #maxVHF === 0 then continue;
        ( keys maxVHF, unique flatten values maxVHF )
      );
      maxHFDict#( c - G#d ) = maxVHFList / last;
      maxVHFList / first
    );
    G' = G#d;
    g' = g#d;
    toList maxHFDict
  );
  ( maxVDict'#0#0, raveledHFs )
);

-----------------------------------------------------------------------------
--- End Complete All
-----------------------------------------------------------------------------


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
--- End of the main algorithm code ------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------



-----------------------------------------------------------------------------
--- functions for extracting hilbert function from the algorithm's return value
-----------------------------------------------------------------------------

--- This method unravels a single hilbert function from the raveled result
---   returned from the Simplified algorithm.

UnravelSimplifiedOne = ( HFs, G, g, lowerBound ) -> (
  targetSum := last G;
  lowerBound' := 0;
  result := { };
  for d in reverse( 0 .. #HFs - 1 ) do (
    --- In this version, we take the minimum hilbert function at this degree.
    ---   This is guarenteed to give a valid result in the Simplified case.
    ---   (In the Complete case, it gives a valid result because only valid
    ---   results are stored in raveledHFs.)
    hd := min( HFs#d#( targetSum - G#d ) );
```

```
    targetSum = targetSum - hd;
    result = (
      --- We don't really start saving the hilbert function until the Macaulay
      ---   bound is a strict inequality.
      if ( #result === 1 and lowerBound' === hd ) then { hd }
      else prepend( hd, result )
    );
    lowerBound' = lowerBound#d#( hd - g#d );
  );
  { result }
);

--- This is essentially the same as UnravelSimplifiedOne. The only difference is
---   that we track all possible triples (targetSum, lowerBound', result).
---   The list of these is kept in partialUnraveled, which is looped over for
---   each degree. Additionally, all possible values for the hilbert function
---   are considered, and not simply the minimum.
UnravelSimplifiedHFs = ( HFs, G, g, lowerBound ) -> (
  partialUnraveled := { ( last G, 0, { } ) };
  for d in reverse( 0 .. #HFs - 1 ) do (
    partialUnraveled = flatten for tlr in partialUnraveled list (
      ( targetSum, lowerBound', result ) := tlr;
      if targetSum < G#d then continue;
      for hd in HFs#d#( targetSum - G#d ) list (
        if hd < lowerBound' then continue;
        (
          targetSum - hd,
          lowerBound#d#( hd - g#d ),
          if ( #result === 1 and lowerBound' === hd ) then { hd }
          else prepend( hd, result )
        )
      )
    )
  );
  partialUnraveled / last
);

--- In this Complete case, this gives a valid result because only valid results
---   are stored in raveledHFs, since we have the extra "k" index.
--- The only difference from UnravelSimplifiedOne is the lowerBound' - g#d index
UnravelCompleteOne = ( HFs, G, g, lowerBound ) -> (
  targetSum := last G;
  lowerBound' := 0;
  result := { };
  for d in reverse( 0 .. #HFs - 1 ) do (
    --- The only difference from UnravelSimplifiedOne is the lowerBound' - g#d
    hd := min( HFs#d#( targetSum - G#d )#( lowerBound' - g#d ) );
    ( targetSum, lowerBound', result ) =
    (
      targetSum - hd,
      lowerBound#d#( hd - g#d ),
      if ( #result === 1 and lowerBound' === hd ) then { hd }
      else prepend( hd, result )
    )
  );
  { result }
);

--- The only difference from UnravelSimplifiedHFs is the lowerBound' - g#d index
UnravelCompleteHFs = ( HFs, G, g, lowerBound ) -> (
  partialUnraveled := { ( last G, 0, { } ) };
  for d in reverse( 0 .. #HFs - 1 ) do (
    partialUnraveled = flatten for tlr in partialUnraveled list (
```

```
      ( targetSum, lowerBound', result ) := tlr;
      if targetSum < G#d then continue;
      --- The only difference from UnravelSimplifiedHFs is the lowerBound' - g#d
      for hd in HFs#d#( targetSum - G#d )#( max( lowerBound' - g#d, 0 ) ) list (
        if hd < lowerBound' then continue;
        (
          targetSum - hd,
          lowerBound#d#( hd - g#d ),
          if ( #result === 1 and lowerBound' === hd ) then { hd }
          else prepend( hd, result )
        )
      )
    )
  );
  partialUnraveled / last
);

-----------------------------------------------------------------------------
--- End unravel functions
-----------------------------------------------------------------------------



-----------------------------------------------------------------------------
--- auxillary methods
-----------------------------------------------------------------------------

--- This is an efficient way of computing the betti numbers of a lexsegment
---   Instead of computing the actual ideal, we loop through each degree and
---   use the max index of the monomial
---   once we have enough monomials in that degree, we move up to the next
---   degree
--- The returned values are the sums of the binomial coefficients of the
---   generators in each degree. They are exactly the values in the
---   Eliahou-Kervaire resolution.
lexBettiArray = ( h, n ) -> (
  b := for i to n list for q to n list binomial( i, q );
  zeroList := for i to n list 0;
  if not h#?0 then return { zeroList } else if h#0 === 0 then return { b#0 };
  if h#0 > 1 or min h < 0 then error( "Not a valid Hilbert function." );
  ( rep, firstNonzeroIndex, firstNonzeroValue ) :=
    ( { n + 1 }, 0, n + 1 );
  for d to #h - 1 list if d === 0 then zeroList else (
    upperBound := sum for i to #rep - 1 list binomial( rep#i + i, i + 1 );
    if h#d > upperBound then error( "Not a valid Hilbert function." );
    s := zeroList;
    for l from h#d to upperBound - 1 do (
      s = s + b#( n + 1 - firstNonzeroValue );
      ( rep, firstNonzeroIndex, firstNonzeroValue ) =
        decrementRep( rep, firstNonzeroIndex, firstNonzeroValue );
    );
    s
  ) do if d =!= 0 then (
    --- move rep up one degree
    rep = prepend( 0, rep );
    firstNonzeroIndex = firstNonzeroIndex + 1
    --- end move rep degree
  )
);


--- Note: n is one less than the number of variables.
--- Additionally, S can have many more variables than we need. We only use the
---   first n+1 variables.
--- This is exactly the same algorithm as lexBettiArray, however, we generate
---   the monomials themselves instead of just the binomial coefficients of the
---   max index.
```

```
createLexIdeal = ( S, h, n ) -> (
  if not h#?0 then return ideal 0_S else if h#0 === 0 then return ideal 1_S;
  if h#0 > 1 or min h < 0 then error( "Not a valid Hilbert function." );
  ( rep, firstNonzeroIndex, firstNonzeroValue ) :=
    ( { n + 1 }, 0, n + 1 );
  gs := flatten for d to #h - 1 list if d === 0 then { } else (
    upperBound := sum for i to #rep - 1 list binomial( rep#i + i, i + 1 );
    if h#d > upperBound then error( "Not a valid Hilbert function." );
    for l from h#d to upperBound - 1 list (
      product for i to d - 1 list S_(
        n + 1 - (
          if rep#i === 0 then firstNonzeroValue
          else if i === 0 or rep#( i - 1 ) === 0 then rep#i
          else rep#i + 1
        )
      )
    ) do (
      ( rep, firstNonzeroIndex, firstNonzeroValue ) =
        decrementRep( rep, firstNonzeroIndex, firstNonzeroValue )
    )
  ) do if d =!= 0 then (
    --- move rep up one degree
    rep = prepend( 0, rep );
    firstNonzeroIndex = firstNonzeroIndex + 1
    --- end move rep degree
  );
  ideal if #gs === 0 then 0_S else gs
);


convertBettiArrayToBettiTally = array -> (
  new BettiTally from flatten append (
    for row in pairs array list (
      d := row#0 - 1;
      for col in pairs row#1 list (
        if col#1 === 0 then continue;
        i := col#0 + 1;
        ( i, { d + i }, d + i ) => col#1
      )
    ),
    ( 0, { 0 }, 0 ) => 1
  )
);


lexBetti = method( Options => { AsTally => true } );
lexBetti ( ZZ, List ) := o -> ( numberOfVariables, h ) -> (
  result := lexBettiArray( h, numberOfVariables - 1 );
  if o.AsTally === true then (
    convertBettiArrayToBettiTally result
  ) else (
    result
  )
);

almostLexBetti = method( Options => { AsTally => true } );
almostLexBetti ( ZZ, List ) := o -> ( numberOfVariables, h ) ->
  lexBetti( numberOfVariables - 1, h - prepend( 0, drop( h, -1 ) ), o );

lexsegmentIdeal = method( TypicalValue => Ideal );
lexsegmentIdeal ( PolynomialRing, List ) := ( S, h ) ->
  createLexIdeal( S, h, dim S - 1 );

almostLexIdeal = method( TypicalValue => Ideal );
```

```
almostLexIdeal ( PolynomialRing, List ) := ( S, h ) ->
  createLexIdeal( S, h - prepend( 0, drop( h, -1 ) ), dim S - 2 );

-------------------------------------------------------------------------------
--- end auxillary methods
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
--- main Method that parses options and delegates to the appropriate algorithm
-------------------------------------------------------------------------------

maxBettiNumbers = method( TypicalValue => MaxBetti, Options => {
  HilbertPolynomial => null,
  HilbertFunctionUpperBound => { },
  HilbertFunctionLowerBound => { },
  HilbertDifferenceUpperBound => { },
  HilbertDifferenceLowerBound => { },
  ResultsCount => "None",
  Algorithm => "Automatic"
} );

maxBettiNumbers ZZ := o -> numberOfVariables -> (
  ---Parse Inputs and handle options
  n := numberOfVariables - 2;
  G := o.HilbertFunctionLowerBound;
  F := o.HilbertFunctionUpperBound;
  g := o.HilbertDifferenceLowerBound;
  f := o.HilbertDifferenceUpperBound;
  p := o.HilbertPolynomial;
  if instance( p, ZZ ) then p = sub( p, QQ( monoid[ getSymbol "i" ] ) );
  algorithm :=
    if o.Algorithm === "Complete" then 1
    else if o.Algorithm === "Simplified" or F === { } then 0
    else -1;
  resultsCount :=
    if o.ResultsCount === "One" or o.ResultsCount === 1 then 1
    else if o.ResultsCount === "AllMaxBettiSum" then 2
    else if o.ResultsCount === "All" then 3
    else 0;
  ---End Parse inputs

  ---Sanitize Inputs
  ( G, F, g, f ) = sanitizeInputs( G, F, g, f, p, n );
  ---End Sanitize Inputs

  ---Automatically select algorithm
  if algorithm === -1 then (
    GFgfsimplified := try sanitizeInputs( G, { }, g, f, p, n ) else null;
    algorithm = if ( G, F, g, f ) ===  GFgfsimplified then 0 else 1;
  );
  algorithmToRun := {
    { SimplifiedNone, SimplifiedSome, SimplifiedSome, SimplifiedAll },
    { CompleteNone, CompleteSome, CompleteSome, CompleteAll }
  }#algorithm#resultsCount;
  unravelToRun := {
    { null, UnravelSimplifiedOne, UnravelSimplifiedHFs, UnravelSimplifiedHFs },
    { null, UnravelCompleteOne, UnravelCompleteHFs, UnravelCompleteHFs }
  }#algorithm#resultsCount;
  ---End Auto select


  ---Run Algorithm
  ( V, lowerBound ) := BuildVLowerBound( g, f, n );
  result := algorithmToRun( G, F, g, f, V, lowerBound );
  ---End Run Algorithm
```

```
  ---Parse Results
  hilbertFunctions :=
    if resultsCount === 0 then null
    else unravelToRun ( result#1, G, g, lowerBound );
  bettiUpperBound := null;
  maximumBettiSum := null;
  maximalBettiNumbers := null;
  if resultsCount === 0 then (
    bettiUpperBound = drop( result, -1 );
    maximumBettiSum = last result;
  ) else if resultsCount === 3 then (
    maximalBettiNumbers = result#0 / ( b -> drop( b, -1 ) );
    bettiUpperBound = max \ transpose maximalBettiNumbers;
    maximumBettiSum = max( last \ result#0 );
  ) else (
    bettiUpperBound = drop( result#0, -1 );
    maximumBettiSum = last result#0;
  );
  realizable := maximumBettiSum === sum bettiUpperBound;
  bettig := sum lexBettiArray( g, n );
  bettiUpperBound = bettiUpperBound + bettig;
  maximumBettiSum = maximumBettiSum + sum bettig;
  if maximalBettiNumbers =!= null then
    maximalBettiNumbers = maximalBettiNumbers / plus_bettig;
  if hilbertFunctions =!= null then
    hilbertFunctions = hilbertFunctions / accumulate_( plus, 0 );
  ---End parse results

  ---Format results
  new MaxBetti from hashTable{
    isRealizable => realizable,
    BettiUpperBound => bettiUpperBound,
    MaximumBettiSum => maximumBettiSum,
    if maximalBettiNumbers =!= null then
      MaximalBettiNumbers => VerticalList maximalBettiNumbers,
    if hilbertFunctions =!= null then
      HilbertFunctions => VerticalList hilbertFunctions
  }
  ---End format results
);

-----------------------------------------------------------------------------
--- end main Method
-----------------------------------------------------------------------------


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
--- Beginning of documentation ----------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------

beginDocumentation( )

doc ///
  Key
    maxBettiNumbers
    (maxBettiNumbers, ZZ)
  Headline
    Upper bounds for total Betti numbers in a family of saturated ideals.
  Usage
    maxBettiNumbers (N,
      HilbertPolynomial => p,
      HilbertFunctionLowerBound => G,
      HilbertFunctionUpperBound => F,
      HilbertDifferenceLowerBound => g,
```

```
    HilbertDifferenceUpperBound => f)
Inputs
  N:ZZ
    the number of variables in the ambient polynomial ring.
  HilbertPolynomial=>RingElement
    the Hilbert polynomial, @TT"p"@, of the ideals in the family.
    See @TO [maxBettiNumbers, HilbertPolynomial]@.
  HilbertFunctionLowerBound=>List
    the lower bound for the Hilbert function, @TT"G"@, of the ideals in the
    family. See @TO [maxBettiNumbers, HilbertFunctionLowerBound]@.
  HilbertFunctionUpperBound=>List
    the upper bound for the Hilbert function, @TT"F"@, of the ideals in the
    family. See @TO [maxBettiNumbers, HilbertFunctionUpperBound]@.
  HilbertDifferenceLowerBound=>List
    the lower bound for the difference Hilbert function, @TT"g"@, of the
    ideals in the family.
    See @TO [maxBettiNumbers, HilbertDifferenceLowerBound]@.
  HilbertDifferenceUpperBound=>List
    the upper bound for the difference Hilbert function, @TT"f"@, of the
    ideals in the family.
    See @TO [maxBettiNumbers, HilbertDifferenceUpperBound]@.
  ResultsCount=>String
    how many Hilbert functions the result should include.
    See @TO [maxBettiNumbers, ResultsCount]@.
  Algorithm=>String
    the algorithm to use. See @TO [maxBettiNumbers, Algorithm]@.
Outputs
  :
    an object with the upper bound and additional information.
Description
  Text
    Consider a polynomial ring, $S$, in @TT"N"@ variables.
    Consider the family of saturated ideals, $I\subset S$, satisfying the
    following constraints. (Note: $h_{S/I}$ will denote the hilbert function
    of $S/I$, and $\Delta$ will denote the difference operator. (i.e.
    $\Delta h_{S/I}(d)=h_{S/I}(d)-h_{S/I}(d-1)$.)

    The functions $G$, $F$, $g$, $f$, and $p$ are arguments to the method.
    In the case where a value is not given, the corresponding constraint is
    removed (i.e. made the trivial constraint). Note: if $p$ is not specified
    $F$ and $G$ must be equal for large degrees.

    @UL{TEX"$G(d)\\leq h_{S/I}(d)\\leq F(d)$ for all $d$",
    TEX"$g(d)\\leq\\Delta h_{S/I}(d)\\leq f(d)$ for all $d$",
    TEX"$h_{S/I}(d)=p(d)$ for large $d$"}@

    @TT"maxBettiNumbers"@ returns the upper bound for the total Betti numbers
    of the ideals along with other information.
    A complete description of the output can be found under @TO MaxBetti@.

    Almost lexsegment ideals have the largest total Betti numbers out of all
    saturated ideals with a given Hilbert function. The function
    @TO almostLexIdeal@ is useful to obtain the ideals with maximal Betti
    numbers.

    The following is an example in $6$ variables where we fix the Hilbert
    polynomial to be $2d+10$, and look at the Betti tables of the ideals that
    realize the maximum total Betti numbers.
  Example
    QQ[d];
    result = maxBettiNumbers(6, HilbertPolynomial => 2*d+10,
      ResultsCount => "All")
    almostLexBetti_6 \ toList result.HilbertFunctions
  Text
    Restricting to ideals with at least one linear element gives us a
```

```
  different result.
Example
  result = maxBettiNumbers(6, HilbertPolynomial => 2*d+10,
    HilbertFunctionUpperBound => {,5}, ResultsCount => "All")
  I = almostLexIdeal(QQ[x_1..x_6], first result.HilbertFunctions)
  betti res I
  hilbertPolynomial(I, Projective=>false)
  (0..6)/(d->hilbertFunction(d,I))
Text
  In most situations, there is an ideal in the family that realizes this
  upper bound. However, there are situations where this is not true.
  This method indicates this with the key @TO isRealizable@. However, there
  is always an ideal that gives the maximum sum of the total Betti numbers.
  This maximum is given by the key @TO MaximumBettiSum@. The following
  example in $5$ variables shows this phenomenon. In it we fix the Hilbert
  polynomial to be $5d+11$, and we restrict $\Delta h_{S/I}(d)\geq 8$ for
  $d=3,4$.
Example
  result = maxBettiNumbers(5, HilbertDifferenceLowerBound => {,,,8,8},
    HilbertPolynomial => 5*d+11);
  sum result.BettiUpperBound
  result.MaximumBettiSum  -- This doesn't match the previous sum.
  result.isRealizable     -- As a result, this is false.
Text
  @HEADER2"Default constraints"@

  Because the inputs can be incomplete or absent, @TT"maxBettiNumbers"@
  assumes the following default values for the missing information.

  @UL{TEX"Lower bounds have a default value of $0$.",
  TEX"Upper bounds have a default value of infinity.",
  TEX"Truncated hilbert functions are assumed to continue, but the
  associated hilbert polynomial is assumed to match the hilbert function at
  last degree where it is specified.",
  TEX"If no Hilbert polynomial is specified, the Hilbert polynomial is
  assumed to be the Hilbert polynomial of $G$ and $F$."}@

  More details can be found under
  @TO [maxBettiNumbers, HilbertFunctionLowerBound]@. In the case where the
  inputs result in constraints that are impossible or invalid, an error is
  thrown.

  @HEADER2"Output Results"@

  In addition to upper bounds for the total Betti numbers, this function can
  optionally output Hilbert functions with maximal total Betti numbers. This
  is specified with the optional argument @TT"ResultsCount"@. More details
  can be found under @TO [maxBettiNumbers, ResultsCount]@.

  @HEADER2"Different Algorithms"@

  There are two different algorithms that get used: the Simplified
  algorithm, which is faster, but is not guarenteed to give sharp bounds,
  and the Complete algorithm, which always gives sharp bounds. The optional
  argument @TT"Algorithm"@ allows the selection of the algorithm. A more
  complete description can be found under @TO [maxBettiNumbers,Algorithm]@.

  @HEADER2"More Examples"@

  We will consider an example where $S$ is the polynomial ring in $5$
  variables.
  This example has only maximal total Betti Numbers, and not maximum total
  Betti numbers.
  Also, the Simplified and Complete algorithms give different results.
  Both of these are somewhat unusual, but give an illuminating example.
  We will choose the following constraints:
```

```
  $$h_{S/I}(6)=41\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
  h_{S/I}(d)=49\ for\ large\ d$$
  $$8\leq \Delta h_{S/I}(3)\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
  8\leq \Delta h_{S/I}(4)\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
  5\leq \Delta h_{S/I}(5)\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
  5\leq \Delta h_{S/I}(6)\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ $$
  Since we will be using these constraints in several examples, we will
  first define a few variables to reduce repetition.
Example
  N = 5;
  g = HilbertDifferenceLowerBound => {,,,8,8,5,5};
  G = HilbertFunctionLowerBound => {,,,,,,41};
  F = HilbertFunctionUpperBound => {,,,,,,41};
  p = HilbertPolynomial => 49;
Text
  We find that $(23, 54, 47, 14)$ is the upper bound for the total Betti
  numbers of all saturated ideals with these constraints.
  Additionally, the maximum for the sum of the Betti numbers is $137$.
  Note that because $23 + 54 + 47 + 14 = 138$, there is no single ideal with
  total Betti numbers of $(23, 54, 47, 14)$.
Example
  maxBettiNumbers(N,p,g,G,F)
Text
  If we want the Hilbert function of an ideal with maximal total Betti
  numbers, we can pass @TT"ResultsCount=>\"One\""@ as an option.
  Note, this gives an ideal with the maximum for the sum of the Betti
  numbers.
Example
  maxBettiNumbers(N,p,g,G,F, ResultsCount=>"One")
Text
  If we want the Hilbert function of all ideals that have the maximum sum of
  the Betti numbers, we can pass @TT"ResultsCount=>\"AllMaxBettiSum\""@ as
  an option.
Example
  maxBettiNumbers(N,p,g,G,F, ResultsCount=>"AllMaxBettiSum")
Text
  Finally, if we want the Hilbert function of all ideals that have maximal
  total Betti numbers, we can pass @TT"ResultsCount=>\"All\""@ as an option.
  In addition to returning the upper bound and Hilbert functions, the
  maximal total Betti numbers of $(23, 54, 45, 13)$ and $(22, 54, 47, 14)$
  are also returned.
Example
  maxBettiNumbers(N,p,g,G,F, ResultsCount=>"All")
Text
  Because we are setting an upper bound of $h_{S/I}(6) \leq 41$, the
  Simplified algorithm will not give sharp bounds. As a result, the Complete
  algorithm is automatically chosen instead. However, we can force the use
  of a different one. In this case, if we specify
  @TT"Algorithm=>\"Simplified\""@, we get an upper bound that is,
  necessarily, larger. Additionally, we are given a Hilbert function that
  appears to be valid, but is not the Hilbert function of any saturated
  ideal.
Example
  maxBettiNumbers(N,p,g,G,F, Algorithm=>"Simplified", ResultsCount=>"One")
Text
  We can compare the speed of the two algorithms with an example of fixing
  the Hilbert polynomial to be $3d^2-6d+175$ in a ring with $6$ variables.
  Because there is no upper bound for $h_{S/I}$, both algorithms give valid
  results, and smallest possible upper bounds.
CannedExample
  i23 : p = HilbertPolynomial => 3*d^2-6*d+175;

  i24 : first timing maxBettiNumbers(6, p,
          Algorithm=>"Simplified", ResultsCount=>"None")
```

```
      o24 = 5.908441668

      o24 : RR (of precision 53)

      i25 : first timing maxBettiNumbers(6, p,
              Algorithm=>"Simplified", ResultsCount=>"All")

      o25 = 7.467646269

      o25 : RR (of precision 53)

      i26 : first timing maxBettiNumbers(6, p,
              Algorithm=>"Complete", ResultsCount=>"None")

      o26 = 21.462211888

      o26 : RR (of precision 53)

      i27 : first timing maxBettiNumbers(6, p,
              Algorithm=>"Complete", ResultsCount=>"All")

      o27 = 28.756055662

      o27 : RR (of precision 53)
  Caveat
    If @TT"Algorithm=>\"Simplified\""@ is forced, this may not return valid
    Hilbert functions for some inputs.
  SeeAlso
    MaxBetti
///
doc ///
  Key
    ResultsCount
///
doc ///
  Key
    HilbertPolynomial
///
doc ///
  Key
    HilbertFunctionLowerBound
///
doc ///
  Key
    HilbertDifferenceLowerBound
///
doc ///
  Key
    HilbertFunctionUpperBound
///
doc ///
  Key
    HilbertDifferenceUpperBound
///
doc ///
  Key
    AsTally
///
doc ///
  Key
    [maxBettiNumbers, HilbertFunctionLowerBound]
    [maxBettiNumbers, HilbertFunctionUpperBound]
    [maxBettiNumbers, HilbertDifferenceLowerBound]
    [maxBettiNumbers, HilbertDifferenceUpperBound]
  Description
```

```
    Text
      The options @TT"HilbertFunctionLowerBound"@,
      @TT"HilbertFunctionUpperBound"@,
      @TT"HilbertFunctionLowerBound"@,
      @TT"HilbertDifferenceLowerBound"@, and
      @TO HilbertPolynomial@ are arguments to
      @TO maxBettiNumbers@.
      Each of these options, other than @TO HilbertPolynomial@, are a list of
      integers starting at degree 0.
      In the case where a value is not given, the corresponding constraint is
      removed (i.e. made the trivial constraint).
      Note: if @TO HilbertPolynomial@ is not specified,
      @TT"HilbertFunctionLowerBound"@ and @TT"HilbertFunctionUpperBound"@
       must be equal for large degrees.

      In the case where no lower
      bound is desired at a specified degree, @TT"0"@, @TT"null"@, or nothing
      can be put instead. For instance, to specify only a lower bound of @TT"4"@
      in degree @TT"3"@ on the Hilbert difference function, the option
      @TT"HilbertDifferenceLowerBound=>{,,,4}"@ can be used.

      Similarly, in the case where no upper bound is desired at a specified
      degree, @TT"infinity"@, @TT"null"@, or nothing can be put instead.
      For instance, to specify only a upper bound of @TT"4"@
      in degree @TT"3"@ on the Hilbert function, the option
      @TT"HilbertFunctionUpperBound=>{,,,4}"@ can be used.

      There are some instances when these options, along with
      @TO HilbertPolynomial@ conflict. In this case, an error is raised.
  SeeAlso
    maxBettiNumbers
    [maxBettiNumbers, HilbertPolynomial]
///
doc ///
  Key
    [maxBettiNumbers, HilbertPolynomial]
  Description
    Text
      The functions @TO HilbertFunctionLowerBound@,
      @TO HilbertFunctionUpperBound@,
      @TO HilbertFunctionLowerBound@,
      @TO HilbertDifferenceLowerBound@, and
      @TT"HilbertPolynomial"@ are arguments to
      @TO maxBettiNumbers@.
      In the case where a value is not given, the corresponding constraint is
      removed (i.e. made the trivial constraint).
      Note: if @TT"HilbertPolynomial"@ is not specified,
      @TO HilbertFunctionLowerBound@ and  @TO HilbertFunctionUpperBound@
       must be equal for large degrees.

      This option can be either a @TO RingElement@ or an integer. In the case
      where this conflicts with the bounds on
      the Hilbert functions, an error is raised.
  SeeAlso
    maxBettiNumbers
    [maxBettiNumbers, HilbertFunctionLowerBound]
///
doc ///
  Key
    lexBetti
    (lexBetti, ZZ, List)
  Headline
    Graded Betti numbers of a lexsegment ideal.
  Usage
    lexBetti (N, h)
  Inputs
```

```
    N:ZZ
      the number of variables in the ambient polynomial ring.
    h:List
      the Hilbert function of the lexsegment ideal.
  Description
    Text
      Consider a polynomial ring in @TT"N"@ variables. For any hilbert function,
      there is a unique lexsegment ideal. Furthermore, this ideal has graded
      Betti numbers that are at least as large as those of any other ideal with
      that hilbert function.

      The Hilbert function of a lexsegment ideal is determined by the values in
      the degrees that are at and below the largest degree of any generator. As
      a result, it makes sense to specify the Hilbert function through the
      largest degree of a generator and then truncate the rest of the function.

      This function returns the graded Betti numbers of a lexsegment ideal with
      the given, tuncated, Hilbert function.
    Example
      lexBetti (4, {1,2,3,3,3,3})
      lexBetti (4, {1,2,3,3,3,3,0})
      lexBetti (5, {1,5,15,35})
  SeeAlso
    lexsegmentIdeal
    almostLexBetti
///
doc ///
  Key
    almostLexBetti
    (almostLexBetti, ZZ, List)
  Headline
    Graded Betti numbers of an almost lexsegment ideal.
  Usage
    almostLexBetti (N, h)
  Inputs
    N:ZZ
      the number of variables in the ambient polynomial ring.
    h:List
      the Hilbert function of the almost lexsegment ideal.
  Description
    Text
      Consider a polynomial ring in @TT"N"@ variables. For any hilbert function
      of a saturated ideal there is a unique almost lexsegment ideal. An almost
      lexsegment ideal is an ideal that is lexsegment in @TT"N-1"@ variables.
      Furthermore, this almost lexsegment ideal is saturated and has graded
      Betti numbers that are at least as large as those of any other saturated
      ideal with that hilbert function.

      The Hilbert function of a lexsegment ideal is determined by the values in
      the degrees that are at and below the largest degree of any generator. As
      a result, it makes sense to specify the Hilbert function through the
      largest degree of a generator and then truncate the rest of the function.
      This also applies to almost lexsegment ideals since they are simply
      lexsegment ideals in a smaller ring.

      This function returns the graded Betti numbers of an almost lexsegment
      ideal with the given Hilbert function.
    Example
      lexBetti (4, {1,2,3,3,3,3})
      almostLexBetti (5, {1,3,6,9,12,15})
      lexBetti (4, {1,2,3,3,3,3,0})
      almostLexBetti (5, {1,3,6,9,12,15,15})
      lexBetti (5, {1,5,15,35})
      almostLexBetti (6, {1,6,21,56})
  SeeAlso
    almostLexIdeal
```

```
    lexBetti
///
doc ///
  Key
    [lexBetti, AsTally]
    [almostLexBetti, AsTally]
  Description
    Text
      This is an option that can be passed to either @TO lexBetti@ or
      @TO almostLexBetti@. If the value of the option is @TT"true"@ a
      @TO BettiTally@ object will be returned. If it is false, a @TO List@ of
      lists will be returned. This latter option is useful if one wishes to
      obtain the total Betti numbers instead of the graded Betti numbers. This
      can easily be done using by applying @TT"sum"@ to the output with
      @TT"AsTally=>false"@.
    Example
      lexBetti (4, {1,2,3,3,3,3}, AsTally => true)
      lexBetti (4, {1,2,3,3,3,3}, AsTally => false)
      sum oo
  SeeAlso
    lexBetti
    almostLexBetti
///
doc ///
  Key
    lexsegmentIdeal
    (lexsegmentIdeal, PolynomialRing, List)
  Headline
    Create a lexsegment ideal.
  Usage
    lexsegmentIdeal (S, h)
  Inputs
    S:PolynomialRing
      the ambient polynomial ring.
    h:List
      the Hilbert function of the lexsegment ideal.
  Description
    Text
      Consider a polynomial ring in @TT"N"@ variables. For any hilbert function,
      there is a unique lexsegment ideal. Furthermore, this ideal has graded
      Betti numbers that are at least as large as those of any other ideal with
      that hilbert function.

      The Hilbert function of a lexsegment ideal is determined by the values in
      the degrees that are at and below the largest degree of any generator. As
      a result, it makes sense to specify the Hilbert function through the
      largest degree of a generator and then truncate the rest of the function.

      This function returns the lexsegment ideal with the given Hilbert
      function.

      Note: this method is significantly faster than the similar @TT"lexIdeal"@
      from the package @TT"LexIdeals"@.
    Example
      lexsegmentIdeal (QQ[x_1..x_4], {1,2,3,3,3,3})
      lexsegmentIdeal (QQ[x_1..x_4], {1,2,3,3,3,3,0}) --Artinian
      lexsegmentIdeal (QQ[x_1..x_5], {1,5,15,35})
  SeeAlso
    lexBetti
    almostLexIdeal
///
doc ///
  Key
    almostLexIdeal
    (almostLexIdeal, PolynomialRing, List)
  Headline
```

```
    Create an almost lexsegment ideal.
  Usage
    almostLexIdeal (S, h)
  Inputs
    S:PolynomialRing
      the ambient polynomial ring.
    h:List
      the Hilbert function of the almost lexsegment ideal.
  Description
    Text
      Consider a polynomial ring in @TT"N"@ variables. For any hilbert function
      of a saturated ideal there is a unique almost lexsegment ideal. An almost
      lexsegment ideal is an ideal that is lexsegment in @TT"N-1"@ variables.
      Furthermore, this almost lexsegment ideal is saturated and has graded
      Betti numbers that are at least as large as those of any other saturated
      ideal with that hilbert function.

      The Hilbert function of a lexsegment ideal is determined by the values in
      the degrees that are at and below the largest degree of any generator. As
      a result, it makes sense to specify the Hilbert function through the
      largest degree of a generator and then truncate the rest of the function.
      This also applies to almost lexsegment ideals since they are simply
      lexsegment ideals in a smaller ring.

      This function returns the almost lexsegment ideal with the given Hilbert
      function.
    Example
      lexsegmentIdeal (QQ[x_1..x_4], {1,2,3,3,3,3})
      almostLexIdeal (QQ[x_1..x_5], {1,3,6,9,12,15})
      lexsegmentIdeal (QQ[x_1..x_4], {1,2,3,3,3,3,0})
      almostLexIdeal (QQ[x_1..x_5], {1,3,6,9,12,15,15})
      lexsegmentIdeal (QQ[x_1..x_5], {1,5,15,35})
      almostLexIdeal (QQ[x_1..x_6], {1,6,21,56})
  SeeAlso
    almostLexBetti
    lexsegmentIdeal
///
doc ///
  Key
    [maxBettiNumbers, Algorithm]
  Description
    Text
      There are two algorithms that can be used to find the upper bounds given
      in @TO maxBettiNumbers@.

      The ``Simplified'' algorithm simply finds the maximum of while ignoring
      the ideal structure of an ideal. In other words, it searches all possible
      numeric functions instead of just the Hilbert functions. This has two
      consequences. First, it is significantly faster because it allows for a
      simplification of the algorithm. Second, it does not always give the
      smallest upper bounds. However, there is one instance where it is
      guarenteed to give the smallest upper bounds: when no upper bound for the
      Hilbert function is specified.

      The ``Complete'' algorithm does not make this simplification, and as a
      result, is slower but give the smallest upper bounds in every situation.

      Ideally, we would like to use the ``Simplified'' algorithm when it gives
      the smallest upper bounds, and use the ``Complete'' algorithm otherwise.
      By default, the algorithm is selected that guarentees the smallest upper
      bounds. However, this can be overridden by passing the @TT"Algorithm"@
      option to @TO maxBettiNumbers@. The possible values are

      @UL{{TT"\"Automatic\"", TEX" - this is the default and chooses the ",
      TEX"algorithm to use based on the other inputs. Note: This will always ",
      TEX"give the smallest upper bounds as well as valid Hilbert functions."},
```

```
      {TT"\"Simplified\"",TEX" - forces use of the ``Simplified'' algorithm. ",
      TEX"Note: if this option is passed, the values in ``HilbertFunctions'' ",
      TEX"may not be actual Hilbert functions."},
      {TT"\"Complete\"",TEX" - forces use of the ``Complete'' algorithm. ",
      TEX"Note: This will always give the smallest upper bounds as well as ",
      TEX"valid Hilbert functions."}}@
  Caveat
    If @TT"Algorithm=>\"Simplified\""@ is forced, this may not return valid
    Hilbert functions for some inputs.
  SeeAlso
    maxBettiNumbers
///
doc ///
  Key
    [maxBettiNumbers, ResultsCount]
  Description
    Text
      The method @TO maxBettiNumbers@ finds the upper bounds for the total Betti
      numbers. In certain instances, there are ideals that realize these upper
      bounds and have maximum possible total Betti numbers. In this case, these
      ideals also must have the maximum possible sum of the total Betti numbers.
      However, there are some instances where there are no ideals that realize
      the upper bounds. In this case, there are only ideals that realize maximal
      total Betti numbers. Some of these also must have the maximum possible sum
      of the total Betti numbers, while others do not.

      @TT"ResultsCount"@ is an option that can be passed to
      @TO maxBettiNumbers@. It determines how many, and what type of ideals are
      collected. The Hilbert function of these ideals is returned in a
      @TO MaxBetti@ object under key @TO HilbertFunctions@.

      There are four possible values, with the default being "None".


      @UL{{TT"\"None\"",TEX" or ",TT"0",
      TEX" - Does not return any Hilbert functions."},
      {TT"\"One\"",TEX" or ",TT"1",
      TEX" - Returns the Hilbert function of an ideal which has the maximum ",
      TEX"possible sum of the total Betti numbers."},
      {TT"\"AllMaxBettiSum\"",TEX" - Returns the Hilbert functions of all ",
      "ideals that have the maximum possible sum of the total Betti numbers."},
      {TT"\"All\"",TEX" - Returns the Hilbert functions of all ideals that ",
      "have maximal total Betti numbers."}}@
  SeeAlso
    maxBettiNumbers
    MaxBetti
///
doc ///
  Key
    MaxBetti
  Description
    Text
      This is the type that is returned by @TO maxBettiNumbers@ it is a
      @TO HashTable@ with the following keys.

      @UL{{TO BettiUpperBound,TEX" - upper bound for the total Betti numbers."},
      {TO HilbertFunctions,
      TEX" - a list of  Hilbert functions with maximal total Betti numbers. ",
      TEX"See ",TO[maxBettiNumbers,ResultsCount],TEX" for more details."},
      {TO isRealizable,TEX" - if there is an ideal with the upper bound as its",
      TEX" total Betti numbers."},
      {TO MaximalBettiNumbers,TEX" - the maximal total Betti numbers."},
      {TO MaximumBettiSum,TEX" - maximum sum of the total Betti numbers."}}@
  SeeAlso
    maxBettiNumbers
```

```
///
doc ///
  Key
    BettiUpperBound
  Description
    Text
      Used as a key in @TO MaxBetti@ with value being a @TO List@.
      The upper bound for the total Betti numbers.
  SeeAlso
    maxBettiNumbers
    MaxBetti
///
doc ///
  Key
    MaximumBettiSum
  Description
    Text
      Used as a key in @TO MaxBetti@ with value being a @TO ZZ@.
      The maximum value of the sum of the total Betti numbers.
  SeeAlso
    maxBettiNumbers
    MaxBetti
///
doc ///
  Key
    HilbertFunctions
  Description
    Text
      Used as a key in @TO MaxBetti@ with value being a @TO VerticalList@.
      A list of truncated Hilbert functions returned by @TO maxBettiNumbers@.
      See @TO[maxBettiNumbers, ResultsCount]@ for more details.
  Caveat
    If @TT"Algorithm=>\"Simplified\""@ is forced, this may not return valid
    Hilbert functions for some inputs.
  SeeAlso
    [maxBettiNumbers, ResultsCount]
    maxBettiNumbers
    MaxBetti
///
doc ///
  Key
    isRealizable
  Description
    Text
      Used as a key in @TO MaxBetti@ with value being a @TO Boolean@.
      Is @TT"true"@ if there is an ideal in the family that has total Betti
      numbers that match @TO BettiUpperBound@ otherwise is @TT"false"@.
      See @TO[maxBettiNumbers, ResultsCount]@ for more details.
  SeeAlso
    [maxBettiNumbers, ResultsCount]
    maxBettiNumbers
    MaxBetti
///
doc ///
  Key
    MaximalBettiNumbers
  Description
    Text
      Used as a key in @TO MaxBetti@ with value being a @TO VerticalList@.
      Each item in the list is a set of total Betti numbers that are maximal.
      In other words, no ideal has total Betti numbers that are simultaneously
      greater than or equal, and there is an ideal with these total Betti
      numbers.
      See @TO[maxBettiNumbers, ResultsCount]@ for more details.
  SeeAlso
```

```
    [maxBettiNumbers, ResultsCount]
    maxBettiNumbers
    MaxBetti
///
doc ///
  Key
    MaxBettiNumbers
  Headline
    Methods to find maximum Betti numbers given bounds on the Hilbert function.
  Description
    Text
      The method @TO maxBettiNumbers@ is the headliner in this package. It
      returns upper bounds for the total Betti numbers in a family that has
      bounds on the Hilbert function and/or the Hilbert difference function.

      The method @TO maxBettiNumbers@ can optionally return special Hilbert
      functions. The methods @TO almostLexBetti@ and @TO almostLexIdeal@ are
      helpful in working with these Hilbert function. The functions
      @TO lexBetti@ and @TO lexsegmentIdeal@ use the same code, and are exported
      from the package in hopes that they are useful. These functions are
      written with a concern for speed and efficiency.
///
--- replace the following with "doc ///" to run this long example
///
  Key
    "Paper Example"
  Description
    Example
      N = 5;
      g = HilbertDifferenceLowerBound => {,,,8,8,5,5};
      G = HilbertFunctionLowerBound => {,,,,,,41};
      F = HilbertFunctionUpperBound => {,,,,,,41};
      p = HilbertPolynomial => 49;
      maxBettiNumbers(N,p,g,G,F)
      maxBettiNumbers(N,p,g,G,F, ResultsCount=>"One")
      maxBettiNumbers(N,p,g,G,F, ResultsCount=>"AllMaxBettiSum")
      maxBettiNumbers(N,p,g,G,F, ResultsCount=>"All")
      almostLexBetti(N, last o9.HilbertFunctions)
      almostLexIdeal(QQ[x_1..x_N], last o9.HilbertFunctions)
      maxBettiNumbers(N,p,g,G,F, Algorithm=>"Simplified", ResultsCount=>"One")
      N = 6;
      QQ[i]; p = HilbertPolynomial => 3*i^2-6*i+175;
      time maxBettiNumbers(N, p, Algorithm=>"Simplified", ResultsCount=>"None");
      time maxBettiNumbers(N, p, Algorithm=>"Simplified", ResultsCount=>"All");
      time maxBettiNumbers(N, p, Algorithm=>"Complete", ResultsCount=>"None");
      time maxBettiNumbers(N, p, Algorithm=>"Complete", ResultsCount=>"All");
      loadPackage "StronglyStableIdeals"
      benchmark("maxBettiNumbers(5, HilbertPolynomial => 25)")
      benchmark("stronglyStableIdeals(25, 5)")
///
--- replace the following with "doc ///" to run this long example
///
  Key
    "Main Example"
  Description
    Example
      QQ[d];
      result = maxBettiNumbers(6, HilbertPolynomial => 2*d+10,
        ResultsCount => "All")
      almostLexBetti_6 \ toList result.HilbertFunctions
      result = maxBettiNumbers(6, HilbertPolynomial => 2*d+10,
        HilbertFunctionUpperBound => {,5}, ResultsCount => "All")
      I = almostLexIdeal(QQ[x_1..x_6], first result.HilbertFunctions)
      betti res I
```

```
      hilbertPolynomial(I, Projective=>false)
      (0..6)/(d->hilbertFunction(d,I))
      result = maxBettiNumbers(5, HilbertDifferenceLowerBound => {,,,8,8},
        HilbertPolynomial => 5*d+11);
      sum result.BettiUpperBound
      result.MaximumBettiSum  -- This doesn't match the previous sum.
      result.isRealizable     -- As a result, this is false.
      N = 5;
      g = HilbertDifferenceLowerBound => {,,,8,8,5,5};
      G = HilbertFunctionLowerBound => {,,,,,,41};
      F = HilbertFunctionUpperBound => {,,,,,,41};
      p = HilbertPolynomial => 49;
      maxBettiNumbers(N,p,g,G,F)
      maxBettiNumbers(N,p,g,G,F, ResultsCount=>"One")
      maxBettiNumbers(N,p,g,G,F, ResultsCount=>"AllMaxBettiSum")
      maxBettiNumbers(N,p,g,G,F, ResultsCount=>"All")
      maxBettiNumbers(N,p,g,G,F, Algorithm=>"Simplified", ResultsCount=>"One")
      p = HilbertPolynomial => 3*d^2-6*d+175;
      first timing maxBettiNumbers(6, p,
        Algorithm=>"Simplified", ResultsCount=>"None")
      first timing maxBettiNumbers(6, p,
        Algorithm=>"Simplified", ResultsCount=>"All")
      first timing maxBettiNumbers(6, p,
        Algorithm=>"Complete", ResultsCount=>"None")
      first timing maxBettiNumbers(6, p,
        Algorithm=>"Complete", ResultsCount=>"All")
///

-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
--- End of documentation ----------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
--- Beginning of tests ------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------

TEST /// --Test a preknown result.
N = 4;
p = 4;
mbn = maxBettiNumbers( N, HilbertPolynomial => p );
assert( mbn.BettiUpperBound === { 6, 8, 3 } );
///

TEST /// --Test that all 8 versions of the algorithm produce the same result.
testMatching = ( N, p ) -> (
  mbn = maxBettiNumbers( N, HilbertPolynomial => p );
  mbn1 = maxBettiNumbers( N, HilbertPolynomial => p,
    Algorithm => "Simplified", ResultsCount => "None" );
  mbn2 = maxBettiNumbers( N, HilbertPolynomial => p,
    Algorithm => "Simplified", ResultsCount => "One" );
  mbn3 = maxBettiNumbers( N, HilbertPolynomial => p,
    Algorithm => "Simplified", ResultsCount => "AllMaxBettiSum" );
  mbn4 = maxBettiNumbers( N, HilbertPolynomial => p,
    Algorithm => "Simplified", ResultsCount => "All" );
  mbn5 = maxBettiNumbers( N, HilbertPolynomial => p,
    Algorithm => "Complete", ResultsCount => "None" );
  mbn6 = maxBettiNumbers( N, HilbertPolynomial => p,
    Algorithm => "Complete", ResultsCount => "One" );
  mbn7 = maxBettiNumbers( N, HilbertPolynomial => p,
    Algorithm => "Complete", ResultsCount => "AllMaxBettiSum" );
```

```
  mbn8 = maxBettiNumbers( N, HilbertPolynomial => p,
    Algorithm => "Complete", ResultsCount => "All" );
  assert( mbn.BettiUpperBound === mbn1.BettiUpperBound );
  assert( mbn.BettiUpperBound === mbn2.BettiUpperBound );
  assert( mbn.BettiUpperBound === mbn3.BettiUpperBound );
  assert( mbn.BettiUpperBound === mbn4.BettiUpperBound );
  assert( mbn.BettiUpperBound === mbn5.BettiUpperBound );
  assert( mbn.BettiUpperBound === mbn6.BettiUpperBound );
  assert( mbn.BettiUpperBound === mbn7.BettiUpperBound );
  assert( mbn.BettiUpperBound === mbn8.BettiUpperBound );
  assert( sort mbn2.HilbertFunctions === sort mbn6.HilbertFunctions );
  assert( sort mbn3.HilbertFunctions === sort mbn7.HilbertFunctions );
  assert( sort mbn4.HilbertFunctions === sort mbn8.HilbertFunctions );
);
for i from 0 to 10 do testMatching( 4, i );
for i from 2 to 10 do testMatching( i, 4 );
///

TEST /// --Test against brute force method
loadPackage "StronglyStableIdeals";
QQ[d]; p = 2*d+10; N = 5;
time ssI = stronglyStableIdeals( p, N );
getTotalBetti = ( N, I ) -> (
  t := new Tally from ( applyKeys( betti res I, first, plus ) );
  for i from 1 to N - 1 list t_i
);
time maxbetti = max \ transpose ( ssI / getTotalBetti_N );
time mbn = maxBettiNumbers( N, HilbertPolynomial => p );
assert( mbn.BettiUpperBound === maxbetti );
///

TEST /// --Test against a preknown result.
N = 5;
g = HilbertDifferenceLowerBound => {,,,8,8,5,5};
G = HilbertFunctionLowerBound => {,,,,,,41};
F = HilbertFunctionUpperBound => {,,,,,,41};
p = HilbertPolynomial => 49;
mbn1 = maxBettiNumbers( N,p,g,G,F );
assert( mbn1.BettiUpperBound === {23,54,47,14} );
assert( mbn1.isRealizable === false );
assert( mbn1.MaximumBettiSum === 137 );
mbn2 = maxBettiNumbers( N,p,g,G,F, ResultsCount => "One" );
assert( #( mbn2.HilbertFunctions ) === 1 );
assert( mbn1.BettiUpperBound === mbn2.BettiUpperBound );
assert( mbn1.isRealizable === mbn2.isRealizable );
assert( mbn1.MaximumBettiSum === mbn2.MaximumBettiSum );
polys = mbn2.HilbertFunctions /
  almostLexIdeal_( QQ[x_1..x_N] ) /
  hilbertPolynomial_( Projective => false );
assert( all( polys, p -> sub( p, ZZ )=== 49 ) );
mbn3 = maxBettiNumbers( N,p,g,G,F, ResultsCount => "AllMaxBettiSum" );
assert( #( mbn3.HilbertFunctions ) === 18 );
assert( mbn1.BettiUpperBound === mbn3.BettiUpperBound );
assert( mbn1.isRealizable === mbn3.isRealizable );
assert( mbn1.MaximumBettiSum === mbn3.MaximumBettiSum );
polys = mbn3.HilbertFunctions /
  almostLexIdeal_( QQ[x_1..x_N] ) /
  hilbertPolynomial_( Projective => false );
assert( all( polys, p -> sub( p, ZZ )=== 49 ) );
mbn4 = maxBettiNumbers( N,p,g,G,F, ResultsCount => "All" );
assert( #( mbn4.HilbertFunctions ) === 36 );
assert( #( mbn4.MaximalBettiNumbers ) === 2 );
```

```
assert( mbn1.BettiUpperBound === mbn4.BettiUpperBound );
assert( mbn1.isRealizable === mbn4.isRealizable );
assert( mbn1.MaximumBettiSum === mbn4.MaximumBettiSum );
polys = mbn4.HilbertFunctions /
  almostLexIdeal_( QQ[x_1..x_N] ) /
  hilbertPolynomial_( Projective => false );
assert( all( polys, p -> sub( p, ZZ )=== 49 ) );
///

-- -------------------------------------------------------------------------
-- -------------------------------------------------------------------------
--- End of tests ----------------------------------------------------------
-- -------------------------------------------------------------------------
-- -------------------------------------------------------------------------


end
```

Bibliography

[1] M. Auslander and D. A. Buchsbaum. Homological Dimension in Local Rings. *Transactions of the American Mathematical Society*, 85(2):390–405, 1957.

[2] A. M. Bigatti. Upper bounds for the Betti numbers of a given Hilbert function. *Communications in Algebra*, 21(7):2317–2334, 1993.

[3] W. Bruns and H. J. Herzog. *Cohen-Macaulay Rings*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2 edition, 1998.

[4] G. Caviglia and S. Murai. Sharp upper bounds for the Betti numbers of a given Hilbert polynomial. *Algebra & Number Theory*, 7(5):1019 – 1064, 2013.

[5] D. Eisenbud. *Commutative Algebra*, volume 150 of *Graduate Texts in Mathematics*. Springer-Verlag, 1995.

[6] S. Eliahou and M. Kervaire. Minimal resolutions of some monomial ideals. *Journal of Algebra*, 129(1):1–25, 1990.

[7] G. Gotzmann. Eine Bedingung für die Flachheit und das Hilbertpolynom eines graduierten Ringes. *Mathematische Zeitschrift*, 158:61–70, 1978.

[8] D. Hilbert. Über die Theorie der algebraischen Formen. *Math. Annalen*, 36:473–534, 1890.

[9] H. A. Hulett. Maximum Betti numbers of homogeneous ideals with a given Hilbert function. *Communications in Algebra*, 21(7):2335–2350, 1993.

[10] F. S. Macaulay. Some Properties of Enumeration in the Theory of Modular Systems. *Proceedings of the London Mathematical Society*, s2-26(1):531–555, 1927.

[11] K. Pardue. Deformation classes of graded modules and maximal Betti numbers. *Illinois Journal of Mathematics*, 40(4):564 – 585, 1996.

[12] I. Peeva. *Graded syzygies.* Springer-Verlag, London, 2011.

Vita

Jay White

Education:

- University of Kentucky, Lexington, KY
  M.A. in Mathematics, Dec 2016
- Cedarville University, Cedarville, OH
  B.S. in Mathematics, May 2015
- Cedarville University, Cedarville, OH
  B.S. in Electrical Engineering, May 2015

Professional Positions:

- Graduate Teaching Assistant, University of Kentucky Fall 2015–Spring 2021

Honors

- Department summer research fellowship, University of Kentucky
- Enochs scholarship in algebra, University of Kentucky