Journal of Symbolic Computation 44 (2009) 358–381

Contents lists available at ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

The Slice Algorithm for irreducible decomposition of monomial ideals

Bjarke Hammersholt Roune [1]

Department of Computer Science, University of Aarhus, IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark

ARTICLE INFO

Article history:
Received 28 February 2008
Accepted 25 August 2008
Available online 23 September 2008

Keywords:
Irreducible decomposition
Monomial ideal
Maximal standard monomial
Socle
Integer programming gap
Frobenius problem
Slice Algorithm
Label Algorithm

ABSTRACT

Irreducible decomposition of monomial ideals has an increasing number of applications from biology to pure math. This paper presents the Slice Algorithm for computing irreducible decompositions, Alexander duals and socles of monomial ideals. The paper includes experiments showing good performance in practice.

1. Introduction

The main contribution of this paper is the Slice Algorithm, which is an algorithm for the computation of the irreducible decomposition of monomial ideals. To irreducibly decompose an ideal is to write it as an irredundant intersection of irreducible ideals.

Irreducible decomposition of monomial ideals has an increasing number of applications from biology to pure math. Some examples of this are the Frobenius problem (Roune, 2008b; Einstein et al., 2007), the integer programming gap (Hoşten and Sturmfels, 2007), the reverse engineering of biochemical networks (Jarrah et al., 2006), tropical convex hulls (Block and Yu, 2006), tropical cyclic polytopes (Block and Yu, 2006), secants of monomial ideals (Sturmfels and Sullivant, 2006), differential powers of monomial ideals (Sullivant, 2008) and joins of monomial ideals (Sturmfels and Sullivant, 2006).

E-mail address: bjarke@daimi.au.dk.
URL: http://www.broune.com/.
1 Tel.: +45 2895 9782.

0747-7171/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jsc.2008.08.002

Irreducible decomposition of a monomial ideal I has two computationally equivalent guises. The first is as the *Alexander dual* of I (Miller, 1998), and indeed some of the references above are written exclusively in terms of Alexander duality rather than irreducible decomposition. The second is as the *socle* of the vector space R/I', where R is the polynomial ring that I belongs to and $I' := I + \langle x_1^t, \ldots, x_n^t \rangle$ for some integer $t >> 0$. The socle is central to this paper, since what the Slice algorithm actually does is to compute a basis of the socle.

Section 2 introduces some basic notions we will need throughout the paper and Section 3 describes an as-simple-as-possible version of the Slice Algorithm. Section 4 contains improvements to this basic version of the algorithm and Section 5 discusses some heuristics that are inherent to the algorithm. Section 6 examines applications of irreducible decomposition, and it describes how the Slice Algorithm can use bounds to solve some optimization problems involving irreducible decomposition in less time than would be needed to actually compute the decomposition. Finally, Section 7 explores the practical aspects of the Slice Algorithm including benchmarks comparing it to other programs for irreducible decomposition.

The Slice Algorithm was in part inspired by an algorithm for Hilbert–Poincaré series due to Bigatti et al. (1993). The Slice Algorithm generalizes versions of the staircase-based algorithm due to Gao and Zhu (2005) (see Section 5.2) and the Label Algorithm due to Roune (2007) (see Section 5.5).

2. Preliminaries

This section briefly covers some notation and background on monomial ideals that are necessary to read the paper. We assume throughout the paper that I, J and S are monomial ideals in a polynomial ring R over some arbitrary field κ and with variables $x_1, \ldots, x_n$ where $n \geq 2$. We also assume that a, b, p, q and m are monomials in R. When presenting examples we use the variables x, y and z in place of x_1, x_2 and x_3 for increased readability.

2.1. Basic notions from monomial ideals

If $v \in \mathbb{N}^n$ then $x^v := x_1^{v_1} \cdots x_n^{v_n}$. We define $\sqrt{x^v} := x^{\mathrm{supp}(v)}$ where $(\mathrm{supp}(v))_i := \min(1, v_i)$. Define $\pi(m) := \frac{m}{\sqrt{m}}$ such that e.g. $\pi\left(x^{(0,1,2,3)}\right) = x^{(0,0,1,2)}$.

The rest of this section is completely standard. A *monomial ideal I* is an ideal generated by monomials, and $\min(I)$ is the unique minimal set of monomial generators. The ideal $\langle M \rangle$ is the ideal generated by the elements of the set M. The *colon ideal* $I : p$ is defined as $\langle m | mp \in I \rangle$.

An ideal I is *artinian* if there exists a $t \in \mathbb{N}$ such that $x_i^t \in I$ for $i = 1, \ldots, n$. A monomial of the form x_i^t is a *pure power*. A monomial ideal is *irreducible* if it is generated by pure powers. Thus $\langle x^2, y \rangle$ is irreducible while $\langle x^2 y \rangle$ is not. Note that $\langle x \rangle \subseteq \kappa[x, y]$ is irreducible and not artinian.

Every monomial ideal I can be written as an irredundant intersection of irreducible monomial ideals, and the set of ideals that appear in this intersection is uniquely given by I. This set is called the *irreducible decomposition* of I, and we denote it by $\mathrm{irr}(I)$. Thus $\mathrm{irr}\left(\langle x^2, xy, y^3 \rangle\right) = \left\{\langle x^2, y \rangle, \langle x, y^3 \rangle\right\}$.

The *radical* of a monomial ideal I is $\sqrt{I} := \left\langle \sqrt{m} \,|\, m \in \min(I) \right\rangle$. A monomial ideal I is *square free* if $\sqrt{I} = I$. A monomial ideal is (strongly) *generic* if no two distinct elements of $\min(I)$ raise the same variable x_i to the same non-zero power (Bayer et al., 1998; Miller et al., 2000). Thus $\langle x^2 y, xy^2 \rangle$ is generic while $\langle xyz^2, xy^2 z \rangle$ is not as both minimal generators raise x to the same power. In Section 7 we informally talk of a monomial ideal being more or less generic according to how many identical non-zero exponents there are in $\min(I)$.

A *standard monomial* of I is a monomial that does not lie within I. The *exponent vector* $v \in \mathbb{N}^n$ of a monomial m is defined by $m = x^v = x_1^{v_1} \cdots x_n^{v_n}$. Define $\deg_{x_i}(x^v) := v_i$. We draw pictures of monomial ideals in 2 and 3 dimensions by indicating monomials by their exponent vector and drawing line segments separating the standard monomials from the non-standard monomials. Thus Fig. 1(a) displays a picture of the monomial ideal $\langle x^6, x^5 y^2, x^2 y^4, y^6 \rangle$.

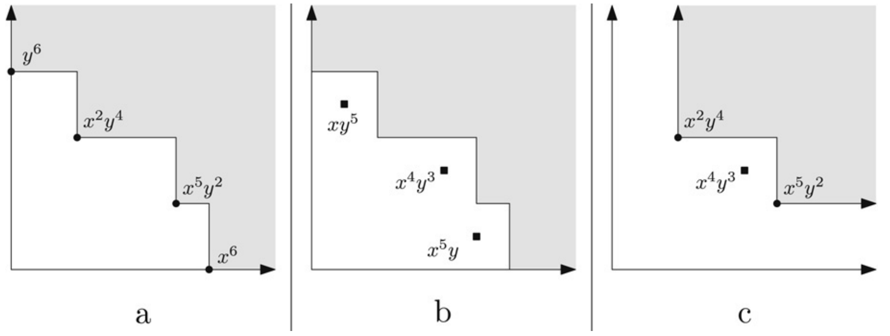

Fig. 1. Examples of monomial ideals.

2.2. Maximal standard monomials, socles and decompositions

In this section we look into socles and their relationship with irreducible decomposition. We also note the well known fact that the maximal standard monomials of I form a basis of the socle of R/I.

Given the generators $\min(I)$ of a monomial ideal I, the Slice Algorithm computes the maximal standard monomials of I. We will need some notation for this.

Definition 1 (*Maximal Standard Monomial*). A monomial m is a *maximal standard monomial* of I if $m \notin I$ and $mx_i \in I$ for $i = 1, \ldots, n$. The set of maximal standard monomials of I is denoted by $\operatorname{msm}(I)$.

The *socle* of R/I is the vector space of those $m \in R/I$ such that $mx_i = 0$ for $i = 1, \ldots, n$. It is immediate that $\{m + I \,|m \in \operatorname{msm}(I)\}$ is a basis of this socle.

Example 2. Let $I := \langle x^6, x^5y^2, x^2y^4, y^6\rangle$ be the ideal in Fig. 1(a). Then $\operatorname{msm}(I) = \{x^5y, x^4y^3, xy^5\}$ as indicated in Fig. 1(b). Let $J := \langle x^5y^2, x^2y^4\rangle$. Then $\operatorname{msm}(J) = \{x^4y^3\}$ as indicated in Fig. 1(c). Finally, $\operatorname{msm}(\langle x^5y^2\rangle) = \emptyset$.

We briefly describe the standard technique for obtaining $\operatorname{irr}(I)$ from $\operatorname{msm}(I)$ (Bayer et al., 1998). Choose some integer $t >> 0$ and define $\phi(x^m) = \left\langle x_i^{m_i+1} \,|m_i + 1 < t\right\rangle$.

Proposition 3 (Miller and Sturmfels (2005, ex. 5.8)). *The map ϕ is a bijection from $\operatorname{msm}(I+\langle x_1^t, \ldots, x_n^t\rangle)$ to $\operatorname{irr}(I)$.*

Example 4. Let $I := \langle x^2, xy\rangle$ and $I' := I + \langle x^t, y^t\rangle = \langle x^2, xy, y^3\rangle$ where $t = 3$. Then $\operatorname{msm}(I') = \{x, y^2\}$ which ϕ maps to $\{\langle x^2, y\rangle, \langle x\rangle\} = \operatorname{irr}(I)$.

2.3. Labels

We will have frequent use for the notion of a label.

Definition 5 (x_i-*label*). Let d be a standard monomial of I and let $m \in \min(I)$. Then m is an x_i-*label* of d if $m|dx_i$.

Note that if m is an x_i-label of d, then $\deg_{x_i}(m) = \deg_{x_i}(d) + 1$. Also, a standard monomial d is maximal if and only if it has an x_i-label m_i for $i = 1, \ldots, n$. So in that case $dx_1 \cdots x_n = \operatorname{lcm}_{i=1}^n m_i$.

Example 6. Let $I := \langle x^2, xz, y^2, yz, z^2\rangle$ be the ideal in Fig. 2(a). Then the maximal standard monomials of I are $\operatorname{msm}(I) = \{xy, z\}$. We see that z has xz as an x-label, yz as a y-label and z^2 as a z-label. Also, xy has x^2 as an x-label and y^2 as a y-label, while it has both of xz and yz as z-labels.

Let $J := I + \langle xy\rangle$ be the ideal in Fig. 2(b). Then $\operatorname{msm}(J) = \{x, y, z\}$. Note that even though xy divides $z \cdot xyz$, it is not a label of z, because it does not divide $z \cdot x$, $z \cdot y$ or $z \cdot z$.

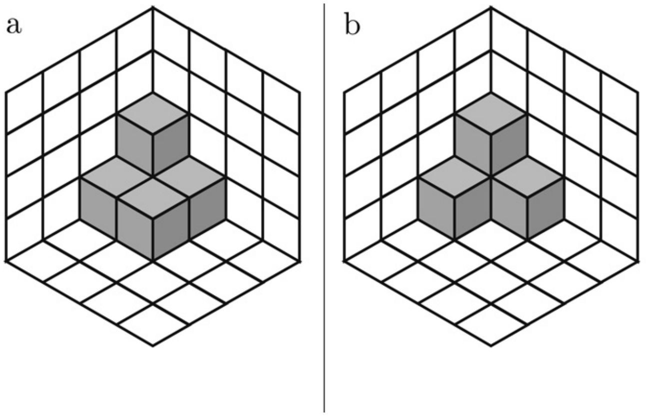

Fig. 2. Examples of monomial ideals.

3. The Slice Algorithm

In this section we describe a basic version of the Slice Algorithm. The Slice Algorithm computes the maximal standard monomials of a monomial ideal given the minimal generators of that ideal.

A fundamental idea behind the Slice Algorithm is to consider certain subsets of $\operatorname{msm}(I)$ that are represented as *slices*. We will define the meaning of the term slice shortly. The algorithm starts out by considering a slice that represents all of $\operatorname{msm}(I)$. It then processes this slice by splitting it into two simpler slices. This process continues recursively until the slices are simple enough so that it is easy to find any maximal standard monomials within them.

From this description, there are a number of details that need to be explained. Section 3.1 covers what slices are and how to split them while Section 3.2 covers the base case. Section 3.3 proves that the algorithm terminates and Section 3.4 contains a simple pseudo-code implementation of the algorithm.

3.1. Slices and splitting

In this section we explain what slices are and how to split them. We start off with the formal definition of a slice and its content.

Definition 7 (*Slice and Content*). A *slice* is a 3-tuple (I, S, q) where I and S are monomial ideals and q is a monomial. The *content* of a slice is defined by $\operatorname{con}(I, S, q) := (\operatorname{msm}(I) \setminus S)q$.

Example 8 shows how this definition is used.

Example 8. Let $I := \langle x^6, x^5y^2, x^2y^4, y^6 \rangle$ and $p := xy^3$. Then I is the ideal depicted in Fig. 3(a), where $\langle p \rangle$ is indicated by the dotted line and $\operatorname{msm}(I) = \{x^5y, x^4y^3, xy^5\}$ is indicated by the squares. We will compute $\operatorname{msm}(I)$ by performing a step of the Slice Algorithm.

Let I_1 be the ideal $I : p = \langle y^3, xy, x^4 \rangle$, as depicted in Fig. 3(b), where $\operatorname{msm}(I_1) = \{x^3, y^2\}$ is indicated by the squares. As can be seen by comparing Figs. 3(a) and 3(b), the ideal I_1 corresponds to the part of the ideal I that lies within $\langle p \rangle$. Thus it is reasonable to expect that $\operatorname{msm}(I_1)$ corresponds to the subset of $\operatorname{msm}(I)$ that lies within $\langle p \rangle$, which turns out to be true, since

$$\operatorname{msm}(I_1)\,p = \{x^4y^3, xy^5\} = \operatorname{msm}(I) \cap \langle p \rangle . \tag{1}$$

It now only remains to compute $\operatorname{msm}(I) \setminus \langle p \rangle$. Let $I_2 := \langle x^6, x^5y^2, y^6 \rangle$ as depicted in Fig. 3(c), where $\operatorname{msm}(I_2) := \{x^5y, x^4y^5\}$ is indicated by the squares. The dotted line indicates that we are ignoring everything inside $\langle p \rangle$. It happens to be that one of the minimal generators of I, namely x^2y^4, lies in the interior of $\langle p \rangle$, which allows us to ignore that minimal generator. We are looking at I_2 because

$$\operatorname{msm}(I_2) \setminus \langle p \rangle = \{x^5y\} = \operatorname{msm}(I) \setminus \langle p \rangle . \tag{2}$$

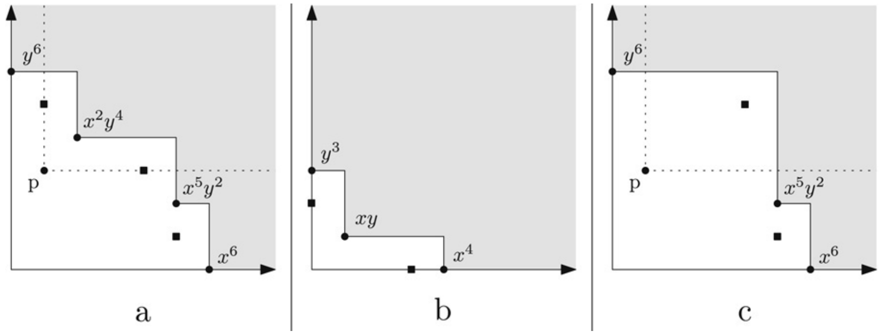

Fig. 3. Illustrations for Example 8.

By combining Eqs. (1) and (2), we can compute $\operatorname{msm}(I)$ in terms of $\operatorname{msm}(I_1)$, $\operatorname{msm}(I_2)$ and p.

Using the language of slices, we have split the slice $A := (I, \langle 0\rangle, 1)$ into the two slices $A_1 := (I_1, \langle 0\rangle, p)$ and $A_2 := (I_2, \langle p\rangle, 1)$. By Eqs. (1) and (2), we see that $\operatorname{con}(A_1) = \operatorname{msm}(I) \cap \langle p\rangle$ and $\operatorname{con}(A_2) = \operatorname{msm}(I) \setminus \langle p\rangle$. Thus

$$\operatorname{con}(A) = \operatorname{msm}(I) = \operatorname{con}(A_1) \cup \operatorname{con}(A_2)$$

where the union is disjoint.

Having defined slices and their content, we can now explain how to *split* a slice into two smaller slices. This is done by choosing some monomial p, called the *pivot*, and then to consider the following trivial equation.

$$\operatorname{con}(I, S, q) = (\operatorname{con}(I, S, q) \cap \langle qp\rangle) \cup (\operatorname{con}(I, S, q) \setminus \langle qp\rangle) . \tag{3}$$

The idea is to express both parts of this disjoint union as the content of a slice. This is easy to do for the last part, since

$$\operatorname{con}(I, S, q) \setminus \langle qp\rangle = \operatorname{con}(I, S + \langle p\rangle, q) .$$

Expressing the first part of the union as the content of a slice can be done using the following equation, which we will prove at the end of this section.

$$\operatorname{msm}(I) \cap \langle p\rangle = \operatorname{msm}(I : p)\, p$$

which implies that (see Example 8)

$$\operatorname{con}(I, S, q) \cap \langle qp\rangle = \operatorname{con}(I : p, S : p, qp) .$$

Thus we can turn Eq. (3) into the following.

$$\operatorname{con}(I, S, q) = \operatorname{con}(I : p, S : p, qp) \ \cup \ \operatorname{con}(I, S + \langle p\rangle, q) . \tag{4}$$

Eq. (4) is the basic engine of the Slice Algorithm. We will refer to it and its parts throughout the paper, and we need some terminology to facilitate this. The process of applying Eq. (4) is called a *pivot split*. We will abbreviate this to just *split* when doing so would not cause confusion.

Eq. (4) mentions three slices, and we give each of them a name. We call the left-hand slice (I, S, q) the *current slice*, since it is the slice we are currently splitting. We call the first right-hand slice $(I : p, S : p, qp)$ the *inner slice*, since its content is inside $\langle qp\rangle$, and we call the second right-hand slice $(I, S + \langle p\rangle, q)$ the *outer slice*, since its content is outside $\langle qp\rangle$.

It is not immediately obvious why it is easier to compute the outer slice's content $\operatorname{con}(I, S + \langle p\rangle, q)$ than it is to compute the current slice's content $\operatorname{con}(I, S, q)$. The following equation shows how it can be easier. See Proposition 11 for a proof.

$$\operatorname{msm}(I) \setminus S = \operatorname{msm}\left(I'\right) \setminus S,\ \ I' := \langle m \in \min(I) \,|\pi(m) \notin S\rangle . \tag{5}$$

This implies that $\operatorname{con}(I, S, q) = \operatorname{con}\left(I', S, q\right)$. In other words, we can discard any element m of $\min(I)$ where $\pi(m)$ lies within S. We will apply Eq. (5) whenever it is of benefit to do so, which it is when $\pi(\min(I)) \cap S \neq \emptyset$. This motivates the following definition.

Definition 9 (*Normal Slice*). A slice (I, S, q) is *normal* when $\pi\ (\min(I)) \cap S = \emptyset$.

Example 10. Let I, p and I_2 be as in Example 8. Then $(I, \langle p\rangle\,, 1)$ is the outer slice after a split on p. This slice is not normal, so we apply Eq. (5) to get the slice $(I_2, \langle p\rangle\,, 1)$, which is the slice A_2 from Example 8. See Fig. 3 for illustrations.

Proposition 11 proves the equations in this section, and it establishes some results that we will need later.

Proposition 11. *Let I be a monomial ideal and let p be a monomial. Then:*

(1) $\gcd(\min(I))$ *divides* $\gcd(\operatorname{msm}(I))$*;*
(2) $\operatorname{msm}(I) \cap \langle p\rangle = \operatorname{msm}(I \cap \langle p\rangle)$*;*
(3) *If* $p|\gcd(\min(I))$*, then* $\operatorname{msm}(I) = \operatorname{msm}(I : p)\, p$*;*
(4) $\operatorname{msm}(I) \cap \langle p\rangle = \operatorname{msm}(I : p)\, p$*;*
(5) $\operatorname{msm}(I) \setminus S = \operatorname{msm}\left(I'\right) \setminus S,\ \ I' := \langle m \in \min(I)\, | \pi\ (m) \notin S\,\rangle$*.*

Proof. (**1**) Let $d \in \operatorname{msm}(I)$. Let l_i be an x_i-label of d and let l_j be an x_j-label of d where $i \neq j$. This is possible due to the assumption in Section 2 that $n \geq 2$. Then $l_i|dx_i$ and $l_j|dx_j$, so $\gcd(\min(I))|\gcd(l_i, l_j)|d$.

(**2**) It follows from Lemma 12 and (1) that

$$\operatorname{msm}(I) \cap \langle p\rangle = \operatorname{msm}(I \cap \langle p\rangle) \cap \langle p\rangle = \operatorname{msm}(I \cap \langle p\rangle)\,.$$

(**3**) If $p|\gcd(\min(I))$ then $p|\gcd(\operatorname{msm}(I))$ by (1), whereby

$$\begin{aligned} d \in \operatorname{msm}(I) &\Leftrightarrow (d/p)p \notin I \text{ and } (d/p)x_i p \in I \text{ for } i = 1, \ldots, n \\ &\Leftrightarrow d/p \notin I : p \text{ and } (d/p)x_i \in I : p \text{ for } i = 1, \ldots, n \\ &\Leftrightarrow d/p \in \operatorname{msm}(I : p) \Leftrightarrow d \in \operatorname{msm}(I : p)\, p. \end{aligned}$$

(**4**) As $p|\gcd(\min(I \cap \langle p\rangle))$ and $(I \cap \langle p\rangle) : p = I : p$, we see that

$$\operatorname{msm}(I) \cap \langle p\rangle = \operatorname{msm}(I \cap \langle p\rangle) = \operatorname{msm}((I \cap \langle p\rangle) : p)\, p = \operatorname{msm}(I : p)\, p.$$

(**5**) Let $d \in \operatorname{msm}(I) \setminus S$ and let $l \in \min(I)$ be an x_i-label of d. Then $l \in \min\left(I'\right)$ since $\pi\ (l)\, |d \notin S$. Thus $dx_i \in I'$ since $l|dx_i$, so $d \in \operatorname{msm}\left(I'\right)$. Also $d \notin I \supseteq I'$.

Suppose instead that $d \in \operatorname{msm}\left(I'\right) \setminus S$. Then $dx_i \in I' \subseteq I$. If $d \in I$ then there would exist an $m \in \min(I) \setminus \min\left(I'\right)$ such that $m|d$, which is a contradiction since then $S \ni \pi\ (m)\, |m|d \notin S$. Thus $d \notin I$ whereby $d \in \operatorname{msm}(I)$. □

Lemma 12. *Let A, B and C be monomial ideals. Then $A \cap C = B \cap C$ implies that* $\operatorname{msm}(A) \cap C = \operatorname{msm}(B) \cap C$.

Proof. Let $d \in \operatorname{msm}(A) \cap C$. We will prove that $d \in \operatorname{msm}(B)$.

$\mathbf{d} \notin \mathbf{B}$: If $d \in B$ then $d \in B \cap C = A \cap C$ but $d \notin A$.

$\mathbf{dx_i} \in \mathbf{B}$: Follows from $dx_i \in A$ and $d \in C$ since then $dx_i \in A \cap C = B \cap C$. □

3.2. The base case

In this section we present the base case for the Slice Algorithm. A slice (I, S, q) is a *base case slice* if I is square free or if $x_1 \cdots x_n$ does not divide $\operatorname{lcm}(\min(I))$. Propositions 13 and 14 show why base case slices are easy to handle.

Proposition 13. *If $x_1 \cdots x_n$ does not divide* $\operatorname{lcm}(\min(I))$*, then* $\operatorname{msm}(I) = \emptyset$*.*

Proof. If $\operatorname{msm}(I) \neq \emptyset$ then there exists some $d \in \operatorname{msm}(I)$. Let $m \in \min(I)$ be an x_i-label of d. Then $x_i|m$, so $x_i|m|\operatorname{lcm}(\min(I))$. □

Proposition 14. *If I is square free and $I \neq \langle x_1, \ldots, x_n\rangle$, then* $\operatorname{msm}(I) = \emptyset$*.*

Proof. Let I be square free and let $d \in \operatorname{msm}(I)$. Let $m_i \in \min(I)$ be an x_i-label of d for $i = 1, \ldots, n$. Then $d = \pi\left(\operatorname{lcm}_{i=1}^{n} m_i\right) = 1$, so $m_i = x_i$. □

3.3. Termination and pivot selection

In this section we show that some quite weak constraints on the choice of the pivot are sufficient to ensure termination. Thus we leave the door open for a variety of different pivot selection strategies, which is something we will have much more to say about in Section 5.

We impose four conditions on the choice of the pivot p. These are presented below, and for each condition we explain why violating that condition would result in a split that there is no sense in carrying out. Note that the last two conditions are not necessary at this point to ensure termination, but they will become so after some of the improvements in Section 4 are applied.

- $\mathbf{p} \notin \mathbf{S}$: If $p \in S$, then the outer slice will be equal to the current slice.
- $\mathbf{p} \neq \mathbf{1}$: If $p = 1$, then the inner slice will be equal to the current slice.
- $\mathbf{p} \notin \mathbf{I}$: See Section 4.4 and Eq. (7) in particular.
- $\mathbf{p}|\pi\ (\mathrm{lcm}(\min\ (\mathbf{I})))$: See Section 4.5 and Eq. (8) in particular.

If a pivot satisfies these four conditions, then we say that it is *valid*. Proposition 15 shows that non-base case slices always admit valid pivots, and Proposition 16 states that selecting valid pivots ensures termination.

Proposition 15. *Let (I, S, q) be a normal slice for which no valid pivot exists. Then I is square free.*

Proof. Suppose I is not square free. Then there exists an x_i such that $x_i^2|m$ for some $m \in \min\ (I)$, which implies that $x_i \notin I$. Also, $x_i \notin S$ since $x_i|\pi\ (m)$ and (I, S, q) is normal. We conclude that x_i is a valid pivot. □

Proposition 16. *Selecting valid pivots ensures termination.*

Proof. Recall that the polynomial ring R is Noetherian, so it does not contain an infinite sequence of strictly increasing ideals. We will use this to show that the algorithm terminates. Suppose we are splitting a non-base case slice $A := (I, S, q)$ on a valid pivot where A_1 is the inner slice and A_2 is the outer slice.

Let f and g be functions mapping slices to ideals, and define them by the expressions $f(I, S, q) := S$ and $g(I, S, q) := \langle \mathrm{lcm}(\min\ (I)) \rangle$. Then the conditions on valid pivots and on non-base case slices imply that $f(A) \subseteq f(A_1)$, $f(A) \subsetneq f(A_2)$, $g(A) \subsetneq g(A_1)$ and $g(A) \subseteq g(A_2)$. Also, if we let A be an arbitrary slice and we let A' be the corresponding normal slice, then $f(A) \subseteq f(A')$ and $g(A) \subseteq g(A')$.

Thus f and g never decrease, and one of them strictly increases on the outer slice while the other strictly increases on the inner slice. Thus there does not exist an infinite sequence of splits on valid pivots. □

3.4. Pseudo-code

This section contains a pseudo-code implementation of the Slice Algorithm. Note that the improvements in Section 4 are necessary to achieve good performance.

The function selectPivot used below returns some valid pivot and can be implemented according to any of the pivot selection strategies presented in Section 5. A simple idea is to follow the proof of Proposition 15 and test each variable $x_1, \ldots, x_n$ for whether it is a valid pivot. If none of those are valid pivots, then I' in the pseudo-code below is square free.

Call the function con below with the parameters $(I, \langle 0 \rangle, 1)$ to obtain $\mathrm{msm}\ (I)$.

function con(I, S, q)
  let $I' := \langle m \in \min\ (I)\ |\pi\ (m) \notin S \rangle$

  if $x_1 \cdots x_n$ does not divide $\mathrm{lcm}(\min\ (I'))$ then return $\emptyset$
  if I' is square free and $I' \neq \langle x_1, \ldots, x_n \rangle$ then return $\emptyset$
  if I' is square free and $I' = \langle x_1, \ldots, x_n \rangle$ then return $\{q\}$

  let $p :=$ selectPivot(I', S)
  return con $(I' : p, S : p, qp)\ \cup\ $ con $(I', S + \langle p \rangle, q)$.

4. Improvements to the basic algorithm

This section contains a number of improvements to the basic version of the Slice Algorithm presented in Section 3.

4.1. Monomial lower bounds on slice contents

Let ql be a monomial lower bound on the slice (I, S, q) in the sense that $ql|d$ for all $d \in \text{con}\,(I, S, q)$. If we then perform a split on l, we can predict that the outer slice will be empty, whereby Eq. (4) specializes to Eq. (6), which shows that we can get the effect of performing a split while only having to compute a single slice.

$$\text{con}\,(I, S, q) = \text{con}\,(I : l, S : l, ql)\,. \tag{6}$$

Proposition 11 provides the simple monomial lower bound $\gcd(\min(I))$, while Proposition 17 provides a more sophisticated bound.

Proposition 17. *Let (I, S, q) be a slice and let $l(I) := \text{lcm}_{i=1}^{n} l_i$ where*

$$l_i := \frac{1}{x_i}\gcd(\min(I) \cap \langle x_i \rangle).$$

Then $ql(I)$ is a monomial lower bound on (I, S, q).

Proof. Let $d \in \text{msm}\,(I)$ and let m be an x_i-label of d. Then $x_i|m$, so $l_i x_i|m|dx_i$ whereby $l_i|d$. Thus $l(I)|d$. □

Example 18. Let $I := \langle x^2y, xy^2, yz, z^2\rangle$. Then $l(I) = y$ and Eq. (6) yields

$$\text{con}\,(I, \langle 0\rangle, 1) = \text{con}\,(I : y, \langle 0\rangle, y)\,,$$

where $I : y = \langle x^2, xy, z\rangle$. As $l(I : y) = x$ we can apply Eq. (6) again to get

$$\text{con}\,(I : y, \langle 0\rangle, y) = \text{con}\,(\langle x, y, z\rangle, \langle 0\rangle, xy) = \{xy\}\,.$$

We can improve on this bound using Lemma 20.

Definition 19 (*x_i-maximal*). A monomial $m \in \min(I)$ is *x_i-maximal* if

$$0 < \deg_{x_i}(m) = \deg_{x_i}(\text{lcm}(\min(I)))\,.$$

Lemma 20. *Let $d \in \text{msm}\,(I)$ and let m be an x_i-label of d. Suppose that m is x_j-maximal for some variable x_j. Then $x_i = x_j$.*

Proof. Suppose that $x_i \neq x_j$ and let l be an x_j-label of d. Then

$$\deg_{x_j}(m) \leq \deg_{x_j}(d) < \deg_{x_j}(l) \leq \deg_{x_j}(\text{lcm}(\min(I))) = \deg_{x_j}(m)\,. \quad \square$$

Corollary 21. *If $m \in \min(I)$ is x_i-maximal for two distinct variables, then $\text{msm}\,(I) = \text{msm}\,(I')$ where $I' := \langle \min(I) \setminus \{m\}\rangle$.*

Corollary 22. *Let (I, S, q) be a slice and let $l_i := \frac{1}{x_i}\gcd(M_i)$ where*

$$M_i := \left\{m \in \min(I) \,\middle|\, x_i \text{ divides } m \text{ and } m \text{ is not } x_j\text{-maximal for any } x_j \neq x_i\right\}.$$

Then $q\,\text{lcm}_{i=1}^{n} l_i$ is a monomial lower bound on (I, S, q).

It is possible to compute a more exact lower bound by defining $M_{(i,j)}$ and computing the gcd of pairs of minimal generators that could simultaneously be respectively x_i- and x_j-labels. However, we expect the added precision to be little, and the computational cost is high. If this is expanded from 2 to n variables, the lower bound is exact, but as costly to compute as the set $\text{msm}\,(I)$ itself.

Corollaries 21 and 22 allow us to make a slice simpler without changing its content, and they can be iterated until a fixed point is reached. We call this process *simplification*, and a slice is *fully simplified* if it is a fixed point of the process. Proposition 23 is an example of how simplification extends the reach of the base case.

Proposition 23. *Let $A := (I, S, q)$ be a fully simplified slice. If $|\min(I)| \leq n$ then A is a base case slice.*

Proof. Assume that $x_1 \cdots x_n | \operatorname{lcm}(\min(I))$. Then for each variable x_i, there must be some $m_i \in \min(I)$ that is x_i-maximal, and these m_i are all distinct. Since $|\min(I)| \leq n$ this implies that $\min(I) = \{m_1, \ldots, m_n\}$. Thus $M_i = \{m_i\}$ where M_i is defined in Corollary 22. Furthermore, since A is fully simplified, $\frac{1}{x_i}\gcd(M_i) = 1$, so $m_i = x_i$ and we are done. □

An argument much like that in the proof of Proposition 23 shows that (I, S, q) is a base case if all elements of $\min(I)$ are maximal. If there is exactly one element m of $\min(I)$ that is not maximal, then one can construct a new base case for the algorithm by trying out the possibility of that generator being an x_i-label for each $x_i|m$. One can do the same if there are k non-maximal elements for any $k \in \mathbb{N}$, but the time complexity of this is exponential in k, so it is slow for large k.

Our implementation does this for $k = 1, 2$, and implementing $k = 2$ did make our program a bit faster. We expect the effect of implementing $k = 3$ would be very small or even negative.

4.2. Independence splits

In this section we define I-independence and we show how this independence allows us to perform a more efficient kind of split. The content of this section was inspired by a similar technique for computing Hilbert–Poincaré series that was first suggested by Bayer and Stillman (1992) and described in more detail by Bigatti et al. (1993).

Definition 24. Let A, B be non-empty disjoint sets such that $A \cup B = \{x_1, \ldots, x_n\}$. Then A and B are *I-independent* if $\min(I) \cap \langle A \rangle \cap \langle B \rangle = \emptyset$.

In other words, A and B are I-independent if no element of $\min(I)$ is divisible by both a variable in A and a variable in B.

Example 25. Let $I := \left\langle x^4, x^2y^2, y^3, z^2, zt, t^2 \right\rangle$. Then $\{x, y\}$ and $\{z, t\}$ are I-independent. It then turns out that we can compute $\operatorname{msm}(I)$ independently for $\{x, y\}$ and $\{z, t\}$, which is reflected in the following equation.

$$\begin{aligned}\operatorname{msm}(I) &= \left\{x^3yz, x^3yt, xy^2z, xy^2t\right\} = \left\{x^3y, xy^2\right\} \cdot \{z, t\} \\ &= \operatorname{msm}(I \cap \kappa[x, y]) \cdot \operatorname{msm}(I \cap \kappa[z, t]) .\end{aligned}$$

Proposition 26 generalizes the observation in Example 25. The process of applying Proposition 26 is called an *independence split*.

Proposition 26. *If A, B are I-independent, then*

$$\operatorname{msm}(I) = \operatorname{msm}(I \cap \kappa[A]) \cdot \operatorname{msm}(I \cap \kappa[B]) .$$

Proof. Let $A' := I \cap \kappa[A]$ and $B' := I \cap \kappa[B]$. If $A' = \langle 0 \rangle$ then $\operatorname{msm}(I) = \emptyset$ by Proposition 13, so we can assume that $A' \neq \langle 0 \rangle$ and $B' \neq \langle 0 \rangle$. It holds that

$$\min(I) = \min\left(A'\right) \cup \min\left(B'\right)$$

so for monomials $a \in \kappa[A]$ and $b \in \kappa[B]$ we get that

$$ab \in I \Leftrightarrow a \in A' \text{ or } b \in B'$$

and thereby

$$ab \notin I \Leftrightarrow a \notin A' \text{ and } b \notin B'$$

which implies that

$$\begin{aligned}ab \in \operatorname{msm}(I) &\Leftrightarrow ab \notin I \text{ and } abx_i \in I \text{ for } x_i \in A \cup B \\ &\Leftrightarrow a \notin A' \text{ and } ax_i \in A' \text{ for } x_i \in A \text{ and} \\ &\ b \notin B' \text{ and } bx_i \in B' \text{ for } x_i \in B \\ &\Leftrightarrow a \in \operatorname{msm}\left(A'\right) \text{ and } b \in \operatorname{msm}\left(B'\right) . \quad \square\end{aligned}$$

Given a slice (I, S, q), this brings up the problem of what to do about S when A and B are I-independent but not S-independent. There are two simple ways to by-pass this issue entirely. The first is to only use pivots that are pure powers, in which case S will be generated by pure powers, so any two sets of variables will be S-independent. The second is to perform independence splits only when there is both I-independence and S-independence.

It is possible to deal with non-S-independence in a more direct way. First remove the elements of $\min(S) \cap \langle A\rangle \cap \langle B\rangle$ from $\min(S)$ when doing the independence split. Then remove those computed maximal standard monomials that lie within $\langle \min(S) \cap \langle A\rangle \cap \langle B\rangle\rangle$.

Example 27. Let I be as in Example 25 and consider the slice $(I, \langle x^3y, y^2z\rangle, 1)$. Then y^2z belongs to neither $\kappa[x, y]$ nor $\kappa[z, t]$, but we can do the independence split on the slice $(I, \langle x^3y\rangle, 1)$ which has content $\{xy^2\} \cdot \{z, t\} = \{xy^2z, xy^2t\}$. We then remove $\langle y^2z\rangle$ from this set, whereby $\mathrm{con}\left(I, \langle x^3y, x^2z\rangle, 1\right) = \{xy^2t\}$.

This idea can be improved by observing that when we know $\mathrm{con}\left(A', S_A, q_A\right)$, we can easily get the monomial lower bound $\gcd\left(\mathrm{con}\left(A', S_A, q_A\right)\right)$, and we can exploit this using the technique from Section 4.1. This might decrease the size of $\min(S) \cap \langle A\rangle \cap \langle B\rangle$, which can help us compute $\mathrm{con}\left(B', S_B, q_B\right)$.

Example 28. Let $I := \langle x^2, xy, xz, yz, a^2, ab, b^2\rangle$ and consider the slice $(I, \langle xa\rangle, 1)$. Then $\{x, y, z\}$ and $\{a, b\}$ are I-independent, and the first slice from the independence split is $\left(\langle x^2, xy, xz, yz\rangle, \langle 0\rangle, 1\right)$, where we are removing xa from $\min(S)$ since it crosses the split. That slice has content $\{x\}$, so x is a lower bound, and we can use the technique from Section 4.1 to go to the inner slice on a pivot split by x, which is

$$(I : x, \langle xa\rangle : x, x) = \left(\langle x, y, z, a^2, ab, b^2\rangle, \langle a\rangle, x\right).$$

Note that while ax crosses the split, $ax : x = a$ does not, so now we also have S-independence while originally we did not.

This leaves the question of how to detect I-independence. This can be done in space $O(n)$ and nearly in time $O(n\,|\min(I)|)$ using the classical union-find algorithm (Galler and Fisher, 1964; Cormen et al., 2001).[2] See the pseudo-code below, where D represents a disjoint-set data structure such that $\mathrm{union}(D, x_i, x_j)$ merges the set containing x_i with the set containing x_j. At the end D is the set of independent sets where $D = \{\{x_1, \ldots, x_n\}\}$ implies that there are no independent sets. The running time claimed above is achieved by using a suitable data structure for D along with an efficient implementation of union. See Galler and Fisher (1964) and Cormen et al. (2001) for details.

let $D := \{\{x_1\}, \ldots, \{x_n\}\}$.
for each $m \in \min(I)$ do
  pick an arbitrary x_i that divides m
  for each x_j that divides m do
    $\mathrm{union}(D, x_i, x_j)$.

This is an improvement on the $O(n^2\,|\min(I)|)$ algorithm for detecting independence suggested by Bigatti et al. (1993). That algorithm is similar to the one described here, the main difference being the choice of data structure.

[2] It can also be done in space $O(n^2)$ and in time $O(n\,|\min(I)| + n^2)$ by constructing a graph in a similar way and then finding connected components.

4.3. A base case of two variables

When $n = 2$ there is a well known and more efficient way to compute $\mathrm{msm}\,(I)$. This is also useful when an independence split has reduced n down to two.

Let $\{m_1, \ldots, m_k\} := \min\,(I)$ where $m_1,\ldots,m_k$ are sorted in ascending lexicographic order such that $x_1 > x_2$. Let $\tau(x^u, x^v) := x^{(v_1, u_2)}$. Then

$$\mathrm{msm}\,(I) = \{\tau(m_1, m_2), \tau(m_2, m_3), \ldots, \tau(m_{k-1}, m_k)\}\,.$$

4.4. Prune S

Depending on the selection strategy used, it is possible for the S in (I, S, q) to pick up a large number of minimal generators, which can slow things down. Thus there is a point to removing elements of $\min\,(S)$ when that is possible without changing the content of the slice. Eq. (7) allows us to do this.

$$\mathrm{con}\,(I, S, q) = \mathrm{con}\left(I, S', q\right), \quad S' := \langle m \in \min\,(S)\,|m \notin I\,\rangle\,. \tag{7}$$

Example 29. Consider the slice $\left(\left\langle x^2, y^2, z^2, yz\right\rangle, \langle xyz\rangle, 1\right)$. Then $p := x$ is a valid pivot, yielding the inner slice $\left(\left\langle x, y^2, z^2, yz\right\rangle, \langle yz\rangle, x\right)$. We can now apply Eq. (7) to turn this into $\left(\left\langle x, y^2, z^2, yz\right\rangle, \langle 0\rangle, x\right)$.

Proposition 16 states that the Slice Algorithm terminates, and we need to prove that this is still true when we use Eq. (7). Fortunately, the same proof can be used, except that the definition of the function f needs to be changed from $f(I, S, q) = S$ to $f(I, S, q) := I + S$. Note that the condition on a valid pivot p that $p \notin I$ is there to make this work.

4.5. More pruning of S

We can prune S using Eq. (8), and for certain splitting strategies this will even allow us to never add anything to S.

$$\mathrm{con}\,(I, S, q) = \mathrm{con}\left(I, S', q\right), \quad S' := \langle m \in \min\,(S)\,|m \text{ divides } \pi\,(\mathrm{lcm}(\min\,(I)))\,\rangle\,. \tag{8}$$

To prove this, observe that any $d \in \mathrm{con}\,(I, S, q)$ divides $\pi\,(\mathrm{lcm}(\min\,(I)))$.

Example 30. Consider the slice $\left(\left\langle x^2, xy, y^2\right\rangle, \langle 0\rangle, 1\right)$. Then $p := x$ is a valid pivot, yielding the normalized outer slice $\left(\left\langle xy, y^2\right\rangle, \langle x\rangle, 1\right)$. We can now apply Eq. (8) to turn this into $\left(\left\langle xy, y^2\right\rangle, \langle 0\rangle, 1\right)$.

Similarly, Eq. (8) will remove any generator of the form x_i^t from S. So if we use a pivot of the special form $p = x_i^t$, and we apply a normalization and Eq. (8) to the outer slice, we can turn Eq. (4) into

$$\mathrm{con}\,(I, S, q) = \mathrm{con}\,(I : p, S : p, qp) \;\cup\; \mathrm{con}\,(\langle \min\,(I) \setminus \langle px_i\rangle\rangle, S, q)$$

which for $S = \langle 0\rangle$ and $q = 1$ specializes to

$$\mathrm{msm}\,(I) = \mathrm{msm}\left(I : x_i^t\right) x_i^t \;\cup\; \mathrm{msm}\left(\left\langle \min\,(I) \setminus \left\langle x_i^{t+1}\right\rangle\right\rangle\right).$$

An implementer who does not want to deal with S might prefer this equation to the more general Eq. (4).

We need to prove that the algorithm still terminates when using Eqs. (7) and (8). We can use the same proof as in Proposition 16, except that we need to replace the definition of f from that proof with $f(I, S, q) := I + S + \left\langle x_1^{u_1}, \ldots, x_n^{u_n}\right\rangle$ where $x^u := \mathrm{lcm}(\min\,(I))$. Note that the condition on a valid pivot p that $p|\pi\,(\mathrm{lcm}(\min\,(I)))$ is there to make this work.

4.6. Minimizing the inner slice

A time-consuming step in the Slice Algorithm is to compute $I : p$ for each inner slice $(I : p, S : p, qp)$. By *minimizing*, we mean the process of computing $\min(I : p)$ from $\min(I)$, which is done by removing the non-minimal elements of $\min(I) : p := \{m : p \mid m \in \min(I)\}$ where $m : p := \frac{m}{\gcd(m,p)}$.

Proposition 31 makes it possible to do this using fewer divisibility tests than would otherwise be required. As seen by Corollary 32, this generalizes both statements of Bigatti (1997, Proposition 1) from p of the form x_i^t to general p.[3] See Bigatti et al. (1993, Section 6) for an even earlier form of these ideas.

Note that the techniques in this section also apply to computing intersections $I \cap \langle p\rangle$ of a monomial ideal with a principal ideal generated by a monomial, since $\min(I \cap \langle p\rangle) = \{\operatorname{lcm}(m, p) \mid m \in \min(I : p)\}$.

The most straightforward way to minimize $\min(I) : p$ is to consider all pairs of distinct $a, b \in \min(I) : p$ and then to remove b if $a|b$. It is well known that this can be improved by sorting $\min(I) : p$ according to some term order, in which case a pair only needs to be considered if the first term comes before the last. This halves the number of divisibility tests that need to be carried out.

We can go further than this, however, because we know that $\min(I)$ is already minimized. Proposition 31 shows how we can make use of this information.

Proposition 31. *Let x^a, x^b and x^p be monomials such that x^a does not divide x^b. Then $x^a : x^p$ does not divide $x^b : x^p$ if it holds for $i = 1, \ldots, n$ that $p_i < a_i \vee a_i \le b_i$.*

Proof. We prove the contrapositive statement, so suppose that x^a does not divide x^b and that $x^u := x^a : x^p$ divides $x^v := x^b : x^p$. Then there is an i such that $a_i > b_i$. As $\max(p_i, a_i) = u_i + p_i \le v_i + p_i = \max(p_i, b_i)$ we conclude that $p_i \ge a_i$. □

This allows us to draw some simple and useful conclusions.

Corollary 32. *Let $a, b \in \min(I)$ and let p be a monomial. Then $a : p$ does not divide $b : p$ if any one of the following two conditions is satisfied.*

(1) $\sqrt{a} = \sqrt{a : p}$
(2) $\gcd(a, p) | \gcd(b, p)$.

Corollary 33. *If $a \in \min(I)$ and $p|\pi(a)$, then $a : p$ is an element of $\min(I : p)$ and $a : p$ does not divide any other element of $\min(I) : p$.*

We can push Proposition 31 further than this. Fix some monomial p and define the binary relation $\prec$ on monomials by

$$a \prec b \text{ if there exists an } i \in \{1, \ldots, n\} \text{ such that } \deg_{x_i}(p) \ge \deg_{x_i}(a) > \deg_{x_i}(b).$$

It is immediate from Proposition 31 that if a does not divide b, and $a \not\prec b$, then $a : p$ does not divide $b : p$. So informally speaking it holds that $a \prec b$ when $a : p$ could divide $b : p$ even when taking Proposition 31 into account.

There is no point to using $\prec$ for the purpose of checking whether a single given element of $\min(I) : p$ divides another, as then we could just as well use an actual check for divisibility. To obtain a benefit from $\prec$, we partition $\min(I)$ into sets such that $\prec$ cannot tell the difference between any two elements from the same set. We define this partition by the equivalence classes of the binary relation $\sim$ defined on monomials by

$$a \sim b \text{ if } \gcd(a, p\sqrt{p}) = \gcd(b, p\sqrt{p}).$$

[3] This provides an answer to the statement of Bigatti (1997, p. 11) that "These remarks drastically reduce the number of divisibility tests, but they do not easily generalize for non-simple-power pivots, not even for power-products with only two indeterminates".

We note that if a, b, c, d are monomials and $a \sim b$ and $c \sim d$, then

$$a \prec c \Leftrightarrow b \prec d.$$

Now define $L(a)$ as the equivalence class that a belongs to according to $\sim$, i.e.

$$L(a) := \{m \in \min(I) \,|m \sim a\}.$$

We can now deal with the equivalence classes $L(a)$ instead of with each individual element of $\min(I)$. This is an advantage if $L(a)$ is large, since a single comparison will tell us how two whole equivalence classes compare according to $\prec$ instead of having to compare each element from the one equivalence class with each element from the other. We summarize the results in this section as follows.

Corollary 34. *Let p be a monomial. If $a, b \in \min(I)$ and $a \not\prec b$, then no element of $L(a) : p$ divides any element of $L(b) : p$. In particular, no element of $L(a) : p$ divides any other.*

Corollary 34 makes use of all the information provided by Proposition 31. Thus it is not surprising that it has Corollaries 32 and 33 as special cases.

This technique works best when most of the non-empty sets $L(a)$ contain considerably more than a single element, which is likely to be true e.g. if p is a small power of a single variable. Even in cases where most of the non-empty sets $L(a)$ consist of only a few elements, it will likely still pay off to consider $L(1)$ and $L(p\sqrt{p})$.[4]

Example 35. Let $I := \langle x^5y, x^2y^2, x^2z^3, xy^3, xyz^3, yz^2\rangle$ and $p := x^3$. Then,

$$\begin{array}{ll}
L(x^4) = \{x^5y\} & L(x^4) : p = \{x^2y\} \\
L(x^2) = \{x^2y^2, x^2z^3\} & L(x^2) : p = \{y^2, z^3\} \\
L(x) \;\, = \{xy^3, xyz^3\} & L(x) : p \;\, = \{y^3, yz^3\} \\
L(1) \;\, = \{yz^2\} & L(1) : p \;\, = \{yz^2\}.
\end{array}$$

We will process these sets from the top down. The set $L(x^4)$ is easy, since $p|\pi\left(x^5y\right)$, so we do not have to do any divisibility tests for $x^5y : p$.

Then comes $L(x^2)$. We have to test if any elements of $L(x^2) : p$ divide any elements of $L(x) : p$ or $L(1) : p$. It turns out that $x^2y^2 : p|xy^3 : p$ and $x^2z^3 : p|xyz^3 : p$, so we can remove all of $L(x)$ from consideration. We do not need to do anything more for $L(1)$, so we conclude that $\min(I : p) = \langle x^2y, y^2, z^3, yz^2\rangle$.

4.7. Reduce the size of exponents

Some applications require the irreducible decomposition of monomial ideals I where the exponents that appear in $\min(I)$ are very large. One example of this is the computation of Frobenius numbers (Roune, 2008b; Einstein et al., 2007).

This presents the practical problem that these numbers are larger than can be natively represented on a modern computer. This necessitates the use of an arbitrary precision integer library, which imposes a hefty overhead in terms of time and space. One solution to this problem is to report an error if the exponents are too large, as indeed the programs Monos (Milowski, 2007) and Macaulay 2 (Grayson and Stillman, 2007) do for exponents larger than $2^{15} - 1$ and $2^{31} - 1$ respectively.

In this section, we will briefly describe how to support arbitrarily large exponents without imposing any overhead except for a quick preprocessing step. The most time-consuming part of this preprocessing step is to sort the exponents.

[4] Our implementation in Frobby considers all the $L(a)$ when p is a pure power, while considering only $L(1)$ and $L(p\sqrt{p})$ when p is not a pure power. In general it should pay off to be more sophisticated about this when p is not a pure power, but p is almost always a pure power when using the default settings for Frobby, so in our case there would be little benefit.

Let f be a function mapping monomials to monomials such that $f(ab) = f(a)f(b)$ when $\gcd(a, b) = 1$. Suppose that $a|b \Rightarrow f(a)|f(b)$ and that f is injective for each i when restricted to the set $\left\{x_i^{v_i} \,|x^v \in \min(I)\right\}$. The reader may verify that then

$$x_1 \cdots x_n \operatorname{msm}(I) = f^{-1}(x_1 \cdots x_n \operatorname{msm}(\langle f(\min(I))\rangle)).$$

The idea is to choose f such that the exponents in $f(\min(I))$ are as small as possible, which can be done by sorting the exponents that appear in $\min(I)$. If this is done individually for each variable, then $|\min(I)|$ is the largest integer that can appear as an exponent in $f(\min(I))$. Thus we can compute $\operatorname{msm}(I)$ in terms of $\operatorname{msm}(\langle f(\min(I))\rangle)$, which does not require large integer computations.

Example 36. If $I := \left\langle x^{100}, x^{40}y^{20}, y^{90}\right\rangle$ then we can choose the function f such that $\langle f(\min(I))\rangle = \left\langle x^2, xy, y^2\right\rangle$.

The underlying mathematical idea used here is that it is the order rather than the value of the exponents that matters. This idea can also be found e.g. in Bayer et al. (1998, Remark 4.6), though in the present paper it is used for a different purpose.

4.8. *Label splits*

In this section we introduce *label splits*. These are based on some properties of labels which pivot splits do not make use of.

Let (I, S, q) be the current slice, and assume that it is fully simplified and not a base case slice. The first step of a label split is then to choose some variable x_i such that $\min(I) \cap \langle x_i\rangle \neq \{x_i\}$. Let $L := \{x^u \in \min(I)\,|u_i = 1\}$. Then L is non-empty since the current slice is fully simplified. Assume for now that $|L| = 1$ and let $l \in L$.

Observe that if $d \in \operatorname{msm}(I)$, then $\frac{l}{x_i}|d$ if and only if l is an x_i-label of d, which is true if and only if x_i does not divide d. This and Eq. (3.1) imply that

$$\begin{aligned}
\operatorname{con}(I, S, 1) \setminus \langle x_i\rangle &= \operatorname{con}(I, S, 1) \cap \left\langle \frac{l}{x_i}\right\rangle = \operatorname{con}\left(I : \frac{l}{x_i}, S : \frac{l}{x_i}, \frac{l}{x_i}\right)\\
\operatorname{con}(I, S, 1) \cap \langle x_i\rangle &= \operatorname{con}(I : x_i, S : x_i, x_i)
\end{aligned}$$

whereby

$$\operatorname{con}(I, S, q) = \operatorname{con}(I : x_i, S : x_i, qx_i) \;\cup\; \operatorname{con}\left(I : \frac{l}{x_i}, S : \frac{l}{x_i}, q\frac{l}{x_i}\right).$$

This equation describes a label split on x_i in the case where $|L| = 1$. In general $|L|$ can be larger than one, so let $L = \{l_1, \ldots, l_k\}$ and define

$$I_j := I\colon \frac{l_j}{x_i}, \quad S_j := \left(S + \left\langle \frac{l_1}{x_i}, \ldots, \frac{l_{j-1}}{x_i}\right\rangle\right) : \frac{l_j}{x_i}, \quad q_j := q\frac{l_j}{x_i}$$

for $j = 1, \ldots, k$. Then $\operatorname{con}\left(I_j, S_j, q_j\right)$ is the set of those $d \in \operatorname{con}(I, S, q)$ such that l_j is an x_i-label of d, and such that none of the monomials $l_1, \ldots, l_{j-1}$ are x_i-labels of d. This implies that

$$\operatorname{con}(I, S, q) = \operatorname{con}(I : x_i, S : x_i, qx_i) \bigcup_{j=1}^{k} \operatorname{con}\left(I_j, S_j, q_j\right) \tag{9}$$

where the union is disjoint. This equation defines a label split on x_i.

An advantage of label splits is that if I is artinian, $S = \langle 0\rangle$ and $|L| = 1$, then none of the slices on the right-hand side of Eq. (9) are empty. These conditions will remain true throughout the computation if the ideal is artinian and generic and we perform only label and independence splits. Example 37 shows that a label split can produce empty slices when $|L| > 1$.

Example 37. Let $I := \langle x^4, y^4, z^4, xy, xz\rangle$. We perform a label split on x, where $l_1 := xy$ and $l_2 := xz$, which yields the following equation.

$$\begin{aligned}
\operatorname{con}(I, \langle 0\rangle, 1) &= \operatorname{con}\left(\langle x^3, y, z\rangle, \langle 0\rangle, x\right) && (\text{this is } (I : x_i, S : x_i, qx_i))\\
&\cup \operatorname{con}\left(\langle x, y^3, z^4\rangle, \langle 0\rangle, y\right) && (\text{this is } (I_1, S_1, q_1))\\
&\cup \operatorname{con}\left(\langle x, y^4, z^3\rangle, \langle y\rangle, z\right) && (\text{this is } (I_2, S_2, q_2))\\
&= \left\{x^3\right\} \cup \left\{y^3z^3\right\} \cup \emptyset = \left\{x^3, y^3z^3\right\}.
\end{aligned}$$

The reason that (I_2, S_2, q_2) is empty is that both l_1 and l_2 are x-labels of y^3z^3.

Using only label splits according to the VarLabel strategy discussed in Section 5.5 makes the Slice Algorithm behave as a version of the Label Algorithm (Roune, 2007). See the External Corner Algorithm (Einstein et al., 2007) for an earlier form of some of the ideas behind the Label Algorithm.

5. Split selection strategies

We have not specified how to select the pivot monomial when doing a pivot split, or when to use a label split and on what variable. The reason for this is that there are many possible ways to do it, and it is not clear which one is best. Indeed, it may be that one split selection strategy is far superior to everything else in one situation, while being far inferior in another. Thus we examine several different selection strategies in this section.

We are in the fortunate situation that an algorithm for computing Hilbert–Poincaré series has an analogous issue of choosing a pivot (Bigatti et al., 1993). Thus we draw on the literature on that algorithm to get interesting pivot selection strategies (Bigatti et al., 1993; Bigatti, 1997), even though these strategies do have to be adapted to work with the Slice Algorithm. The independence and label strategies are the only ones among the strategies below that are not similar to a known strategy for the Hilbert–Poincaré series algorithm.

It is assumed in the discussion below that the current slice is fully simplified and not a base case slice. Note that all the strategies select valid pivots only. We examine the practical merit of these strategies in Section 7.2.

5.1. The minimal generator strategy

We abbreviate this as *MinGen*.

Selection This strategy picks some element $m \in \min(I)$ that is not square free and then selects the pivot $\pi(m)$.

Analysis This strategy chooses a pivot that is maximal with respect to the property that it removes at least one minimal generator from the outer slice. This means that the inner slice is easy, while the outer slice is comparatively hard since we can be removing as little as a single minimal generator.

5.2. The pure power strategies

There are three pure power strategies.

Selection These strategies choose a variable x_i that maximizes $|\min(I) \cap \langle x_i\rangle|$ provided that $x_i^2 \mid \operatorname{lcm}(\min(I))$. Then they choose some positive integer e for which it holds that $x_i^{e+1} \mid \operatorname{lcm}(\min(I))$ and select the pivot x_i^e.

The strategy *Minimum* selects $e := 1$ and the strategy *Maximum* selects $e := \deg_{x_i}(\operatorname{lcm}(\min(I))) - 1$. The strategy *Median* selects e as the median exponent of x_i from the set $\min(I) \cap \langle x_i\rangle$.

Note that the Minimum strategy makes the Slice Algorithm behave as a version of the staircase-based algorithm due to Gao and Zhu (2005).

Analysis The pure power strategies have the advantage that the minimization techniques described in Section 4.6 work especially well for pure power pivots. Maximum yields an easy inner slice and a hard outer slice, while Minimum does the opposite. Median achieves a balance between the two.

5.3. The random GCD strategy

We abbreviate this as *GCD*.

Selection Let x_i be a variable that maximizes $|\min(I) \cap \langle x_i \rangle|$, and pick three random monomials $m_1, m_2, m_3 \in \min(I) \cap \langle x_i \rangle$. Then the pivot is chosen to be $p := \pi(\gcd(m_1, m_2, m_3))$. If $p = 1$, then the GCD strategy fails, and we might try again or use a different selection strategy.

Analysis We consider this strategy because a similar strategy has been found to work well for the Hilbert–Poincaré series algorithm mentioned above.

5.4. The independence strategy

We abbreviate this as *Indep*.

Selection The independence strategy picks two distinct variables x_i and x_j, and then selects the pivot $p := \pi\left(\gcd\left(\min(I) \cap \left\langle x_i x_j \right\rangle\right)\right)$. If $p = 1$, then the independence strategy fails, and we might try again or use a different selection strategy.

Analysis The pivot p is the maximal monomial that will make every minimal generator that is divisible by both x_i and x_j disappear from the outer slice. The idea behind this is to increase the chance that we can perform an independence split on the outer slice while having a significant impact on the inner slice as well.

5.5. The label strategies

There are three label strategies.

Selection These strategies choose a variable x_i such that $\min(I) \cap \langle x_i \rangle \neq \{x_i\}$ and then perform a label split on x_i. The strategy MaxLabel maximizes $|\min(I) \cap \langle x_i \rangle|$, VarLabel minimizes i and MinLabel minimizes $|\{x^u \in \min(I) \,|\, v_i = 1\}|$ while breaking ties according to MaxLabel.

Note that the VarLabel strategy makes the Slice Algorithm behave as a version of the Label Algorithm (Roune, 2007).

Analysis MaxLabel chooses the variable that will have the biggest impact, while MinLabel avoids considering as many empty slices by keeping $|\min(S)|$ small. VarLabel is being considered due to its relation to the Label Algorithm.

6. Applications to optimization

Sometimes we compute a socle or an irreducible decomposition because we want to know some property of it rather than because we are interested in knowing the socle or decomposition itself. This kind of situation often has the form

$$\text{maximize } v(J) \text{ subject to } J \in \operatorname{irr}(I),$$

where v is some function mapping $\operatorname{irr}(I)$ to $\mathbb{R}$. We call such a problem an Irreducible Decomposition Program (IDP). As described in Sections 6.3 and 6.4, applications of IDP include computing the integer programming gap, Frobenius numbers and the codimension of a monomial ideal.

The Slice Algorithm can solve some IDPs in much less time than it would need to compute all of $\operatorname{irr}(I)$, and that is the subject of this section. Section 6.1 explains the general principle of how to do this, while Section 6.2 provides some useful techniques for making use of the principle. Sections 6.3 and 6.4 present examples of how to apply these techniques.

6.1. Branch and bound using the slice algorithm

In this section we explain the general principle of solving IDPs using the Slice Algorithm.

The first issue is that the Slice Algorithm is concerned with computing maximal standard monomials while IDPs are about irreducible decomposition. We deal with this by using the function ϕ from Section 2.2 to reformulate an IDP of the form

$$\text{maximize } v'(J) \text{ subject to } J \in \operatorname{irr}\left(I'\right)$$

into the form

$$\text{maximize } v(d) \text{ subject to } d \in \operatorname{msm}(I)$$

where $v(d) := v'(\phi(d))$ and $I := I' + \left\langle x_1^t, \ldots, x_n^t \right\rangle$ for some $t >> 0$.

It is a simple observation that there is no reason to compute all of $\operatorname{msm}(I)$ before beginning to pick out the element that yields the greatest value of v. We might as well not store $\operatorname{msm}(I)$, and only keep track of the greatest value of v found so far.

We define a function $b(I, S, q)$ that maps slices (I, S, q) to real numbers to be an *upper bound* if $d \in \operatorname{con}(I, S, q)$ implies that $v(d) \leq b(I, S, q)$. We will now show how to use such an upper bound b to turn the Slice Algorithm into a *branch and bound* algorithm.

Suppose that the Slice Algorithm is computing the content of a slice (I, S, q), and that $b(I, S, q)$ is less than or equal to the greatest value of v found so far. Then we can skip the computation of $\operatorname{con}(I, S, q)$, since no element of $\operatorname{con}(I, S, q)$ improves upon the greatest value of v found so far.

We can take this a step further by extending the idea of monomial lower bounds from Section 4.1. The point there was that if we can predict that the outer slice of some pivot split will be empty, then we should perform that split and ignore the outer slice. That way we get the benefit of a split while only having to examine a single slice. In the same way, if we can predict that one slice of some pivot split will not be able to improve upon the best value found so far, we should perform the split and ignore the non-improving slice. The hard part is to come up with a way to find pivots where such a prediction can be made. Sections 6.3 and 6.4 provide examples of how this can be done.

A prerequisite for applying the ideas in this section is to construct a bound b. It is not possible to say how to do this in general, since it depends on the particulars of the problem at hand, but Section 6.2 presents some ideas that can be helpful.

6.2. Monomial bounds

In this section we present some ideas that can be useful when constructing upper bounds for IDPs of the form

$$\text{maximize } v(d) \text{ subject to } d \in \operatorname{msm}(I)\,.$$

Suppose that v is decreasing in the sense that if $a|b$ then $v(a) \geq v(b)$. Then $b(I, S, q) := v(q)$ is an upper bound, since if $d \in \operatorname{con}(I, S, q)$ then $q|d$, so $v(d) \leq v(q)$.

Suppose instead that v is increasing in the sense that if $a|b$ then $v(a) \leq v(b)$. Then $b(I, S, q) := v(q\pi(\operatorname{lcm}(\min(I))))$ is an upper bound, since if $d \in \operatorname{con}(I, S, q)$ then $d|q\pi(\operatorname{lcm}(\min(I)))$ by Proposition 38, so $v(d) \leq v(q\pi(\operatorname{lcm}(\min(I))))$. Any monomial upper bound on $\operatorname{con}(I, S, q)$ yields an upper bound in the same way.

Proposition 38. *If* $d \in \operatorname{msm}(I)$ *then* $d|\pi(\operatorname{lcm}(\min(I)))$.

Proof. Let $d \in \operatorname{msm}(I)$ and let $m_i \in \min(I)$ be an x_i-label of d for $i = 1, \ldots, n$. Then $d = \pi\left(\operatorname{lcm}_{i=1}^n m_i\right)$ divides $\pi(\operatorname{lcm}(\min(I)))$. $\square$

Sections 6.3 and 6.4 provide examples of how these ideas can be applied.

6.3. Linear IDPs, codimension and Frobenius numbers

Let $r \in \mathbb{R}^n$ and define the function $v_r(x^u) := u \cdot r$. Then we refer to IDPs of the form (10) as *linear*.

$$\text{maximize } v_r(d) \text{ subject to } d \in \operatorname{msm}(I)\,. \tag{10}$$

It is well known that the *codimension* of a monomial ideal I' equals the minimal number of generators of the ideals in $\operatorname{irr}(I')$. The reader may verify that this is exactly the optimal value of the IDP (10) if we let $I := \sqrt{I'} + \langle x_1^2, \ldots, x_n^2 \rangle$ and $r = (1, \ldots, 1)$, noting the well known fact that the codimension of an ideal does not change by taking the radical. This implies that solving IDPs is NP-hard since computing codimensions of monomial ideals is NP-hard (Bayer and Stillman, 1992, Proposition 2.9). Linear IDPs are also involved in the computation of Frobenius numbers (Roune, 2008b; Einstein et al., 2007).

Let us return to the general situation of r and I being arbitrary. Our goal in this section is to solve IDPs of the form (10) efficiently by constructing a bound. The techniques from Section 6.2 do not immediately seem to apply, since v_r need neither be increasing nor decreasing. To deal with this problem, we will momentarily restrict our attention to some special cases.

Let $a \in \mathbb{R}^n_{\geq 0}$ be a vector of n non-negative real numbers, and define $v_a(x^u) := u \cdot a$. We will construct a bound for the IDP

$$\text{maximize } v_a(d) \text{ subject to } d \in \operatorname{msm}(I)\,.$$

This is now easy to do, since v_a is increasing so that we can use the techniques from Section 6.2. Specifically, $v_a(d) \leq v_a(q\pi\,(\operatorname{lcm}(\min(I))))$ for all $d \in \operatorname{con}(I, S, q)$.

Similarly, let $b \in \mathbb{R}^n_{\leq 0}$ be a vector of n non-positive real numbers, and define $v_b(x^u) := u \cdot b$. We will construct a bound for the IDP

$$\text{maximize } v_b(d) \text{ subject to } d \in \operatorname{msm}(I)\,.$$

This is also easy, since v_b is decreasing so that we can use the techniques from Section 6.2. Specifically, $v_b(d) \leq v_b(q)$ for all $d \in \operatorname{con}(I, S, q)$.

We now return to the issue of constructing a bound for IDP (10). Choose $a \in \mathbb{R}^n_{\geq 0}$ and $b \in \mathbb{R}^n_{\leq 0}$ such that $r = a + b$. Then we can combine the bounds for v_a and v_b above to get a bound for v. So if $d \in \operatorname{con}(I, S, q)$, then

$$v(d) = v_a(d) + v_b(d) \leq v_a(q\pi\,(\operatorname{lcm}(\min(I)))) + v_b(q) =: b(I, S, q).$$

Now that we have a bound b, we follow the suggestion from Section 6.1 that we should devise a way to find pivots where we can predict that one of the slices will be non-improving. Let (I, S, q) be the current slice and let $x^u := \operatorname{lcm}(\min(I))$.

Suppose that r_i is positive and consider the outer slice (I', S', q') from a pivot split on x_i. We can predict that the exponent of x_i in our monomial upper bound will decrease from $\deg_{x_i}(q) + u_i - 1$ down to $\deg_{x_i}(q)$. Thus we get that

$$r_i(u_i - 1) \leq b(I, S, q) - b(I', S', q'),$$

whereby

$$b(I', S', q') \leq b(I, S, q) - r_i(u_i - 1),$$

which implies that the outer slice is non-improving if

$$b(I, S, q) - r_i(u_i - 1) \leq \tau, \tag{11}$$

where τ is the best value found so far. We can do a similar thing if r_i is negative by considering the value of $\deg_{x_i}(q')$ on the inner slice of a pivot split on $x_i^{u_i - 1}$.

As we will see in Section 7.4, this turns out to make things considerably faster. One reason is that checking Eq. (11) for each variable x_i is very fast, because it only involves computations on the single monomial $\operatorname{lcm}(\min(I))$. Another reason is that we can iterate this idea, as moving to the inner or outer slice can reduce the bound, opening up the possibility for doing the same thing again. We can also apply the simplification techniques from Section 4.1 after each successful application of Eq. (11).

6.4. The integer programming gap

Let $c \in \mathbb{Q}^n$ and $d \in \mathbb{Z}^k$, and let A be a $k \times n$ integer matrix. The *integer programming gap* of a bounded and feasible integer program of the form

$$\text{minimize } c \cdot x \text{ subject to } Ax = d,\ x \in \mathbb{N}^n$$

is the difference between its optimal value and the optimal value of its *linear programming relaxation*, which is defined as the linear program

$$\text{minimize } c \cdot x \text{ subject to } Ax = d,\ x \in \mathbb{R}^n_{\geq 0}.$$

A paper of Hoşten and Sturmfels (2007) describes a way to compute the integer programming gap that involves the sub-step of computing an irreducible decomposition $\operatorname{irr}\left(I'\right)$ of a monomial ideal I'. Our goal in this section is to show that this sub-step can be reformulated as an IDP whose objective function v satisfies the property that $a|b \Rightarrow v(a) \leq v(b)$ whereby we can construct a bound using the technique from Section 6.2.

First choose $t >> 0$ and let $I := I' + \left\langle x_1^{t+1}, \ldots, x_n^{t+1} \right\rangle$ so that we can consider $\operatorname{msm}(I)$ in place of $\operatorname{irr}\left(I'\right)$. Define $\psi : \mathbb{N}^n \mapsto \mathbb{N}^n$ by the expression

$$(\psi(u))_i := \begin{cases} u_i, & \text{for } u_i < t, \\ 0, & \text{for } u_i \geq t. \end{cases}$$

So if $t = 4$ then $\psi(3, 4, 5) = (3, 0, 0)$. Define $v(u)$ for $u \in \mathbb{N}^n$ as the optimal value of the following linear program. We say that this linear program is *associated to u*.

$$\begin{array}{rl} \text{maximize} & c \cdot (\psi(u) - w) \\ \text{subject to} & A(\psi(u) - w) = 0,\ w \in \mathbb{R}^n \\ \text{and} & w_i \geq 0 \text{ for those } i \text{ where } u_i < t. \end{array}$$

The IDP that the algorithm of Hoşten and Sturmfels (2007) needs to solve is then

$$\text{maximize } v(u) \text{ subject to } x^u \in \operatorname{msm}(I).$$

By Proposition 39, we can construct a bound for this IDP using the technique from Section 6.2. Note that we can use this bound to search for non-improving outer slices for pivots of the form x_i in the exact same way as described for linear IDPs in Section 6.3.

Proposition 39. *The function v satisfies the condition that $x^a | x^b \Rightarrow v(a) \leq v(b)$.*

Proof. Let $e_i \in \mathbb{N}^n$ be a vector of zeroes except that the ith entry is 1. It suffices to prove that $v(u) \leq v(u + e_i)$ for $u \in \mathbb{N}^n$. Let $w \in \mathbb{R}^n$ be some optimal solution to the linear program associated to u. We will construct a feasible solution w' to the linear program associated to $u + e_i$ that has the same value. We will ensure this by making w' satisfy the equation $\psi(u) - w = \psi(u + e_i) - w'$.

The case $\mathbf{u_i + 1 < t}$: Let $w' := w + e_i$.

The case $\mathbf{u_i + 1 = t}$: Let $w' := w - u_i e_i$. Note that the non-negativity constraint on the ith entry of w' is lifted due to $u_i + 1 = t$.

The case $\mathbf{u_i + 1 > t}$: Let $w' := w$. Note that this case is not relevant to the computation since no upper bound will be divisible by x_i^{t+1}. □

7. Benchmarks

We have implemented the Slice Algorithm in the software system Frobby (Roune, 2008a), and in this section we use Frobby to explore the Slice Algorithm's practical performance. Section 7.1 describes the test data we use, Section 7.2 compares a number of split selection strategies, Section 7.3 compares Frobby to other programs and finally Section 7.4 evaluates the impact of the bound optimization from Section 6.

Table 1
Information about the test data

Name	n	$\lvert\min(I)\rvert$	$\lvert\mathrm{irr}(I)\rvert$	max. exponent
generic-v10g40	10	40	52,131	29,987
generic-v10g80	10	80	163,162	29,987
generic-v10g120	10	120	411,997	29,991
generic-v10g160	10	160	789,687	29,991
generic-v10g200	10	200	1,245,139	29,991
nongeneric-v10g100	10	100	19,442	10
nongeneric-v10g150	10	150	52,781	10
nongeneric-v10g200	10	200	79,003	10
nongeneric-v10g400	10	400	193,638	10
nongeneric-v10g600	10	600	318,716	10
nongeneric-v10g800	10	800	435,881	10
nongeneric-v10g1000	10	1,000	571,756	10
squarefree-v20g100	20	100	3,990	1
squarefree-v20g500	20	500	11,613	1
squarefree-v20g2000	20	2,000	22,796	1
squarefree-v20g4000	20	4,000	30,015	1
squarefree-v20g6000	20	6,000	30,494	1
squarefree-v20g8000	20	8,000	35,453	1
squarefree-v20g10000	20	10,000	37,082	1
J51	89	3,036	9	1
J60	89	3,432	10	1
smalldual	20	160,206	20	9
frobn12d11	12	56,693	4,323,076	87
frobn13d11	13	170,835	24,389,943	66
k4	16	61	139	3
k5	31	13,313	76,673	6
model4vars	16	20	64	2
model5vars	32	618	6,550	4
tcyc5d25p	125	3,000	20,475	1
tcyc5d30p	150	4,350	40,920	1

7.1. The test data

In this section we briefly describe the test data that we use for the benchmarks. Table 1 displays some information about each input. The data used is publicly available from the supplementary data included in the Appendix.

Generation of random monomial ideals

The random monomial ideals referred to below were generated using the following algorithm, which depends on a parameter $N \in \mathbb{N}$. We start out with the zero ideal. A random monomial is then generated by pseudo-randomly generating each exponent within the range $[0, N]$. Then this monomial is added as a minimal generator of the ideal if it does not dominate or divide any of the previously added minimal generators of the ideal. This process continues until the ideal has the desired number of minimal generators. The random number generator used was the standard C rand() function.

Description of the input data

This list provides information on each test input.

generic These ideals are nearly generic due to choosing $N = 30.000$.
nongeneric These ideals are non-generic due to choosing $N = 10$.
square free These ideals are square free due to choosing $N = 1$.

Table 2
Empirical comparison of split selection strategies

Strategy	generic-v10g200	nongeneric-v10g400	squarefree-v20g10000	J60
MaxLabel	13 s	13 s	224 s	19 s
MinLabel	14 s	13 s	203 s	2 s
VarLabel	18 s	13 s	213 s	13 s
Minimum	13 s	14 s	19 s	3 s
Median	**12 s**	**11 s**	**20 s**	**3 s**
Maximum	35 s	43 s	19 s	3 s
MinGen	59 s	201 s	19 s	4 s
Indep	13 s	12 s	21 s	3 s
GCD	18 s	20 s	19 s	3 s

J51, J60 These ideals were generated using the reverse engineering algorithm of Jarrah et al. (2006), and they were kindly provided by M. Paola Vera Licona. They have the properties of having many variables, being square free and having a small irreducible decomposition.

smalldual This ideal has been generated as the Alexander dual of a random monomial ideal with 20 minimal generators in 20 variables. Thus it has many minimal generators and a small decomposition.

t5d25p, t5d30p These ideals are from the computation of cyclic tropical polytopes, and they have the special property of being generated by monomials of the form $x_i x_j$ (Block and Yu, 2006). They were kindly provided by Josephine Yu.

k4, k5 These ideals come with the program Monos written by Milowski (2007). They are involved in computing the integer programming gap of a matrix (Hoşten and Sturmfels, 2007).

model4vars, model5vars These ideals come from computations on algebraic statistical models, and they were generated using the program 4ti2 (4ti2 team , 2006) with the help of Seth Sullivant.

frobn12d11, frobn13d11 These ideals come from the computation of the Frobenius number of respectively 12 and 13 random 11-digit numbers (Roune, 2008b).

7.2. Split selection strategies

In this section we evaluate the split selection strategies described in Section 5. Table 2 shows the results.

The most immediate conclusion that can be drawn from Table 2 is that label splits do well on ideals that are somewhat generic, while they fare less well on square free ideals when compared with pivot splits. It is a surprising contrast to this that the MinLabel strategy is best able to deal with J60.

Table 2 also shows that the pivot strategies are very similar on square free ideals. This is not surprising, as the only valid pivots on such ideals have the form x_i, and the pivot strategies all pick the same variable.

The final conclusion we will draw from Table 2 is that the Median strategy is the best split selection strategy on these ideals, so that is the strategy we will use in the rest of this section. The Minimum strategy is a very close second.

7.3. Empirical comparison to other programs

In this section we compare our implementation in Frobby (Roune, 2008a) of the Slice Algorithm to other programs that compute irreducible decompositions. There are two well known fast algorithms for computing irreducible decompositions of monomial ideals.

Alexander Duality This algorithm is based on Alexander duality and intersection of ideals. Its advantage is speed on highly non-generic ideals. See Miller (1998), Hoşten and Smith (2002) and Milowski (2004).

Scarf Complex This algorithm enumerates the facets of the Scarf complex by walking from one facet to adjacent ones. The advantage of the algorithm is speed for generic ideals, while the drawback is that highly non-generic ideals lead to high memory consumption and bad performance. This is because the algorithm internally transforms the input ideal into a corresponding generic ideal that can have a much larger decomposition. See Bayer et al. (1998) and Milowski (2004)

We have benchmarked the following three programs.

Macaulay 2 version 1.0 This program (Grayson and Stillman, 2007) includes an implementation of the Alexander Dual Algorithm. The time-consuming parts of the algorithm are written in C++.

Monos version 1.0 RC 2 This program[5] (Milowski, 2007) is Alexander Milowski's implementation in Java of both the Alexander Dual Algorithm and the Scarf Complex Algorithm.

Frobby version 0.6 This C++ program (Roune, 2008a) is our implementation of the Slice Algorithm.

How these programs compare depend on what kind of input is used, so we use all the inputs described in Section 7.1 to get a complete picture. In order to run these benchmarks in a reasonable amount of time, we have allowed each program to run for one hour on each input and no longer. Each program has been allowed to use 512 MB of RAM and no more, not including the space used by other programs. We use the abbreviation OOT for "out of time", OOM for "out of memory" and RE for "runtime error".

The benchmarks have all been run on the same Linux machine with a 2.4 GHz Intel Celeron CPU. The reported time is the user time as measured by the Unix command line utility "time".

All of the data can be seen on Table 3. The data show that Frobby is faster than the other programs on all inputs except for smalldual. This is because the Alexander Dual Algorithm does very well on this kind of input, due to the decomposition being very small compared to the number of minimal generators. The decompositions of J51 and J60 are also small compared to the number of minimal generators, though from the data not small enough to make the Alexander Dual Algorithm win out.

It is clear from Table 3 that Macaulay 2 has the fastest implementation of the Alexander Dual Algorithm when it does not run out of memory. As expected, the Scarf Complex Algorithm beats the Alexander Dual Algorithm on generic ideals, while the positions are reversed on square free ideals.

As can be seen from Table 3, the other programs frequently run out of memory. In the case of Macaulay 2, this is clearly in large part due to some implementation issue. However, the issue of consuming large amounts of memory is fundamental to both the Alexander Dual Algorithm and the Scarf Complex Algorithm, since it is necessary for them to keep the entire decomposition in memory, and these decompositions can be very large — see frobn13d11 as an example. The Slice Algorithm does not have this issue.

An advantage of the Slice Algorithm is that the inner and outer slices of a pivot split can be computed in parallel, making it simple to make use of multiple processors. The Scarf Complex Algorithm is similarly easy to parallelize, while the Alexander Dual Algorithm is not as amenable to a parallel implementation.

Although Frobby, Macaulay 2 and Monos can make use of no more more than a single processor, multicore systems are fast becoming ubiquitous. Algorithmic research and implementations must adapt or risk wasting almost all of the available processing power on a typical system. For example, a non-parallel implementation on an eight-way system will use only 13% of the available processing power.

7.4. The bound technique

In this section we examine the impact of using the bound technique from Section 6 to compute Frobenius numbers.

[5] There are two different versions of Monos that have both been released as version 1.0. We are using the newest version, which is the version 1.0 RC2 that was released in 2007.

Table 3
Empirical comparison of programs for irreducible decomposition

Input	Frobby (Slice)	Macaulay2 (Alexander)	Monos (Alexander)	Monos (Scarf)
generic-v10g40	<1 s	512 s[a]	1632 s	14 s
generic-v10g80	1 s	OOM	OOT	82 s
generic-v10g120	4 s	OOM	OOT	332 s
generic-v10g160	8 s	OOM	OOT	OOM
generic-v10g200	12 s	OOM	OOT	OOM
nongeneric-v10g100	<1 s	138 s[a]	770 s	191 s
nongeneric-v10g150	1 s	OOM	OOT	OOT
nongeneric-v10g200	1 s	OOM	OOT	OOT
nongeneric-v10g400	4 s	OOM	OOT	OOM
nongeneric-v10g600	8 s	OOM	OOT	OOM
nongeneric-v10g800	11 s	OOM	OOT	OOM
nongeneric-v10g1000	15 s	OOM	OOT	OOM
squarefree-v20g100	<1 s	17 s	27 s	1015 s
squarefree-v20g500	1 s	80 s	608 s	OOM
squarefree-v20g2000	4 s	OOM	OOT	OOM
squarefree-v20g4000	9 s	OOM	OOT	OOM
squarefree-v20g6000	13 s	OOM	OOT	OOT
squarefree-v20g8000	19 s	OOM	OOT	OOT
squarefree-v20g10000	21 s	OOM	OOT	OOT
J51	2 s	8 s	6 s	OOM
J60	3 s	10 s	7 s	OOM
smalldual	1961 s	RE	559 s	RE
frobn12d11	285 s	OOM	OOT	OOT
frobn13d11	2596 s	RE	OOT	RE
k4	<1 s	2 s	2 s	22 s
k5	108 s	OOM	OOT	OOM
model4vars	<1 s	1 s	1 s	2 s
model5vars	2 s	OOM	896 s	OOM
tcyc5d25p	7 s	OOM	OOM	OOM
tcyc5d30p	16 s	OOM	OOT	OOM

[a] This time has been included in spite of using more than 512 MB of memory.

Table 4
Empirical evaluation of the bound technique

Strategy	frob-n11d11 without bound	frob-n11d11 using bound	frob-n12d11 without bound	frob-n12d11 using bound
Frob	**66 s**	**22 s**	**204 s**	**93 s**
Median	76 s	35 s	256 s	147 s
Maximum	226 s	189 s	805 s	712 s
Minimum	731 s	761 s	3205 s	3388 s

Table 4 displays the time[6] needed to solve a Frobenius problem IDP with and without using the bound technique for some split selection strategies. We have included a new selection strategy Frob that works as Median, except that it selects the variable that maximizes the increase of the lower bound value on the inner slice.

It is clear from Table 4 that the Frob and Median split selection strategies are much better than the others for computing Frobenius numbers, and that Frob is a bit better than Median. We also see that

[6] It may be noted that the time used on frobn12d11 when using the Median split selection strategy has been reported in Table 3 to be 285 s, while Table 4 reports it to be 256 s. The 29 additional seconds are accounted for by the time needed to generate a textual representation of the output msm (I) and writing it to disk. When computing the Frobenius number without the bound technique, Frobby may still compute all of msm (I), but the only output that needs to be generated is just the Frobenius number itself.

applying the bound technique to the Frob split selection strategy reduces the runtime to somewhere between one third and one half of what it is when not using the bound technique.

Appendix. Supplimentary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.jsc.2008.08.002.

References

4ti2 team, 2006. 4ti2 version 1.3 – A software package for algebraic, geometric and combinatorial problems on linear spaces. Available at: http://www.4ti2.de.

Bayer, D., Peeva, I., Sturmfels, B., 1998. Monomial resolutions. Mathematical Research Letters 5 (1–2), 31–46.

Bayer, D., Stillman, M., 1992. Computation of hilbert functions. Journal of Symbolic Computation 14 (1), 31–50.

Bigatti, A.M., 1997. Computation of Hilbert-Poincaré series. Journal of Pure and Applied Algebra 119 (3), 237–253. Available at: http://cocoa.dima.unige.it/research/publications.html.

Bigatti, A.M., Conti, P., Robbiano, L., Traverso, C., 1993. A divide and conquer algorithm for Hilbert-Poincaré series, multiplicity and dimension of monomial ideals. In: Applied Algebra, Algebraic Algorithms and Error-correcting Codes (San Juan, PR, 1993). In: Lecture Notes in Comput. Sci., vol. 673. Springer, Berlin, pp. 76–88. Available at: http://cocoa.dima.unige.it/research/publications.html.

Block, F., Yu, J., 2006. Tropical convexity via cellular resolutions. Journal of Algebraic Combinatorics 24 (1), 103–114.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2001. Data structures for disjoint sets. In: Introduction to Algorithms, 2nd ed. MIT Press and McGraw-Hill, pp. 498–524. (Chapter 21).

Einstein, D., Lichtblau, D., Strzebonski, A., Wagon, S., 2007. Frobenius numbers by lattice point enumeration. Integers 7. Available at: http://www.integers-ejcnt.org/.

Galler, B.A., Fisher, M.J., 1964. An improved equivalence algorithm. Communications of the ACM 7 (5), 301–303.

Gao, S., Zhu, M., 2005. Irreducible decomposition of monomial ideals. SIGSAM Bulletin 39 (3), 99–99. URL: http://portal.acm.org/citation.cfm?id=1113458.

Grayson, D.R., Stillman, M.E., 2007. Macaulay 2 version 1.0, a software system for research in algebraic geometry. Available at: http://www.math.uiuc.edu/Macaulay2/.

Hoşten, S., Smith, G.G., 2002. Monomial ideals. In: Eisenbud, D., Grayson, D.R., Stillman, M., Sturmfels, B. (Eds.), Computations in Algebraic Geometry with Macaulay 2. In: Algorithms and Computation in Mathematics, vol. 8. Springer-Verlag, pp. 73–100.

Hoşten, S., Sturmfels, B., 2007. Computing the integer programming gap. Combinatorica 27 (3).

Jarrah, A.S., Laubenbacher, R., Stigler, B., Stillman, M., 2006. Reverse-engineering of polynomial dynamical systems. Advances in Applied Mathematics.

Miller, E., 1998. Alexander duality for monomial ideals and their resolutions. ArXiv:math/9812095.

Miller, E., Sturmfels, B., 2005. Combinatorial Commutative Algebra. In: Graduate Texts in Mathematics, vol. 227. Springer.

Miller, E., Sturmfels, B., Yanagawa, K., 2000. Generic and cogeneric monomial ideals. Journal of Symbolic Computation 29 (4–5), 691–708. Available at: http://www.math.umn.edu/~ezra/papers.html.

Milowski, R.A., 2004. Computing irredundant irreducible decompositions and the scarf complex of large scale monomial ideals. Master's thesis, San Francisco State University, Available at: http://www.milowski.com/.

Milowski, R.A., 2007. Monos version 1.0 rc2 – A software package for monomial computations. Available at: http://code.google.com/p/monos-algebra/.

Roune, B.H., 2007. The label algorithm for irreducible decomposition of monomial ideals. ArXiv:0705.4483.

Roune, B.H., 2008a. Frobby version 0.6 – A software system for computations with monomial ideals. Available at: http://www.broune.com/frobby/.

Roune, B.H., 2008b. Solving thousand digit frobenius problems using grobner bases. Journal of Symbolic Computation 43 (1), 1–7. URL: http://www.broune.com/.

Sturmfels, B., Sullivant, S., 2006. Combinatorial secant varieties. Pure and Applied Mathematics Quarterly 2 (3).

Sullivant, S., 2008. Combinatorial symbolic powers. Journal of Algebra 319 (1), 115–142.