Journal of Software for Algebra and Geometry

```
gap> g:= SymmetricGroup( 4 );
Sym( [ 1 .. 4 ] )
gap> tbl:= CharacterTable( g );;  HasIrr( tbl );
false
gap> tblmod2:= CharacterTable( tbl, 2 );
BrauerTable( Sym( [ 1 .. 4 ] ), 2 )
gap> tblmod2 = CharacterTable( tbl, 2 );
true
gap> tblmod2 = BrauerTable( tbl, 2 );
true
gap> tblmod2 = BrauerTable( g, 2 );
true
gap> libtbl:= CharacterTable( "M" );
CharacterTable( "M" )
gap> CharacterTableRegular( libtbl, 2 );
BrauerTable( "M", 2 )
gap> BrauerTable( libtbl, 2 );
fail
gap> CharacterTable( "Symmetric", 4 );
CharacterTable( "Sym(4)" )
gap> ComputedBrauerTables( tbl );
[ , BrauerTable( Sym( [ 1 .. 4 ] ), 2 ) ]
```

```
i5 : betti(t,Weights=>{1,0})

            0 1  2  3 4
o5 = total: 1 4 13 14 4
         0: 1 .  .  . .
         1: . 2  2  4 2
         2: . 2  5  6 .
         3: . .  4  . 2
         4: . .  .  4 .
         5: . .  2  . .

o5 : BettiTally
i6 : betti(t,Weights=>{0,1})

            0 1  2  3 4
o6 = total: 1 4 13 14 4
         0: 1 .  .  . .
         1: . 2  2  4 2
         2: . 2  5  6 .
         3: . .  4  . 2
         4: . .  .  4 .
         5: . .  2  . .

o6 : BettiTally
i7 : t1 = betti(t,Weights=>{1,1})

            0 1  2  3 4
o7 = total: 1 4 13 14 4
         0: 1 .  .  . .
         1: . .  .  . .
         2: . .  .  . .
         3: . 2  .  . .
         4: . .  .  . .
         5: . 2  .  . .
         6: . .  1  . .
         7: . .  8  6 .
         8: . .  4  8 4

o7 : BettiTally
i8 : peek t1

o8 = BettiTally{(0, {0, 0}, 0) => 1 }
                (1, {2, 2}, 4) => 2
                (1, {3, 3}, 6) => 2
                (2, {3, 7}, 10) => 2
                (2, {4, 4}, 8) => 1
                (2, {4, 5}, 9) => 4
                (2, {5, 4}, 9) => 4
                (2, {7, 3}, 10) => 2
                (3, {4, 7}, 11) => 4
                (3, {5, 5}, 10) => 6
                (3, {7, 4}, 11) => 4
                (4, {5, 7}, 12) => 2
                (4, {7, 5}, 12) => 2
```

```
ring r1 = 32003,(x,y,z),ds;
int a,b,c,t=11,5,3,0;
poly f = x^a+y^b+z^(3*c)+x^(c+2)*y^(c-1)+x^
          x^(c-2)*y^c*(y^2+t*x)^2;
option(noprot);
timer=1;
ring r2 = 32003,(x,y,z),dp;
poly f=imap(r1,f);
ideal j=jacob(f);
vdim(std(j));
==> 536
vdim(std(j+f));
==> 195
timer=0;  // reset timer
```

Primary decomposition of squarefree pseudomonomial ideals

Alan Veliz-Cuba

vol 12   2022

JSAG **12** (2022), 27–32
https://doi.org/10.2140/jsag.2022.12.27

The Journal of Software for
Algebra and Geometry

Primary decomposition of squarefree pseudomonomial ideals

ALAN VELIZ-CUBA

ABSTRACT: A *squarefree pseudomonomial* in $K[x_1, x_2, \dots, x_n]$ is a polynomial of the form $z_1 z_2 \cdots z_r$ such that $z_j \in \{x_{i_j}, x_{i_j} - 1\}$ for some $i_j \in \{1, \dots, n\}$ and the i_j's are distinct. A *squarefree pseudomonomial ideal* is an ideal generated by squarefree pseudomonomials. Here we present the package *Pseudomonomial-PrimaryDecomposition* for the computation of the primary decomposition of squarefree pseudomonomial ideals. Internally, the package performs computation using bitwise logical operations on integers.

1. Introduction.

A squarefree pseudomonomial of a ring $K[x_1, x_2, \dots, x_n]$ is a polynomial of the form $z_1 z_2 \cdots z_r$ such that $z_j \in \{x_{i_j}, x_{i_j} - 1\}$ for some $i_j \in \{1, \dots, n\}$ and the i_j's are distinct. A squarefree pseudomonomial ideal is an ideal that is generated by squarefree pseudomonomials. Squarefree pseudomonomial ideals appear in the study of problems such as reverse engineering of finite dynamical systems and in the study of neural coding, where the primary decomposition of these ideals contains important combinatorial information [Jarrah et al. 2007; Veliz-Cuba 2012; Curto et al. 2013].

In this manuscript we present the theory, algorithm, and implementation of the primary decomposition computation for squarefree pseudomonomial ideals. We use the structure of squarefree pseudomonomial ideals to design an efficient algorithm to compute their primary decomposition, similar to existing algorithms for monomial ideals.

2. Theory.

The theory behind the algorithm is based on the work presented in [Curto et al. 2013] following similar properties of squarefree monomial ideals [Miller and Sturmfels 2005; Sturmfels 2002]. Here we summarize those results. In this section K denotes a field.

Lemma 2.1. *Consider a squarefree pseudomonomial ideal $J \subset K[x_1, \dots, x_n]$. If f is a squarefree pseudomonomial such that $f \in \langle J, z\rangle$, where $z = x_t$ or $z = 1 - x_t$ for some $t \in \{1, \dots, n\}$, then $f \in \langle z\rangle$ or $(1 - z) f \in J$.*

Proof. Adapted from [Curto et al. 2013]. Since f is a squarefree pseudomonomial, we have that up to a sign $f = z_1 z_2 \cdots z_r$ such that $z_j \in \{x_{i_j}, 1 - x_{i_j}\}$ and the i_j's are distinct. Since J is a squarefree pseudomonomial ideal, we can write

$$J = \langle z g_1, \dots, z g_k, (1 - z) f_1, \dots, (1 - z) f_l, h_1, \dots, h_m\rangle,$$

MSC2010: primary 13P05, 14Q99, 68W30; secondary 92B05, 92B20.
Keywords: primary decomposition, minimal primes, monomials, pseudomonomials.
PseudomonomialPrimaryDecomposition version 0.3

where g_j, f_j, and h_j are squarefree pseudomonomials that contain no factor z or $1-z$. Thus, f is of the form

$$f = \sum_{j=1}^{k} u_j z g_j + (1-z)\sum_{j=1}^{l} v_j f_j + \sum_{j=1}^{m} w_j h_j + yz,$$

for some $u_j, v_j, w_j, y \in K[x_1, \dots, x_n]$. Setting $z = 0$ (i.e., $x_t = 0$ if $z = x_t$ and $x_t = 1$ if $z = 1 - x_t$) and multiplying by $1-z$ yields

$$(1-z)f|_{z=0} = (1-z)\sum_{j=1}^{l} v_j|_{z=0} f_j + (1-z)\sum_{j=1}^{m} w_j|_{z=0} h_j,$$

where "$|_{z=0}$" means evaluated at $z = 0$. Note that since $(1-z)f_j, h_j \in J$, we have that $(1-z)f|_{z=0} \in J$. Now we have 3 cases.

If z is a factor of f, then $f \in \langle z\rangle$.

If $1-z$ is a factor of f, say $f = (1-z)z_2\cdots z_r$, then $f = (1-z)f|_{z=0} \in J$.

If neither z or $1-z$ is a factor of f, then $(1-z)f = (1-z)f|_{z=0} \in J$.

In all cases we obtain $f \in \langle z\rangle$ or $(1-z)f \in J$. □

Proposition 2.2. *Consider a squarefree pseudomonomial ideal $J \subseteq K[x_1, \dots, x_n]$ and let $z_1 \cdots z_r$ be a squarefree pseudomonomial. Then, $\left\langle J, \prod_{j=1}^{r} z_i\right\rangle = \bigcap_{i=1}^{r}\langle J, z_i\rangle$.*

Proof. Adapted from [Curto et al. 2013]. Note that we can consider $z_j \in \{x_{i_j}, 1 - x_{i_j}\}$.

It is clear that $\left\langle J, \prod_{i=1}^{r} z_i\right\rangle \subseteq \cap_{i=1}^{r}\langle J, z_i\rangle$.

We now consider $f \in \cap_{i=1}^{r}\langle J, z_i\rangle$. We have three cases.

If $f \in \langle z_i\rangle$ for all i, then $f \in \left\langle \prod_{i=1}^{r} z_i\right\rangle \subseteq \left\langle J, \prod_{i=1}^{r} z_i\right\rangle$.

If $f \notin \langle z_i\rangle$ for some i's, then without loss of generality we consider $f \notin \langle z_i\rangle$ for $i = 1, \dots, m$ and $f \in \langle z_i\rangle$ for $i = m+1, \dots, r$. We can write

$$f = (1-z_1)f + z_1(1-z_2)f + \cdots + z_1\cdots z_{m-1}(1-z_m)f + z_1\cdots z_m f.$$

From Lemma 2.1, $(1-z_j)f \in J$ for $j = 1, \dots, m$, since $f \in \langle J, z_i\rangle$ and $f \notin \langle z_i\rangle$. Then, the first m terms are in J. Also, since $f \in \langle z_i\rangle$ for $i = m+1, \dots, r$, it follows that $f \in \langle z_{m+1}\cdots z_r\rangle$ and then $z_1\cdots z_m f \in \langle z_1\cdots z_r\rangle$. Thus $f \in \left\langle J, \prod_{i=1}^{r} z_i\right\rangle$. □

Proposition 2.2 provides a concrete way to write an ideal as an intersection of simpler ideals. Using this result recursively, we will obtain at the end an intersection of ideals of the form $\langle x_{i_1} - a_1, \dots, x_{i_k} - a_k\rangle$, where $a_j \in \{0, 1\}$, which provide the primary decomposition. We remark that Proposition 2.2 is similar to Lemma 2.1 in [Sturmfels 2002, monomial ideals chapter]. Also, since each ideal in the intersection is prime, the ideal is radical, and so the primary decomposition is the set of minimal primes of the ideal. Before describing the algorithm we show two examples.

Example 2.3. Consider $I = \langle x_1, x_2(1-x_3), (1-x_2)x_4\rangle$. Using Proposition 2.2 we obtain

$$\begin{aligned} I &= \langle x_1, x_2(1-x_3), 1-x_2\rangle \cap \langle x_1, x_2(1-x_3), x_4\rangle \\ &= \langle x_1, x_2, 1-x_2\rangle \cap \langle x_1, 1-x_3, 1-x_2\rangle \cap \langle x_1, x_2, x_4\rangle \cap \langle x_1, 1-x_3, x_4\rangle. \end{aligned}$$

Since the first ideal in the last equality contains 1, it is not proper, thus we obtain

$$I = \langle x_1, 1-x_3, 1-x_2\rangle \cap \langle x_1, x_2, x_4\rangle \cap \langle x_1, 1-x_3, x_4\rangle.$$

Example 2.4. Consider $I = \langle x_1(1-x_2), (1-x_1)x_2, x_1x_2\rangle$. Using Proposition 2.2 we obtain

$$\begin{aligned} I &= \langle x_1(1-x_2), (1-x_1)x_2, x_1\rangle \cap \langle x_1(1-x_2), (1-x_1)x_2, x_2\rangle \\ &= \langle (1-x_1)x_2, x_1\rangle \cap \langle x_1(1-x_2), x_2\rangle \\ &= \langle 1-x_1, x_1\rangle \cap \langle x_2, x_1\rangle \cap \langle x_1, x_2\rangle \cap \langle 1-x_2, x_2\rangle = \langle x_1, x_2\rangle. \end{aligned}$$

3. Algorithm.

The algorithm uses Proposition 2.2 recursively as well as some simplifications seen in Examples 2.3 and 2.4.

Algorithm for the computation of the primary decomposition of squarefree pseudomonomial ideals. The algorithm is adapted from [Curto et al. 2013].

Input: A squarefree pseudomonomial ideal $J \subseteq K[x_1, \dots, x_n]$.
Output: Primary decomposition of J, a list $\mathcal{P}$ of primary ideals such that $J = \cap_{I\in\mathcal{P}} I$.

```
Step 1. Set $\mathcal{P} = \varnothing$ and $\mathcal{D} = \{J\}$. Remove redundant generators.
Step 2. For each ideal $I \in \mathcal{D}$ define $\mathcal{D}_I = \{\langle I, z_1\rangle, \dots, \langle I, z_m\rangle\}$, where $z_1z_2\cdots z_m$ is one of the generators
        of $I$ of degree greater than 1 and $z_j \in \{x_{i_j}, x_{i_j}-1\}$.
Step 3. For each $\langle I, z\rangle$ found in Step 2, remove redundant generators. Also, remove ideals that have 1
        as a generator.
Step 4. Define $\mathcal{T} = \bigcup_{I\in\mathcal{D}} \mathcal{D}_I$ and set $\mathcal{P} := \mathcal{P} \cup \{I \in \mathcal{T} : I$ has linear generators only$\}$ and $\mathcal{D} := \{I \in \mathcal{T} :$
        $I$ has nonlinear generators$\}$.
Step 5. Repeat Steps 2–4 until $\mathcal{D} = \varnothing$.
Step 6. Remove redundant ideals of $\mathcal{P}$.
```

The algorithm is guaranteed to give the primary decomposition because:

- Proposition 2.2 guarantees that $I = \bigcap_{K\in\mathcal{D}_I} K$ in Step 2.
- At every step we have $J = \bigcap_{I\in\mathcal{P}} I \cap \bigcap_{I\in\mathcal{D}} I$.
- Step 2 reduces the number of nonlinear generators, so the algorithm will terminate after a finite number of iterations.
- Step 6 results in $J = \cap_{I\in\mathcal{P}} I$ such that each ideal in $\mathcal{P}$ has linear generators only and hence is primary.

4. Implementation and examples.

We implemented the algorithm described in Section 3 using bitwise logical operations on integers in [Macaulay2]. A squarefree pseudomonomial

$$P = x_{i_1}\cdots x_{i_r}(x_{k_1}-1)\cdots(x_{k_m}-1)$$

is encoded as a list of two integers $B_P := \left\{\sum_{j=1}^r 2^{i_j}, \sum_{j=1}^m 2^{k_j}\right\}$. Note that the integer $\sum_{j=1}^r 2^{i_j}$ represents the binary string with 1's at positions i_j and 0's elsewhere, and the integer $\sum_{j=1}^{m} 2^{k_j}$ represents the binary string with 1's at positions k_j and 0's elsewhere. A similar approach was used to compute the Gröbner basis for Boolean polynomials [Hinkelmann and Arnold 2010]. The advantage of using bitwise representation is that comparison between polynomials can be done quickly. For example, consider a polynomial P with bitwise representation $B_P = \{a, b\}$ and a polynomial Q with bitwise representation $B_Q = \{c, d\}$. Then, to check whether P divides Q it is enough to check whether $a\&c = a$ and $b\&d = b$, where "&" is the bitwise AND operator.

The package *PrimaryDecompositionPseudomonomial* uses bitwise logical operations on integers. However, the input and output of the package are in polynomial form, so the user does not need to manipulate polynomials in bitwise form. To ensure a correct implementation of the package, we computed the primary decomposition of over 10,000,000 squarefree pseudomonomial ideals and compared our results with those of the current implementation of `primaryDecomposition` in Macaulay2. In all cases, our results were correct. The package *PrimaryDecompositionPseudomonomial* exports two functions: `primaryDecompositionPseudomonomial` and `isSquarefreePseudomonomialIdeal`; the first computes the primary decomposition and the second determines whether an ideal is a squarefree pseudomonomial ideal. Below are some examples in Macaulay2.

Example 4.1.

```
i1 : needs "PseudomonomialPrimaryDecomposition.m2"
i2 : R = QQ[x1,x2,x3,x4];
i3 : I = ideal(x1,x2*(1-x3),(1-x2)*x4);
o3 : Ideal of R
i4 : primaryDecomposition I
o4 = {ideal (x4, x3 - 1, x1), ideal (x3 - 1, x2 - 1, x1), ideal (x4, x2, x1)}
o4 : List
i5 : primaryDecompositionPseudomonomial I
o5 = {ideal (x1, x4, x2), ideal (x1, x3 - 1, x2 - 1), ideal (x1, x3 - 1, x4)}
o5 : List
```

Note that the order of ideals and generators used by `primaryDecompositionPseudomonomial` and `primaryDecomposition` may be different.

Example 4.2.

```
i1 : needs "PseudomonomialPrimaryDecomposition.m2"
i2 : R = ZZ/3[x1,x2];
i3 : I = ideal(x1*(1-x2),(1-x1)*x2, x1*x2);
o3 : Ideal of R
i4 : primaryDecomposition I
o4 = {ideal (x2, x1, x1*x2)}
o4 : List
i5 : primaryDecompositionPseudomonomial I
o5 = {ideal (x2, x1)}
o5 : List
```

Note that `primaryDecompositionPseudomonomial` finds a minimal set of generators for each ideal.

Example 4.3.

```
i1 : needs "PseudomonomialPrimaryDecomposition.m2"
i2 : R = ZZ/3[x1,x2];
i3 : I = ideal(x1*(1-x2),(1-x1)*x2, x1*x2,(x1-1)*(x2-1));
o3 : Ideal of R
i4 : primaryDecomposition I
o4 = {}
o4 : List
i5 : primaryDecompositionPseudomonomial I
o5 = {}
o5 : List
```

The ideal is not proper, so both functions return the empty set.

Example 4.4.

```
i1 : needs "PseudomonomialPrimaryDecomposition.m2"
i2 : R = ZZ/3[x1,x2];
i3 : I = ideal(x1*(1-x1),(1-x1)*x2, x1*x2);
o3 : Ideal of R
i4 : primaryDecompositionPseudomonomial I
stdio:4:1:(3): error: Not a squarefree pseudomonomial ideal.
i5 : isSquarefreePseudomonomialIdeal I
o5 = false
```

The algorithm works only with squarefree pseudomonomial ideals.

Example 4.5.

```
i1 : needs "PseudomonomialPrimaryDecomposition.m2"
i2 : R = ZZ/2[x1,x2,x3,x4,x5,x6,x7,x8,x9];
i3 : I = ideal( (x8+1)*(x7+1)*(x4+1)*x3, (x7+1)*(x6+1)*(x4+1)*x3*(x1+1),
     (x8+1)*(x4+1)*x2, (x7+1)*x3*(x1+1), x5*(x4+1)*x3*x1, x5*(x2+1)*x1,
     x8*(x6+1)*x5*(x4+1)*x3, x8*(x6+1)*x5*(x2+1), x9*(x7+1)*x5,
     x9*(x8+1)*x5*(x4+1)*(x3+1)*x2*x1, x6*(x4+1)*(x3+1)*x2*x1, x8*(x7+1)*x6,
     (x8+1)*x5*(x4+1), (x7+1)*x5*x3*(x2+1)*(x1+1), x7*x5*(x4+1)*x2*x1,
     x8*x5*x3, x1*(x2+1)*(x3+1)*(x4+1)*(x5+1)*(x6+1)*x7*x8*(x9+1),
     x7*x5*(x3+1)*x1, x8*x5*x4*(x2+1) );
o3 : Ideal of R
i4 : timing(primaryDecompositionPseudomonomial I;)
o4 = -- .415975 seconds
o4 : Time
i5 : timing(primaryDecomposition I;)
o5 = -- 42.7676 seconds
o5 : Time
```

In some cases `primaryDecompositionPseudomonomial` can be orders of magnitude faster than the standard algorithm.

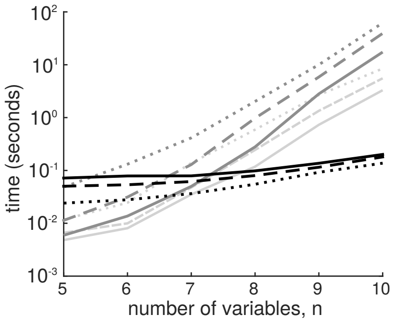

Figure 1. Timings of the function `primaryDecompositionPseudomonomial` (black) and `primaryDecomposition` (dark gray). Timings for `minimalPrimes` (light gray) are shown as well, since the ideals are known to be radical. The data was generated by calculating the average time of the computation of the primary decomposition of squarefree pseudomonomial ideals $I = \langle p_1, p_2, \dots, p_m \rangle$ of $\mathbb{Z}_2[x_1, \dots, x_n]$, where p_i is a randomly generated squarefree pseudomonomial. The curves show the timings as a function of n for $m = 10$ (solid), $m = 20$ (dashed), and $m = 30$ (dotted). The sample size for each n and m was 1000. Some computations were done on the Ohio Supercomputer Center.

Figure 1 shows a comparison between `primaryDecompositionPseudomonomial` and the current implementation of `primaryDecomposition`. For a small number of variables the current implementation is faster on squarefree pseudomonomial ideals than the implementation we describe in this paper. However, as the number of variables increases, our implementation is faster for that class of ideals.

SUPPLEMENT. The online supplement contains version 0.3 of *PseudomonomialPrimaryDecomposition*.

REFERENCES.

[Curto et al. 2013] C. Curto, V. Itskov, A. Veliz-Cuba, and N. Youngs, "The neural ring: an algebraic tool for analyzing the intrinsic structure of neural codes", *Bull. Math. Biol.* **75**:9 (2013), 1571–1611. MR Zbl

[Hinkelmann and Arnold 2010] F. Hinkelmann and E. Arnold, "Fast Gröbner basis computation for Boolean polynomials", 2010. arXiv 1010.2669

[Jarrah et al. 2007] A. S. Jarrah, R. Laubenbacher, B. Stigler, and M. Stillman, "Reverse-engineering of polynomial dynamical systems", *Adv. in Appl. Math.* **39**:4 (2007), 477–489. MR

[Macaulay2] D. R. Grayson and M. E. Stillman, "Macaulay2: a software system for research in algebraic geometry", available at http://www.math.uiuc.edu/Macaulay2.

[Miller and Sturmfels 2005] E. Miller and B. Sturmfels, *Combinatorial commutative algebra*, vol. 227, Springer, 2005. MR Zbl

[Sturmfels 2002] B. Sturmfels, "Ideals, varieties and Macaulay 2", pp. 3–15 in *Computations in algebraic geometry with Macaulay* 2, edited by D. Eisenbud et al., Algorithms Comput. Math. **8**, Springer, 2002. MR Zbl

[Veliz-Cuba 2012] A. Veliz-Cuba, "An algebraic approach to reverse engineering finite dynamical systems arising from biology", *SIAM J. Appl. Dyn. Syst.* **11**:1 (2012), 31–48. MR Zbl

RECEIVED: 4 Apr 2016 REVISED: 5 Jan 2021 ACCEPTED: 18 Jul 2022

ALAN VELIZ-CUBA:
avelizcuba1@udayton.edu
Department of Mathematics, University of Dayton, Dayton, OH, United States