THE F_4 ALGORITHM
SPEEDING UP GRÖBNER BASIS COMPUTATIONS USING LINEAR ALGEBRA

BJARKE HAMMERSHOLT ROUNE

Introduction

Gröbner bases are very useful, but the original Buchberger algorithm often cannot compute them in any reasonable amount of time. The algorithm F_4 [1][2] by Faugère is an improved version of the Buchberger algorithm that makes it possible to compute some previously intractable Gröbner bases, as can be seen by the fact that the well known cyclic-9 Gröbner basis problem was first cracked using F_4.

F_4 is very close to a drop-in replacement of the polynomial reduction step in the Buchberger algorithm and it can easily be combined with conventional criteria for eliminating useless S-pairs. F_4 speeds up the reduction step by exchanging multiple polynomial divisions for row-reduction of a single matrix.

High Level Perspective

Suppose f is a polynomial that we want to reduce by polynomials $r_1, \ldots, r_k$. Then F_4 will perform the following steps.

- Construct a matrix A based on f and $r_1, \ldots, r_k$.
- Compute a row-echelon form $\tilde{A}$ of A.
- Read off a reduced form of f from $\tilde{A}$.

The matrix A can be constructed such that many polynomials can be reduced at the same time, and the reduced polynomials can all be read off from $\tilde{A}$. This and the speed that has been attained by linear algebra software is what makes F_4 so efficient. Note that most of the entries of A will be zero, so it is necessary to use sparse matrix techniques.

An Example of F_4 in Action

If we use Gaussian elimination to reduce A, then the computations performed by F_4 will correspond with those of the polynomial division algorithm. We will start by looking at an example of polynomial division and then see how F_4 performs the same computation.

We will reduce $2X^2 - Y$ by $\{X - 1, Y + 2\}$ using the lexicographic term order where $Y \leq X$. The steps in the polynomial division algorithm are as follows.

- $(2X^2 - Y) - 2X(X - 1) = 2X - Y$
- $(2X - Y) - 2(X - 1) = -Y + 2$
- $(-Y + 2) + (Y + 2) = 4$

F_4 performs these same computations by reducing the matrix below. Columns correspond to monomials and rows correspond to polynomials. Note that the

columns are sorted in descending order to ensure that the first non-zero entry from the left in each row corresponds to the initial term of the corresponding polynomial.

$$\begin{array}{cccc} X^2 & X & Y & 1 & \\ \left[\begin{array}{cccc} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 2 \\ 2 & 0 & -1 & 0 \end{array}\right. & & & & \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right. \begin{array}{l} \text{(corresponds to } X(X-1) = X^2 - X\text{)} \\ \text{(corresponds to } X - 1\text{)} \\ \text{(corresponds to } Y + 2\text{)} \\ \text{(corresponds to } 2X^2 - Y\text{)} \end{array} \end{array}$$

Gaussian elimination on this matrix does the same thing as the polynomial division algorithm did before. The result is the following matrix.

$$\left[\begin{array}{cccc} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 4 \end{array}\right]$$

Note that the bottom row corresponds to $(0,0,0,4) \cdot (X^2, X, Y, 1) = 4$, which is the reduced polynomial. This shows the basic way that F_4 works, but of course there are a number of loose ends. We will now clear these up.

Constructing The Matrix A

We will construct A such that A makes it possible to emulate polynomial division by using Gaussian elimination. Note that an efficient implementation of F_4 will not actually emulate polynomial division; we merely need A to make it possible. We assume for simplicity that all polynomials have initial coefficient 1.

At each step the polynomial division algorithm looks at the initial monomial of the intermediate result h and attempts to find an r_i such that $\text{in}_{\leq}(r_i) \,|\, \text{in}_{\leq}(h)$. If such an r_i is found, then the next intermediate result will be $h - tr_i$ where $t := \frac{\text{in}_{\leq}(h)}{\text{in}_{\leq}(r_i)}$.

We want to make this same computation possible within A. Thus if h is one of the intermediate results in the polynomial division algorithm and h can be reduced by some r_i, then we need a row in A that corresponds to tr_i.

We do not need to know the exact value of $h - tr_i$ in order to predict which r_j will be used in the next step of the polynomial division algorithm — it is sufficient to know which monomials appear in $h - tr_i$. To be on the safe side, we can assume that $h - tr_i$ contains all the monomials of f and tr_i except the initial monomial.

These ideas lead to the following algorithm for constructing A.

(1) Add f to A and let M be the set of monomials of f.
(2) If M is empty, then we are done.
(3) Let m be the maximal monomial in M and remove m from M.
(4) If m cannot be reduced by any r_i, then go to step 2.
(5) Choose a monomial t and an r_i such that $\text{in}_{\leq}(tr_i) = m$.
(6) Add tr_i to A, add the monomials of tr_i except m to M and go to step 2.

This algorithm terminates since the polynomial ring of a field is Noetherian, and the ideal generated by all monomials that have at some point been removed from M in step 3 strictly increases each time that step is executed.

The monomial that is removed from M in step 3 does not need to be maximal, but then it becomes necessary to ensure that already processed monomials are not

added in the last step. The algorithm still terminates, as any monomial added to M will eventually also be added to M by the unmodified algorithm above.

By constructing A in this way, we ensure that we can emulate polynomial division within A. The matrix A might contain some unnecessary rows, since we have assumed that no terms cancel in $h - tr_i$, which might not be true. This is rarely a problem in practice, but consider the case of reducing $x^{10^{100}} - x^{10^{100}-1}$ by $x - 1$.

Replacing Gaussian Elimination

So far we have used Gaussian elimination to reduce A, and we have argued that this emulates polynomial division which proves that the result will be correct.

To make F_4 efficient we will have to use some more efficient algorithm in place of Gaussian elimination, and that algorithm may not produce the same row-echelon form of A as Gaussian elimination does. Thus we need to prove that *any* row-echelon form of A will give us a correctly reduced polynomial.

Proof. This would be immediately obvious if row-echelon forms were unique, but of course they are not. The *reduced* row-echelon form is, however, unique. If we use Gaussian elimination to compute the reduced row-echelon form of A, then that corresponds to fully reducing f using polynomial division.[1]

Let $\tilde{A}$ be some row-echelon form of A and let $\tilde{A}'$ be the reduced row-echelon form of A. Then we can transform $\tilde{A}'$ into $\tilde{A}$ using row operations that leave the pivots unchanged. This corresponds to adding multiples of $r_1, \ldots, r_k$ to the fully reduced polynomial without changing the initial term (if any). That still gives us a correctly reduced polynomial. □

The reduced polynomial will either be zero or have an initial term that cannot be reduced by any of $r_1, \ldots, r_k$. Thus we can spot the row in $\tilde{A}$ that corresponds to the reduced polynomial as it either is all zeroes or has its pivot in a column of $\tilde{A}$ that does not contain a pivot in A.

Computing Multiple Reductions In One Step

The Buchberger algorithm computes many polynomial reductions one after the other. Suppose we need to reduce polynomials $f_1, \ldots, f_n$. Then we could construct a matrix for each f_i and reduce these matrices separately. It is more efficient to construct a single matrix A that contains all the rows of the matrices $A_1, \ldots, A_n$ excluding duplicates. Then reducing A will reduce all of $f_1, \ldots, f_n$ simultaneously.

The only difference between reducing the polynomials one at a time and reducing them simultaneously is that then the reduced polynomials will be reducing one another linearly, but this it not a problem for the Buchberger algorithm.

References

[1] Jean-Charles Faugère. A new efficient algorithm for computing gröbner bases (f_4). *Journal of Pure and Applied Algebra*, 139(1):61–88, June 1999.

[2] Clint McKay. An analysis of improvements to buchberger's algorithm for groebner basis computation. Master's thesis, University of Maryland, College Park, 2004. `hdl.handle.net/1903/2119`.

[1] Note that reducing a polynomial only implies that the initial term cannot be further reduced, while fully reducing implies that no term can be further reduced.