Implementation of an Involutive Method for Computing Gröbner Bases of Zero-dimensional Ideals

Hoon Hong
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria
e-mail: hhong@risc.uni-linz.ac.at

Andreas Neubacher
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria
e-mail: aneubach@risc.uni-linz.ac.at

June 19, 1995

Abstract

A fundamental problem in constraint solving is that of solving systems of nonlinear equations. The method of Gröbner basis generalizes the well known Gaussian elimination of the linear case to the nonlinear case. In this paper, we give an overview of the mathematical foundations of a new algorithm by A.Yu. Zharkov and Yu.A. Blinkov for computing Gröbner bases for zero-dimensional ideals. Furthermore, a report on an implementation of this algorithm with descriptions of experimental results (esp. with regard to runtime behavior) is given.

1 Introduction

Involutive systems are a type of Gröbner bases first described by A.Yu. Zharkov and Yu.A. Blinkov in [6] and [7]. Their findings are based on the work of J.F. Pommaret [4], C.H. Riquier [5], and M. Janet [3]. Involutive methods are a new and promising approach towards computing Gröbner bases.

After a short introduction to the theory of involutive systems we describe our experiences with implementing and parallelizing a sequential algorithm for computing involutive systems for zero-dimensional ideals as given by Zharkov and Blinkov in [7]. Algorithms are listed in pseudo code and the results of experiments with a concrete implementation in C++ are discussed in detail.

Section 2 gives a short overview of those parts of the theory of involutive systems which are required for an understanding of the algorithm. Section 3 presents the sequential algorithm, its implementation, and timing results. Section 4 finally lists our plans for future research.

2 Theoretic Background for Involutive Systems

Throughout this section we assume that the reader has already had some introduction to polynomial ideal theory and esp. Gröbner bases theory.

As we only want to give an overview of the notions relevant to the algorithm we do omit proofs and do not present various results not directly related to the algorithm. To give the interested reader pointers towards the related literature we always list a reference to the paper in which the corresponding definition first appears or in which a proof of the corresponding theorem is presented.

2.1 Basic Definitions and Notation

Following the notation used by Zharkov and Blinkov in [6] and [7], we will use the notational conventions given below in the remainder of this paper:

- K: an arbitrary field of characteristic zero.
- a, b: elements of K.
- $K[x_1, \ldots, x_m]$: a polynomial ring over K.
- G, P, Q: finite subsets of $K[x_1, \ldots, x_m]$.
- g, h, p, q, r: elements of G, P, Q, etc.
- u, v, w: power products over $x_1, \ldots, x_m$.
- $\deg(u)$: the total degree of u.
- $\mathrm{vars}(u)$: the set of variables in u whose exponent is greater than zero.
- $\mathrm{cf}(p, u)$: the coefficient of u in p.
- $\mathrm{ideal}(G)$: the ideal generated by G.

Definition 1 (Admissible ordering) *A total ordering $<$ on the power products over $x_1, \ldots, x_m$ is called an* admissible ordering *iff*

(i) $\forall u \neq 1 : 1 < u$, *and*

(ii) $\forall u, v, w : u < v \Rightarrow uw < vw$.

From now on, let $<$ be an admissible ordering with the property $x_1 < x_2 < \ldots < x_m$. Based on this we define the following notions:

- $\mathrm{lpp}(p)$: the leading power product in p w.r.t. $<$.
- $\mathrm{lcf}(p)$: the coefficient of the leading power product in p.
- $\mathrm{lt}(p)$: the leading term of p, i.e. $\mathrm{lcf}(p)\mathrm{lpp}(p)$.
- $\mathrm{red}(p)$: the reductum of p, i.e. $p - \mathrm{lt}(p)$.

Definition 2 (Class – [6]) *The* class of a power product u *(Symbol:* $\mathrm{class}(u)$*) is the smallest index i for which x_i appears with a non-zero exponent in u. The class of a polynomial is the class of its leading power product.*

Definition 3 (Multiplicativity – [6], [4]) *A variable x_i is called* multiplicative *for a power product u (Symbol: $x_i \in \mathrm{Mult}(u)$), iff $i \leq \mathrm{class}(u)$. Otherwise, x_i is called* non-multiplicative *for u (Symbol: $x_i \in \mathrm{Nonmult}(u)$).*

This extends to polynomials by defining $\mathrm{Mult}(p) := \mathrm{Mult}(\mathrm{lpp}(p))$ *and* $\mathrm{Nonmult}(p) := \mathrm{Nonmult}(\mathrm{lpp}(p))$.

2.2 Janet Reduction

Based on the notion of multiplicativity one can define a reduction relation that is stronger than the reduction relations used in Gröbner bases theory, but which will serve a similar purpose.

Notation: The product $u \cdot v$ will be denoted by $u \times v$ if $\mathrm{vars}(v) \subseteq \mathrm{Mult}(u)$ or $v = 1$. This extends to polynomials by writing $p \cdot v$ as $p \times v$ if $\mathrm{vars}(v) \subseteq \mathrm{Mult}(p)$ or $v = 1$.

□

Definition 4 (Janet divisor – [6]) *The power product u is called a* Janet divisor *for the term w (Symbol: $u|_J w$), iff there exists a power product v such that $w = u \times v$.*

The following definition is similar to the reduction relation used in Gröbner bases theory.

Definition 5 (Janet reduction – [6]) *A polynomial p can be* Janet-reduced *to a polynomial q modulo G (Symbol: $p \xrightarrow{G}_J q$), iff there exist a $g \in G$ and a power product u in p such that*

- $\mathrm{lt}(g)|_J u$, *and*
- $a := \frac{\mathrm{cf}(p,u)}{\mathrm{lcf}(g)\mathrm{lpp}(g)} \neq 0$, *and*
- $q = p - aug$.

A polynomial p is called Janet-irreducible *modulo G, iff there is no polynomial q such that $p \xrightarrow{G}_J q$.*

The transitive closure of the Janet-reduction is denoted by $p \xrightarrow{G}{}^*_J q$.

As in Gröbner bases theory, the definition of a reduction relation leads to the notion of a normal form.

Definition 6 (Janet normal form – [6]) *A polynomial q is called a* Janet normal form *of p modulo G, iff*

- $p \xrightarrow{G}{}^*_J q$, *and*
- q *is Janet-irreducible modulo G.*

This definition immediately raises the question of uniqueness: Is there exactly one Janet normal form for any given polynomial? It can be shown that the answer is yes for certain sets G, namely for those which are *autoreduced*.

Definition 7 (Autoreduced – [6]) *G is called* autoreduced *iff*

$$\forall g, h \in G, g \neq h : \mathrm{lpp}(g) \not|_J \mathrm{lpp}(h).$$

Theorem 1 (Uniqueness of the Janet normal form – [6]) *Let G be autoreduced. Then the Janet normal form modulo G is unique for any polynomial p.*

Notation: The Janet normal form modulo G of a polynomial p will be denoted as $\mathrm{NF}_J(p, G)$.

□

2.3 Involutive Systems

The following definition captures the central mechanism used for defining and computing involutive systems.

Definition 8 (Prolongation – [6]) *The* prolongation *of a polynomial p by a variable x is the product px. If $x \in \mathrm{Mult}(p)$ the prolongation is called* multiplicative, *otherwise* non-multiplicative.

Now we are ready to give the central definition. This is what we want to compute.

Definition 9 (Involutive system – [6]) *G is called an* involutive system, *iff*

- *G is autoreduced, and*
- $\forall g \in G \forall x \in \mathrm{Nonmult}(G) : \mathrm{NF}_J(gx, G) = 0$.

In general one is interested in a slightly more restricted form of involutive systems, namely involutive bases. The corresponding definition is given below.

Definition 10 (Normalized – [6]) *G is called* normalized *iff* $\forall g \in G : \mathrm{lcf}(g) = 1$.

Definition 11 (Involutive basis – [6]) *G is called an* involutive basis *of* ideal(G) *iff it is an involutive system and normalized.*

The relation between involutive systems and Gröbner bases is given in the following corollary to Theorem 6 given in [6].

Corollary 1 (Involutive systems and Gröbner bases – [6]) *Any involutive system is a Gröbner basis w.r.t. the term ordering used.*

3 The Implementation

3.1 Algorithm Description

The algorithm described in this section is taken from [7]. The basic idea behind the algorithm for computing involutive systems is the iterated application of the Definitions 9 (Involutive System) and 12 (Completeness).

3.2 Additional Theory

To guarantee termination of the algorithm the following modification and extension of the theory given in Section 2 are required:

From now on let the ordering $<$ on power products be compatible with the total degree ordering, i.e. $u < v$ if $\deg(u) < \deg(v)$.

Furthermore, extend the notion of *degree* to polynomials and finite sets of polynomials by

- $\deg(p)$: $\deg(\mathrm{lpp}(p))$.
- $\deg(G)$: $\max\{\deg(g) | g \in G\}$.

The notion of *completeness* defined below differs from Definition 11 (Involutive basis) only in the fact that the degrees of the prolongations gx considered for normal form computations must not be greater than the degree of the system G.

Definition 12 (Completeness – [7]) *G is complete iff*

- *it is autoreduced,*

$G \leftarrow$ `InvolutiveBasis`(F)

Computes an involutive basis.

Inputs:
F : A finite subset of $K[x_1, \ldots, x_m]$

Outputs:
G : An involutive basis of ideal(F).

Begin

Step 1: *Initialize*

$G \leftarrow$ `CompleteSystem`(F)

Step 2: *Iterate*

while there exist $g \in G$ and $x \in$ Nonmult(g) such that $\deg(gx) > \deg(G)$ and $\mathrm{NF}_{\mathrm{J}}(gx, G) \neq 0$ **do**
$G \leftarrow$ `CompleteSystem`$(G \cup \{\mathrm{NF}_{\mathrm{J}}(gx, G)\})$
endwhile

End

Algorithm 1: `InvolutiveBasis`

- *normalized, and*
- $\forall g \in G \forall x \in \mathrm{Nonmult}(g) : \deg(gx) \leq \deg(G) \Rightarrow \mathrm{NF}_{\mathrm{J}}(gx, G) = 0$.

This additional definition may seem spurious at first glance, but the termination proof given in [7] depends on it. Straightforward implementation of Definition 11 (Involutive basis) may lead to larger and larger polynomials being added while trying to complete the system.

3.3 Algorithm Analysis

Research on complexity results for Gröbner bases computations has shown that it is very hard to derive general complexity bounds. Therefore we decided not to concentrate on this aspect of the algorithms.

Nevertheless, one may expect the following:

- Computing $\mathrm{NF}_{\mathrm{J}}(.,.)$ is done in the check of every **while**-loop. Therefore a significant part of the total computing time should be spent on this operation, which we regard as a black box for purposes of analysis.
- The order on power products influences the selection of the leading power product of a polynomial. As this is central to the reduction operation and the choice of polynomials in the **while**-loops different orders should give different computation times.

3.4 Algorithm Implementation and Timings

The algorithm described in this section was implemented in C++ using the Sturm library [1]. The implementation was done as a straightforward translation of the pseudo-code algorithms to C++.

$G \leftarrow$ `CompleteSystem`(F)

Computes a complete system.

Inputs:
F : A finite subset of $K[x_1, \ldots, x_m]$

Outputs:
G : A complete system such that ideal(F) = ideal(G).

Begin

Step 1: *Initialize*

$G \leftarrow$ `AutoreducedSystem`(F)

Step 2: *Iterate*

while there exist $g \in G$ and $x \in$ Nonmult(g) such that $\deg(gx) \leq \deg(G)$ and $\mathrm{NF}_J(gx, G) \neq 0$ **do**
$G \leftarrow$ `AutoreducedSystem`$(G \cup \{\mathrm{NF}_J(gx, G)\})$
endwhile

Step 3: *Normalize*

Divide all polynomials in G by their leading coefficients.

End

Algorithm 2: `CompleteSystem`

For purposes of parallelization, we are interested in the overall behaviour of the algorithm. Therefore we decided upon randomly generating many small examples and computing mean values for the parameters to be measured instead of computing timings for a few large special cases. The unpredictable effect of small changes to the input sets on running times of Gröbner bases methods also suggests this approach, because it is hard to find input sets showing "typical" behaviour of the algorithm.

The following list gives the parameters we used for randomly generating systems of polynomials and the values chosen for these parameters:

- **The order on power products**: Total degree lexicographic, total degree reverse lexicographic.
- **The number of variables**: 2 – 4.
- **The number of polynomials per system**: Same as the number of variables.
- **The total degree of the polynomials**: 3 – 4.
- **The number of terms per polynomial**: 4 – 8.
- **Coefficient size in bits**: 1 – 32.

The values measured were

- The time spent computing `CompleteSystem`, `AutoreducedSystem`, and $\mathrm{NF}_J(.,.)$ relative to the total computing time of `InvolutiveBasis`.
- The number of times these algorithms were called.

$G \leftarrow$ `AutoreducedSystem`(F)

Computes an autoreduced system.

Inputs:
F : A finite subset of $K[x_1, \ldots, x_m]$

Outputs:
G : An autoreduced system such that ideal(F) = ideal(G).

Begin

Step 1: *Initialize*

$G \leftarrow F$

Step 2: *Iterate*

while there exist $g, h \in G$ such that $\mathrm{lpp}(h) |_J \mathrm{lpp}(g)$ **do**
$\bar{g} \leftarrow g - h \frac{\mathrm{lt}(g)}{\mathrm{lt}(h)}$
if $\bar{g} = 0$ **then** $G \leftarrow G \setminus \{g\}$
else $G \leftarrow G \setminus \{g\} \cup \{\bar{g}\}$
endwhile

End

Algorithm 3: `AutoreducedSystem`

- The number of polynomials in the involutive basis.
- The maximum degree of the polynomials in the involutive basis.

3.5 Conclusion

The timing experiments lead to the following main observations, which influence further development and especially parallelization of the algorithm:

- More than 90% of the total computing time was spent on computing $\mathrm{NF}_J(.,.)$. The average time spent on a single computation of $\mathrm{NF}_J(.,.)$ was in general less than 0.5% of the total computing time.
- The order on power products seems to influence the running times: Using the lexicographic order instead of the reverse lexicographic order nearly doubled the average computing time for a single $\mathrm{NF}_J(.,.)$ and also increased the number of normal forms computed by about 10%. It should be noted that the numbers of polynomials in the resulting involutive bases and the degrees of the polynomials in general did not vary by more than ± 1.

  The results here are not entirely clear as there are examples where the lexicographic order was as fast or faster than the reverse lexicographic order, but the great majority of test results shows the behaviour described here.

4 Further Research

It does not seem promising to try improving the algorithm as it stands. We expect that parallelization will allow us to speed up the basis computations. Work in this direction is

already on the way.

Acknowledgements

We would like to thank Pedro A. García-Sánchez for an excellent presentation of the involution method.

References

[1] Hong, H. and Neubacher, A. and Stahl, V. *The STURM Library Manual.* RISC-Linz Report Series No. 94-30, RISC Linz, Johannes Kepler University, 4040 Linz, Austria, 1994.

[2] Hong, H. and Neubacher, A. and Schreiner, W. and Stahl, V. *The C++ Interface to the STURM Multi-Processor Kernel.* RISC-Linz Report Series No. 94-31, RISC Linz, Johannes Kepler University, 4040 Linz, Austria, 1994.

[3] Janet, M. *Leçons sur les systémes d'équations aux deriveés partielles.* Gauthier-Villars, Paris, 1929.

[4] Pommaret, J.F. *Systems of Partial Differential Equations and Lie Pseudogroups.* Gordon and Breach, New York, 1978.

[5] Riquier, J.H. *Les systèmes d'équations aux deriveés partielles.* Gauthier-Villars, Paris, 1910.

[6] Zharkov, Yu.A. and Blinkov, A.Yu. *Involution Approach to Solving Systems of Algebraic Equations.* Proceedings of IMACS'93, 1993.

[7] Zharkov, Yu.A. and Blinkov, A.Yu. *Involutive Bases of Zero-Dimensional Ideals.* ??? (to appear), 1994.