CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY IN PRAGUE

BACHELOR THESIS

Minimal Problem Solver Generator

Pavel Trutman

pavel.trutman@fel.cvut.cz

April 14, 2015

Available at
http://cmp.felk.cvut.cz/~trutmpav/theses/bsc-pavel-trutman.pdf

Thesis Advisor: Ing. Tomáš Pajdla, PhD.

Acknowledge grants here. Use centering if the text is too short.

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Minimal Problem Solver Generator

Pavel Trutman

April 14, 2015

Text of acknowledgements…

Abstract

Text of abstract…

Resumé

Text of resumé...

Contents

Abbreviations

AHA! Some optional explanation before the list. Indentation can be set by the command `\setlength{\AbbrvIndent}{5em}`.

1D	one dimension(al)
2D, 3D, ...	two dimension(al), three dimension(al), two dimension(al), three dimension(al), two dimension(al), three dimension(al), two dimension(al), three dimension(al), ...
AAM	active appearance model
AI	artificial intelligence
ASM	active shape model
B-rep	boundary representation
BBN	Bayesian belief networks

1 Introduction

Here comes introduction.

2 Polynomial system solving

Firstly we review the state of the art algorithms for computing Gröbner basis. Better understanding of these algorithms helps us to integrate them more efficiently into polynomial solving algorithms based on Gröbner basis computation.

2.1 Buchberger's Algorithm

Buchberger's Algorithm [2], which was invented by Bruno Buchberger, was the first algorithm for computing Gröbner basis. The algorithm is described in details in [3, 1].

2.1.1 First implementation

The first and easy, but very inefficient implementation of this algorithm is based on the observation that we can extend a set F of polynomials to a Gröbner basis only by adding all non-zero remainders $\overline{S(f_i, f_j)}^F$ of all pairs from F into F until there is no non-zero remainder. The main disadvatage of this simple algorithm is that so constructed Gröbner basis are often bigger than necessary.

Algorithm 1 Simple Buchberger Algorithm

Input:
  F a finite set of polynomials
Output:
  G a finite set of polynomials

$G \leftarrow F$
$B \leftarrow \{\{g_1, g_2\} \mid g_1, g_2 \in G,\ g_1 \neq g_2\}$
while $B \neq \emptyset$ **do**
  select $\{g_1, g_2\}$ from B
  $B \leftarrow B \setminus \{\{g_1, g_2\}\}$
  $h \leftarrow \overline{S(g_1, g_2)}^G$
  if $h \neq 0$ **then**
    $B \leftarrow B \cup \{\{g, h\} \mid g \in G\}$
    $G \leftarrow G \cup \{h\}$
  end if
end while
return G

2.2 F_4 Algorithm

The F_4 Algorithm [4] by Jean-Charles Faugère is an improved version of the Buchberger's Algorithm. The F_4 replaces the classical polynomial reduction found in the Buchberger's Algorithm by a simultaneous reduction of several polynomials. This reduction mechanism is achieved by a symbolic precomputation followed by Gaussian elimination implemented using sparse linear algebra methods. F_4 speeds up the reduction step by exchanging multiple polynomial divisions for row-reduction of a single matrix.

2.2.1 Improved Algorithm F_4

The main function of F_4 Algorithm consists of two parts. The goal of the first part is to initialize the whole algorithm. First, it generates required pairs and initializes Gröbner basis G. Then it takes each polynomial from the input set and calls function *Update* on it, which updates the set P of pairs and the set G of basic polynomials.

The second part of the algorithm generates new polynomials and includes them into the set G. In each iteration, it selects same pairs from P using function *Sel*. How to best select the pairs, is still an open question. Some selection strategies are described in the section 2.2.6 on page 7. Then, it splits each pair into two tuples. The first tuple contains the first polynomial f_1 of the pair and monomial t_1 such that $\mathrm{LM}(t_1 \times f_1) = \mathrm{lcm}(\mathrm{LM}(f_1), \mathrm{LM}(f_2))$. The second tuple is constructed in the same way from the second polynomial of the pair. All tuples from all selected pairs are put into the set L, i.e. duplicates are removed.

Next, it calls function *Reduction* on the set L and stores result in the set $\tilde{F}^+$. In the end it iterates throught all new polynomials in the set $\tilde{F}^+$ and calls function *Update* on each of them. This generates new pairs into the set P and extends Gröbner basis G.

This algorithm terminates when the set P of pairs is empty. Then the set G is a Gröbner basis and it is the output of the algorithm.

Algorithm 2 Improved Algorithm F_4

Input:

F a finite set of polynomials

Sel a function $List(Pairs) \to List(Pairs)$ such that $Sel(l) \neq \emptyset$ if $l \neq \emptyset$

Output:

G a finite set of polynomials

$G \leftarrow \emptyset$
$P \leftarrow \emptyset$
$d \leftarrow 0$
while $F \neq \emptyset$ **do**
  $f \leftarrow first(F)$
  $F \leftarrow F \backslash \{f\}$
  $(G, P) \leftarrow Update(G, P, f)$
end while
while $P \neq \emptyset$ **do**
  $d \leftarrow d + 1$
  $P_d \leftarrow Sel(P)$
  $P \leftarrow P \backslash P_d$
  $L_d \leftarrow Left(P_d) \cup Right(P_d)$
  $(\tilde{F}_d^+, F_d) \leftarrow Reduction(L_d, G, (F_i)_{i=1,\ldots,(d-1)})$
  for $h \in \tilde{F}_d^+$ **do**
    $(G, P) \leftarrow Update(G, P, h)$
  end for
end while
return G

2.2.2 Function Update

In this algorithm is used standard implementation of Buchberger Criteria **CITE Gebauer and Moller** [**GM88**].

2.2.3 Function Reduction

Task of this function is simple, it performs polynomial division using methods of linear algebra.

Input of this function is a set L containing tuples of monomial and polynomial, which were made in the main function of the F_4 Algorithm.

First, this function calls function *Symbolic Preprocessing* on the set L. This returns a set F of polynomials to be reduced. To use linear algebra methods to perform polynomial division the polynomials have to be represented by a matrix. Each column of the matrix corresponds to a monomial and the columns have to be ordered with respect to the ordering used so that the most right column corresponds to "1". Each row of the matrix corresponds to a polynomial from the set F. Construction of the matrix is simple. On the (i, j) posistion in the matrix, we put the coefficient of the term corresponding to j-th monomial from the i-th polynomial from the set F.

If we have constructed matrix like this we can reduce it to a row echelon form using, for example, Gauss-Jordan elimination. Note that this matrix is typically sparse so we can use sparse linear algebra methods to save computation time and memory. After elimination, we construct resulting polynomials by multiplying the reduced matrix by a vector of monomials from the **?????**.

In the end, the function returns the set of reduced polynomials that have leading monomials which were not amongs polynomials before reduction.

Algorithm 3 Reduction

Input:

L a finite set of tuples of monomial and polynomial

G a finite set of polynomials

$\mathcal{F} = (F_i)_{i=1,\ldots,(d-1)}$, where F_i is finite set of polynomials

Output:

$\tilde{F}^+$ a finite set of polynomials

F a finite set of polynomials

1: $F \leftarrow \mathit{Symbolic\ Preprocessing}(L, G, \mathcal{F})$
2: $\tilde{F} \leftarrow$ Reduction to Row Echelon Form of F
3: $\tilde{F^+} \leftarrow \left\{ f \in \tilde{F} \mid \mathrm{LM}(f) \notin \mathrm{LM}(F) \right\}$
4: **return** $(\tilde{F}^+, F)$

2.2.4 Function Symbolic Preprocessing

Function *Symbolic Preprocessing* starts with a set L of tuples containing monomial and polynomial. These tuples were made from selected pairs. Then, these tuples are simplified by function *Simplify* and after multiplying polynomials with corresponding monomials the results are put into the set F.

Next, the function goes through all monomials in the set F and for each monomial m looks for some polynomial f from Gröbner basis G such $m = m' \times \mathrm{LM}(f)$ where m'

is a some monomial. All such polynomials f and monomials m' are after simplification multiplied and put into the set F. The goal of this search is to have for each monomial in F some polynomial in F with the same leading monomial. This will ensure that all added polynomials will be reduced for G after polynomial division (using linear algebra).

Algorithm 4 Symbolic Preprocessing

Input:

L a finite set of tuples of monomial and polynomial
G a finite set of polynomials
$\mathcal{F} = (F_i)_{i=1,\dots,(d-1)}$, where F_i is finite set of polynomials

Output:

F a finite set of polynomials

$F \leftarrow \{multiply(Simplify(m, f, \mathcal{F})) \mid (m, f) \in L\}$
$Done \leftarrow \mathrm{LM}(F)$
while $\mathrm{M}(F) \neq Done$ **do**
    m an element of $\mathrm{M}(F) \setminus Done$
    $Done \leftarrow Done \cup \{m\}$
    if m is top reducible modulo G **then**
        $m = m' \times \mathrm{LM}(f)$ for some $f \in G$ and some monomial m'
        $F \leftarrow F \cup \{multiply(Simplify(m', f, \mathcal{F}))\}$
    end if
end while
return F

2.2.5 Function Simplify

Function *Simplify* simplifies a polynomial which is a product of the multiplication of a given monomial m and a polynomial f.

The function recursively looks for a monomial m' and a polynomial f' such $\mathrm{LM}(m' \times f') = \mathrm{LM}(m \times f)$. The polynomial f' is selected from all polynomials that has been reduced in previous iterations (sets $\tilde{F}^+$). We select polynomial f' such that total degree of m' is minimal.

This is done to insert into the set F (set of polynomials ready to reduce) polynomials such that are mostly reduced and have small number of monomials. This of course speeds up following reduction.

2.2.6 Selection strategy

For the speed of the F_4 Algorithm is very important how to select in each iteration critical pairs from the list of all critical pairs P. This of course depends on the implemetation of the function *Sel*. There are more possible implemetations:

- The easiest implementation is to select all pairs from P. In this case we reduce all criticals pairs at the same time.
- If the function *Sel* selects only one critical pair then the F_4 Algorithm is the Buchberger's Algorithm. In this case the *Sel* function corresponds to the selection strategy in the Buchberger's Algorithm.

Algorithm 5 Simplify

Input:

m a monomial

f a polynomial

$\mathcal{F} = (F_i)_{i=1,\ldots,(d-1)}$, where F_i is finite set of polynomials

Output:

(m', f') a non evaluated product of a monomial and a polynomial

```
for u ∈ list of all divisors of m do
    if ∃j (1 ≤ j ≤ d) such that (u × f) ∈ F_j then
        F̃_j is the Row Echelon Form of F_j
        there exists a (unique) p ∈ F̃_j^+ such that LM(p) = LM(u × f)
        if u ≠ m then
            return Simplify(m/u, p, F)
        else
            return (1, p)
        end if
    end if
end for
return (m, f)
```

- The best function that Faugère has tested is to select all critical pairs with a minimal total degree. Faugère calls this strategy the *normal strategy for* F_4.

Algorithm 6 Sel – The normal strategy for F_4

Input:

P a list of critical pairs

Output:

P_d a list of critical pairs

```
d ← min {deg(lcm(p)) | p ∈ P}
P_d ← {p ∈ P | deg(lcm(p)) = d}
return P_d
```

2.3 F_5 Algorithm

3 Automatic generator

3.1 Reimplementation

3.2 Multiple eliminations solver

3.3 Removing unnecessary polynomials

3.4 Matrix partitioning

3.5 F4 strategy

4 Experiments

5 Conclusion

Bibliography

[1] Thomas Becker and Volker Weispfenning. *Gröbner Bases, A Computational Approach to Commutative Algebra.* Number 141 in Graduate Texts in Mathematics. Springer-Verlag, New York, NY, 1993. 4

[2] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, Mathematical Institute, University of Innsbruck, Austria, 1965. 4

[3] David Cox, John Little, and Donald O'Shea. *Ideals, Varieties, and Algorithms : An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Undergraduate Texts in Mathematics. Springer, New York, USA, 2nd edition, 1997. 4

[4] Jean-Charles Faugère. A new efficient algorithm for computing gröbner bases (f_4). *Journal of pure and applied algebra*, 139(1–3):61–88, 7 1999. 4