An Efficient Algorithm for Computing Gröbner Bases for Confidentiality Problems

Stephen Roehrig
The Heinz School
Carnegie Mellon University

Introduction

The Buchberger algorithm solves this well-known problem from commutative algebra (see e.g., Cox et al. 1996).

> Given a subset F in $k[x_1, x_2, \ldots x_n]$, the set of polynomials over a field k, find $G \subset k[x_1, x_2, \ldots x_n]$, such that Ideal$(F)$ = Ideal(G) and G is a reduced Gröbner basis.

Diaconis and Sturmfels (1998) showed how the solution to this algebraic problem can be used to examine the set of tables having a fixed set of marginal sums. Thus finding a Gröbner basis has a very practical application in statistical data protection: If a database administrator publishes some seemingly non-sensitive marginal totals derived from a sensitive n-dimensional table, to what extent can a "data snooper" reconstruct the sensitive information?

While routines for computing Gröbner bases are provided in packages such as *Mathematica* and Maple, they are very slow. The current favorite for this task is a system called *Macaulay* (`http://www.math.uiuc.edu/Macaulay2`), written by a group at the University of Illinois. While *Macaulay* is much faster than *Mathematica*, it is still a general purpose program that doesn't take full advantage of the special structure of the statistical disclosure problem. In this note we provide a number of improvements to the standard implementation of Buchberger's algorithm that give a speed increase of about 20 million over *Macaulay*.

Background

Let $T = [t_{i_1,\ldots,i_n}]$ be an n-dimensional table, and let $M = \{M_1, \ldots, M_p\}$ be a collection of lower-dimensional marginal sums derived from T. The collection M implies a set of linear equality constraints on the cells of T. The data protection problem is to determine what a data snooper can discover about T from the published marginals M. One technique the snooper might use is to apply an optimization method such as linear or integer programming to find upper and lower bounds on the sensitive values of T. Very narrow bounds for a cell suggest a partial disclosure of its value. Another approach is to enumerate all tables T' that have M as their marginal totals, and then consider the distribution of each cell value. If a cell distribution is too tightly focused around the true value, disclosure has again occurred. Of course, for larger tables, complete enumeration is impractical, so a sampling scheme would be used to estimate the distributions.

what can a thief discover

As shown first by Conti and Traverso (1991), the theory of polynomial ideals, and in particular the Gröbner basis of a special ideal, can be used to solve the integer programming problem associated with finding upper and lower bounds. The work of Diaconis and Sturmfels extends this by showing that this same Gröbner basis provides the means for sampling from the space of tables T' that agree with the published marginals M.

The Conti and Traverso framework is as follows. Let $x_j, j = 1, \ldots, n$ be variables associated with the cell values in T, and let A be an $m \times n$ constraint matrix arising from the marginals M. Introduce variables $y_i, i = 1, \ldots, m$, and set $\alpha_j = \prod_i y_i^{a_{ij}}$. If k is a field (say, the rationals), consider the polynomials $k[m] = k[y_1, \ldots, y_m]$ and $k[n] = k[\alpha_1, \ldots, \alpha_n]$. Finally, let $\phi_j = \alpha_j - x_j$. Then Conti and Traverso show that with a suitable term ordering on $k[y, x]$, the ideal $(\phi_j) \cap k[x]$ is generated by those elements in a Gröbner basis of (ϕ_j) that are in $k[x]$.

The ideal $(\phi_j) \cap k[x]$ represents the set of "moves" that take one table T', that satisfies the marginal constraints, to another table with the same margins. For example, suppose the cells in a $3 \times 3 \times 2$ table T are numbered as $x_{111} \rightarrow x_1, x_{112} \rightarrow x_2, \ldots x_{332} \rightarrow x_{18}$. Then one possible move that leaves the two-way margins fixed is represented by the polynomial $x_1 x_6 x_{14} x_{17} - x_2 x_5 x_{13} x_{18}$. The exponents in the first monomial are interpreted as increments to the corresponding table cell values, while those in the second monomial represent decrements. Because the exponents are restricted to non-negative integers, increments and decrements will necessarily be integer values, resulting in a new integer table. An alternative description of this move is the following, where a "+" indicates a unit increment and a "−" indicates a unit decrement, and the two levels $(k = 1, k = 2)$ of the table are shown side by side.

+	0	−	−	0	+
0	0	0	0	0	0
−	0	+	+	0	−

The Gröbner basis is the minimal set of moves that generate the set of all tables satisfying the known marginal totals. In practice, one starts with a table known to have the given marginals (such as the true table T), then applies the moves in the Gröbner basis at random. As Diaconis and Sturmfels show, this random walk moves uniformly through the space of tables agreeing with the margins, allowing a statistically valid sampling.

One difficulty with the Conti and Traverso setup is the inclusion of the extra variables y_i. The procedure is to compute the basis over the set of polynomials $k[y, x]$, then "throw away" those basis elements not exclusively in the xs. Since the complexity of computing a Gröbner basis depends exponentially on the total number of variables, this is wasteful. We show below how to overcome this difficulty.

Bigatti's algorithm.

Buchberger's Algorithm

The most widely used version of Buchberger's algorithm for finding a Gröbner basis is shown in Figure 1 (Buchberger 1985). Here, F, G, P and B are sets of polynomials, f, g, b and u are individual polynomials, and $\leq_T$ is a term ordering on polynomials.

The algorithm uses the special function NormalForm(F, g), which is essentially $\overline{g}^F$ (see Cox et al. (1996)), except that at each division step, the divisor is chosen in a special way. The notation $g \rightarrow_{f,b,u}$ used in the definition is read "g is reducible using f, b, u" and is true iff

$$\text{Coefficient}(g, u \cdot \text{LeadingPowerProduct}(f)) \neq 0$$

and

$$b = \text{Coefficient}(g, u \cdot \text{LeadingPowerProduct}(f)/\text{LeadingCoefficient}(f).$$

Here is the definition:

Algorithm ($h :=$ NormalForm(F, g)).

$h := g$

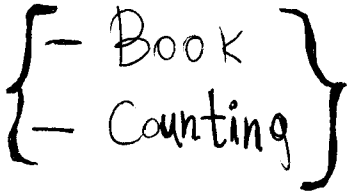
while exist $f \in F, b, u$ such that $h \rightarrow_{f,b,u}$ *do* choose $f \in F, b, u$ such that $h \rightarrow_{f,b,u}$ and $u \cdot \text{LeadingPowerProduct}(f)$ is maximal (w.r.t $<_T$)

$h := h - b \cdot u \cdot f$

Buchberger remarks that this special choice of the divisor is crucial to the performance of the algorithm. This assertion was examined empirically by keeping the set F in an C++ standard template library `map` collection, where the key is the leading power product. The importance of the special choice was tested by taking binomials from the map, first from the beginning (smallest w.r.t. $<_T$), and then from the end (largest). There was no difference in the number of total division steps, so this special choice seems irrelevant, likely because of the special form of our polynimials (monomial differences).

The Gröbner basis algorithm uses two criteria to decide whether a B-pair should even be processed. Criterion1 is the same criterion that appears in Cox et al. (1996) p. 107. Buchberger's Criterion2 is the same as the Cox et al. Proposition 4 in Chapter 2, Section 9 (p. 101).

A difference between Buchberger's algorithm and the one presented in Cox et al. is that Buchberger uses the additional sets R and P. These hold new polynomials, that are reduced by polynomials currently in the basis. This technique results in a reduced Gröbner basis at the termination of the algorithm.



Algorithm:
$R := F;\ P := \emptyset;\ G := \emptyset;\ B := \emptyset$
ReduceAll(R, P, G, B); NewBasis(P, G, B)
while $B \neq 0$ *do*
    $\{f_1, f_2\}$:= a pair in B whose $LCM(LP(f_1), LP(f_2))$ is minimal w.r.t. $<_T$
    $B := B - \{\{f_1, f_2\}\}$
    if (*not* Criterion1(f_1, f_2, G, B) *and not* Criterion2(f_1, f_2)) *then*
        $h :=$ NormalForm$(G,$ SPolynomial$(f_1, f_2))$
        if $h \neq 0$ *then*
            $G_0 := \{g \in G \mid LP(h) \leq_T LP(G)\}$
            $R := G_0;\ P := \{h\};\ G := G - G_0$
            $B := B - \{\{f_1, f_2\} \mid f_1 \in G_0$ *or* $f_2 \in G_0\}$
            ReduceAll(R, P, G, B); NewBasis(P, G, B).

Subalgorithm ReduceAll(R, P, G, B):
while $R \neq \emptyset$ *do*
    $h :=$ an element in R; $R := R - \{h\}$
    $h :=$ NormalForm$(G \cup P, h)$
    if $h \neq \emptyset$ *then*
        $G_0 := \{g \in G \mid LP(h) \leq_T LP(G)\}$
        $P_0 := \{p \in P \mid LP(h) \leq_T LP(p)\}$
        $G := G - G_0$
        $P := P - P_0$
        $R := R \cup G_0 \cup P_0$
        $B := B - \{\{f_1, f_2\} \in B \mid f_1 \in G_0$ or $f_2 \in G_0\}$
        $P := P \cup \{h\}$
Subalgorithm NewBasis(P, G, B):
    $G := G \cup P$
    $B := B \cup \{\{g, p\} \mid g \in G, p \in P, g \neq p\}$
    $H := G;\ K := \emptyset$
    while $H \neq \emptyset$ *do*
        $h :=$ an element in $H; H := H - \{h\}$
        $k :=$ NormalForm$(G - \{h\}, h);\ K := K \cup \{k\}$
    $G := K$.

Subalgorithm Criterion1$(f_1, f_2, G, B) :\Leftrightarrow$ there exists a $p \in G$ such that
    $f_1 \neq p, p \neq f_2$,
    $LP(p) \leq_T LCM(LP(f_1), LP(f_2))$,
    $\{f_1, p\}$ not in B and $\{p, f_2\}$ not in B.

Subalgorithm Criterion2$(f_1, f_2) :\Leftrightarrow$
    $LCM(LP(f_1), LP(f_2)) = LP(f_1) \cdot LP(f_2)$.

Figure 1: Buchberger's Algorithm for Gröbner Bases

Improvements

All the improvements discussed here are based on using Conti and Traverso's description of the ideal (Conti and Traverso, 1991), and lex ordering on the monomials. For example, suppose we want the Gröbner basis for the moves on a $3 \times 3 \times 2$ table with all three 2-D margins given. Numbering the 3-D cell variables as $x_{111} \rightarrow x_1, x_{112} \rightarrow x_2, \ldots x_{332} \rightarrow x_{18}$, the row-reduced form of the line sum equations is

```
    x1 x2 x3 x4 x5 x6 x7 x8 x9x10x11x12x13x14x15x16x17x18
 y1  1  0  0  0  0  0  0  0  0  1  0  1  0  0  0  1  0  1
 y2  0  1  0  0  0  0  0  0  0 -1  0 -1  0  0  0 -1  0 -1
 y3  0  0  1  0  0  0  0  0  0 -1  0  0  0  0  0 -1  0  0
 y4  0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  1  0  0
 y5  0  0  0  0  1  0  0  0  0  0  0 -1  0  0  0  0  0 -1
 y6  0  0  0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  1
 y7  0  0  0  0  0  0  1  0  0 -1  0 -1  0  0  0  0  0  0
 y8  0  0  0  0  0  0  0  1  0  1  0  1  0  0  0  0  0  0
 y9  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0
y10  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0
y11  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0 -1  0 -1
y12  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  0  1
y13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0
y14  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1
```

In Conti and Traverso's setup, there are 14 y variables, and one t variable. The ideal-defining binomials are

```
y1-x1
y2-x2
y3-x3
y4-x4
y5-x5
y6-x6
y7-x7
y8-x8
y9-x9
y1*y4*y8*y9-y2*y3*y7*x10
y10-x11
y1*y6*y8*y10-y2*y5*y7*x12
y11-x13
y12-x14
y13-x15
y1*y4*y12*y13-y2*y3*y11*x16
y14-x17
y1*y6*y12*y14-y2*y5*y11*x18
```

```
y1*y2*y3*y4*y5*y6*y7*y8*y9*y10*y11*y12*y13*y14*t-1
```

The lex order of the variables is $y_1, y_2, \ldots, y_{14}, t, x_1, x_2, \ldots, x_{18}$.

In the Buchberger algorithm, after the first pass through ReduceAll and NewBasis, the current basis is the following.

```
+x1x6x14x17-x2x5x13x18
+x1x6x8x11-x2x5x7x12
+x1x4x14x15-x2x3x13x16
+x1x4x8x9-x2x3x7x10
+tx2^2x3x4x5^2x7x8x9x11x13^2x15x18-1
+y14-x17
+y13-x15
+y12-x14
+y11-x13
+y10-x11
+y9-x9
+y8-x8
+y7-x7
+y6-x6
+y5-x5
+y4-x4
+y3-x3
+y2-x2
+y1-x1
```

The first four binomials are the result of eliminating all ys in the original four 4th-degree binomials. These four, now exclusively in terms of xs, are moves of the form

$$\begin{array}{ccc|ccc} + & 0 & - & - & 0 & + \\ 0 & 0 & 0 & 0 & 0 & 0 \\ - & 0 & + & + & 0 & - \end{array}$$

The four binomials, exclusively in the xs, can be read off directly from the initial row-reduced matrix as follows. Any column in that matrix consisting of a single "1" is ignored. The other columns—there are as many of these as there are degrees of freedom in the constraint set—each give rise to a polynomial in the xs. Looking for example at the `x12` column, the polynomial has in its leading monomial `x1`, `x6`, `x8` and `x10`, since these are the xs directly associated with the rows y_i in which the 1s appear. This just corresponds to the Conti and Traverso polynomials (`+y1-x1` etc.). The second monomial consists of `x2`, `x5`, `x7`, and `x12`; the `x12` appears because this is the `x12` column. The polynomial is thus

$$\texttt{x1x6x8x10} - \texttt{x2x5x7x12}.$$

Executing now the master *while* loop of the Buchberger algorithm, we find that, because of the way the B-pairs are extracted from the set B, *all* the binomials consisting entirely of xs are produced *first*. There are 15 of these. The set of B-pairs is built as an STL set, ordered with lex on the LCM of the leading monomials of the binomials in the pair. This ensures that pairs of binomials consisting entirely of xs will be chosen first. Following these, 56 other binomials are computed, rounding out the complete Gröbner basis. Since, however, we are only interested in those binomials having only x terms, the *while* loop can be stopped immediately on the production of a binomial containing a y or t. This is the first significant improvement.

A second observation is that, after the initial pass through ReduceAll and NewBasis, any B-pair containing a binomial in y_i or t will result in a reduced SPolynomial still containing y_i or t. This implies that once the algorithm has processed all pairs containing binomials exclusively in the xs, there is no need, as above, to continue. It follows that there is no need to keep in the set B any pair with a binomial containing y_i or t. This makes B vastly smaller, and improves performance considerably. The upshot of this is, so far as the algorithm is concerned, it is as though everything is done in $k[x_1, x_2, \ldots x_n]$.

A third observation is that, once inside the *while* loop, if h is nonzero, there are some steps to compute G_0 and R. For our problem, G_0, and so R, is always empty. Thus these steps are unnecessary. As a consequence, the routine ReduceAll does nothing, and can be eliminated, since no updating of G, P, R or B need be done there.

A fourth observation is that in the routine NewBasis, everything in $G \cup P$ is already reduced w.r.t. itself. This is a consequence, ultimately, of the way the B-pairs are drawn from B. Therefore the only thing that NewBasis needs to do is update B. Since P always consists of a single binomial, this is fast to do. The assignment to H, and the *while* loop over it, are unnecessary. Finally, note that Criterion2 is best applied *before* adding a pair to B, keeping this set as small as possible.

Next, it was observed empirically that Criterion1 was more time-consuming than the NormalForm calls it is designed to eliminate. Divisions on binomials are relatively fast, but searching through the STL set B, as required by Criterion1, is expensive.

After this much improvement of the algorithm, the Gröbner basis for the $3 \times 3 \times 3$ example above is calculated in about 250 milliseconds, requiring the creation of just 79 S-polynomials. This is about a 100,000-fold improvement over the general-purpose program *Macauley*, which took a little less than seven hours on the same machine.

Other Improvements

It's possible and prudent to use whatever knowledge is available in generating a Gröbner basis. For example, once a basis is available for the $3 \times 3 \times 3$ problem

with all 2-way margins known, it can serve as a starting point for constructing, say, a basis for the slightly larger $4 \times 3 \times 3$ problem. Simply include the $3 \times 3 \times 3$ basis, suitably embedded in the larger problem, along with the starting set of polynomials described above. This leverages computation already performed, and appears to provide at least an additional $\times 20$ speed improvement.

We used this technique to compute the Gröbner basis for the $4 \times 3 \times 3$ problem, which took 20 minutes and had 628 polynomials. Also, the $5 \times 3 \times 3$ problem was solved, resulting in 3,236 polynomials. However, even with the roughly 10^6 speed improvement of the revised algorithm, this computation took two months on a reasonably fast UNIX platform.

Finally, we remark that the same program can be used to *verify* a Gröbner basis rather than to generate it from scratch. The verification process is itself vastly quicker than generating the basis, so if one has an idea of what the basis should be, it is quick to check. An examination of the the progression of bases so far discovered may lead to a way of predicting the basis for a larger table. In this way, it may be possible to extend the set of known bases considerably.

References

1. Buchberger, B. (1985), "Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory," in *Multidimensional Systems Theory*, edited by N.K. Bose, D. Reidel Publishing Co., Dordrecht, 184–232.

2. Conti, P. and C. Traverso (1991), "Buchberger Algorithm and Integer Programming," in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC-9)*, edited by H. Mattson, T. Mora and T. Rao, Lecture Notes in Computer Science **539**, Springer-Verlag, New York, 130–139.

3. Cox, D., J. Little and D. O'Shea (1996), *Ideals, Varieties, and Algorithms*, Springer-Verlag, New York.

4. Diaconis, P. and B. Sturmfels (1998), "Algebraic Algorithms For Sampling From Conditional Distributions," *Annals of Statistics*, Vol. 28, No. 1, 363–397.