Hensel Lifting via Gröbner bases

Daniel Lichtblau

Wolfram Research, Inc.
100 Trade Centre Dr.
Champaign IL USA, 61820

danl@wolfram.com

ACA ’05, Nara, Japan
August 1, 2005

Abstract

In this talk I will show how one may use Gröbner bases over Euclidean domains to perform Hensel lifting in some polynomial rings. The algorithm is quite simple. Moreover, for the ring of univariate polynomials over the integers, dedicated polynomial arithmetic code of around two dozen lines can implement this method quite efficiently (it compares well to tree lifting, which appears to be the most effective approach known). We will also see how the Gröbner basis approach to lifting may be applied to bivariate polynomials over finite fields.

Introduction

We begin with Gröbner bases over Euclidean domains. This goes back to work by Buchberger, Kandri−Rody and Kapur, Möller, L. Pan, and others in the 1980s, with roots in earlier work. Roughly, we find such bases in the same way as when working over a field except we use the Euclidean algorithm in lieu of division. An analogy to keep in mind is that of row reduction vs. Hermite normal form computation, as these are what ordinary Gröbner bases and bases over Euclidean rings become when the input polynomials are linear.

The material for this talk originates in work I did a few years ago. The primary purpose was to simplify the theory and methods in this earlier body of work on Gröbner bases over Euclidean rings. The next step was to investigate various applications. There are several that involve special cases of such bases. The form of Hensel lifting to be shown is among them.

This appears to be perhaps related to work by Gianni and Trager, also from the 1980s, describing a way to do multivariate polynomial factorization over a finite field. They present a Gröbner basis computation to do, in one step, Hensel lifting and recombination of factors over a specialization to a univariate ring.

```
poly =
 Expand[(x^5 + 18 x^4 + 34 x^3 +
     5 x^2 + 21 x + 30) *
   (x^4 + 24 x^3 + 22 x^2 + 17 x + 15)]
```

$$450 + 825 x + 1092 x^2 + 1777 x^3 + 1492 x^4 + 1210 x^5 + 1234 x^6 + 488 x^7 + 42 x^8 + x^9$$

We will first factor the polynomial modulo a small prime, removing the constant factor.

```
mod = 11;
fax =
  FactorList[poly, Modulus → mod];
fax = First /@ Rest[fax]
```

$$\{4 + 6 x + 2 x^3 + x^4,\ 8 + 10 x + 5 x^2 + x^3 + 7 x^4 + x^5\}$$

Next we wish to make the factors correct modulo a power of the prime. This correction step is of course our desired Hensel lifting , used in most algorithms for factoring polynomials over the rationals. It is typically done by iterations of Newton’s method in a $p-$adic setting, but we will instead use Gröbner bases. In effect we take separate "$p-$adic gcds" of our polynomial and each factor raised to the indicated power. These gcds are the lifted factors.

For this particular example we take the factors, square them, compute Gröbner bases over the integers of the set `{poly,squaredfactor,squaredmodulus}`, and extract the last elements of these bases. This will correspond to quadratic Hensel lifting, insofar as factors correct modulo p becomes correct modulo p^2.

```
(Last[GroebnerBasis[
     {mod^2, poly, #1},
     CoefficientDomain ->
  Integers]] &) /@ (fax^2)
```

```
{15 + 17 x + 22 x^2 + 24 x^3 + x^4,
 30 + 21 x + 5 x^2 + 34 x^3 + 18 x^4 + x^5}
```

Lo and behold, we recover the correct factors in this simple example. In real life we only get $p-$adic images that might need to be recombined. There is a nice knapsack algorithm due to van Hoeij (2002) that will do this.

Theorem: Given a square free univariate polynomial f over the rationals, and an integer p such that the leading coeffi− cient of f is not divisible by p, f is square free modulo p, and $f \equiv_p g_0 h_0$. Assume $s = \text{GCD}\left[g_0^2, f\right]$ exists modulo p^2.Then s is the Hensel lift of g_0 modulo p^2.

Remark: the notion of a gcd existing in the setting of an inte- gral domain is just a convenient shorthand for the result of running the Euclidean algorithm under the assumption that we encounter no zero divisors along the way. Also note that this p−adic "gcd" may be computed, as above, by a Gröbner basis over the integers.

Remark: One could lift further than quadratically. The for− mulation above is convenient in that the proof is simple and iterated quadratic lifting is generally more efficient than going higher in one lift.

Remark: As we increase powers we are effectively working in the univariate ring $\mathbb{Z}_{p^n}$. This is a finite chain ring, and these were discussed in work by Norton and Sălăgean (2001).

Proof: We are given $f \equiv_p g_0 h_0$. Suppose the quadratically lifted equation is $f \equiv_{p^2} g_1 h_1$ where $g_1 \equiv_p g_0$ and $h_1 \equiv_p h_0$. The assumptions imply that the degrees of g_0 and g_1 are equal, and likewise with the h cofactors. We may write $g_1 = g_0 + p\, t_0$. Then a simple computation shows that $g_1 (g_0 - p\, t_0) \equiv_{p^2} g_0^2$. We see that $g_1 \mid f$ and $g_1 \mid g_0^2$ mod−ulo p^2. Now let $s = \mathrm{GCD}\left[g_0^2,\ f\right]$. Then we have $g_1 \mid s$. In order to show these are equal up to unit multiples (which proves the theorem), it suffices to show that $\mathrm{degree}(g_1) \geq \mathrm{degree}(s)$.

Suppose $\mathrm{degree}(s) > \mathrm{degree}(g_1)$. Then $\mathrm{degree}(s) > \mathrm{degree}(g_0)$. Since $s \mid f$ modulo p^2 we have $s \mid f$ modulo p. But also $s \mid g_0^2$ so the strict degree inequality implies that s is not square free modulo p. Hence f is not square free modulo p, in contradiction to the assumption that it is. □

Here is an example from van Hoeij's 2002 paper on knap−sack factorization. Timings are on a 3 GHz Pentium 4 run−ning under Linux.First we create the polynomial in ques−tion. Its roots are comprised of sums of pairs of roots of a smaller polynomial.

```
poly1 = x^20 - 5 x^18 + 864 x^15 -
   375 x^14 - 2160 x^13 + 1875 x^12 +
   10 800 x^11 + 186 624 x^10 -
   54 000 x^9 + 46 875 x^8 + 270 000 x^7 -
   234 375 x^6 - 2 700 000 x^5 -
   1 953 125 x^2 + 9 765 625;
rts = x /. Solve[poly1 == 0, x];
sums =
  Flatten[Table[rts[[i]] + rts[[j]],
    {i, 19}, {j, i + 1, 20}]];
newpoly = Expand[Times @@
    (x - N[sums, 200])];
newpoly = Chop[newpoly] /.
   a_Real → Round[a];
```

We now extract factors modulo a certain prime.

```
mod = Prime[4000];
fax = FactorList[
   newpoly, Modulus → mod];
fax = First /@ Rest[fax];
```

We lift to the 36th power of our prime.

```
liftFactors[fax_, poly_, mod_,
  pow_] := Module[{modpow = mod,
   top = Ceiling[Log[2, pow]],
   liftedfax = fax},
  Do[modpow = If[j == top,
     mod^pow, modpow^2];
   liftedfax = Expand[liftedfax^2,
     Modulus → modpow];
   liftedfax = Map[
     Last[GroebnerBasis[
        {modpow, poly, #},
        CoefficientDomain →
         Integers]] &,
     liftedfax], {j, top}];
  liftedfax]
```

```
Timing[liftedfax = liftFactors[
    fax, newpoly, mod, 36];]
```

```
{2.87056 Second, Null}
```

Remark: Dedicated univariate polynomial code will do this still faster. Up to fairly high lifting size it appears to be gen−erally more efficient than Shoup’s tree−lifting (described in the 1999 book by von zur Gathen and Gerhard), though asymptotically the latter is superior. Below we show dedi−cated code and apply it to our previous example.

```
liftFactors2[ofax_, opoly_, mod_, bnd_] :=
 Module[{p, p1, pow = bnd, pow1, liftedfax, liftedfax2,
    liftedfax3, top, tm, PT, tot = 0, i, fax, poly, x},
   x = First[Variables[opoly]];
   fax = CoefficientList[ofax, x];
   poly = CoefficientList[opoly, x];
   liftedfax = fax;
   p = p1 = mod;
   top = Ceiling[Log[2, pow]];
   pow1 = 2 * pow;
   PT = Table[pow1 = Ceiling[pow1 / 2], {top}];
   pow1 = 1;
   Do[p = p^2;
    pow1 *= 2;
    If[pow1 ≠ PT[[top - i + 1]], pow1 = PT[[top - i + 1]];
     p = p / p1;
     liftedfax = Mod[liftedfax, p]];
    liftedfax2 = Map[Algebra‘PolynomialTimesModList[#, #, p] &, liftedfax];
    liftedfax = Map[Algebra‘PolynomialGCDModList[poly, #, p] &, liftedfax2];
    tot = tot + tm;
    , {i, top}];
   Map[Internal‘FromCoefficientList[#, x] &, liftedfax]]
```

```
Timing[
 liftedfax2 = liftFactors2[fax,
    newpoly, mod, 36];]
```

```
{0.934858 Second, Null}
```

We now show an example in a bivariate ring modulo a prime. The upshot is that we can do Hensel lifting in the same way, resorting to Gröbner bases over the Euclidean domain given by univariate polynomials modulo the prime.

```
randpoly[deg_, mod_, x_, y_] :=
 Sum[Random[Integer,
    {If[i + j == deg && i * j == 0, 1,
      0], mod - 1}] * x ^ i * y ^ j,
  {i, 0, deg}, {j, 0, deg - i}]
```

```
mod = 19;
SeedRandom[1111];
totdeg = 6;
poly1 =
  randpoly[totdeg / 2, mod, x, y];
poly2 = randpoly[
   totdeg / 2, mod, x, y];
poly = Expand[poly1 * poly2,
  Modulus → mod]
```

$$18 x + 6 x^2 + 18 x^3 + 16 x^4 + 12 x^5 + 8 x^6 + 13 y + 12 x y + 17 x^2 y + 11 x^3 y + 7 x^4 y + 16 x^5 y + 16 y^2 + 17 x y^2 + 6 x^2 y^2 + 2 x^3 y^2 + 14 x^4 y^2 + 14 y^3 + 18 x y^3 + 5 x^2 y^3 + 2 x^3 y^3 + y^4 + 4 x y^4 + 16 x^2 y^4 + 18 y^5 + 18 x y^5 + 7 y^6$$

We will evaluate at $x = 11$ and factor, removing the con−stant term.

```
fax = Map[First,
  Drop[FactorList[poly /. x → 11,
    Modulus → mod], 1]]
```

```
{8 + y, 14 + 3 y + y^2,
 14 + 13 y + 9 y^2 + y^3}
```

We will lift a factor modulo a power of the ideal $(x - 11)$ that is sufficient to reclaim factors of degree 3 in x. Note that here we lift to 12th power in one step. We might instead have fashioned this as an iterated quadratic lifting, as in earlier examples.

```
subst = x - 11;
power = 12;
substpower = subst^power;
liftedfactor =
 Last[GroebnerBasis[{poly,
    substpower, fax[[1]]^power},
   y, Modulus → mod,
   CoefficientDomain ->
    Polynomials[x]]]
```

```
3 + 14 x + 2 x^2 + 15 x^3 +
 2 x^4 + 15 x^5 + 18 x^6 + 18 x^7 +
 18 x^8 + 13 x^9 + 5 x^10 + x^11 + y
```

As in A. Lenstra’s 1985 paper, we can use this to recover the true factor via reducing appropriately in a univariate polynomial lattice. This requires some basic tools which we also built from Gröbner bases. First is a routine to compute a basis over a module. This is not completely general by any means in that term orderings are not general, but it will suffice to give us what we’ll need for lattice reduction.

```
moduleGroebnerBasis[polys_,
  vars_, cvars_, opts___] :=
 Module[{newpols, rels,
   len = Length[cvars],
   gb, j, k, ruls},
  rels = Flatten[
    Table[cvars[[j]] * cvars[[k]],
     {j, len}, {k, j, len}]];
  newpols = Join[polys, rels];
  gb = GroebnerBasis[newpols,
    Join[cvars, vars], opts];
  rul = Map[(# → {}) &, rels];
  gb = Flatten[gb /. rul];
  Collect[gb, cvars]]
```

Now we can implement the lattice reduction using the mod−ule Gröbner basis computation. The key thing is to weight polynomials by total degree.

```
polynomialLatticeReduce[
  mat_?MatrixQ,
  mod_:0] := Module[
  {len = Length[First[mat]],
   newvars, generators, mgb},
  newvars = Array[v, len];
  generators = mat.newvars;
  mgb = moduleGroebnerBasis[
    generators, Variables[mat],
    newvars, CoefficientDomain →
     Rationals, Modulus → mod,
    MonomialOrder →
     DegreeReverseLexicographic];
  Outer[D, Reverse[mgb], newvars]]
```

We next create the appropriate lattice from our lifted factor.

```
deg = Exponent[liftedfactor, y];
lattice1 =
  Table[If[i == j, substpower, 0],
   {i, deg}, {j, totdeg - 2}];
coeffs = PadRight[CoefficientList[
    liftedfactor, y], totdeg - 2];
lattice2 = Table[
   RotateRight[coeffs, j],
   {j, 0, totdeg - 3 - deg}];
lattice = Join[lattice1, lattice2]
```

```
{{(-11 + x)^12, 0, 0, 0},
 {3 + 14 x + 2 x^2 + 15 x^3 + 2 x^4 +
   15 x^5 + 18 x^6 + 18 x^7 + 18 x^8 +
   13 x^9 + 5 x^10 + x^11, 1, 0, 0},
 {0, 3 + 14 x + 2 x^2 + 15 x^3 + 2 x^4 +
   15 x^5 + 18 x^6 + 18 x^7 + 18 x^8 +
   13 x^9 + 5 x^10 + x^11, 1, 0},
 {0, 0, 3 + 14 x + 2 x^2 + 15 x^3 +
   2 x^4 + 15 x^5 + 18 x^6 + 18 x^7 +
   18 x^8 + 13 x^9 + 5 x^10 + x^11, 1}}
```

A reduction will now recover an actual factor, up to a multi−plier in the base field $\mathbb{Z}_{19}$.

```
fac = First[redlat =
      polynomialLatticeReduce[
       lattice, mod]].
   y^Range[0, totdeg / 2]
```

$$12x + 4x^2 + x^3 + (15 + 2x + 11x^2)\, y + 5x y^2 + 5y^3$$

```
poly2
```

$$10x + 16x^2 + 4x^3 + 3y + 8xy + 6x^2 y + x y^2 + y^3$$

```
PolynomialMod[5 * poly2 - fac, mod]
0
```

Summary

We used Gröbner bases over the integers to perform Hensel lifting of factors of univariate polynomials. This is quite computationally efficient, and requires but a few lines of code. We then saw how essentially the same idea applies to bivariate polynomials, using univariate polynomials over a field as our coefficient ring for the Gröbner bases. We also saw in brief how one may recover true factors using Lenstra’s method, again with Gröbner as the engine to do lattice reduction in a univariate polynomial ring.

We also mention that code similar to the polynomial lattice reduction can be used to compute Hermite normal forms. The tandem can be put to use finding low degree solutions to univariate polynomial systems. While efficiency is less than the best possible, it is by no means unreasonable, and the code simplicity is hard to beat. Hence we have Gröbner bases playing a role both in Hensel lifting and lattice algorithms over polynomial rings.

References

W. Adams and P. Loustaunau (1994). An Introduction to Gröbner Bases. Graduate Studies in Mathematics 3. Ameri−can Mathematical Society.

B. Buchberger (1985). Gröbner bases: An algorithmic method in polynomial ideal theory. In Multidimensional Sys-tems Theory, chap 6. N. K. Bose, ed. D. Reidel Publishing Company.

J. von zur Gathen and J. Gerhard (1999). Modern Computer Algebra. Cambridge University Press.

P. Gianni and B. Trager (1985). GCD’s and factoring multi−variate polynomials using Gröbner bases. Proceedings of Eurocal 1985.

M. van Hoeij (2002). Factoring polynomials and the knap−sack problem. Journal of Number Theory 95:167−181.

A. Kandri−Rody and D. Kapur (1988). Computing a Gröb−ner basis of a polynomial ideal over a Euclidean domain. Journal of Symbolic Computation 7:55−69.

A. Lenstra(1985). Factoring multivariate polynomials over finite fields. Journal of Computer and System Sciences **30**:235−248.

D. L. Revisiting strong Gröbner bases over Euclidean domains. Submitted.

H. M. Möller (1988). On the construction of Gröbner bases using syzygies. Journal of Symbolic Computation 6: 345−359.

G. Norton and A. Sălăgean (2001). Strong Gröbner bases and cyclic codes over a finite−chain ring. Workshop on Coding and Cryptography, Paris 2001. Preprint.

L. Pan (1989). On the D−bases of polynomial ideals over principal ideal domains. Journal of Symbolic Computation 7: 81−88.