Implicitization via the Gröbner Walk

Daniel Lichtblau
Wolfram Research, Inc.
100 Trade Center Dr.
Champaign IL USA 61820
danl@wolfram.com

ACA 2007
Approximate Algebraic Computation Session
Oakland University
Rochester MI
July 19−22, 2007

Abstract: The Gröbner walk is a useful method for conversion from a "simple" Gröbner basis to a different one in a desired term order.Various issues along the way include coefficient swell (similar to that seen in the classical Buchberger algo−rithm), polynomials with many initial elements in the "end−game" phase, and the like.We take as benchmark the implicitiza-tion of the 32 bicubic parametric patches in the Newell (Utah) teapot.We will see how the Gröbner walk can be used to implic−itize all of them, using in some cases approximate arithmetic and an early abort strategy, with a result that can be certified a posteriori.To the best of my knowledge the four spout patches have never before been amenable to a Gröbner basis approach. We show some other nontrivial examples from the implicitiza−tion literature.

Introduction

What is the Gröbner walk?

It is a method to convert from an "easy" Gröbner basis to a desired one that might be more difficult to compute directly.

What is it used for?

Many things. Among them, implicitization of parametrized curves and surfaces. This is the application we consider in this talk.

How does it work?

In brief, one moves between cones in the Gröbner fan, beginning with the initial order and moving along a "line" in weight vector space toward the terminal (desired) term order. Good references are (Collart et al 1997) and (Amrhein et al 1997).

The term orders are regarded as weight matrices with each vector comprised of integers. General theory then says they must be square, have full rank, and have the first nonzero in each column be positive.

One could opt for more generality. We do not.

One starts by computing a Gröbner basis with respect to the start order.

Polynomial "initials" are the sum of those monomials in a given polynomial that are equal with respect to the first weight vector.

Each move is relatively simple because it involves only ideals of initials of polynomials in the current basis, and these are mostly monomials.

Convert the initials ideal to get into the neighboring cone.

Use this to carry along the full polynomial set ("lifting" step).

(Optionally) interreduce.

If not yet in terminal cone, restart by looking for the next neighboring cone.

Complexity issues

There are many ways for this process to bog down. The point of this talk is to discuss some, and ways to make them less intolerable. Here are a few items.

The initial basis may be slow to compute.

One typically uses a degree−reverse−lexicographic term order but of course any valid term order might be used.

Coefficient swell as we go through cones can be a problem.

We might encounter cone traversals where the initials ideal contains polynomials with many terms. This more or less defeats the purpose of the Gröbner walk. It is especially common once we are on the boundary of the final cone.

For purposes of variable elimination (needed e.g. for implicitization) it might be unnecessary and inefficient to compute the full basis with respect to a given elimination ordering.

Interreduction of intermediate bases is not actually necessary.

If not done, then one encounters "false cones" problem wherein ties in new initials do not actually get us always to a new cone.

In practice the false cones problem is MUCH worse for efficiency than the slowness of interreduction. We always interreduce.

Tactics we use or otherwise discuss

Perturbation of initial weight vector

This is important because if not done we are more likely to bog down in intermediate Gröbner basis computations involving initials. Reason: We may make a cone crossing that is not through the interior of a proper face, and in this circumstance the initials may have many more monomials than would otherwise be the case. The point of the Gröbner walk is to work with ideals generated by small initials.

Handling of final weight vector

This is important in those (very common) cases where the first weight vector for the final cone puts us at a border of many cones, by virtue of being not in the interior of that cone. There are many ways discussed in the literature for dealing with this.

Do nothing and bear the consequences of a slow final conversion.

(Amrhein et al 1997) Heuristic perturbation. One adds decreasing multiples of later weight vectors to the first one. With probability 1 this puts us in the interior of SOME cone. If it turns out not to be the correct cone we make the multiples smaller and repeat.

(Amrhein et al 1997) Other methods such as fractal walk.

(Tran 2000) Deterministic perturbation. One computes a term order that is a priori in the interior of the final cone.

(Fukuda et al 2007) Use a symbolic perturbation.

My synopsis

Heuristic or simulated perturbation are the methods of choice.

Deterministic perturbation can generate weight vectors with HUGE integers, and will hang in exponent arithmetic, if there are more than 5 or so variables present.

Doing nothing will hang for same reason as Buchberger algorithm.

Other methods are more trouble for less gain than heuristic and simulated perturbation.

Elimination order

As we do elimination in the examples herein, we use a destination term order well suited for this.

Weight monomials with elimination variables higher than monomials without them.

Break ties based on degree reverse lexicographic ordering of the entire varaible set.

This term order is discussed in (Cox et al 2007).

Initial order

It is helpful to use a "sensible" initial ordering.

Could use degree reverse lexicographic.

> *We do this, and pay a cost in the need to compute an initial basis with respect to this term order.*

If we begin with a rational parametrization, could use an initial order such that input, with denominators cleared, is already a Gröbner basis. This would have a weight matrix with initial vector (0,...,0,1,1,...,1).

> *In practice such orders can sometimes cause huge inefficiency in the walk process. We will not use them.*

Early abort for elimination orders

In implicitization and other elimination tasks we can check at intermediate stages of the walk to see if a basis happens to be an elimination basis for the polynomials.

This is defined as a basis G for the ideal I such that, if the noneliminated variable set is Y and the field is K, then $\mathbb{K}[Y] \bigcap G$ generates the elimination ideal $K[Y] \bigcap I$.

If so, we may often use it just as we would the basis we were trying for. We simply take those polynomials in the basis that are free of the variables we were eliminating (see next subsection for why this works). For many purposes, e.g. surface implicitization, this is a single polynomial.

This applies even though the intermediate Gröbner basis is NOT a Gröbner basis with respect to the provided termination ordering. Hence it is a bona fide "short cut".

Early version of this called "sudden death" (Amrhein et al 1997).

Recent version called "ideal−specific elimination order" (Tran 2004).

Me, I’ve always viewed it as "opportunistic intermediate processing".

Algorithmic underpinnings of this early abort strategy

The condition is simple to check algorithmically due to a basic result.

Proposition (Tran 2004): A Gröbner basis G for an ideal I is an elimination basis for a set of variables X if, for each polynomial $g \in G$, either the head term of g contains variables in X, or else g is entirely free of variables in X.

Proof: As above, denote the complementary variable set as Y and the field as $\mathbb{K}$. We refer to polynomials in $\mathbb{K}[Y] \bigcap G$ as G_Y. Take $f \in \mathbb{K}[Y] \bigcap I$. Without loss of generality assume f is reduced with respect to G_Y and the given term order. If $f \neq 0$ then, since G is a Gröbner basis with respect to that term order, f can be further reduced by some $g \in G - G_Y$. But the head term condition shows no such g can reduce f, so $f = 0$. So $f \in G_Y$. Hence G_Y generates $\mathbb{K}[Y] \bigcap I$.□

Approximate arithmetic

In some cases we will use finite precision to "guess" a result. This allows us to avoid intermediate coefficient swell. The result can be verified after the fact. We show this in some examples below.

```
ListPlot::lpn : datalist8k[5] is not a list of numbers or pairs of numbers. >>
```

◄ | ►

Example 5 (Tran 2004)

Here we have a rational parametrization wherein the fourth polynomial is the common denominator. We will implicitize via an old method from (Kalkbrenner 1990).

```
rational6 = {-168 * t - 336 * s * t + 378 * s^2 * t + 36 * t^2 - 1728 * s * t^2 + 1719 * s^2 * t^2 +
    32 * t^3 + 2064 * s * t^3 - 2072 * s^2 * t^3, -100 + 25 * s^2 - 600 * s * t + 600 * s^2 * t +
    132 * t^2 - 144 * s * t^2 + 111 * s^2 * t^2 - 32 * t^3 + 744 * s * t^3 - 736 * s^2 * t^3,
   6 - 6 * s + s^2 + 18 * s * t - 18 * s^2 * t + 36 * s * t^2 - 36 * s^2 * t^2 - 54 * s * t^3 + 54 * s^2 * t^3,
   1 + 6 * s * t - 6 * s^2 * t + 12 * s * t^2 - 12 * s^2 * t^2 - 18 * s * t^3 + 18 * s^2 * t^3};
denom = Last[rational6];
polys6 = Join[vars - Most[rational6] * drecip, {denom * drecip - 1}]
```

```
{-drecip (-168 t - 336 s t + 378 s^2 t + 36 t^2 - 1728 s t^2 + 1719 s^2 t^2 + 32 t^3 + 2064 s t^3 - 2072 s^2 t^3) + x,
 -drecip (-100 + 25 s^2 - 600 s t + 600 s^2 t + 132 t^2 - 144 s t^2 + 111 s^2 t^2 - 32 t^3 + 744 s t^3 - 736 s^2 t^3) +
  y, -drecip (6 - 6 s + s^2 + 18 s t - 18 s^2 t + 36 s t^2 - 36 s^2 t^2 - 54 s t^3 + 54 s^2 t^3) + z,
 -1 + drecip (1 + 6 s t - 6 s^2 t + 12 s t^2 - 12 s^2 t^2 - 18 s t^3 + 18 s^2 t^3)}
```

◄ | ►

Results

Example 1

```
Timing[
 gbHoffmann = GroebnerBasis[HoffmannPolys, vars, elims, MonomialOrder → EliminationOrder,
    Sort → True, Method → {"GroebnerWalk", "EarlyEliminate" → True}];]
```

```
{21.6854, Null}
```

The early elimination check is vital for this example; without it the computation takes around 500 seconds.

Example 2, modulo a reasonably large prime

We can do this.

```
Timing[gbmod = GroebnerBasis[TranPolys, vars, elims, Sort -> True,
    MonomialOrder → EliminationOrder, Modulus → Prime[1 000 001], Method → "GroebnerWalk"];]
```

```
{581.84, Null}
```

But the early elimination check helps tremendously for this example.

```
Timing[gbmod =
   GroebnerBasis[TranPolys, vars, elims, Sort -> True, MonomialOrder → EliminationOrder,
    Modulus → Prime[1 000 001], Method → {"GroebnerWalk", "EarlyEliminate" → True}];]
```

```
{157.078, Null}
```

Of note: the input is already a Gröbner basis with respect to the monomial order given (with variable order $s > t > x > y > z$) by the weight matrix below.

```
{{0, 0, 1, 1, 1}, {1, 1, 0, 0, 0},
  {0, 0, 0, 0, -1}, {0, 0, 0, -1, 0}, {0, -1, 0, 0, 0}}}
```

If we use this as initial order for the Gröbner walk then it instead takes longer than I was willing to wait. So we do not use this order.

Example 2 in characteristic zero

How long to do this in characteristic zero?

```
Timing[gb = GroebnerBasis[TranPolys, vars, elims, Sort -> True,
    MonomialOrder → EliminationOrder, Method → {"GroebnerWalk", "EarlyEliminate" → True}];]
```

```
{17656.9, Null}
```

Ouch. The gremlin in the machinery is the coefficient swell. We will now see how to keep this at bay.

Example 2 approximated, then rationalized

We can obtain a result in reasonable time using approximation, then rationalization, then verification. For this particular example the total time, including verification, is around 12 minutes. We start by computing a numerical Gröbner basis to high precision.

```
Timing[gbapprox = GroebnerBasis[TranPolys, vars, elims,
    MonomialOrder → EliminationOrder, CoefficientDomain → InexactNumbers[600],
    Method → {"GroebnerWalk", "EarlyEliminate" → True}];]
```

```
{383.9, Null}
```

```
implicit = Rationalize[gbapprox[[1]]];
Precision[implicit]
```

```
Infinity
```

```
rules = First[Solve[Tranpolys == 0, vars]];
Timing[Expand[implicit /. rules] == 0]
```

```
{340.485, True}
```

See (Shirayanagi 1996), (Lichtblau 2000) for discussion regarding computation of numerical Gröbner bases.

Another possibility for example 2

Use a prime modulus, then Hensel lift or else use skeleton of result to solve a linear system.

I tried the latter and got a huge sparse system I was unable to solve without blowing up memory.

Might do better when I get some sparse linear algebra for matrices with arbitrary size integers.

Example 3, modulo a reasonably large prime

We will attempt the implicitization of all patches, using a modulus to keep coefficient swell from occurring.

```
Table[Timing[implicitmod[j] = GroebnerBasis[patches3D[[j]], vars, elims,
      MonomialOrder → EliminationOrder, Modulus → Prime[1 000 001], Method →
       {"GroebnerWalk", "EarlyEliminate" → True}];][[1]], {j, 1, Length[patches3D]}]
```

```
{0.032002, 0.032002, 0.028002, 0.032002, 0.088006, 0.052003, 0.052003,
 0.048003, 0.076005, 0.108007, 0.136008, 0.144009, 4.63229, 6.90443, 11.6047,
 11.5567, 147.553, 169.359, 503.495, 548.386, 3.43621, 3.49622, 3.48822, 3.49622,
 0.188011, 0.188012, 0.188012, 0.192012, 0.176011, 0.176011, 0.176011, 0.176011}
```

We do them all, many in "real time". We will show timings for the nonmodular case for all but the four spout parametrizations (patches 17−20), where we will resort to different methods to get the implicitizations.

Example 3 in characteristic zero

```
Table[Timing[implicit[j] =
     GroebnerBasis[patches3D[[j]], vars, elims, MonomialOrder → EliminationOrder,
      Method → {"GroebnerWalk", "EarlyEliminate" → True}];][[1]], {j, 1, 16}]
```

```
{0.044002, 0.032002, 0.032002, 0.032002, 0.108007, 0.112007, 0.116007, 0.116007,
  0.192012, 0.184012, 0.184011, 0.196013, 8.23251, 9.39659, 17.3571, 17.7891}}
```

```
Table[Timing[implicit[j] =
     GroebnerBasis[patches3D[[j]], vars, elims, MonomialOrder → EliminationOrder, Method →
       {"GroebnerWalk", "EarlyEliminate" → True}];][[1]], {j, 21, Length[patches3D]}]
```

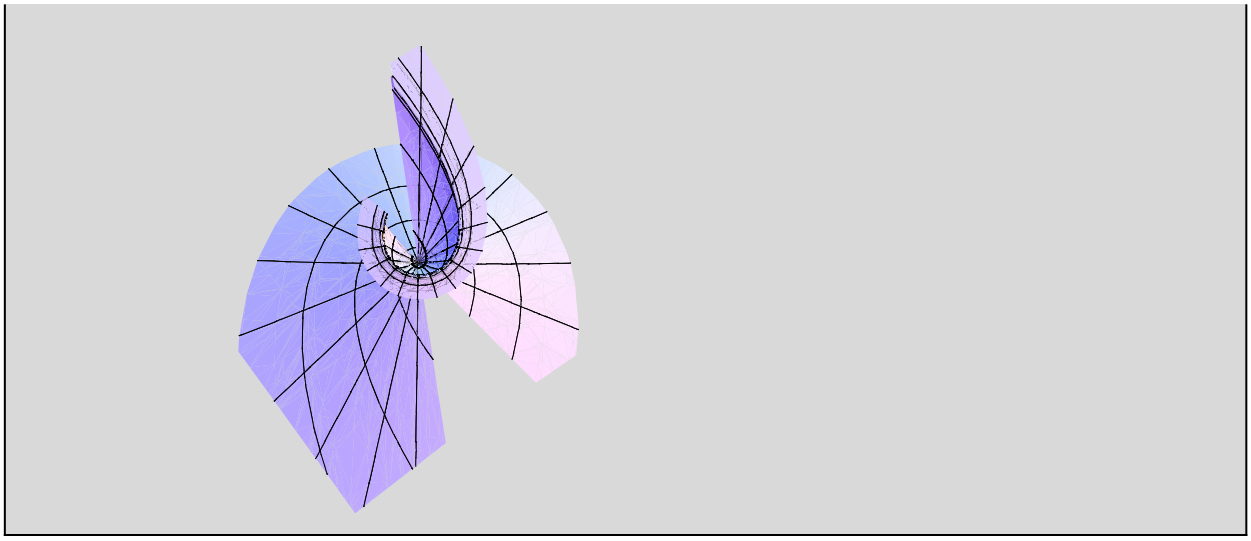
```
{5.67636, 5.73236, 5.78836, 5.77636, 0.220014, 0.224014,
 0.224014, 0.216014, 0.196012, 0.224014, 0.216013, 0.204013}
```

It is interesting to note that without doing a perturbation of the initial weight vector, several of these take substantially longer (more than a factor of 10). They seem to get caught in large intermediate Gröbner basis computations because ideal initials are not always small.

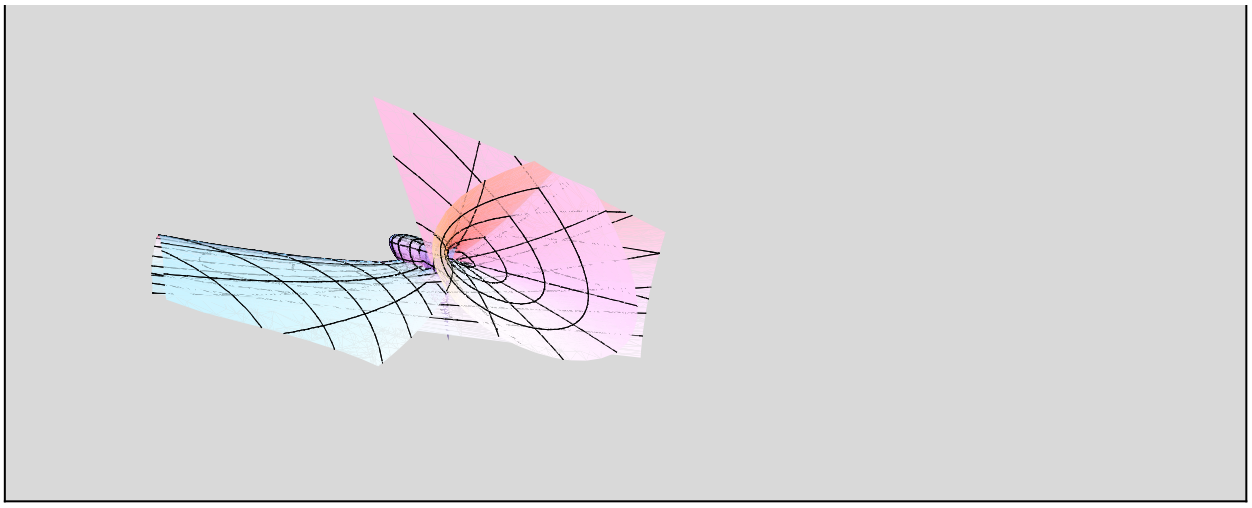
Plots of a few patches may give some idea of why the implicitization difficulty varies so widely.

patch 1

patch 11



patch 19



◄ | ►

One might expect from comparing the modular and characteristic zero timings that the remaining four cases could be handled in reasonable time (an hour each, say). This turns out not to be the case, at least with option settings I have tried. We instead resort to numerical computations followed by rationalization.

```
Timing[implicit17approx = GroebnerBasis[patches3D[[17]], vars, elims,
    MonomialOrder → EliminationOrder, CoefficientDomain → InexactNumbers[700],
    Method → {"GroebnerWalk", "EarlyEliminate" → True}];]
```

```
{445.356, Null}
```

We rationalize, clear denominators (not necessary, but nice to have result with integer coefficients), plug in original relations and check that it expands to zero.

```
implicit17attempt = Rationalize[implicit17approx];
imp2 = Apply[List, implicit17attempt[[1]]];
imp2 = imp2 /. {x → 1, y → 1, z → 1};
den = Denominator[imp2];
den = Apply[LCM, den];
implicit17 = Expand[den * implicit17attempt[[1]]];
rules17 = First[Solve[patches3D[[17]] == 0, vars]];
Timing[Expand[implicit17 /. rules17] == 0]
```

```
{280.038, True}
```

Example 3 cont'd

We just show timings for computing the remaining three patch implicitizations to 700 digits. That turns out to suffice for rationalizing and recovering the correct exact polynomials.

```
Timing[implicit18approx = GroebnerBasis[patches3D[[18]], vars, elims,
    MonomialOrder → EliminationOrder, CoefficientDomain → InexactNumbers[700],
    Method → {"GroebnerWalk", "EarlyEliminate" → True}];]
```

```
{465.629, Null}
```

```
Timing[implicit19approx = GroebnerBasis[patches3D[[19]], vars, elims,
    MonomialOrder → EliminationOrder, CoefficientDomain → InexactNumbers[700],
    Method → {"GroebnerWalk", "EarlyEliminate" → True}];]
```

```
{1216.11, Null}
```

```
Timing[implicit20approx = GroebnerBasis[patches3D[[20]], vars, elims,
    MonomialOrder → EliminationOrder, CoefficientDomain → InexactNumbers[700],
    Method → {"GroebnerWalk", "EarlyEliminate" → True}];]
```

```
{1291.09, Null}
```

Example 4

This next example requires use of a perturbation of the target weight vector. Moreover an a priori computation a la (Tran 2000) is out of the question because the number of variables would cause the exponent vectors to have too many digits to allow the arithmetic to be feasible. We use the method of (Amrheim et al) though the simulated perturbation of (Fukuda et al) would also work here.

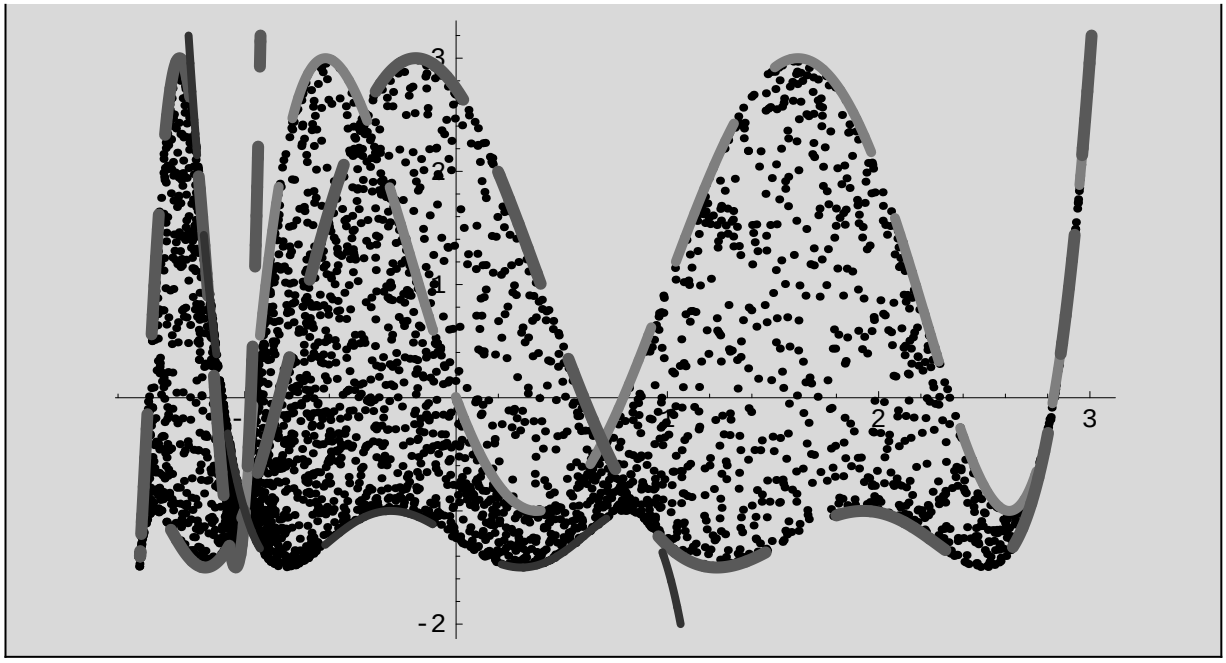
```
Timing[implicitpoly =
  Factor[First[GroebnerBasis[polys, mainvars, elimvars, Sort → True, MonomialOrder →
      EliminationOrder, Method -> {"GroebnerWalk", "EarlyEliminate" → True}]]]]
```

```
{125.79586199999999,
 (-5 * x + 5 * x^2 + 5 * x^3 - 5 * x^4 + x^5 - y) * (5 + 5 * x - 20 * x^3 + 16 * x^5 + 4 * y) *
  (5400 + 11 700 * x - 8675 * x^2 - 37 800 * x^3 - 20 600 * x^4 + 14 880 * x^5 +
    13 680 * x^6 - 1920 * x^7 - 3200 * x^8 + 256 * x^10 - 1980 * y - 6230 * x * y -
    7280 * x^2 * y - 3800 * x^3 * y - 800 * x^4 * y - 32 * x^5 * y + y^2)}
```

Note that the result factors nontrivially, that is, it is not prime. So it could not be the implicitization of a rational parametrization.

"The envelope, please."

(I always wanted to say that).



◄ | ►

Example 5

```
Timing[gbrat6 =
   GroebnerBasis[polys6, vars, Prepend[elims, drecip], MonomialOrder → EliminationOrder,
    Sort → True, Method → {"GroebnerWalk", "EarlyEliminate" → True}];]
```

```
{64.244, Null}
```

By using approximate arithmetic of 300 digits one can obtain the same result in about 24 seconds. Not quite "real time" but getting reasonably close...

Summary

What we can do

Implicitization of "difficult" polynomial, rational, and algebraic parametrizations from the literature.

Many other elimination and related problems are also amenable to effective handling via Gröbner walk computations.

Necessary tactics

Perturbation of initial weight vector.

Other than when reaching boundary of final cone, this will help to avoid "large initials" problem.

Always interreduce intermediate bases.

Failure to do so will cause huge blowup due to "false cones" problem.

It will probably also cause crop failures. At least, it has not been proven otherwise.

When indicated, perturbation of final weight vector. Viable possibilities are

Heuristic

Simulated

Early finish when we discover we have attained an elimination ideal

Rarely hurts, never by much, and sometimes helps substantially.

Within variable classes (elimination, not elimination), sort using some sensible heuristic

Usually improves on input order unless you know a "good" variable ordering a priori.

In some cases we must use approximate arithmetic, then rationalize to recover the correct exact result

For some purposes an approximate result, particularly one to high precision, might suffice.

Open bottlenecks

Computation of the initial basis can be costly.

For implicitization one can use initial orders that make the input, in effect, a basis already.

This is not true for nonrational parametrizations.

It generally does not help, and in some cases makes the overall walk much slower.

Provisional conclusion: Use degree reverse lexicographic as initial ordering unless there is a compelling reason not to do so.

Interreduction of intermediate bases is generally the most time consuming step.

Per earlier remarks, it is at present unavoidable.

"Lifting" from initials basis to full basis seems to be second most costly.

Method of (Fukuda et al 2007) might improve on pedestrian method from earlier literature?

The sheer number of cones, large polynomials that complicate the lifting and/or interreduction step, and coefficient swell still pose serious bottlenecks in some problems.

But, as examples above indicate, we are showing progress using the Gröbner walk for implicitization and other computations.

References

B. Amrhein, O. Gloor, and W. Küchlin. On the walk.Theoretical Computer Science **187**:179−202. 1997.

S. Collart, M. Kalkbrenner, and D. Mall. Converting bases with the Gröbner walk. Journal of Symbolic Computation **24**:465−469. 1997.

D. Cox, J. Little, and D. O'Shea. Ideals, Varieties, and Algorithms: An Introduction to Computational ALgebraic Geometry and Commutative Algebra (third edition). Undergraduate Texts in Mathematics. Springer−Verlag, 2007.

K. Fukuda, A. N. Jensen, N Lauritzen, and R. Thomas. The generic Gröbner walk. Journal of Symbolic Computation **42**:198−213. 2007.

M. Kalkbrenner. Implicitization of rational parametric curves and surfaces. AAECC−8, Lecture Notes in Computer Science **508**:249−259, S. Sakata, ed. Springer, 1990.

R. Lewis. Private communication, 2007.

D. Lichtblau. Solving finite algebraic systems using numeric Gröbner bases and eigenvalues. Proceedings of SCI 2000 **10**:555−560. 2000.

D. Lichtblau. Computing curves bounding trigonometric planar maps: symbolic and hybrid methods. Proceedings of the Workshop on Automated and Deductive Geometry (ADG 2004). Lecture Notes in Artificial Intelligence **3763**:70−91, H. Hong and D. Wang, eds. Springer−Verlag, 2006.

K. Shirayanagi. Floating point Gröbner bases. Mathematics and Computers in Simulation **42**:509−528. 1996.

Q−N Tran. A fast algorithm for Gröbner basis conversion and its applications. Journal of Symbolic Computation **30**:451−467. 2000.

Q−N Tran. Efficient Groebner walk conversion for implicitization of geometric objects. Computer Aided Geometric Design **21**:837−857. 2004.

Q−N Tran. A new class of term orders for elimination. Journal of Symbolic Computation **42**:533−548. 2007.