

Computer Assisted Mathematics: Tools and Tactics for Solving Hard Problems

Daniel Lichtblau
danl@wolfram.com

Wolfram Research, Inc.
October/November, 2006

This talk was given at IMA on November 2, 2006. It is a minor variant of one given at the *Mathematica* Technology Conference in October 2006. Caveat: This was prepared using the version of *Mathematica* under development as of October 2006. A few examples will not work quite the same way in version 5.

Abstract

In this talk I will present several problems that have caught my attention over the past few years. We will go over *Mathematica* formulations and solutions. Along the way we will meet with a branch-and-bound loop in its natural habitat, some rampaging Gröbner bases, a couple of tamed logic puzzles, and at least a dozen wild beasts (well...would you believe... a *Mathematica* developer wearing Halloween fangs?). You'll laugh. You'll cry. You'll write your holiday cards (be sure to spell my name correctly).

As the purpose is to illustrate a few of the many ways in which *Mathematica* can be used to advantage in tackling difficult problems, we will go into a bit of detail in selected examples. Do not let this deter you; there will be no exam, and it is the methods, not the problems, that are of importance. The examples are culled from problems I have seen on Usenet groups (primarily *MathGroup*), in articles, or have been asked in person.

(1) A simple math problem

At a workshop in 2004 I was posed the following: Find a positive integer such that, when multiplied by 9, you get the same result as by moving the least significant digit to the most significant. The person asking is editor-in-chief of the *Journal of Symbolic Computation*. He told me the (smallest) result had 44 digits, and it had taken him substantial computational effort to find it.

We code this as a simple integer programming problem, directly invoking the *Mathematica* function `Reduce` in order to solve it.

```
In[139]:=
timesBy9OrRotateDigits[j_] :=
Module[{n, m, k, res}, res = {ToRules[Reduce[{n == 10 * m + k, 9 * n == m + 10^j * k,
Element[{n, m, k}, Integers], m >= 0, 1 <= k <= 9, n >= 10^j}, {n, m, k}]]];
If[res === {}, {}, {j + 1, First[n /. res]}]
```

We confirm below that the smallest solution has 44 digits. It can be shown to be the repeating block of the fraction $9/89$.

```
In[140]:= Timing[Table[timesBy9OrRotateDigits[j], {j, 45}] /. {} → Sequence[]]
Out[140]= {0.192011, {{44, 10 112 359 550 561 797 752 808 988 764 044 943 820 224 719}}}
```

His comment: "That's correct! But you cheated, using *Mathematica*:--)"

He, alas, had used another program...

(2) Algebraic replacement of expressions

This is a topic of some apparent interest insofar as it shows up with regularity on the *Mathematica* user's Usenet group comp.softsys.math.mathematica (aka *MathGroup*). One has a perhaps complicated expression, and wishes to replace subexpressions using perhaps a new "variable" to encapsulate them. The issue is that they might not be amenable to the familiar syntactic replacement rules of *Mathematica*.

For example, one might wish to replace $a^2 b$ with a new variable, and likewise have $a^4 b^2$ replaced by the square of that variable, etc. The particular example below comes from a recent *MathGroup* question [algebraic replacement].

```
In[11]:= expr = -((1 + 2 * n) * ((a^4 * k^2 + a^2 * (-1 + k^2 * (q - z)^2) + 2 * (q - z)^2) *
    Cos[k * Sqrt[a^2 + (q - z)^2]] - k * (a^2 - 2 * (q - z)^2) *
    Sqrt[a^2 + (q - z)^2] * Sin[k * Sqrt[a^2 + (q - z)^2]]) *
    Sin[(1 + 2 * n) * Pi * z / L] / (8 * Pi * w * (a^2 + (q - z)^2)^(5 / 2));
```

The desire was to replace $\sqrt{a^2 + (q - z)^2}$ with a new variable R .

I have written code to address this sort of question on a few occasions, and it seems to get a bit more complicated each time as the mission of the replacement gets more elaborate. Below is the current form I use. The key is to use the function `PolynomialReduce` to do a sort of "generalized division" in order to get an algebraic replacement.

```
In[12]:= replacementFunction[expr_, rep_, vars_] :=
    With[{num = Numerator[expr], den = Denominator[expr], hed = Head[expr]},
        If[PolynomialQ[num, vars] && PolynomialQ[den, vars],
            PolynomialReduce[num, rep, vars][[2]] / PolynomialReduce[den, rep, vars][[2]],
            If[Head[hed] === Symbol && MemberQ[Attributes[hed], NumericFunction],
                Map[replacementFunction[#, rep, vars] &, expr], expr]
        ]
```

There is a price to pay for getting the power of algebraic reduction. We really need to use polynomials in our replacements, and that means we cannot work directly with the desired radical. This is unfortunate in that it will leave us with radicals of powers, but we can fix that using e.g. `PowerExpand` or (as a post-processing step) a syntactic replacement rule.

```
In[13]:= replacementFunction[expr, a^2 + (q - z)^2 - R^2, {a, q, z}] /.
    (R^aa_) ^ (bb_) :> R^ (aa * bb)
```

```
Out[13]= -\frac{1}{8 \pi R^5 w} (1 + 2 n) ((-R^2 + R^4 + q^2 (3 - R^2) + q (-6 + 2 R^2) z + (3 - R^2) z^2) Cos[R] -
    R (-3 q^2 + R^2 + 6 q z - 3 z^2) Sin[R]) Sin\left[\frac{(\pi + 2 n \pi) z}{L}\right]
```

Had we multiple replacements, perhaps with interdependencies, then we might want to form a Gröbner basis prior to use of `PolynomialReduce`. That's another topic.

(3) Gröbner bases

I do not propose to explain in any detail what these are. For our purposes they are to be regarded as an algebraic form of "rewriting rules". That is to say, we have polynomials and we want to rewrite according to some (polynomial) equivalence relations, perhaps eliminating some variables along the way.

Various tactics are used to extend this idea to handling rational functions and radicals. We will see examples of this in later sections.

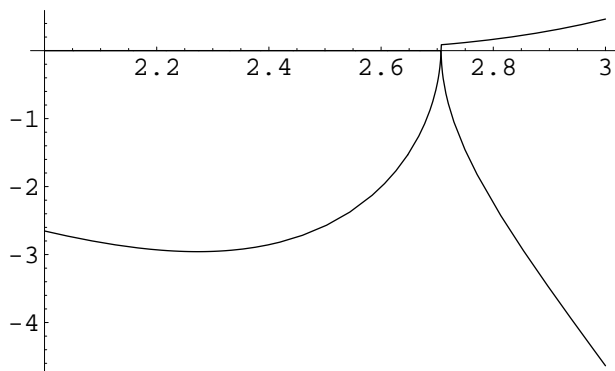
(4) Unraveling a parametrized nested radical

This next problem was posed last year on *MathGroup* [parametrized root].

$$\begin{aligned} \text{In}[1]:= \text{expr} = & \left(-1 + \sqrt{2} - \sqrt{(1 + 2\sqrt{2} - s)(-1 + s) + s} \right) \\ & \left(-1 - \sqrt{2} + \sqrt{2}s - \sqrt{-1 - 2\sqrt{2} + 2(2 + \sqrt{2} - s)s} \right) - \\ & \left(-1 - \sqrt{2} + \sqrt{2}s + \sqrt{-1 - 2\sqrt{2} + 2(2 + \sqrt{2} - s)s} \right) \\ & \left(-2 - 2\sqrt{2} + 2s + 2\sqrt{\left(1 - \frac{1}{4} \left(-1 - \sqrt{2} + \sqrt{(1 + 2\sqrt{2} - s)(-1 + s) + s}\right)^2\right)} \right); \end{aligned}$$

This is an algebraic function and we wish to find the exact solution where it vanishes around $s \approx 2.707$. A brief look shows that, at the root, it "goes complex" as the parameter s increases.

```
In[2]:= Plot[{Re[expr], Im[expr]}, {s, 2, 3}]
```



```
Out[2]= - Graphics -
```

One way is to create a Gröbner basis (in effect, describing the vanishing set `expr==0`) and let it do all the hard work of sorting out the algebraic dependencies.

```
In[6]:= gb = GroebnerBasis[expr, s];
```

In general we might hope there is a "reasonable" element in the basis, in particular one that is explicitly polynomial in `s`. And indeed it turns out the first element satisfies that description.

```
In[151]:= poly = First[gb]
```

```
Out[151]= 541 248 + 382 720  $\sqrt{2}$  + (-4 396 864 - 3 109 056  $\sqrt{2}$ ) s + (16 646 016 + 11 770 496  $\sqrt{2}$ ) s2 +
(-38 899 008 - 27 505 600  $\sqrt{2}$ ) s3 + (62 625 368 + 44 282 664  $\sqrt{2}$ ) s4 +
(-73 400 488 - 51 902 592  $\sqrt{2}$ ) s5 + (64 510 372 + 45 616 892  $\sqrt{2}$ ) s6 +
(-43 135 784 - 30 501 092  $\sqrt{2}$ ) s7 + (22 023 157 + 15 570 312  $\sqrt{2}$ ) s8 +
(-8 531 504 - 6 031 944  $\sqrt{2}$ ) s9 + (2 464 184 + 1 744 512  $\sqrt{2}$ ) s10 +
(-513 920 - 364 928  $\sqrt{2}$ ) s11 + (73 888 + 51 584  $\sqrt{2}$ ) s12 + (-7168 - 3968  $\sqrt{2}$ ) s13 + 512 s14
```

We now want to figure out a minimal polynomial component that vanishes near 2.707. First we find the vanishing point more precisely.

```
In[8]:= acc = 100;
sroot = FindRoot[expr == 0, {s, 2.7},
AccuracyGoal -> acc, PrecisionGoal -> acc, WorkingPrecision -> (1.2 * acc)];
```

Now we factor our polynomial and find the factor that vanishes at `sroot`.

```
In[11]:= fax = Drop[Map[First, FactorList[poly, Extension -> Automatic]], 1];
minpoly = First[Select[fax, Chop[# /. sroot, 10^(-acc)] == 0 &]]
```

```
Out[12]= -136 - 96  $\sqrt{2}$  + (512 + 364  $\sqrt{2}$ ) s + (-804 - 578  $\sqrt{2}$ ) s2 +
(676 + 486  $\sqrt{2}$ ) s3 + (-329 - 220  $\sqrt{2}$ ) s4 + (96 + 44  $\sqrt{2}$ ) s5 - 16 s6
```

We now use `Solve` to find our value as a `Root` object.

```
In[176]:= First[Select[s /. Solve[minpoly == 0, s], Abs[# - (s /. sroot)] < 10^(-acc / 5) &]]
```

```
Out[176]= Root[64 + 512 #1 - 6112 #12 + 21 024 #13 - 32 136 #14 + 10 832 #15 + 43 568 #16 -
86 152 #17 + 81 425 #18 - 46 080 #19 + 15 872 #110 - 3072 #111 + 256 #112 &, 8]
```

We now use some number theory functionality, buried under the hood and based on lattice reduction, to deduce the same result from the high precision approximating value. We assume the result will have a minimal polynomial of degree no larger than 20. This formerly required an add-on package.

```
In[177]:= RootApproximant[s /. sroot, 20]
```

```
Out[177]= Root[64 + 512 #1 - 6112 #12 + 21 024 #13 - 32 136 #14 + 10 832 #15 + 43 568 #16 -
86 152 #17 + 81 425 #18 - 46 080 #19 + 15 872 #110 - 3072 #111 + 256 #112 &, 8]
```

It turns out that for this particular example, if we let `RootApproximant` guess the degree it will come up with the right result.

```
In[178]:= RootApproximant[s/.sroot]
Out[178]= Root[64 + 512 #1 - 6112 #1^2 + 21024 #1^3 - 32136 #1^4 + 10832 #1^5 + 43568 #1^6 - 86152 #1^7 + 81425 #1^8 - 46080 #1^9 + 15872 #1^10 - 3072 #1^11 + 256 #1^12 &, 8]
```

(5) Implicitizing a rational surface

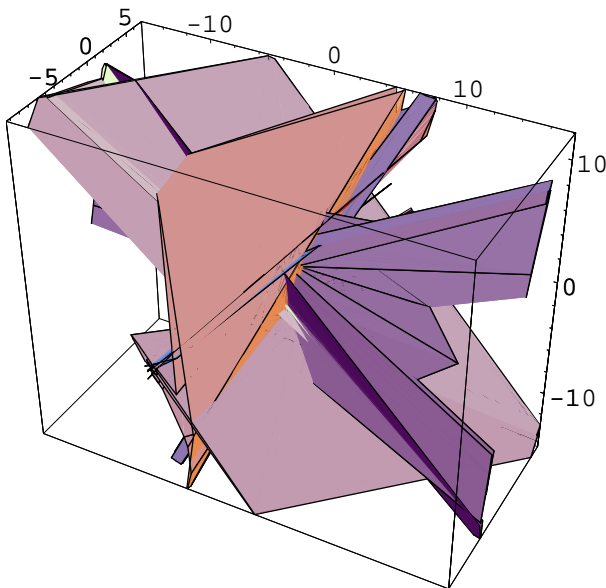
We present an example of a parametrized rational surface, wherein we use a Gröbner basis to find the implicit form. The example comes from fairly recent literature [parametrized surface]. Why do we want to implicitize it? Well, there might be many reasons, but one is that it actually is a bit easier to visualize because the plot routines seem to handle that form better. I think this is because the curve where the parametric form denominator vanishes is problematic, and moreover there are several base points (where numerators ALSO vanish), near which we are likely to confuse the plotting routines.

We regard the parametric form as a system of polynomials x, y, z each parametrized by s, t but we also have a common denominator w likewise parametrized.

```
In[1]:= w = t^3 + t^2 - t + s^2 t - 1 - s + s^2 + s^3;
polys = {2 t^3 + 4 t^2 + 2 t + 4 s t + s^2 t + 2 + 3 s + s^2,
-2 s t^2 - 2 t - s t + 2 + s - 2 s^2 - s^3, 2 t^2 - 3 s t^2 - 2 t - 3 s t - 2 s^2 t - 2 s - 3 s^2 - s^3};
```

So let's have a look at this thing. I confess it is a bit tricky to get the parametric plot to do anything nice with it.

```
In[21]:= L = 2.2;
ParametricPlot3D[Evaluate[polys/w], {s, -L, L}, {t, -L, L}, PlotPoints -> 15];
```



A bit of a mess. For future reference the code below, contributed by Michael Trott MMAL ("*Mathematica* Master At Large"), will do a reasonable job in the next release of *Mathematica*.

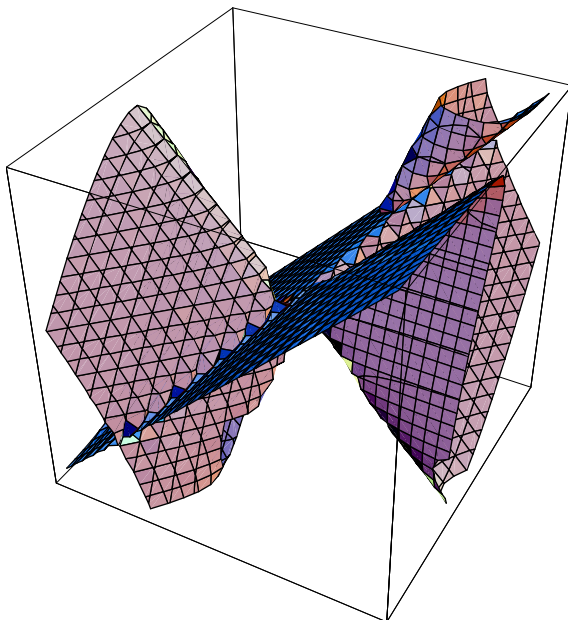
```
L = 2.2;
ParametricPlot3D[Evaluate[polys / w],
  {s, -L, L}, {t, -L, L}, Mesh -> False, PlotPoints -> 40,
  PlotRange -> 5, Exclusions -> {w == 0, Offset -> 0.01, MaxRecursion -> 3},
  ColorFunction -> (Hue[#5] &)]
```

We will implicitize. How? What one might try is to form a Gröbner basis of polynomials $\{w x - x(s, t), w y - y(s, t), w z - z(s, t)\}$ where w is an explicit polynomial in the parameters, (x, y) are distinct variables, and $(x(s, t), y(s, t))$ are again explicit polynomials. The goal would be to eliminate the parameters (s, t) . This alone does not work due to presence of base points. One repairs the situation by adding a polynomial $w r - 1$ in a new "reciprocal" variable r . This relation forces the denominator to be nonzero. We add this new variable to the list of those we seek to eliminate.

```
In[3]:= vars = {x, y, z};
        elims = {s, t, r};
        Timing[implicit = First[GroebnerBasis[
          Append[w*vars - polys, w*r - 1], vars, elims, MonomialOrder -> EliminationOrder]]]
Out[5]= {0.341 Second, -4 x + 2 x^2 - 3 x y + 3 x^2 y - y^2 - 3 x y^2 + 2 z - x z - x^2 z + 4 y z + 3 y^2 z - 3 y z^2 + z^3}
```

Nothing to it. Now we do a contour plot.

```
In[6]:= << Graphics`ContourPlot3D`
In[15]:= l = 20;
          ContourPlot3D[Evaluate[implicit], {x, -l, l},
            {y, -l, l}, {z, -l, l}, Contours -> {0}, PlotPoints -> 6];
```



Nicer, I think. And MUCH nicer in future...

(6) Puzzle: Johnny's ideal woman

This puzzle appeared in a university publication in 1998. It was asked in *MathGroup* in 2003 [[Johnny's ideal woman](#)].

Johnny's ideal woman is brunette, blue eyed, slender, and tall. He knows four women: Adele, Betty, Carol, and Doris. Only one of the four women has all four characteristics that Johnny requires.

1. Only three of the women are both brunette and slender.
2. Only two of the women are both brunette and tall.
3. Only two of the women are both slender and tall.
4. Only one of the women is both brunette and blue eyed.
5. Adele and Betty have the same color eyes.
6. Betty and Carol have the same color hair.
7. Carol and Doris have different builds.
8. Doris and Adele are the same height.

At IMS 2004 I showed a method that had been presented in `comp.soft-sys.math.mathematica` by Lars Rasmussen. It involves setting up and solving a system of equations (which, under the hood, uses a Gröbner basis).

Here I will show a logic programming approach. It is simple, yet effective. Also of interest is the contrast to a method we will cover later. The method below is similar to that of a couple of responses that appeared in the forum. I think it is generally more effective, however, in that it makes use of intermediate pruning steps so that set sizes do not get very large. Another place this approach was used to advantage was by Michael Trott (MMAL) in solving the notorious "Who owns the zebra" puzzle [[Trott TMJ article](#)].

The idea is to enlarge, via enumeration of possibilities, and then prune the search space iteratively as new variables are taken into account. The enumeration steps consist of adding all possible values of a new variable to each partial solution. These are then pruned using available constraints in an effort to keep the size of the partial solution set reasonable. We will illustrate the method on this example. We begin by showing the power set of eye combinations that can go with the four women (we will not show this for the remaining features).

```
In[68]:= gals = {Adele, Betty, Carol, Doris};
         features = {blueeye, slender, brunette, tall};
```

```
In[70]:= eyes = Flatten[Outer[List, Sequence @@ Table[{blueeye, Not[blueeye]}, {4}], 3]
```

```
Out[70]= {{blueeye, blueeye, blueeye, blueeye}, {blueeye, blueeye, blueeye, !blueeye},
          {blueeye, blueeye, !blueeye, blueeye}, {blueeye, blueeye, !blueeye, !blueeye},
          {blueeye, !blueeye, blueeye, blueeye}, {blueeye, !blueeye, blueeye, !blueeye},
          {blueeye, !blueeye, !blueeye, blueeye}, {blueeye, !blueeye, !blueeye, !blueeye},
          {!blueeye, blueeye, blueeye, blueeye}, {!blueeye, blueeye, blueeye, !blueeye},
          {!blueeye, blueeye, !blueeye, blueeye}, {!blueeye, blueeye, !blueeye, !blueeye},
          {!blueeye, !blueeye, blueeye, blueeye}, {!blueeye, !blueeye, blueeye, !blueeye},
          {!blueeye, !blueeye, !blueeye, blueeye}, {!blueeye, !blueeye, !blueeye, !blueeye}}
```

We form the lists of possibilities combining women with eye color.

```
In[71]:= galsAndEyes = Flatten[Outer[Thread[And[###] &, {gals}, eyes, 1], 1]

Out[71]= {{Adele && blueeye, Betty && blueeye, Carol && blueeye, Doris && blueeye},
  {Adele && blueeye, Betty && blueeye, Carol && blueeye, Doris && ! blueeye},
  {Adele && blueeye, Betty && blueeye, Carol && ! blueeye, Doris && blueeye},
  {Adele && blueeye, Betty && blueeye, Carol && ! blueeye, Doris && ! blueeye},
  {Adele && blueeye, Betty && ! blueeye, Carol && blueeye, Doris && blueeye},
  {Adele && blueeye, Betty && ! blueeye, Carol && blueeye, Doris && ! blueeye},
  {Adele && blueeye, Betty && ! blueeye, Carol && ! blueeye, Doris && blueeye},
  {Adele && blueeye, Betty && ! blueeye, Carol && ! blueeye, Doris && ! blueeye},
  {Adele && ! blueeye, Betty && blueeye, Carol && blueeye, Doris && blueeye},
  {Adele && ! blueeye, Betty && blueeye, Carol && blueeye, Doris && ! blueeye},
  {Adele && ! blueeye, Betty && blueeye, Carol && ! blueeye, Doris && blueeye},
  {Adele && ! blueeye, Betty && blueeye, Carol && ! blueeye, Doris && ! blueeye},
  {Adele && ! blueeye, Betty && ! blueeye, Carol && blueeye, Doris && blueeye},
  {Adele && ! blueeye, Betty && ! blueeye, Carol && blueeye, Doris && ! blueeye},
  {Adele && ! blueeye, Betty && ! blueeye, Carol && ! blueeye, Doris && blueeye},
  {Adele && ! blueeye, Betty && ! blueeye, Carol && ! blueeye, Doris && ! blueeye}}
```

Now we use the relevant constraint to prune this list.

```
In[72]:= galsAndEyes = Cases[galsAndEyes, {AorB_ && a_, ___, BorA_ && a_, ___} /;
  MatchQ[AorB, Adele | Betty] && MatchQ[BorA, Adele | Betty]]

Out[72]= {{Adele && blueeye, Betty && blueeye, Carol && blueeye, Doris && blueeye},
  {Adele && blueeye, Betty && blueeye, Carol && blueeye, Doris && ! blueeye},
  {Adele && blueeye, Betty && blueeye, Carol && ! blueeye, Doris && blueeye},
  {Adele && blueeye, Betty && blueeye, Carol && ! blueeye, Doris && ! blueeye},
  {Adele && ! blueeye, Betty && ! blueeye, Carol && blueeye, Doris && blueeye},
  {Adele && ! blueeye, Betty && ! blueeye, Carol && blueeye, Doris && ! blueeye},
  {Adele && ! blueeye, Betty && ! blueeye, Carol && ! blueeye, Doris && blueeye},
  {Adele && ! blueeye, Betty && ! blueeye, Carol && ! blueeye, Doris && ! blueeye}}
```

We add an attribute for girth.

```
In[73]:= widths = Flatten[Outer[List, Sequence@@Table[{slender, Not[slender]}, {4}], 3];
galsAndEyesAndWidth = Flatten[Outer[Thread[And[###] &, galsAndEyes, widths, 1], 1];
```

This time we can prune with more than one constraint.

```
In[92]:= galsAndEyesAndWidth =
  Cases[galsAndEyesAndWidth, {___, And[CorD_, ___, a_], ___, And[DorC_, ___, b_], ___} /;
  MatchQ[CorD, Carol | Doris] && MatchQ[DorC, Carol | Doris] && b == Not[a]];
galsAndEyesAndWidth = Select[galsAndEyesAndWidth,
  Length[Select[#, Function[x, MatchQ[x, _ && blueeye && slender]]] == 3 &];
```

Now add hair coloring.

```
In[79]:= hairs = Flatten[Outer[List, Sequence@@Table[{brunette, Not[brunette]}, {4}], 3];
galsAndEyesAndWidthAndHair =
  Flatten[Outer[Thread[And[###] &, galsAndEyesAndWidth, hairs, 1], 1];
```

Again we prune.

```
In[83]:= galsAndEyesAndWidthAndHair = Cases[galsAndEyesAndWidthAndHair,
  {___, And[BorC_, ___, a_], ___, And[CorB_, ___, a_], ___} /;
  MatchQ[BorC, Betty | Carol] && MatchQ[CorB, Betty | Carol]];
galsAndEyesAndWidthAndHair = Select[galsAndEyesAndWidthAndHair,
  Length[Select[#, Function[x, MatchQ[x, _ && blueeye && _ && brunette]]] == 1 &];
```

Finally add height.

```
In[85]:= heights = Flatten[Outer[List, Sequence@@Table[{tall, Not[tall]}, {4}], 3];
galsAndEyesAndWidthAndHairAndHeight =
  Flatten[Outer[Thread[And[###] &, galsAndEyesAndWidthAndHair, heights, 1], 1];
galsAndEyesAndWidthAndHairAndHeight = Cases[galsAndEyesAndWidthAndHairAndHeight,
  {___, And[AorD_, _, _, a_], ___, And[DorA_, _, _, a_], ___} /;
  MatchQ[AorD, Adele | Doris] && MatchQ[DorA, Adele | Doris]];
```

This time we prune with three constraints. When finished, we will have only one ideal woman (who, as it happens, repeats herself. Possibly Johnny was not aware of this trait.)

```
In[88]:= galsAndEyesAndWidthAndHairAndHeight =
  Flatten[Outer[Thread[And[###] &, galsAndEyesAndWidthAndHair, heights, 1], 1];
galsAndEyesAndWidthAndHairAndHeight = Cases[galsAndEyesAndWidthAndHairAndHeight,
  {___, And[AorD_, _, _, a_], ___, And[DorA_, _, _, a_], ___} /;
  MatchQ[AorD, Adele | Doris] && MatchQ[DorA, Adele | Doris]];
galsAndEyesAndWidthAndHairAndHeight = Select[galsAndEyesAndWidthAndHairAndHeight,
  Length[Select[#, Function[x, MatchQ[x, _ && blueeye && _ && _ && tall]]] == 2 &];
galsAndEyesAndWidthAndHairAndHeight = Select[galsAndEyesAndWidthAndHairAndHeight,
  Length[Select[#, Function[x, MatchQ[x, __ && brunette && tall]]] == 2 &];

Out[91]= {{Adele && blueeye && slender && brunette && tall,
  Betty && blueeye && slender && !brunette && tall,
  Carol && blueeye && slender && !brunette && !tall,
  Doris && !blueeye && !slender && brunette && tall},
  {Adele && blueeye && slender && brunette && tall,
  Betty && blueeye && slender && !brunette && !tall,
  Carol && blueeye && slender && !brunette && tall,
  Doris && !blueeye && !slender && brunette && tall}}
```

We see that only Adele has the requisite features.

(7) The envelope, please (a curvy boundary)

Or: How Gröbner bases saved me a lot of work, after I already did it

This is a problem I first encountered around eight years ago, and have spoken about on a few occasions [boundary curve]. It originated in a doctoral dissertation in number theory. The investigation focused on a trigonometric map between planar regions, and one question concerned finding the envelope curve(s) bounding the range.

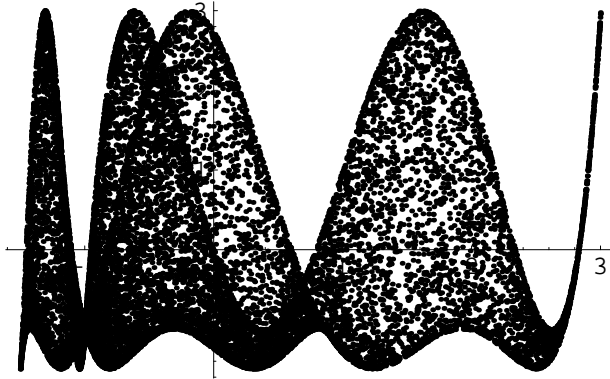
```
In[17]:= trigpoly[n_, a_, b_] = Cos[n a] + Cos[n b] + Cos[n (a - b)];
```

I will work with a variant that is computationally more challenging than the original one.

```
In[18]:= x[a_, b_] = trigpoly[1, a, b];
y[a_, b_] = trigpoly[5, a, b];
```

We plot the images of around 16000 random points from the square $[0, 2\pi] \times [0, 2\pi]$.

```
In[25]:= pointlist[m_] := Table[2 * Pi * {Random[], Random[]}, {2^m}]
points2k = pointlist[14];
data[{point__}] := {x[point], y[point]};
datalist2k = Map[data, points2k];
ListPlot[datalist2k, PlotRange -> All];
```



There are several ways to go about finding the boundary curve. The one I used was to cast as an optimization problem using Lagrange multipliers (one might alternatively look for a vanishing Jacobian). The idea is for fixed x coordinate, find corresponding y values that are minimal/maximal. As we work with trig functions we will make them polynomial variables and augment with the usual identity $\sin^2 + \cos^2 = 1$. There are alternatives but this is what I found to work well.

```
In[41]:= parameters = {a, b};
mainvars = {x, y};
xpoly = x - x[a, b];
ypoly = y - y[a, b];
xypolys = {xpoly, ypoly};
grad[expr_, a_, b_] := {D[expr, a], D[expr, b]};
trigsubs = {Cos[a] -> ca, Sin[a] -> sa, Cos[b] -> cb, Sin[b] -> sb};
gradientpolys = grad[xpoly, a, b] - λ grad[ypoly, a, b];
trigidentities = {ca^2 + sa^2 - 1, cb^2 + sb^2 - 1};
elimvars = {λ, ca, cb, sa, sb};
polys = TrigExpand[Join[trigidentities, xypolys, gradientpolys]] /. trigsubs
```

```
Out[51]= {-1 + ca^2 + sa^2, -1 + cb^2 + sb^2, x - ca - cb - ca cb - sa sb,
y - ca^5 - cb^5 - ca^3 cb^2 + 10 ca^3 sa^2 + 10 ca^3 cb^5 sa^2 - 5 ca sa^4 - 5 ca cb^5 sa^4 - 25 ca^4 cb^4 sa sb + 50 ca^2 cb^4 sa^3 sb -
5 cb^4 sa^5 sb + 10 cb^3 sb^2 + 10 ca^5 cb^3 sb^2 - 100 ca^3 cb^3 sa^2 sb^2 + 50 ca cb^3 sa^4 sb^2 + 50 ca^4 cb^2 sa sb^3 - 100 ca^2 cb^2 sa^3 sb^3 +
10 cb^2 sa^5 sb^3 - 5 cb sb^4 - 5 ca^5 cb sb^4 + 50 ca^3 cb sa^2 sb^4 - 25 ca cb sa^4 sb^4 - 5 ca^4 sa sb^5 + 10 ca^2 sa^3 sb^5 - sa^5 sb^5,
sa - 25 λ ca^4 sa + cb sa - 25 λ ca^4 cb^5 sa + 50 λ ca^2 sa^3 + 50 λ ca^2 cb^5 sa^3 - 5 λ sa^5 - 5 λ cb^5 sa^5 -
ca sb + 25 λ ca^5 cb^4 sb - 250 λ ca^3 cb^4 sa^2 sb + 125 λ ca cb^4 sa^4 sb + 250 λ ca^4 cb^3 sa sb^2 -
500 λ ca^2 cb^3 sa^3 sb^2 + 50 λ cb^3 sa^5 sb^2 - 50 λ ca^5 cb^2 sb^3 + 500 λ ca^3 cb^2 sa^2 sb^3 - 250 λ ca cb^2 sa^4 sb^3 -
125 λ ca^4 cb sa sb^4 + 250 λ ca^2 cb sa^3 sb^4 - 25 λ cb sa^5 sb^4 + 5 λ ca^5 sb^5 - 50 λ ca^3 sa^2 sb^5 + 25 λ ca sa^4 sb^5,
-cb sa + 25 λ ca^4 cb^5 sa - 50 λ ca^2 cb^5 sa^3 + 5 λ cb^5 sa^5 + sa + ca sb - 25 λ cb^4 sb - 25 λ ca^5 cb^4 sb +
250 λ ca^3 cb^4 sa^2 sb - 125 λ ca cb^4 sa^4 sb - 250 λ ca^4 cb^3 sa sb^2 + 500 λ ca^2 cb^3 sa^3 sb^2 - 50 λ cb^3 sa^5 sb^2 +
50 λ cb^2 sb^3 + 50 λ ca^5 cb^2 sb^3 - 500 λ ca^3 cb^2 sa^2 sb^3 + 250 λ ca cb^2 sa^4 sb^3 + 125 λ ca^4 cb sa sb^4 -
250 λ ca^2 cb sa^3 sb^4 + 25 λ cb sa^5 sb^4 - 5 λ sb^5 - 5 λ ca^5 sb^5 + 50 λ ca^3 sa^2 sb^5 - 25 λ ca sa^4 sb^5}
```

We have our system of polynomials. We now compute an elimination ideal, getting rid of the trigonometric variables and the Lagrange multiplier.

Originally I could not use purely symbolic GroebnerBasis to find the envelope curves (it hung). The computation involved the Mathematica functions NMinimize, FindRoot, NDSolve, NullSpace, RowReduce, and Rationalize.

With a dash of `LatticeReduce` for extra measure. Altogether a fascinating foray into symbolic-numeric computation.

But it turns out *Mathematica's* `GroebnerBasis` can (now) handle this, if asked just right. This involves using the Gröbner walk method. It also involved repairing some trouble in that code, which is why it was not handling the problem three years ago when I first tried it. Remark: This next computation cannot be done in version 5.2 of *Mathematica*. It requires the kernel under development at the time this talk was given.

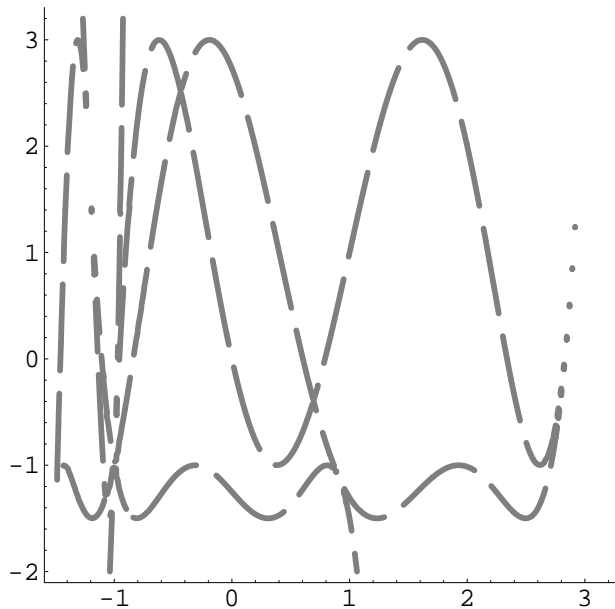
```
In[59]:= Timing[
  implicitpoly = Factor[First[GroebnerBasis[polys, mainvars, elimvars, Sort -> True,
    MonomialOrder -> EliminationOrder, Method -> "GroebnerWalk"]]]]
```

```
Out[59]= {135.836, (-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)}
```

Let's have a look at it.

```
In[53]:= Needs["Graphics`"]
```

```
In[56]:= ImplicitPlot[Evaluate[implicitpoly == 0],
  {x, -1.5, 3.2}, {y, -2, 3.2}, PlotPoints -> 200, AspectRatio -> 1,
  PlotStyle -> {{Thickness[.01], GrayLevel[.5], Dashing[ {.15, .04}]}, {Automatic}}];
```



(8) Independent vertices in a graph

This was discussed not long ago on the Usenet news group `sci.math.symbolic` [[independent vertices](#)]. We wish to find a maximally independent set of vertices in a graph. This is an NP-complete problem from combinatorial optimization. Undaunted, we proceed as below. First we need code to create a graph with a set of independent vertices of some given size (with low probability, there could be a larger set). The code is adapted from some that Maxim Rytin showed in the news group thread.

```
In[57]:= shuffle[n_Integer, m_Integer] :=
Module[{res = Range[n]}, Do[rand = Random[Integer, {j, n}];
res[{{j, rand}}] = res[{{rand, j}}];, {j, m}];
Take[res, m]]
```

Now we create a graph with 500 vertices and an independent set of 100 vertices. We show the set as it is what we hope to reproduce.

```
In[132]:=
{n, m} = {500, 100};
density = 1 / 4;
SeedRandom[1111];
indepset = Sort[shuffle[n, m]]
mat = Array[
Boole[#1 < #2 && Random[] < density && Complement[{{##}, indepset] != {}] &, {n, n}];
```

```
Out[135]=
{2, 4, 17, 18, 21, 26, 28, 34, 37, 50, 51, 53, 59, 61, 62, 68, 79, 85, 87, 93,
95, 97, 105, 113, 134, 137, 143, 144, 148, 152, 154, 155, 158, 161, 162, 168,
176, 178, 179, 189, 191, 193, 194, 202, 205, 208, 212, 215, 219, 224, 228, 229,
231, 232, 234, 235, 237, 245, 249, 260, 261, 264, 273, 275, 285, 293, 296, 298,
299, 300, 306, 308, 309, 312, 315, 321, 327, 329, 357, 360, 377, 382, 390, 394,
400, 408, 416, 420, 422, 433, 436, 438, 442, 447, 463, 465, 466, 477, 491, 492}
```

The variable *mat* is the familiar matrix of edges, where each row and column represents a vertex.

We will recover this independent set using quadratic programming and an appropriate objective function. I show how to do this with `FindMinimum`. One uses the (new) `QuadraticProgramming` method as it seems to do well with a nonconvex problem having linear constraints. Some of the ideas behind this approach, using quadratic programming, are discussed in a technical report [[Karmarker Resende Ramakrishnan report](#)].

The key is to come up with an appropriate objective function. Well, before that, we need an appropriate formulation of the problem. We will use a variable for each vertex and constrain it to take on values between zero and one. A value of zero means the variable is not in the independent set, while one means it is, so we will further encourage them to take on one or the other of the two boundary values by adept use of the objective function.

This objective will have three components. The first is the product of matrix values above the diagonal times the corresponding vertex variables. A typical monomial looks like `mat[[i, j]]*x[i]*x[j]` and a value of one means we regard the two vertices as being in the independent set, but really they cannot be because there is a vertex between them. We could, I suppose, have quadratic constraints that force all these to be zero or at least "small", but I think it is better to keep the constraints simple and put our effort into the objective function.

The second component to this function will be the negated total of the variables. Without a component such as this, they'd all be zero, which would give us no vertices in our independent set (bad). So we want the objective function to try to push some variables to be large rather than small. We will weight this term by the reciprocal of the expected size of the set (this is cheating, because we know the size, but in practice one could take reasonable guesses as to what might be good weights). We use this smallish weighting because otherwise the objective function might be encouraged to give a result with too many elements in the "independent" set, in particular containing some that gave contributions near one from the monomials in the first component discussed above.

The third component to this function is similar to the second one. It is simply the sum of squares of all variables, similarly weighted.

If we have values of one for all vertex variables that are in the independent set, and values of zero for the rest, then we get values of 0, -1, and -1 respectively for the three objective function components, for a total objective function value of -2 (this, as noted above, assumes we got the weight "correct"). If other vertex variables take on positive values then the first component increases by more than the others decrease. Offhand I do not know if it is possible to decrease some of the one values, and increase some of the zero values, in such a way that the total objective function decreases. Maybe. But this is only a heuristic method, albeit one that seems to work well.

```
In[59]:= independentVerticesQP[mat_, n_, m_] := Module[
  {x, vars, poly1, poly2, poly3, poly, sol},
  vars = Array[x, n];
  poly1 = Sum[mat[[i, j]] * x[i] * x[j], {i, n - 1}, {j, i + 1, n}];
  poly2 = Total[vars];
  poly3 = vars.vars;
  poly = poly1 - poly2 / m - poly3 / m;
  sol = FindMinimum[
    {poly, Flatten[{(0 ≤ # ≤ 1 &) /@ vars}]}, vars, Method → QuadraticProgramming];
  {sol[[1]], Flatten[Position[Round[Chop[vars /. sol[[2]]]],
    _? (# != 0 &), Heads → False]]}]
```

```
In[85]:= Timing[iset = independentVerticesQP[mat, n, m]]
```

```
Out[85]= {2.856179,
  {-2., {2, 4, 17, 18, 21, 26, 28, 34, 37, 50, 51, 53, 59, 61, 62, 68, 79, 85, 87, 93, 95,
  97, 105, 113, 134, 137, 143, 144, 148, 152, 154, 155, 158, 161, 162, 168, 176, 178,
  179, 189, 191, 193, 194, 202, 205, 208, 212, 215, 219, 224, 228, 229, 231, 232,
  234, 235, 237, 245, 249, 260, 261, 264, 273, 275, 285, 293, 296, 298, 299, 300,
  306, 308, 309, 312, 315, 321, 327, 329, 357, 360, 377, 382, 390, 394, 400,
  408, 416, 420, 422, 433, 436, 438, 442, 447, 463, 465, 466, 477, 491, 492}}}
```

Lo and behold, we have recovered our independent set.

We will illustrate the independent set on a smaller example (so the picture is not overcrowded with vertices and edges). Again this will require the next version of *Mathematica* after 5.2, both to do the minimization and the graph plot.

```
In[60]:= {n, m} = {100, 20};
density = 1 / 8;
SeedRandom[1111];
indepset = Sort[shuffle[n, m]]
mat = Array[
  Boole[#1 < #2 && Random[] < density && Complement[{###}, indepset] != {}] &, {n, n}];
```

```
Out[63]= {2, 3, 6, 9, 12, 16, 17, 18, 25, 27, 32, 46, 52, 58, 71, 72, 79, 81, 83, 97}
```

```
In[65]:= Timing[indies = independentVerticesQP[mat, n, m]]
```

Well, we found a set of independent vertices a bit larger than the one we started with. This is no doubt due to the low density of the graph. We will show this with blue vertices in the set and red ones outside. We'll have light green edges connecting vertices not in the set, red edges connecting vertices in the set to vertices outside, and blue edges connecting pairs in the set (the salient feature being we do not want to see any blue edges).

```
In[66]:= iset = Indies[[2]];
iset = indepset;
GraphPlot[mat, Method → "CircularEmbedding", VertexRenderingFunction →
  Function[{c, v}, {If[MemberQ[iset, v], Blue, Red], PointSize[Medium], Point[c]}],
  VertexLabeling → Automatic, EdgeRenderingFunction →
  Function[{r, v, l}, ii = Intersection[iset, v];
    {If[ii == {}, Green, If[Length[ii] == 2, Blue, Red]], Line[r]}]]
```

(9) Depicting cylinders containing five given points (CENSORED)

Over the past several years I have worked on various aspects of the following problem: given five points in space, find all right circular cylinders containing them. A bit of thought indicates that, generically, one needs four values to define the axial line, and a fifth necessary value is the radius, so having an equation to be satisfied by each point means five is the number we expect will give us finitely (but not zero) actual cylinders. Well, we might need to go to complex space, but... Anyway, I do not propose to go into a lengthy discussion on this topic (I have done so in past, e.g. [cylinders through five points](#)).

One configuration that is of interest is that where the five points are the vertices of two regular tetrahedra glued together along a common face. We orient so that this face lies in the horizontal plane, and thus the last two points are aligned vertically. In this case there happen to be six real cylinders containing all five points, and it can be shown that this is the largest (other than the nongeneric cases where there are infinitely many, e.g. the points are all collinear).

Some general theory indicates that if we raise the top point vertically, we continue to have six cylinders until we hit a particular value at which...poof...they all disappear (which is to say, solutions to the radial and axial parameters become complex valued). I always found this interesting and wanted to visualize it. I think we can do a nice job of this but, alas, this uses technology I cannot yet show outside the company (sorry).

(10) A logic puzzle with 0–1 inequality constraints

This next example illustrates a type of popular logic puzzle often found in supermarket and airport publications. This one is called "Dinner on the run" [[Jean Hannagan puzzle](#)]; I cribbed it from my son's puzzle magazine. A description of the problem is as follows. Five men order five sandwiches from Max's Deli, such that each has a distinct fillings, toppings, spreads, and breads. Further clues are as below, and we are to deduce the components of each person's sandwich. We follow customary rules of interpretation of such puzzles, e.g. "Neither the sandwich with cheese nor the one with mayo was on white bread" means, among other things, that the cheese and mayo sandwiches are themselves distinct.

- (1) One filling is salami. One topping is tomatoes. One spread is butter. One bread is pumpernickel. Frank placed the order.
- (2) Regardless of order, Max refuses to put mustard or ketchup on tuna salad, or onions on turkey.
- (3) Tim does not like onions.
- (4) The roast beef was not on rye, and did not contain pickles.
- (5) The tuna salad was not on seven-grain bread.
- (6) Neither tuna salad nor sandwich with lettuce was spread with mayo.
- (7) The turkey, which was on whole wheat, did not have mustard.
- (8) The sandwich with lettuce did not have mustard.
- (9) Neither Nick nor the person requesting ketchup and onions used white bread.
- (10) The ham sandwich had neither pickles nor onions.
- (11) The sandwich with pickles did not have mayo.
- (12) Jim did not order relish.
- (13) The sandwich with lettuce did not have relish.
- (14) The sandwich with cheese did not have mustard. Moreover it did not go to Tim.
- (15) Neither the sandwich with cheese nor the one with mayo was on white bread.
- (16) Elmer does not eat rye bread.
- (17) Tim did not order his with whole wheat bread.
- (18) The sandwich with whole wheat did not have mayo.
- (19) Nick did not order tuna salad.

- (20) Jim did not order seven-grain bread.
- (21) Neither Jim nor Nick used lettuce.

We first use the information just to figure out the five items in each of five categories. The men are Frank, Jim, Nick, Tim, and Elmer. The fillings are salami, tuna salad, turkey, ham, and roast beef. The toppings are tomatoes, lettuce, onions, cheese, and pickles. The spreads are butter, ketchup, mustard, mayo, and relish. The breads are pumpernickel, seven-grain, white, rye, and wheat. Note that this preprocessing step is in itself a bit of a challenge. (Is relish a topping or spread? Is cheese a filling or topping?)

As there are a scant $(5!)^4$ possibilities (around 200 million, that is) we could, with some work, do a brute force search. We will use a constraint satisfaction approach via linear programming as we believe it is instructive to demonstrate handling of inequations in this setting.

Conceptually, we arbitrarily number the men Frank=1,...,Elmer=5. We create variables for each of the categories: filling, top, spread, and bread. For each such variable we create subvariables, one per sandwich number. These must be zero or one, and they sum to one (that is, exactly one has value one). For example, we will have the equation

$$\sum_{j=1}^5 \text{mayo}[j] = 1$$

Moreover we have transposes of the above type of equation. That is, we know the sum of all subvariables for a particular sandwich and category must be also be one. For example, we have

$$\text{mayo}[1] + \text{ketchup}[1] + \text{mustard}[1] + \text{relish}[1] + \text{butter}[1] = 1$$

We furthermore have equations relating the basic variables to their corresponding subvariables, such as

$$\text{mayo} = \text{mayo}[1] + 2 \text{mayo}[2] + 3 \text{mayo}[3] + 4 \text{mayo}[4] + 5 \text{mayo}[5]$$

For efficiency we only work with the knapsack variables, using tactics equivalent to these above equations at the end to put the solution into a reasonable form.

Finally we have all the inequations and the handful of equations implied by the 21 rules above. These we handle as follows. From rule 7, say, we know that turkey = wheat. This means we equate all the corresponding subvariables. From rule 11 we know that pickles \neq mayo. Some reflection shows that this equivalent to the set of subvariable inequalities $\text{mayo}[j] + \text{pickles}[j] \leq 1$ for $1 \leq j \leq 5$. When we equate a pair of items, e.g. turkey and wheat, we realize that by equating each pair of their five subvariables we can remove one. Similarly observe that, for example, Jim did not take lettuce, so we have an equation for the variable representing lettuce used by Jim (it is zero). Again we can remove it from further consideration after we use it to simplify whatever constraints contained that variable. Use of such equations as means to remove variables and constraints tends to make the computation much faster. In this case it gave a speed improvement by a factor of around 10.

As all constraints are linear, we will solve this by treating it as a linear programming problem. Curiously we are not actually optimizing anything; it is an example of a "constraint satisfaction" problem. This gives us some freedom to make up objective functions, and in general that can be used to improve efficiency, but we will forego that in the interest of simplicity.

The idea of the branching method is to solve what are termed relaxations of the problem wherein integrality is not enforced but inequality constraints are in use. For any solution one looks for noninteger parts. If all are integer the solution is fine. Otherwise one takes, say, the first coordinate that is not an integer and spawns a pair of new problems, one constraining the corresponding variable to be less or equal to the floor of the value in the solution, and the other constraining it to be greater or equal to the ceiling of that value. This is the "branching". We do not utilize the "bounding" part in this example, but in cases where there is an actual objective function of interest, one

uses it to remove subproblems for which we know there is no hope of improving on the current "best" solution (once a viable solution has been found). Note that this approach is also useful for what are referred to as mixed problems, wherein some but not all variables may be constrained to take on integer values. One simply branches only on those so constrained.

The following code will quit after finding one valid solution. We are trusting that the author used adequate conditions to ensure that a solution exists and is unique (which is in fact the case). It is not hard to modify the code to allow for other possibilities. Note that in contrast to the method I presented yesterday [ILP talk notebook] for handling Frobenius instance problems, we now have many integer variables but constrained to take on values in a small set. In this case it is neither necessary nor even desirable to preprocess with lattice reduction, or to base branching decisions thereon.

```
In[141]:=
findSandwiches[] :=
Module[{consumers, fills, tops, spreads, breads, frank, jim, nick, tim, elmer, salami,
  tuna, turkey, ham, roastbeef, tomatoes, lettuce, onions, cheese, pickles, butter,
  ketchup, mustard, mayo, relish, pumpernickel, sevengrain, white, rye, wheat, fillvars,
  topvars, spreadvars, breadvars, varlists, all01vars, pv, prodvars, constraints1,
  constraints2, constraints3, constraints4, constraints, vars, program, stack,
  soln, soln01, badpos, counter = 0, eps = 10-6, names, suffix, partsoln, psvars},
consumers = {frank, jim, nick, tim, elmer};
fills = {salami, tuna, turkey, ham, roastbeef};
tops = {tomatoes, lettuce, onions, cheese, pickles};
spreads = {butter, ketchup, mustard, mayo, relish};
breads = {pumpernickel, sevengrain, white, rye, wheat};
fillvars = Outer[#1[#2] &, fills, consumers];
topvars = Outer[#1[#2] &, tops, consumers];
spreadvars = Outer[#1[#2] &, spreads, consumers];
breadvars = Outer[#1[#2] &, breads, consumers];
varlists = {fillvars, topvars, spreadvars, breadvars};
all01vars = Flatten[varlists];
constraints1 = Map[0 ≤ # ≤ 1 &, all01vars];
constraints2 =
  Map[Apply[Plus, #] == 1 &, {fillvars, topvars, spreadvars, breadvars}, {2}];
constraints3 = Map[Apply[Plus, #] == 1 &,
  Map[Transpose, {fillvars, topvars, spreadvars, breadvars}], {2}];
ineq[v1_, v2_, v3_] := {ineq[v1, v2], ineq[v1, v3], ineq[v2, v3]};
ineq[v1_, v2_] := Map[v1[#] + v2[#] ≤ 1 &, consumers];
eq[v1_, v2_] := Map[v1[#] == v2[#] &, consumers];
constraints4 = {ineq[mustard, tuna], ineq[ketchup, tuna],
  ineq[onions, turkey], ineq[roastbeef, rye], ineq[roastbeef, pickles],
  ineq[tuna, sevengrain], ineq[tuna, lettuce, mayo], ineq[turkey, mustard],
  ineq[mustard, lettuce], ineq[ketchup, white], ineq[ham, pickles],
  ineq[ham, onions], ineq[pickles, mayo], ineq[lettuce, relish],
  ineq[cheese, mustard], ineq[cheese, mayo, white], ineq[mayo, wheat]};
constraints = Flatten[Join[constraints1, constraints2, constraints3, constraints4]];
partsoln =
  Flatten[{eq[turkey, wheat], eq[ketchup, onions], onions[tim] == 0, rye[elmer] == 0,
    wheat[tim] == 0, white[nick] == 0, onions[nick] == 0, relish[jim] == 0, cheese[tim] == 0,
    tuna[nick] == 0, sevengrain[jim] == 0, lettuce[jim] == 0, lettuce[nick] == 0}];
psvars = Map[First, partsoln];
partsoln = partsoln /. Equal → Rule;
constraints = constraints //. partsoln /.
  {True → Sequence[], ((aa_ /; Head[aa] != Plus) ≤ 1) → Sequence[]};
all01vars = Complement[all01vars, psvars];
program = constraints;
stack = {program, {}};
While[stack != {}, counter++; program = stack[[1]]; stack = stack[[2]];
  Internal`DeactivateMessages[soln = NMinimize[{1, program}, all01vars]];
  If[! FreeQ[soln, Indeterminate], Continue[]];
  soln = Chop[soln[[2]], eps];
  soln01 = all01vars /. soln;
  badpos =
```

```

Position[soln01, (aa_ /; Chop[aa - Round[aa], eps] != 0), {1}, 1, Heads → False];
If[badpos == {}, partsoln = partsoln /. soln;
soln = Join[soln, partsoln];
soln =
  Split[Sort[(soln /. aa_?NumberQ => Round[aa]) /. {HoldPattern[_ → 0] => Sequence[],
    HoldPattern[hh_ [bb_] → 1] => bb == hh, HoldPattern[
      Rule[_ , bb_ /; Not[NumberQ[bb]]] => Sequence[]}], #1[[1]] === #2[[1]] &];
names = Map[#[[1, 1]] &, soln];
soln = MapThread[Prepend, {soln /. aa_ == bb_ → bb, names}];
soln = Map[ToString, soln, {2}];
suffix = soln[[1, 1]];
suffix = StringDrop[suffix, StringPosition[suffix, "$"][[1, 1]] - 1];
soln = Map[StringReplace[#, suffix → "" ] &, soln, {2}];
Break[], badpos = badpos[[1, 1]];
stack = {Append[program, all01vars[[badpos]] == 0], stack};
stack = {Append[program, all01vars[[badpos]] == 1], stack};];];];
{counter, soln}

```

After all that work, solving is a breeze. Who'da thunk it?

```

In[142]:=
SetOptions[LinearProgramming, Method → InteriorPoint];

```

```

In[143]:=
Timing[orders = findSandwiches[]]

```

```

Out[143]=
{1.13207, {33, {{elmer, butter, lettuce, turkey, wheat},
  {frank, cheese, relish, rye, tuna}, {jim, ketchup, onions, pumpernickel, roastbeef},
  {nick, ham, mayo, sevengrain, tomatoes}, {tim, mustard, pickles, salami, white}}}}

```

Similar methods can be used for solving, say, sudoku puzzles.

(11) Units conversion (a final bit of integer linear programming)

I was recently asked in-house about the possibility of finding, given some dimensional monomial, equivalent minimal units in terms of sums of absolute values of exponents. We start with some set of units equivalents, e.g. $\text{mass} = \text{length}^2 / \text{time}^2$.

First clear denominators and then convert to linear polynomials in "exponent vectors". That is, something like $f^2 m^2$ will become $2 f + 3 m$. Except I rename variables to $x[f]$ etc. This is because I also want to handle negative exponents using "reciprocal" variables e.g. $xr[f]$ and corresponding relations such as $x[f] + xr[f] == 0$.

I use `Minimize` to find a smallest equivalent, the use `Reduce` to find all of them.

```

In[1]:= polys = {-(l*m) + f*t^2, -(l^2*m) + e*t^2, f*k*m^2 - c^2*s^2, -(k*m^2) + c^2*o*s,
  -k+m*p*s^2, -(k*m^2) + s^3*wa, -(k*m^2) + c*s*we, -k+c*s*t, c^2*h-k*m^2,
  -(k*m) + n*s^2, -(k*m^2) + j*s^2, -(k*m^2) + c*s^2*v, -c+a*s};

```

```

In[12]:= vars = Variables[polys];
tmp1 = polys /. {Plus -> pplus, Times -> llist};
tmp2 = tmp1 /. a_Symbol^e1_ /. MemberQ[vars, a] => e1*x[a];
tmp3 = tmp2 /. llist[-1, a_] => -Map[-1*#&, llist[a]];
lpoll = tmp3 /. {llist -> Plus, pplus -> Subtract};
newvars = Join[Map[x[#] &, vars], Map[xr[#] &, vars]];
lpol2 = Map[x[#] + xr[#] &, vars];
lpolys = Join[lpoll, lpol2];
nulls = LatticeReduce[Normal[CoefficientArrays[lpolys, newvars][[2]]]];
myvars = Array[mm, Length[nulls]];

In[22]:= minimalUnits[expr_] :=
Module[{input, llist, soll, newpolys, constrnt, obj, min, vals, allsols},
input = ((expr /. Times -> llist) /. a_Symbol^e1_ /. MemberQ[vars, a] => e1*x[a]) /.
llist -> Plus;
soll = Normal[CoefficientArrays[input, newvars][[2]]];
newpolys = soll + Apply[Plus, myvars * nulls];
constrnt = Map[# >= 0 &, newpolys];
obj = Apply[Plus, newpolys];
{min, vals} =
Minimize[{obj, Append[constrnt, Element[myvars, Integers]]}, myvars];
allsols = Reduce[Flatten[{obj == min, constrnt, Element[myvars, Integers]}], myvars];
allsols = {ToRules[allsols]};
Map[Apply[Times, newvars^(newpolys /. #) /. {x[a_] => a, xr[a_] => 1/a}] &, allsols]]

```

Here is a simple example.

```

In[23]:= Timing[minimalUnits[c^2*m^4*p*k^3*v^2/(h^3*t^7*wa^2)]]

```

```

Out[23]= {2.76017, {
  {
     $\frac{c^6 f^6 n w a}{l^2}$ ,  $\frac{a f^6 j n}{l^2}$ ,  $\frac{c f^5 j w a}{l t^2}$ ,  $\frac{a f^5 j^2}{l t^2}$ ,  $\frac{c f^5 j n}{l^2 o}$ ,
     $\frac{c f^4 j^2}{l o t^2}$ ,  $\frac{a c^2 f^5 n}{l^2}$ ,  $\frac{c^3 f^4 w a}{l t^2}$ ,  $\frac{a c^2 f^4 j}{l t^2}$ ,  $\frac{c^3 f^4 n}{l^2 o}$ ,  $\frac{c^3 f^3 j}{l o t^2}$ ,  $\frac{a c^4 f^3}{l t^2}$ ,  $\frac{c^5 f^2}{l o t^2}$ 
  }
}}

```

We see that our results have total degree of 11, whereas the input total was 24.

(12) Summary

Well, here we are. As promised, we touched upon Gröbner bases, logic puzzles, branch-and-bound, and more. The methods presented apply to a vast array of problems in mathematical optimization and elsewhere.

Should I survive the expected tomato attack, I hope to return to problems from MathGroup and elsewhere in a future talk, but perhaps with more of a number theory and calculus focus.

(13) Acknowledgements

I thank Jeff Bryant, Brett Champion, Yifan Hu, Michael Trott, and Robby Villegas for assisting in various parts of the preparation of these slides. Mostly small matters, such as gathering the shards of my fractured sanity.

(14) Selected references

[Algebraic replacement]

<http://forums.wolfram.com/mathgroup/archive/2006/Aug/msg00283.html>

[Parametrized nested radical]

<http://forums.wolfram.com/mathgroup/archive/2005/May/msg00502.html>

[Parametrized rational surface]

A. Chtcherba and D. Kapur. Conditions for determinantal formula for resultant of a polynomial system. Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation (ISSAC 2006), 55–62. ACM Press 2006.

See also: J. Zheng, T. Sederberg, E–W Chionh, and D Cox. Implicitizing rational surfaces with base points using the method of moving surfaces. In: Topics in Algebraic Geometry and Geometric Modeling. Contemporary Mathematics volume 334, 151–168. AMS 2003.

[Johnny's ideal woman]

<http://forums.wolfram.com/mathgroup/archive/2003/Sep/msg00219.html>

<http://forums.wolfram.com/mathgroup/archive/2003/Sep/msg00258.html>

The original statement of the problem is from Georgia College & State University BITS & BYTES 3:3, February 1998.

http://www.gcsu.edu/acad_affairs/coll_artsci/mathcomp_sci/bits/V3N3Feb98.html

This work has also appeared in The Mathematica Journal 9(2):289–291, "Tricks of the Trade" column, edited by P. Abbott.

[Who own's the zebra?]

M. Trott. Solving puzzles with *Mathematica*. In "Trott's Corner" column, The *Mathematica* Journal 7(3):291–307 (puzzle 3). 1999.

[Trigonometric boundary curve]

D. Lichtblau. Computing curves bounding trigonometric planar maps: symbolic and hybrid methods. Proceedings of the 2004 Workshop on Automated Deduction in Geometry (ADG 2004). Lecture Notes in Computer Science 3763, 70–91. 2006.

<http://www.springerlink.com/content/2267x4v48wpg1680/?p=0e3d41cd8764421c9518b74e42de8892&pi=12>

[Independent vertices]

<http://groups.google.com/group/sci.math.symbolic/msg/c06cbe2ba0a3ace7?dmode=source>

[Karmarker, Resende, Ramakrishnan report]

An interior point approach to the maximum independent set problem in dense random graphs. Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ. 1989.

[Cylinders through five points]

<http://webs.uvigo.es/adg2006/talks.html>

Daniel Lichtblau. Cylinders Through Five Points: Complex and Real Enumerative Geometry.

[Hannagan's "Dinner on the Run"]

J. Hannagan. Dinner on the run. Dell Variety Puzzles and Word Games 135:62. May 2004.

[Integer linear programming intro]

D. Lichtblau. An Introduction to Integer Linear Programming in *Mathematica*. Notebook for presentation at Wolfram Technology Conference 2006.

<http://library.wolfram.com/infocenter/Conferences/6449>