Making Change and Finding Repfigits: Balancing a Knapsack

Daniel Lichtblau

100 Trade Center Dr
Champaign IL 61820
USA
danl@wolfram.com

Abstract. We will discuss knapsack problems that arise in certain computational number theory settings. A common theme is that the search space for the standard real relaxation is large; in a sense this translates to a poor choice of variables. Lattice reduction methods have been developed in the past few years to improve handling of such problems. We show explicitly how they may be applied to computation of Frobenius instances, Keith numbers (also called "repfigits"), and as a first step in computation of Frobenius numbers.

Key words. Frobenius instance solving, lattice reduction, integer linear programming, change−making problem, Frobenius numbers, Keith numbers, repfigits.

A shorter, differently formatted version of this article appeared in *Proceedings of the Second International Congress on Mathematical Software (ICMS 2006)*, A. Iglesias and N. Takayama, eds. Lecture Notes in Computer Science **4151**:182−193. Springer−Verlag.2006.

1. Introduction

Various problems in the realm of computational number theory have as key steps the solving of a linear equation or system over the integers, subject to some linear inequality constraints. The Frobenius instance problem (also known as the change−making problem) is a well−known example. The problem of finding what are called repfigits, to be described below, is another. These may be regarded as a class of knapsack problems wherein one is allowed to take only certain integer multiples of various items in forming a "valid" combination. As such these fall into the category of integer linear programming (ILP).

Classical methods for solving such problems include branch−and−bound and cutting plane methods [8] [16]. These approaches alone are often inadequate for certain classes of problems due to a phenomenon roughly described as "unbalanced bases". A method for dealing with this deficiency was developed in [1] [2]. In essence it involves working with a basis that is reduced (in the sense of [11]) and ordered by size, in conjunction with standard branch−and−bound. We will describe and illustrate this for the types of problem mentioned above . We remark that the handling of Frobenius instances is by no means new, having been discussed in the aforementioned references. We also show how the method is applied in a new algorithm for finding Frobenius numbers, a task that is substantially harder than solving Frobenius instances.

The algorithms described in this paper have been implemented in *Mathematica* [18]. Selected code is provided in the appendix.

2. Frobenius Instances

Suppose we have a set of positive integers $A = (a_1, \ ..., a_n)$ and a target M, a positive integer. We seek nonnegative integer multipliers $X = (x_1, \ ..., x_n)$ such that $X \cdot A = M$. This is a standard problem in integer linear programming. A classical method [16] for solving this would be to solve the relaxed problem wherein we enforce all inequality constraints but all variables to be real rather than integer valued. If in the solution we encounter a variable a with value s that is not an integer then we spawn two subproblems where we enforce respectively that $a \leq \lfloor s \rfloor$ and $a \geq \lceil s \rceil$. We continue this process of solving relaxed subproblems, splitting when a variable has a noninteger value. It can be shown that eventually either we exhaust all possibilities or we obtain an integer valued solution [16]; in either case clearly the algorithm terminates.

A drawback to this approach is that the search space for the relaxed subproblems might appear to be "large", in the sense of having many points with not all coordinates integer valued. In particular it may be the case that a standard LP solver will find real valued solutions to the restricted subproblems without making rapid progress to an (entirely) integer valued solution, because it might be possible to subdivide the (real valued) solution polytope in such a way that integer points do not readily appear at corners.

The method of [1], [2] was developed as a way to improve on this situation. Roughly it proceeds as follows. First we find a description of a solution set to our equations that is a priori integer valued but possibly does not satisfy the required inequalities. We arrange that this solution set has "good" basis vectors, such that when we solve relaxed problems with these we more rapidly walk through our (real valued) solution polytope. More correctly, the polytope is likely to intersect fewer hyperplanes orthogonal to larger direction vectors and spaced by integral multiples of those vectors.

With respect to the Frobenius instance problem it goes as follows. First we find a solution vector X over $\mathbb{Z}^n$ and a basis V for the integer null space (that is, $n-1$ independent vectors $V_j \in \mathbb{Z}^n$ with $V_j \cdot A = 0$); for this we use a method based on the Hermite normal form [5]. We use multiples of the basis vectors to find a "small" solution which we still call X. The tactic utilized to do this is sometimes called the embedding method. It apparently has been independently discovered several times; variants appear in [1], [12], [13], and [14], with short code provided in the appendix of this paper. In starting with a solution over $\mathbb{Z}^n$ rather than the nonnegatives $\mathbb{N}^n$ we are working with what is called an integer relaxation of the nonnegativity constraint. We will later enforce nonnegativity via the more common LP relaxations wherein integrality is not enforced.

We define variables $t = (t_1, \ldots, t_{n-1})$ so that solution vectors are given by $X + t\,V$. For purposes of finding a valid solution to the problem at hand we need to impose two requirements. The first is that all components are nonnegative and the second is that all are integers. The first can be met by standard linear programming. For the second, as in the classical approach, we will use branching on subproblems. Specifically we find solutions to relaxed LP problems wherein we now work over nonnegative reals. We then branch on noninteger values in those solutions. For example, if a solution has, say, $t_j = 5/4$, we create two subproblems identical to the one we just solved, but with the new constraints $t_j \leq \lfloor 5/4 \rfloor = 1$ and $t_j \geq \lceil 5/4 \rceil = 2$, respectively (though note that if other variables also had noninteger solutions then we need not have chosen this particular one). As observed above, this branching process will terminate eventually, with either a valid solution or the information that no such solution exists.

We explain again, in slightly different terms, why it is important to work with a small integer solution to the integer relaxation (that is, allowing negative values) and a lattice reduced basis for the null vectors. As our methodology is to take combinations of these null vectors, effectively they define directions in a solution polytope for the transformed problem. By working with a reduced basis we in effect change our coordinate system to one where the various search directions are roughly orthogonal. This helps us to avoid the possibility of taking many steps in similar directions in searching the polytope of nonnegative solutions for one that is integer valued. Thus we explore it far more efficiently. Moreover, in starting with a small solution we begin closer to the nonnegative orthant. Heuristically this seems to make the sought−for multipliers of the null vectors relatively small, and this is good for computational speed. This is discussed in section 2 of [1], with further explanation and illustrations found in [3].

A further efficiency, from [2], is to choose carefully the variable on which to branch. We order by increasing size the reduced lattice of null vectors. Branching will be done on the noninteger multiplier variable correspond− ing to the largest of these basis vectors. This has the effect of exploring the solution polytope in directions in which it is relatively thin, thus more quickly finding integer lattice points therein or exhausting the space. This refinement is important for handling pathological examples of the sort presented in [1] and [2].

As we have a constraint satisfaction problem we are also free to impose any linear integer objective function of our choosing. Thus an optimization is to obtain extremal values for linear forms with integer coefficients and use these results as simple cutting planes [16]. For example we can minimize or maximize the various coordi− nate values t_j, selecting one either in some specific order or at random for each subproblem. This process amounts to finding the width of the polytope along the directions of our lattice basis vectors, and enforcing integrality of the optimized variable helps to further restrict the search space.

As reported in [2] this method is very effective in solving Frobenius instances. We illustrate with an example from [17].

$A = (10\,000\,000\,000,\ 10\,451\,674\,296,\ 18\,543\,816\,066,\ 27\,129\,592\,681,\ 27\,275\,963\,647,$
$29\,754\,323\,979,\ 31\,437\,595\,145,\ 34\,219\,677\,075,\ 36\,727\,009\,883,\ 43\,226\,644\,830,$
$47\,122\,613\,303,\ 57\,481\,379\,652,\ 73\,514\,433\,751,\ 74\,355\,454\,078,\ 78\,522\,678\,316,$
$86\,905\,143\,028,\ 89\,114\,826\,334,\ 91\,314\,621\,669,\ 92\,498\,011\,383,\ 93\,095\,723\,941)$

We let the target be 862 323 776. In several seconds the instance solver returns the empty set. This has implica−tions for bounding the Frobenius number of A; we will discuss that in a later section.

3. Keith Numbers

Keith numbers, also known as repfigits, were introduced in 1987 by Michael Keith [10] as a sort of computa−tional novelty that relates a Fibonacci−like sequence to a linear equation involving its seed. They are defined as follows. Suppose we are given a number s of n digits (we work in base 10, but these can be defined with respect to arbitrary bases). We may form a sequence in Fibonacci style as follows. The first n elements are the digits themselves. The $(n+1)^{\text{th}}$ element is the sum of the first n digits. Subsequent elements are the sums of the preceding n elements. Then s is called a Keith number provided it appears in this sequence. As an example, the sequence for 197 is {1, 9, 7, 17, 33, 57, 107, 197, ...} and so 197 is a Keith number. Keith originally referred to these as repfigits, for "replicating Fibonacci digits".

Keith numbers tend to be quite rare (there are only 71 of them below 10^{19}). Prior methods for finding them involved clever segmentation of an enumeration. While flawless (in the sense that they find all of them), these are limited in range due to algorithmic complexity and memory requirements. At the time the present work was begun the state of the art, from [10], was that all such numbers up to 19 digits had been found but no larger ones were known. We will take this substantially further.

To begin we must find equations to describe these things. If the digits are $\{d_0, d_1, \ldots, d_{n-1}\}$ then the number in question is $\sum_{j=0}^{n-1} d_j\, 10^{n-1-j}$. We form the appropriate sequence using a Fibonacci matrix of dimension n. This is simply a matrix that, when operating on a vector, replaces each element up to the last by its successor, and replaces the last by the sum of the elements. For example, for $n = 3$ it is $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$.

If we multiply this matrix by itself $k-1$ times then the dot product of the bottom row with the digit sequence will give the $(n+k)^{\text{th}}$ term in the sequence. Some simple inequality considerations will give fairly tight bounds on how many such multiples can possibly work for a given number of digits n. We will use each possibility to form a homogeneous linear diophantine equation. In the actual code we take advantage of the structure of the matrix to avoid forming explicit matrix products.

We demonstrate with a short example. We start by obtaining the set of candidate equation vectors for 5 digit examples. One of them is (−8207, 1705, 3069, 3395, 3524). That is, we will seek a solution to the system

$$-8207\, x_1 + 1705\, x_2 + 3069\, x_3 + 3395\, x_4 + 3524\, x_5 = 0$$

with each $x_j \in \{0, 1, \ldots 9\}$ and $x_1 \geqslant 1$.

As in the last section, the first step in the process of [1] and [2] is to find a full set of integer solutions to such a system. Since these are homogeneous equations we require only the integer null space. This can be obtained readily from the Hermite normal form for the matrix comprised of the vector for the homogeneous equation, augmented by an identity matrix. Again we want to work with vectors that are small and close to orthogonal so we apply lattice reduction to get a "good" set of vectors spanning the same solution set. We obtain (−3, −1, −3, −3, −1), (−2, −4, −3, 3, −3), (1, 6, −5, 6, −2), and (7, −3, −15, 5, 26).

Notice that for any solution vector, its negative is also a solution vector. Looking at the first vector in our solution basis we thus see that 31 331 is a Keith number of five digits. That was too easy; there is no guarantee we will have a solution vector with all components in the desired range. We look at a slightly larger example to see this. For six digits one candidate equation−defining vector is $(-96\,160, -4224, 5752, 7144, 7482, 7616)$. This time we get the following small null vectors: $(0, 3, -3, 0, 4, 0)$, $(-1, 1, -3, -2, -4, -4)$, $(0, -6, -3, 1, 0, -2)$, $(0, -1, 1, 5, 0, -6)$, and $(0, 4, -12, 15, -12, 9)$. So we have a generating set of small null vectors none of which have entirely nonnegative or entirely nonpositive values (with the first being nonzero, in order that they give a legitimate six digit number). We now need a way to recombine these so that the first component is positive and the rest are nonnegative.

Again we have something we can tackle with standard branch−and−bound iterations [8] [16]. Let $\{v_1, \ldots, v_n\}$ be our null space basis (here n is one less than the number of digits). We seek an integer vector of the form $a_1 v_1 + \ldots + a_n v_n$ such that all components lie in the range $\{0, 1, \ldots, 9\}$ with the first component strictly posi− tive. For this we create variables $\{a_1, \ldots, a_n\}$ which will ultimately be required to take on integer values. In order to effect this we will solve relaxed linear programming problems with appropriate inequality constraints. As noted earlier, these are simply constraint satisfaction problems, so we can use arbitrary linear objective functions as a means of obtaining integer cuts cheaply. When some variables do not take on integer values in the solution we choose one such on which to branch and spawn a pair of subproblems. Our choice again is from [2]; we branch on that variable of noninteger solution value whose corresponding basis vector is largest.

To summarize, we first find the appropriate sets of integer equations. For each we find spanning sets of solu− tions that do not in general satisfy the digit inequality constraints. We lattice reduce these. We use ILP methods to find all possible solutions subject to the usual inequality constraints on digits.

Code that implements this is found in the appendix. We used this to find all Keith numbers up through 29 digits. We show all repfigits between 20 and 29 digits below.

```
{1, 2, 7, 6, 3, 3, 1, 4, 4, 7, 9, 4, 6, 1, 3, 8, 4, 2, 7, 9}
{2, 7, 8, 4, 7, 6, 5, 2, 5, 7, 7, 9, 0, 5, 7, 9, 3, 4, 1, 3}
{4, 5, 4, 1, 9, 2, 6, 6, 4, 1, 4, 4, 9, 5, 6, 0, 1, 9, 0, 3}
{8, 5, 5, 1, 9, 1, 3, 2, 4, 3, 3, 0, 8, 0, 2, 3, 9, 7, 9, 8, 9}
{7, 6, 5, 7, 2, 3, 0, 8, 8, 2, 2, 5, 9, 5, 4, 8, 7, 2, 3, 5, 9, 3}
{2, 6, 8, 4, 2, 9, 9, 4, 4, 2, 2, 6, 3, 7, 1, 1, 2, 5, 2, 3, 3, 3, 7}
{3, 6, 8, 9, 9, 2, 7, 7, 5, 9, 3, 8, 5, 2, 6, 0, 9, 9, 9, 7, 4, 0, 3}
{6, 1, 3, 3, 3, 8, 5, 3, 6, 0, 2, 1, 2, 9, 8, 1, 9, 1, 8, 9, 6, 6, 8}
{2, 2, 9, 1, 4, 6, 4, 1, 3, 1, 3, 6, 5, 8, 5, 5, 5, 8, 4, 6, 1, 2, 2, 7}
{9, 8, 3, 8, 6, 7, 8, 6, 8, 7, 9, 1, 5, 1, 9, 8, 5, 9, 9, 2, 0, 0, 6, 0, 4}
{1, 8, 3, 5, 4, 9, 7, 2, 5, 8, 5, 2, 2, 5, 3, 5, 8, 0, 6, 7, 7, 1, 8, 2, 6, 6}
{1, 9, 8, 7, 6, 2, 3, 4, 9, 2, 6, 4, 5, 7, 2, 8, 8, 5, 1, 1, 9, 4, 7, 9, 4, 5}
{9, 8, 9, 3, 8, 1, 9, 1, 2, 1, 4, 2, 2, 0, 7, 1, 8, 0, 5, 0, 3, 0, 1, 3, 1, 2}
{1, 5, 3, 6, 6, 9, 3, 5, 4, 4, 5, 5, 4, 8, 2, 5, 6, 0, 9, 8, 7, 1, 7, 8, 3, 4, 2}
{1, 5, 4, 6, 7, 7, 8, 8, 1, 4, 0, 1, 0, 0, 7, 7, 9, 9, 9, 7, 4, 5, 6, 4, 3, 3, 6}
{1, 3, 3, 1, 1, 8, 4, 1, 1, 1, 7, 4, 0, 5, 9, 6, 8, 8, 3, 9, 1, 0, 4, 5, 9, 5, 5}
{1, 5, 4, 1, 4, 0, 2, 7, 5, 4, 2, 8, 3, 3, 9, 9, 4, 9, 8, 9, 9, 9, 2, 2, 6, 5, 0}
{2, 9, 5, 7, 6, 8, 2, 3, 7, 3, 6, 1, 2, 9, 1, 7, 0, 8, 6, 4, 5, 2, 2, 7, 4, 7, 4}
{9, 5, 6, 6, 3, 3, 7, 2, 0, 4, 6, 4, 1, 1, 4, 5, 1, 5, 8, 9, 0, 3, 1, 8, 4, 1, 0}
{9, 8, 8, 2, 4, 2, 3, 1, 0, 3, 9, 3, 8, 6, 0, 3, 9, 0, 0, 6, 6, 9, 1, 1, 4, 1, 4}
{9, 4, 9, 3, 9, 7, 6, 8, 4, 0, 3, 9, 0, 2, 6, 5, 8, 6, 8, 5, 2, 2, 0, 6, 7, 2, 0, 0}
{4, 1, 7, 9, 6, 2, 0, 5, 7, 6, 5, 1, 4, 7, 4, 2, 6, 9, 7, 4, 7, 0, 4, 7, 9, 1, 5, 2, 8}
{7, 0, 2, 6, 7, 3, 7, 5, 5, 1, 0, 2, 0, 7, 8, 8, 5, 2, 4, 2, 2, 1, 8, 8, 3, 7, 4, 0, 4}
```

We remark that lattice methods alone can find sporadic large Keith numbers. One approach, from [15], improves the chances of getting a valid result from the lattice reduction step. The idea is to augment each null vector with a zero, and augment the lattice with a row consisting of some nonzero value (typically one) in the new column of zeros, and $\frac{9}{2}$ everywhere else. Thus if there is a valid solution then this augmented lattice contains the vector consisting of that nonzero value (or its negative) and the remaining entries in the range $\left\{-\frac{9}{2}, \frac{9}{2}\right\}$. As this would be a fairly "small" vector, one can hope that it will appear in the reduced basis (this is essentially the idea used by Schnorr and Euchner, in a binary setting, to raise the density at which one can typically solve subset sum problems). In practice we get a few Keith numbers this way as well as several more near misses.

It might be effective to combine this different lattice formulation with the branch−and−bound regimen described above. This is not entirely trivial as that required that the vectors span a solution space for a homoge− neous equation, whereas the vectors in this modified lattice need not satisfy that equation.

Observe that the Keith numbers beginning at 24 digits but smaller than 29 digits all have leading digit in the set {1, 2, 9}. One might well wonder if there is a deep reason for this, and whether the trend returns after 29 digits. Also all known Keith numbers from 25 digits onward have a final digit that is either even or 5, and hence they cannot be prime. Again, one might wonder whether this trend continues, and, if so, whether there is an interest− ing reason behind it.

We will say a bit about the practical complexity of the method we have described for solving ILPs. While in principle branching can have very bad performance, in practice we find that the complexity scales reasonably well with problem size. Although we cannot claim that the methods described in this paper are polynomial time even in fixed dimension, in practice they do seem to scale that way. The behavior with respect to dimension is of course not so nice. For Keith number computations we find that, on average, the time spent for handling $n+1$ digits is roughly twice the time needed for n digits. Considering that each additional digit multiplies the search space by a factor of 10 this is still not so bad. To give an indication of computational speed, the implemen tation in the appendix was able to handle 29 digits in around three days on a 3.0 GHz machine. We emphasize that this is but a crude measure both of complexity and actual performance, as various sorts of optimizations could have a significant impact on each. For example, preliminary experiments with the software in [6] indicate room for improvement in the handling of the ILP solving after the lattice reduction phase.

4. Frobenius Numbers

We are given a set $A = (a_1, \ldots, a_n)$ of positive integers with $\gcd(A) = 1$. For later purposes we assume that the set is in ascending order. It can be shown that there are at most finitely many numbers not representable as a nonnegative integer combination of elements in A. The largest such nonrepresentable is called the Frobenius number of the set. The "Frobenius number problem" is to find it. Generally speaking this tends to be a different and usually harder problem than the Frobenius instance solving discussed earlier. We will give a much abbrevi− ated discussion here in order to indicate how the sort of knapsack solving under discussion plays a role in computation of these numbers. References may be found in [4] and [17].

It turns out that this is trivial for $n = 2$ (this was shown by Sylvester in the late 1800’s, even before the problem was popularized by Frobenius early in the twentieth century). Moreover in the 1980’s some very good methods appeared for the case $n = 3$. For larger n, if one orders the elements by increasing size and restricts a_1 to be less than around 10^7 then there are effective algorithms to find the Frobenius number for A. This is roughly indepen− dent of the size of n; they can handle $n = 100$, for example. Our interest is in handling the case where a_1, the smallest element in A, is large (say, up to 10^{100}). As the problem is known to be intrinsically difficult we cannot hope to have both a_1 and n large. Hence we limit the latter to 10 or so.

In [17] one finds a definition of a "fundamental domain" which is a generalization that subsumes both lattice diagrams in earlier literature and a circulant graph description from [4]. Similar ideas, expressed in the terminol− ogy of Minimal Distance Diagrams, appear in [9]. The Frobenius number will be determined by the furthest corner from the origin, in a suitably weighted l_1 norm. As we will see, an important domain feature is what we term "elbows". We give a quick idea of what is this fundamental domain for our set $A = (a_1, \ldots, a_n)$.

We start with the lattice of integer combinations of $\{a_2, \ldots, a_n\}$ that are zero modulo a_1. This is a full dimen− sional lattice in $\mathbb{Z}^{n-1}$. The set of residues of $\mathbb{Z}^{n-1}$ modulo this lattice gives rise to the fundamental domain, which lives in a space of dimension one less than the size of our set. It is not hard to see that there are a_1 distinct residue classes, so we know the cardinality of this domain. We now define the "weight" of a vector $v \in \mathbb{Z}^{n-1}$ as

$v.(a_2, \ldots, a_n)$. It can be shown that every residue class has at least one element with all nonnegative entries. From those we choose one of minimal weight. In case of a tie we choose the one that is lexicographically last. This uniquely defines the set of residues that we take to comprise the fundamental domain.

This domain has several interesting properties. (1) It is a staircase: if it contains a lattice element then it contains all nonnegative vectors with any coordinate strictly smaller. (2) It tiles $\mathbb{Z}^{n-1}$. (3) It is a cyclic group $\mathbb{Z}/a_1\mathbb{Z}$. (4) It can be given a circulant graph structure. It is this structure that was utilized in various shortest−path graph methods. Old and new approaches using such methods are discussed at length in [4]. In contrast, the method put forth in [17] primarily makes use of the staircase structure.

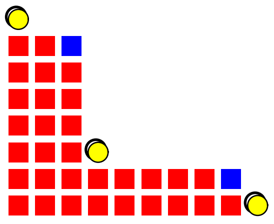
From the staircase property the fundamental domain has turning points we refer to as elbows. It has extremal points called corners. Specifically, a corner is a point c in the domain, such that $c + e_j$ is not in the domain, where e_j is the j^{th} coordinate vector. An elbow is a point x that is not in the domain, but is such that, for each j, either $x_j = 0$ or $x - e_j$ is in the domain. An elbow with all but one coordinate zero is called an "axial" elbow. It indicates how far one can go along a given axis and still remain inside the domain. There are two other defini− tions that play a role in the algorithm. We will not descibe them too carefully but, roughly, there are as follows.

(1) Protoelbows. These have both positive and negative coordinates and correspond to certain "minimal" equivalences (that is, reducing relations) in the lattice.

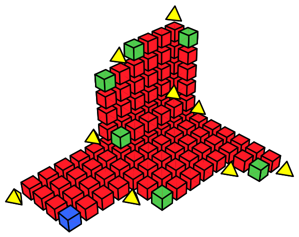
(2) Preelbows. These are the positive parts of the protoelbows. Elbows are minimal elements in the partially ordered (ascending by inclusion) set of preelbows.

With respect to these domains the Frobenius number corresponds to the farthest corner from the origin where distance is an l_1 metric weighted by element sizes. In brief one sees this as follows. Recall that each element in the domain corresponds to a residue modulo a_1 that satisfies a minimal nonnegativity property. Thus values in the same residue class but of smaller weight cannot be attained using nonnegative combinations of elements of A, whereas values of equal or larger weight are attained as such combinations. We conclude that the largest nonattainable value for the residue class of the element $X = (x_2, \ldots x_n)$ is $a_2 x_2 + \ldots + a_n x_n - a_1$. From this and the staircase property of the fundamental domain it is clear that the largest nonattainable value overall is a_1 less than the largest weight of a corner element.

Below are pictures (provided courtesy of Stan Wagon) of fundamental domains corresponding to $n = 3$ and $n = 4$ respectively (recall that the fundamental domain lives in a space of dimension one less than n). In the planar diagram the elbows are the lattice points on the axes that bound the diagram, and the lattice point in the interior just outside the "ell". The corners are the two extremal points reached by intersecting vertical and horizontal lines through the elbows. This picture tells the entire story as regards the $n = 3$ case because it can be shown that there is at most one interior elbow and two corners, and finding them is easy. The three elbows (two axial, one internal) are denoted by circles.



In the three dimensional diagram the elbows are again the axial bounding lattice points as well as bounding points where the staircase goes up in the coordinate planes and in the interior. They are demarcated by tetrahedra.



With this brief background we now say a bit about how knapsack solving can play a role in the computation of Frobenius numbers. First, it turns out that axial elbows are defined by an integer programming problem (see [17]) which, in complexity if not details of definition, is similar to the Frobenius instance problem. From the axial elbows we immediately get good lower and upper bounds on the Frobenius number (each elbow less one is in the domain, and the farthest corner is bounded by the vector with all components given by corresponding axial elbows less one). More importantly for our purposes is that they give a search space from which to find all elbows. They are particular integer points in a polyhedron that satisfy certain inequality conditions. Full details, including a method for finding them, are provided in [17]. From the elbows one can find all corners and in particular the one that gives the Frobenius number of the set.

Another tactic presented in [17] makes direct use of Frobenius instance solving to find axial elbows. One uses a bisection approach, working down from an a priori bound on the axial elbow values. The goal is to find the smallest value for which a certain set of Frobenius instances have no solution.

Still another point of overlap between Frobenius instances and numbers is the obvious fact that whenever a Frobenius instance solver returns an empty solution we automatically have a lower bound on the Frobenius number. One can test random values that are, say, an order of magnitude below the heuristic approximation for the Frobenius number presented in [4]. If any such test gives no solution we thereby establish a lower bound that is often better than a priori bounds to be found in the literature.

There is also a heuristic method in [17] for more efficiently "guessing" the likely Frobenius number from a restricted set of elbows. It gives an a priori upper bound, and a single Frobenius instance invocation can then verify whether it is in fact the actual value. In random examples this appears always to be the case.

We sketched above how efficient ILP knapsack solving, of the sort used for Frobenius instances, also may be applied to the (generally much harder) problem of finding Frobenius numbers. From the fundamental domain pictures one realizes a possible alternative approach. Working an axis at a time, a branch−and−bound strategy might be directly applied to get to extremal vertices (corners) in the domain. Thus we could have a bilevel branching algorithm, with the outer level iterating over these extrema, and the inner one using relaxed LPs to solve the ILPs needed to move to new vertices. This might become an alternative to the algorithm described in [17].

We remark that there is a connection between another knapsack solving technique and the problem of comput−ing Frobenius numbers. It is well known that integer programming e.g. for knapsack problems can be done with toric Gröbner bases [7]. What is not so obvious is that they may also be used to deduce the stairway structure of the fundamental domain. An algorithm for this purpose is presented in [17]. The basic idea is to formulate a term ordering so that the staircase structure of the fundamental domain is captured by the staircase of the Gröbner basis lead monomials. This has the added virtue of finding all elbows at once, so no elaborate method is needed to search a bounding box defined by axial elbows. An implementation by the author has handled Frobenius number problems involving as many as 7 numbers of 40 digits. While it is not competitive with the main approach in [17] (which has handled sets of up to 11 numbers), Frobenius number problems of this size are apparently larger than what can be handled by other methods from the published literature.

Another link between Frobenius numbers and methods from ideal theory appears implicitly in [9] as well as other literature concerning what are called "multiple−loop networks". There, as in [4], one works with a circulant directed graph based on residues modulo a positive integer. Now, however, the modulus is the largest rather than smallest element of the given set. While the authors did not explicitly consider the connection to Frobenius numbers, some of the ideas are quite similar. In particular the maximum diameter of this graph, which is similar to the Frobenius number (though using an unweighted metric), plays an important role in their work. They discuss several aspects of the related domain (actually family of domains, as they do not impose uniqueness conditions) in terms of monomial ideals. They define a generalized ell shape and prove their domains are always of such a shape; this corresponds to the uniqueness of the interior elbow as shown in [17]. It would be interesting to understand better how their ideas, in particular regarding use of monomial ideals, relate to the toric ideal construction of the fundamental domain given in [17].

5. Summary

We have investigated several examples of integer linear programs with the common feature that a straightforward branch−and−bound approach, working with real relaxations, will tend to bog down in searching a large polytope. We utilize a method based on solving of an integer relaxation to the set of equality constraints. We reformulate the problem as one of adding combinations of null vectors to a specific solution. The vectors in question tend to be well suited to the problem at hand because we use lattice reduction to make them close to orthogonal. We then enforce inequality constraints via branch−and−bound on real relaxations of the new problem. We use a branching choice that tends to make the polytope thin in the search direction and thus helps to exhaust it efficiently.

We reviewed this approach as it was applied to the change−making problem [1] [2]. We then used it to find all Keith numbers through 29 digits; previous methods had gone only through 19 digits. This brought us to a range where we could observe curious patterns in leading and trailing digits, something not present in the smaller Keith numbers. We also gave a brief idea of how this method is used in a new algorithm to compute Frobenius numbers.

A further direction would be to incorporate effective cutting planes. The code in the appendix only attempts a very naive sort of cut (by using random coordinate variables as objective function). Preliminary experiments with an external library [6] indicate that more serious cutting plane efforts can give substantial speed improvement; we have seen ILP examples that improve by an order of magnitude. We emphasize that this still requires preprocessing with lattice reduction in the manner described in this paper.

Acknowledgements

I thank Stan Wagon for providing the diagrams of fundamental domains and for his careful reading and many suggestions that improved the exposition. I thank Victor Moll for inviting me to attend the 2005 Clifford Lectures conference at Tulane, where I presented an earlier version of this work. I thank the two anonymous referees for their several suggestions which improved readability, and the second referee moreover for pointing out the relevance of [9] and related literature on multiple−loop networks.

6. Appendix: *Mathematica* Implementation of Selected Algorithms

We solve a system of integer equations over integers and extract a small solution by using combinations of null vectors to decrease the size of a specific solution. This is used as a first step in solving Frobenius instances, for example.

```
systemSolve[(mat_)?MatrixQ, (rhs_)?VectorQ] :=
 Module[{newmat, modrows, hnf, j = 1, len = Length[mat], zeros, solvec, nullvecs},
   newmat = Prepend[Transpose[mat], rhs];
   newmat = Transpose[Join[Transpose[newmat], IdentityMatrix[Length[newmat]]]];
   hnf = HermiteDecomposition[newmat][[2]];
   zeros = Table[0, {len}];
   While[j ≤ Length[hnf] && Take[hnf[[j]], len] =!= zeros, j++];
   solvec = Drop[hnf[[j]], len + 1] / (-hnf[[j, len + 1]]);
   nullvecs = (Drop[#1, len + 1] &) /@ Drop[hnf, j];
   {solvec, LatticeReduce[nullvecs]}
  ] /; Length[rhs] == Length[mat]
```

```
smallSolution[(sol_)?VectorQ, (nulls_)?MatrixQ] := Module[
  {max, dim = Length[nulls] + 1, weight, auglat, lat, k, soln}, lat = Prepend[nulls, sol];
  max = Max[Flatten[Abs[lat]]];
  weight = dim * max ^ 2;
  auglat = (Prepend[#1, 0] &) /@ lat;
  auglat[[1, 1]] = weight;
  lat = LatticeReduce[auglat];
  For[k = 1, lat[[k, 1]] == 0, k++];
  soln = lat[[k]];
  Which[soln[[1]] == weight, Drop[soln, 1],
   soln[[1]] == -weight, -Drop[soln, 1],
   True, sol]]
```

The code below will find the set of Keith number linear equations for a given number of digits. Integer solutions to any of these equations, subject to the constraints that the first variable be positive and all variables lie between 0 and 9, will give Keith numbers.

```
keithEquations[len_Integer /; len > 0] := Module[{matrow, n, list, res, vecs},
  res = list[];
  Do[matrow[j] = Table[KroneckerDelta[k, j + 1], {k, len}], {j, len - 1}];
  matrow[len] = Table[1, {len}];
  n = len;
  While[9 * Apply[Plus, matrow[n]] < 10 ^ (len - 1), n++;
   matrow[n] = Sum[matrow[k], {k, n - len, n - 1}];];
  While[First[matrow[n]] ≤ 10 ^ (len - 1), res = list[res, matrow[n]];
   n++;
   matrow[n] = Sum[matrow[k], {k, n - len, n - 1}];];
  vecs = Apply[List, Flatten[res, Infinity, list]];
  Map[(# - 10 ^ Range[len - 1, 0, -1]) &, vecs]]
```

Here we give a fairly straightforward implementation of the Keith number solver. The input is a set of integer vectors spanning the solution space for a particular Keith number homogeneous linear equation. The program will find all possible combinations that have all components between 0 and 9 and the first one nonzero (so they correspond to digits with the leading one positive), or else terminate with an empty solution set.

```
keithSolutions[onulls_] :=
 Module[{nulls, vars, x, len = Length[onulls], vecs, constraints, program,
   stack, soln, solns = {}, badvar, varvals, val, counter = 1, var, extra,
   maxs, mins, ctmp = {}, octmp, bad = False, vnum, vval, eps = 1/10^5, rndvar},
  nulls = Reverse[onulls[[Ordering[Norm /@ N[onulls]]]]]; vars = Array[x, len];
  vecs = vars.nulls; constraints = Join[{1 ≤ First[vecs] ≤ 9}, (0 ≤ #1 ≤ 9 &) /@ Rest[vecs]];
  While[ctmp =!= octmp, mins = Table[
     Internal`DeactivateMessages[val = NMinimize[{vars[[j]], Join[constraints, ctmp]}, vars];
      If[Head[val] === NMinimize || ! FreeQ[val, Indeterminate], bad = True; Break[]];
      val = First[val], NMinimize::"nsol"], {j, len}]; maxs = Table[
     Internal`DeactivateMessages[val = NMaximize[{vars[[j]], Join[constraints, ctmp]}, vars];
      If[Head[val] === NMaximize || ! FreeQ[val, Indeterminate], bad = True; Break[]];
      val = First[val], NMaximize::"nsol"], {j, len}]; octmp = ctmp;
   ctmp = Join[Thread[vars ≤ Floor[maxs + eps]], Thread[vars ≥ Ceiling[mins - eps]]];];
  If[bad, Return[{counter, {}}]]; constraints = Join[constraints, ctmp];
  program = constraints; stack = {program, {}};
  While[stack =!= {}, counter++; program = stack[[1]]; stack = stack[[2]];
   rndvar = vars[[RandomInteger[{1, len}]]]; program = {rndvar, program};
   Internal`DeactivateMessages[vals = NMinimize[program, vars], NMinimize::"nsol"];
   If[Head[vals] == NMinimize, Continue[]]; vval = Ceiling[First[vals] - eps];
   vals = Chop[vals[[2]]]; soln = Chop[vecs /. vals];
   If[! FreeQ[soln, Indeterminate], Continue[]];
   constraints = program[[2]]; varvals = vars /. vals;
   badvar = Position[varvals, a_ /; Chop[a - Round[a]] =!= 0, {1}, 1, Heads → False];
   If[badvar == {}, soln = Round[soln]; solns = {soln, solns};
    Do[extra = Table[vecs[[k]] == soln[[k]], {k, j - 1}];
     stack = {Join[constraints, Append[extra, vecs[[j]] ≤ soln[[j]] - 1]], stack};
     stack = {Join[constraints, Append[extra, vecs[[j]] ≥ soln[[j]] + 1]], stack};,
     {j, Length[soln]}]; Continue[]]; badvar = badvar[[1, 1]]; var = vars[[badvar]];
   val = var /. vals; stack = {Join[constraints, {rndvar ≥ vval, var ≤ Floor[val]}], stack};
   stack = {Join[constraints, {rndvar ≥ vval, var ≥ Ceiling[val]}], stack};];
  {counter, Partition[Flatten[solns], Length[First[nulls]]]}]
```

Not surprisingly there are various ways to improve on this sort of solving once one begins with a good basis set. Some dedicated ILP solvers seem to do these perhaps an order of magnitude faster.

The next snippets of code will generate all possible Keith number equations for 2 through 29 digits and call the solver above on each.

```
integerNullSpace[vec : {_Integer ..}] := Module[{mat, hnf},
  mat = Transpose[Join[{vec}, IdentityMatrix[Length[vec]]]];
  hnf = Last[Developer`HermiteNormalForm[mat]];
  LatticeReduce[Map[Drop[#, 1] &, Drop[hnf, 1]]]]
Do[
  keqns[j] = keithEquations[j];
  Do[vecs = integerNullSpace[keqns[j][[k]]];
   nulls[j, k] = Reverse[vecs[[Ordering[Map[Norm, N[vecs]]]]]], {k, Length[keqns[j]]}],
  {j, 2, 29}];
Table[{keithSolutions[nulls[j, k]]}, {j, 2, 29}, {k, Length[keqns[j]]}]
```

We can readily check whether a given sequence represents a Keith number.

```
KeithQ[n_] := Last[NestWhile[Append[#, Total[Take[#, -Length[IntegerDigits[n]]]]] &,
     IntegerDigits[n], Last[#] < n &]] == n;
```

The routine below is an adaptation of one in [15] for solving low density binary knapsack problems. We modify in order to try for solutions with variables taking nonnegative single digit values.

```
integerNullSpace2[origvec :{_Integer ..}] :=
 Module[{vec, mat, hnf, red, vecs, m}, vec = origvec ;
  mat = Transpose[Join[{vec}, IdentityMatrix[Length[vec]]]];
  hnf = Drop[Last[Developer‘HermiteNormalForm[mat]], 1];
  vec = Table[-9 / 2, {Length[vec] + 1}];
  vec[[1]] = 1;
  hnf = LatticeReduce[hnf];
  hnf = Prepend[hnf, vec];
  red = LatticeReduce[hnf];
  vecs = Cases[red, {1 | -1, ___}];
  vecs = Map[Rest[# / Sign[First[#]]] &, vecs];
  vecs + 9 / 2]
```

7. References

[1] K. Aardal, C. A. J. Hurkens, and A. K. Lenstra. Solving a system of linear diophantine equations with lower and upper bounds on the variables. *Mathematics of Operations Research* **25**:427−442, 2000.

[2] K. Aardal and A. K. Lenstra. Hard equality constrained knapsacks. In *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization* (IPCO 2002), W. J. Cook and A. S. Schulz, eds. Lecture Notes in Computer Science 2337, 350−366. Springer−Verlag, 2002.

[3] K. Aardal, R. Weismantel, and L. A. Wolsey. Non−standard approaches to integer programming. *Discrete Applied Mathematics* **123**:5−74, 2002.

[4] D. Beihoffer, J. Hendry, A. Nijenhuis, and S. Wagon. Faster algorithms for Frobenius numbers. *Elec. J. Combinatorics* **12**, 2005.

[5] W. A. Blankenship. Algorithm 288: Solution of simultaneous linear diophantine equations. *Communications of the ACM* **9**(7):514, 1966.

[6] COmputational INfrastructure for Operations Research (COIN−OR). Home page URL:
http://www.coin−or.org/documentation.html

[7] P. Conti and C. Traverso. Gröbner bases and integer programming. *Proceedings of the 9th International Symposium on Applied Algebra, Algebraic Algorithms and Error−Correcting Codes (AAECC−9).* H. F. Mattson, T. Mora, and T. R. N. Rao, eds. Lecture Notes in Computer Science 539, 130−139. Springer−Verlag, 1991

[8] G. Dantzig. *Linear Programming and Extensions*. Princeton Landmarks in Mathematics and Physics. Princeton University Press, 1963 (Reprinted 1998).

[9] D. Gómez−Perez, J. Gutierrez, and Á. Ibeas. Circulant digraphs and monomial ideals. *Proceedings of the 8th Interna− tional Workshop on Computer Algebra in Scientific Computing (CASC 2005)*, V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, eds. Lecture Notes in Computer Science 3718, 196−205. Springer−Verlag, 2005.
Extended version available electronically at:
http://personales.unican.es/ibeasaj/circula/

[10] M. Keith. Determination of all Keith Numbers up to 10^{19}. Electronic manuscript, 1998.
Available electronically at:
http://users.aol.com/s6sj7gt/keithnum.htm
See also:
http://users.aol.com/s6sj7gt/mikekeit.htm

[11] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen* **261**:515−534, 1982.

[12] D. Lichtblau. Revisiting strong Gröbner bases over Euclidean domains. Manuscript, 2003.

[13] K. R. Matthews. Short solutions of A X=B using a LLL−based Hermite normal form algorithm. Manuscript, 2001.

[14] P. Nguyen. Cryptanalysis of the Goldreich−Goldwasser−Halevi cryptosystem from Crypto ’97. Advances in Cryptol− ogy, *Proceedings of CRYPTO 1999*, Santa Barbara, CA, 1999. Lecture Notes in Computer Science 1666, 288−304. Springer−Verlag, 1999.
Available electronically at:
http://www.di.ens.fr/~pnguyen/pub.html#Ng99

[15] C. P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. In *Proceedings of the 8th International Conference on Fundamentals of Computation Theory*, 1991. L. Budach, ed. Lecture Notes in Computer Science 529, 68−85. Springer−Verlag, 1991.

[16] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley−Interscience Series in Discrete Mathematics and Optimization, 1986.

[17] S. Wagon, D. Einstein, D. Lichtblau, and A. Strzebonski. Frobenius numbers by lattice enumeration. Submitted.

[18] S. Wolfram. *The Mathematica Book* (5th edition). Wolfram Media, 2003.