J. Symbolic Computation (1995) **19**, 507–526

A Fast Method for Finding the Basis of Non-negative Solutions to a Linear Diophantine Equation

MIGUEL FILGUEIRAS AND ANA PAULA TOMÁS

Universidade do Porto

R. do Campo Alegre 823, 4150 Porto, Portugal [†]

(Received May 1993, revised June 1994 and September 1995)

We present a complete characterization of the set of minimal solutions of a single linear Diophantine equation in three unknowns over the natural numbers. This characterization, for which we give a geometric interpretation, is based on well-known properties of congruences and we use it as the foundation of direct algorithms for solving this particular kind of equation. These direct algorithms and an enumeration procedure are then put together to build an algorithm for solving the general case of a Diophantine equation over the naturals. We also put forth a statistical method for comparing algorithms for solving Diophantine equations which is more sound than comparisons based on times observed for small sets of equations. From an extensive comparison with algorithms described by other authors it becomes clear that our algorithm is the fastest known to date for a class of equations. Typically the equations in this class have a small number of unknowns in one side, the maximum value for their coefficients being greater than 3.

1. Introduction

Research on algorithms for solving linear Diophantine equations on the non-negative integers, or systems of such equations, had a significant increase in the recent past. This may be explained by their use in term rewriting techniques, namely in unification algorithms of terms with associative and commutative function symbols (the so-called AC-unification) (Stickel, 1981; Huet, 1978; Guckenbiehl & Herold, 1985), and also by the surge of programming languages and systems based on constraint solvers — examples are Constraint Logic Programming languages under the CLP-scheme (Jaffar *et al.*, 1986; Jaffar & Lassez, 1987), and problem solvers emerging from work on Operations Research and Artificial Intelligence.

It is well known that efficient algorithms do exist for solving these kind of equations (or systems of them) when the domain is the integers (Chou & Collins, 1982). However, restricting the domain to the naturals makes those algorithms unsuitable. This also happens with the work on geometric lattices described, for instance, in Bachem and Kannan (1984).

[†] E-mail: mig@ncc.up.pt, apt@ncc.up.pt

Several specific algorithms have been put forth to find the basis of non-negative solutions to linear Diophantine equations or systems of such equations†. In other words, they find the set of minimal solutions which are defined as the non-zero solutions that are minimal in a component-wise ordering. This set may be viewed as the basis of the commutative monoid of all solutions. The oldest algorithm we know of is due to Elliott (1903) in the context of Number Theory. Domenjoud (1991) gives an overview of other algorithms.

In the sequel we describe a new algorithm, which we named *Slopes* (Filgueiras & Tomás, 1992*a*, 1992*b*), that solves a single homogeneous Diophantine equation. It may be easily extended to the non-homogeneous case along the lines of Guckenbiehl and Herold (1985) and Contejean (1993). First we give a complete characterization of the set of minimal solutions of a Diophantine equation in three unknowns, based on well-known properties of congruences. We use this characterization as the foundation of direct algorithms for solving this particular kind of equation. The basic idea in Slopes is that of combining these very efficient algorithms with an enumeration procedure for the values of some unknowns. We have used a similar approach in developing *CBA*, for *congruence-based algorithm*, (Tomás & Filgueiras, 1991*a*, 1991*b*) and the *Rectangles Algorithm* (Filgueiras & Tomás, 1993).

We also put forth a statistical method for comparing algorithms for solving Diophantine equations which is more sound than comparisons based on times observed for small sets of equations. An extensive comparison with other algorithms was made and we conclude that Slopes is the fastest for a class of equations which in most cases have a small number of unknowns in one side and a maximum value for the coefficients greater than 3.

In the next sections we start by presenting the basic ideas used in Slopes, and a series of examples and geometric interpretations of the algorithms for solving equations in three unknowns. We then proceed with the formal details that prove the soundness of these algorithms, and a description of the general method and of its implementation. Comparisons of efficiency with other algorithms are then addressed, followed by the conclusions.

2. General ideas

The method we describe in this paper to solve

$$\sum_{i}^{N} a_i \cdot x_i = \sum_{j}^{M} b_j \cdot y_j \qquad a_i, b_j, x_i, y_j \in \mathbb{N} \tag{2.1}$$

has the following characteristics:

(i) solving the problem in non-negative integers is replaced by solving a family of sub-problems in *positive* integers; each sub-problem is simply the result of setting some of the unknowns to 0;
(ii) for each sub-problem, enumeration is performed *for all but three* of the unknowns.
(iii) enumeration is controlled by the bounds of Huet (1978) and Lambert (1987) and bounds derived from certain kinds of solutions;

† We will not discuss general algorithms from Integer Programming that could be applied to the present problem. See Abdulrab & Pécuchet (1986) for tentative work in this vein with seemingly unconvincing results.

(iv) for each tuple fixed by enumeration, an equation of the form (2.2) is solved.

$$a \cdot x = b \cdot y + c \cdot z + v, \qquad a, b, c, x, y, z \in \mathbb{N} - \{0\}, \quad v \in \mathbb{Z} - \{0\} \tag{2.2}$$

We have found that (2.2) can be solved *directly* in the sense that all the tuples generated by the solver are minimal solutions.

The basic result is that, if minimal solutions of (2.2) are ordered with z strictly increasing†, then both the solution with the smallest z and the differences (called *spacings* in the sequel) between consecutive solutions can be computed algebraically.

In establishing this result the following geometric characterization was of use. Let us consider the convex hull of the projections of the minimal solutions onto the YZ-plane. Then, *when $v \geq 0$ all the minimal solutions and only minimal solutions lie on its border*‡, which we can easily characterize (see Corollary 4.0.1). When $v < 0$ the minimal solutions lie on a polygonal line which can also be easily found out.

3. Solving equations in three unknowns — examples

Solving (2.2) turns out to be simpler when viewed in terms of congruences. It is well-known that each solution of (2.2) verifies the congruence

$$b \cdot y + c \cdot z \equiv -v \pmod{a} \tag{3.1}$$

and reciprocally, that each solution of (3.1) corresponds to some integral solution of (2.2).

As an example, suppose the equation $8x = 6y + 5z$. Table 1 illustrates the distribution of $6y + 5z$ mod 8 in the YZ–plane for $(y, z) \in \mathbb{N} \times \mathbb{N}$. Each 0-entry in the Table corresponds to a solution of the equation.

Table 1. Distribution of $6y + 5z$ mod 8 throughout the YZ–plane

Z																
9	5	3	1	7	5	3	1	7	5	3	1	7	5	3	1	
8	0	6	4	2	0	6	4	2	0	6	4	2	0	6	4	
7	3	1	7	5	3	1	7	5	3	1	7	5	3	1	7	
6	6	4	2	0	6	4	2	0	6	4	2	0	6	4	2	
5	1	7	5	3	1	7	5	3	1	7	5	3	1	7	5	
4	4	2	0	6	4	2	0	6	4	2	0	6	4	2	0	
3	7	5	3	1	7	5	3	1	7	5	3	1	7	5	3	
2	2	0	6	4	2	0	6	4	2	0	6	4	2	0	6	
1	5	3	1	7	5	3	1	7	5	3	1	7	5	3	1	
0	0	6	4	2	0	6	4	2	0	6	4	2	0	6	4	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Y

† In this presentation we suppose that solutions are computed in this order to which we will refer as z-increasing order. However, similar results hold if a y-increasing order is taken instead.

‡ More accurately, the edge linking the first to the last solution should not be included, unless the projection is a line segment.

There is a repetitive pattern, the plane being partitioned into identical rectangles $y_{\max} \times z_{\max}$.

In the above example, $y_{\max} = 4$ and $z_{\max} = 8$.

A trivial conclusion is that the projection along YZ of each positive minimal solution of (2.2) for $v = 0$ lays on $R_0 = [0, y_{\max} - 1] \times [0, z_{\max} - 1]$. In fact, one just has to note that if $s = (y, z)$ is a solution then $s' = (y \bmod y_{\max}, z \bmod z_{\max})$ is also a solution, and $s' \leq s$. This is still true if $v > 0$, but may be false when $v < 0$ for in this case the value of x corresponding to s' may be negative. The congruence is, however, still useful as a guide in the search for positive solutions, since it gives an accurate idea of the place and pattern of distribution of integral solutions.

We often denote solutions as pairs (y, z) instead of triples (x, y, z) for x is fixed by (y, z). Moreover, we often use a similar abbreviate notation for spacings, that is $(-\delta_y, \delta_z)$ appears instead of a triple, $(\delta_x, -\delta_y, \delta_z)$. Note that solutions are computed in z-increasing order so that the variation in y must be negative.

In the rest of this section, we present some examples so as to give some intuition on the method.

Example 1. Find minimal $(x, y, z) \in \mathbb{N}^3$ such that $13x = 6y + 8z$.
The values of $6y + 8z \bmod 13$ are shown in Table 2. All the positive minimal solutions are in $[0, 12] \times [0, 12]$, thus Table 2 presents just that rectangle and the two minimal solutions $(0, 13)$ and $(13, 0)$. The entries corresponding to the minimal (resp. non-minimal) solutions are denoted by ⊗ (resp. ○).

Table 2. Solving $13x = 6y + 8z$

Z															
13	⊗														
12	5	11	4	10	3	9	2	8	1	7	○	6	12		
11	10	3	9	2	8	1	7	○	6	12	5	11	4		
10	2	8	1	7	○	6	12	5	11	4	10	3	9		
9	7	⊗	6	12	5	11	4	10	3	9	2	8	1		
8	12	5	11	4	10	3	9	2	8	1	7	○	6		
7	4	10	3	9	2	8	1	7	○	6	12	5	11		
6	9	2	8	1	7	○	6	12	5	11	4	10	3		
5	1	7	⊗	6	12	5	11	4	10	3	9	2	8		
4	6	12	5	11	4	10	3	9	2	8	1	7	○		
3	11	4	10	3	9	2	8	1	7	○	6	12	5		
2	3	9	2	8	1	7	○	6	12	5	11	4	10		
1	8	1	7	⊗	6	12	5	11	4	10	3	9	2		
0	○	6	12	5	11	4	10	3	9	2	8	1	7	⊗	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	Y

The algorithm computes the starting solution $(13, 0)$, and the *minimum spacing* $\Delta = (-10, 1)$ between minimal solutions (by Properties 4.1 and 4.2 in subsection 4.1). Then, $(3, 1) = (13, 0) + \Delta$. Since, Δ cannot be added to $(3, 1)$ because y would become negative, some formula (see Theorem 4.1 in subsection 4.2) is applied to compute the next solution $(2, 5)$. A new spacing $\Delta' = (-1, 4) = (2, 5) - (3, 1)$ is obtained, which is then used to derive other minimal solutions.

A spacing $(-\delta_y, \delta_z)$ is called *the minimum spacing* if δ_z is the minimum positive spacing in z between integral solutions, and the corresponding $\delta_y < y_{\max}$ (see Property 4.2 in subsection 4.1). By *m-spacing* we mean a spacing between consecutive minimal solutions.

In the previous example, only two m-spacings are used: $(-10, 1)$, and $(-1, 4)$.

Example 2. Find minimal $(x, y, z) \in \mathbb{N}^3$ such that $13x = 6y + 8z + 9$.
This equation may be viewed as $6y + 8z \equiv 4 \pmod{13}$, so that if we were to solve it by hand, we could go into Table 2 looking for the 4's. Table 3 shows the minimal solutions (marked by $\otimes$) of this problem. The solution with the smallest z is $(5, 0)$, and the first m-spacing is $(-4, 3)$.

Table 3. Solving $13x = 6y + 8z + 9$

Z														
13	0													
12	5	11	4	10	3	9	2	8	1	7	0	6	12	
11	10	3	9	2	8	1	7	0	6	12	5	11	4	
10	2	8	1	7	0	6	12	5	11	4	10	3	9	
9	7	0	6	12	5	11	4	10	3	9	2	8	1	
8	12	5	11	4	10	3	9	2	8	1	7	0	6	
7	$\otimes$	10	3	9	2	8	1	7	0	6	12	5	11	
6	9	2	8	1	7	0	6	12	5	11	4	10	3	
5	1	7	0	6	12	5	11	4	10	3	9	2	8	
4	6	12	5	11	4	10	3	9	2	8	1	7	0	
3	11	$\otimes$	10	3	9	2	8	1	7	0	6	12	5	
2	3	9	2	8	1	7	0	6	12	5	11	4	10	
1	8	1	7	0	6	12	5	11	4	10	3	9	2	
0	0	6	12	5	11	$\otimes$	10	3	9	2	8	1	7	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
														Y

Suppose $s = (y_0, z_0)$ is some minimal solution of (2.2), with $v \geq 0$. If $\Delta = (-\delta_y, \delta_z)$ is the m-spacing that must be added to s to get the next minimal solution, δ_z is the smallest positive feasible variation such that $\delta_y \leq y_0$. When $v < 0$, an additional condition must be satisfied: that x is kept non-negative.

As we will see in Subsection 4.3 (see Corollary 4.1.1) the m-spacings required to solve the family of problems $\{a \cdot x = b \cdot y + c \cdot z + v\}_{v \geq 0}$ are in a one-to-one correspondence with the positive minimal solutions of some homogeneous equation in 3 unknowns: $(-\delta_y, \delta_z)$ is a m-spacing if and only if (δ_y, δ_z) is a positive minimal solution of that equation. As we explain in Subsection 4.3, such equation results from the m-spacings being somehow multiples of the minimum spacing $(-\delta_{y_1}, \delta_{z_1})$, as follows:

$$\delta_y \equiv \delta_{y_1} \cdot (\delta_z / \delta_{z_1}) \pmod{y_{\max}}. \tag{3.2}$$

In the examples above, where $a = 13$, $b = 6$, and $c = 8$, the non-trivial m-spacings (that is, those for which neither the variation in z nor that in y is null) are $\{(-10, 1), (-7, 2), (-4, 3), (-1, 4)\}$. Note that $(-7, 2)$ is the first m-spacing if, for instance, $6y + 8z \equiv 9 \pmod{13}$ was to be solved. These m-spacings correspond to the minimal solutions of $\delta_y \equiv 10 \cdot \delta_z \pmod{13}$.

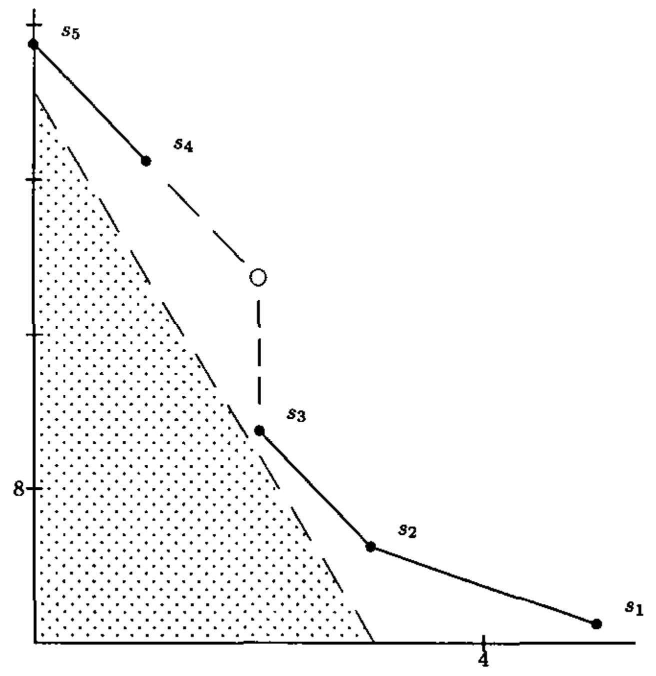

Figure 1. The case $v < 0$

When $v < 0$, the correspondence is still simple but slightly different. As we show in Subsection 4.3 (Proposition 4.1) and the following example illustrates, if $(-\delta_y, \delta_z)$ is a m-spacing, then $(-\delta_y, \delta_z) = t \cdot (0, z_{\max}) + (-\delta'_y, \delta'_z) + r \cdot (-y_{\max}, 0)$ for some $t \geq 0$, some $r \geq 0$ and some positive minimal solution (δ'_y, δ'_z) of (3.2). Thus, the trivial m-spacings $(-y_{\max}, 0)$ and $(0, z_{\max})$ which are never used if $v \geq 0$, may have to be used when $v < 0$.

Example 3. Find minimal $(x, y, z) \in \mathbb{N}^3$ such that $8x = 46y + 5z - 139$. Because $46y + 5z - 139 \equiv 6y + 5z - 3 \pmod 8$, we recall Table 1 which shows the behaviour of $6y + 5z \bmod 8$. The m-spacings are obtained from the set of minimal solutions of $\delta_y \equiv 3 \cdot (\delta_z/2) \bmod 4$. Because, in this case, the variation in x matters, we represent the spacings associated to those minimal solutions by triples, $(\delta_x, -\delta_y, \delta_z)$:

$$\{(-23, -4, 0), (-16, -3, 2), (-9, -2, 4), (-2, -1, 6), (5, 0, 8)\}.$$

Figure 1 shows the projections of the minimal solutions onto the YZ-plane. The starting solution, which is labelled by s_1, is $(-11, 1, 1) + (23, 4, 0) = (12, 5, 1)$. Then, $s_2 = s_1 + (-9, -2, 4) = (3, 3, 5)$, and $s_3 = s_2 + (-2, -1, 6) = (1, 2, 11)$. The idea is to apply the first (if in δ_z-increasing order) spacing that keeps both y and x non-negative. Now, none of the non-trivial m-spacings in the set can be applied to s_3. The rule, then, is to compute the minimum $t > 0$ such that a spacing other than that corresponding to $(0, z_{\max}) = (0, 8)$ can be added to $s_3 + t \cdot (5, 0, 8)$. Thus, $s_4 = (s_3 + (5, 0, 8)) + (-2, -1, 6) = (4, 1, 25)$, and finally $s_5 = s_4 + (-2, -1, 6) = (2, 0, 31)$. For the proof of the correctness of this method refer to Proposition 4.1.

In the next section, the results that support our algorithm are stated more formally and some of the proofs are given or sketched.

4. Solving equations in three unknowns — formal details

The problem is to find (x, y, z) minimal in component-wise ordering such that

$$a \cdot x = b \cdot y + c \cdot z + v, \qquad a, b, c, x, y, z \in \mathbb{N}, \;\; v \in \mathbb{Z}. \tag{4.1}$$

As we have seen, the general idea of our algorithm to solve (4.1) is the following: solutions are obtained in z-increasing order, each solution but the first being computed from the previous one and some m-spacing. In this section, some technical details are given, namely how the starting solution and the m-spacings (and in particular, the minimum spacing) are computed. Moreover, the correctness of the method is justified.

4.1. THE STARTING SOLUTION AND THE MINIMUM SPACING

Clearly, if $v = 0$ then $s_0 = (b/\gcd(a,b), y_{\max}, 0)$ and $s_L = (c/\gcd(a,c), 0, z_{\max})$ are minimal solutions of (4.1) where $y_{\max} = a/\gcd(a,b)$, and $z_{\max} = a/\gcd(a,c)$. Thus, in this case the starting solution, that is the one with the smallest z is s_0.

It is not difficult to check the following properties.

PROPERTY 4.1. *A non-negative integer z satisfies (4.1) with $v = 0$ iff $z = \delta_{z_1} \cdot t$, being $\delta_{z_1} = \gcd(a,b)/\gcd(a,b,c)$, and $t \in \mathbb{N}$.*

PROPERTY 4.2. *The minimum spacing is $(-\delta_{y_1}, \delta_{z_1})$ where*

$$\delta_{y_1} = \frac{c}{\gcd(a,b,c)} \cdot m_b \bmod y_{\max} \qquad \delta_{z_1} = \frac{\gcd(a,b)}{\gcd(a,b,c)},$$

being m_b an integer such that[†] $\gcd(a,b) = m_a \cdot a + m_b \cdot b$.

PROPERTY 4.3. *The spacing between any two solutions is of the form*

$$(-\delta_{y_1} \cdot k + y_{\max} \cdot t, \delta_{z_1} \cdot k)$$

for some integers k and t.

PROPERTY 4.4. *Provided that $\gcd(a,b,c)$ divides v, the starting solution s_0 of (4.1) is given by*

$$(y_0, z_0) + k \cdot (y_{\max}, 0)$$

where $k = max\{0, \lceil(-b \cdot y_0 - c \cdot z_0 - v)/(b \cdot y_{\max})\rceil\}$ and

$$z_0 = \frac{-v \cdot M_c}{\gcd(a,b,c)} \bmod \delta_{z_1} \qquad y_0 = \frac{(-v - z_0 \cdot c) \cdot m_b}{\gcd(a,b)} \bmod y_{\max}.$$

Here, M_c is any integer satisfying $b \cdot M_b + c \cdot M_c + a \cdot M_a = \gcd(a,b,c)$, for some integers M_a, and M_b, and $y_{\max}$, δ_{z_1}, and m_b are as defined above.

[†] This is the same as saying that m_b is the inverse of $b/\gcd(a,b)$ in $\mathbb{Z}[y_{\max}]$; and $n \bmod m$ denotes the (positive) remainder of n/m.

Property 4.1 is a simple consequence of the fact that Equation (4.1) with $v = 0$ has some integral solution if and only if $\gcd(a, b)$ divides $c \cdot z$. Properties 4.2 and 4.3 follow from Property 4.1, because each spacing $(-\delta_y, \delta_z)$ satisfies $b \cdot (-\delta_y) + c \cdot \delta_z \equiv 0 \pmod{a}$. In what concerns Property 4.4, the detail that may be less trivial is that $z = (-v \cdot M_c)/\gcd(a, b, c)$ verifying (4.1) as an integral solution implies that the smallest non-negative z_0 must be as defined. However, one just has to note that if z satisfies (4.1), also does $z + p \cdot \delta_{z_1}$, for $p \in \mathbb{Z}$. Now, because δ_{z_1} is the minimum spacing, $z \bmod \delta_{z_1}$ is the smallest non-negative value. Finally, the correction $k \cdot (y_{\max}, 0)$ is needed when $v < 0$ and the value x_0 determined by (y_0, z_0) is negative.

4.2. THE HOMOGENEOUS EQUATION

In this section, one of the major results is stated and proved. As a corollary, the minimal solutions of the homogeneous equation in three unknowns can be computed directly. That is, there is an algorithm that yields the minimal solutions without generating superfluous candidate solutions.

THEOREM 4.1. *Let $s_2 = (y_k, z_k)$ and $s_1 = (y_k + \delta_{y_k}, z_k - \delta_{z_k})$ be two minimal solutions of (4.1) with $v = 0$ such that $0 < y_k < \delta_{y_k}$ and there is no other minimal solution with y-component in $]y_k, y_k + \delta_{y_k}[$. Then, taking $F_k = \lceil \delta_{y_k}/y_k \rceil$, the minimal solution with maximum y-component less than y_k is*

$$(y', z') = (F_k \cdot y_k - \delta_{y_k}, F_k \cdot z_k + \delta_{z_k})$$

In the example of Figure 2, we have $F_k = 3$, $p_2 = F_k \cdot s_2$ and $s_3 - p_2 = s_2 - s_1 = (\delta_{y_k}, \delta_{z_k})$. The proof below can be easily understood if one notes that the rectangles with dashed sides (e.g. the one whose sides contain the (integral) solutions n_1, n_2, and p_1) cannot contain any interior solution (e.g. like s), as this would imply the existence of a solution in $]0, y_k[\times]0, \delta_{z_k}[$ (e.g., as s'') and this contradicts the minimality of s_1 and s_2.

PROOF. That (y', z') is indeed a solution follows trivially from s_1, s_2 being solutions.

That (y', z') is a minimal solution with maximum y-component $< y_k$ is now proved by showing that the existence of a solution $s = (y, z)$ in the rectangle $]0, y_k[\times[\delta_{z_k}, F_k \cdot z_k + \delta_{z_k}[$ leads to a contradiction[†]. Let the natural n, $1 \leq n \leq F_k$, be such that

$$(n-1) \cdot z_k + \delta_{z_k} \leq z < n \cdot z_k + \delta_{z_k}$$

. As (y, z), $(y_k + \delta_{y_k}, z_k - \delta_{z_k})$, and (y_k, z_k) are solutions, so is

$$s'' = (y'', z'') = (y_k + \delta_{y_k} + y - n \cdot y_k, \;\; z_k - \delta_{z_k} + z - n \cdot z_k)$$

From $0 < y < y_k$ and $1 \leq n$ we have $y - n \cdot y_k < 0$, and because $n \leq F_k$ and $F_k \cdot y_k$ is the minimum multiple of y_k greater than δ_{y_k} we have $(n-1) \cdot y_k < \delta_{y_k}$, or

[†] Note that in the rectangle $]0, y_k[\times]0, \delta_{z_k}[$ there can be no solutions because otherwise s_1, s_2 would not be minimal.

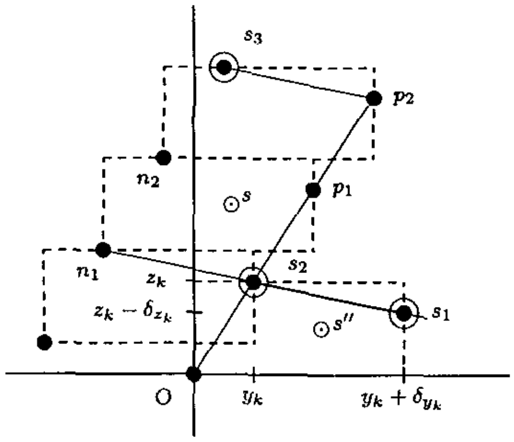

Figure 2. Finding the next m-spacing

equivalently $n \cdot y_k < y_k + \delta_{y_k}$. It follows

$$0 < y_k + \delta_{y_k} + y - n \cdot y_k < y_k + \delta_{y_k} \qquad 0 < y'' < y_k + \delta_{y_k}. \tag{4.2}$$

From the definition of n,

$$n \cdot z_k - z_k + \delta_{z_k} \leq z \qquad z - n \cdot z_k - \delta_{z_k} < 0$$

so that

$$0 \leq z_k - \delta_{z_k} + z - n \cdot z_k < z_k \qquad 0 < z'' < z_k. \tag{4.3}$$

But (4.2) and (4.3) contradict the assumption that s_1 and s_2 are minimal solutions. □

In other words, the theorem provides an expression of the next minimal solution when the current m-spacing, denoted by $(-\delta_{y_k}, \delta_{z_k})$ cannot be added to the current solution (s_2 in Figure 2).

From the expression of y' given by this theorem we have either $y' = 0$ or $y' = y_k - (\delta_{y_k} \bmod y_k)$. The first case leads to the solution $(0, z_{\max})$ and the new m-spacing is $(-y_k, z_{\max} - z_k)$, while in the latter we have a new m-spacing

$$(-\delta_{y_{k+1}}, \delta_{z_{k+1}}) = (-(\delta_{y_k} \bmod y_k), \lfloor \delta_{y_k}/y_k \rfloor \cdot z_k + \delta_{z_k}).$$

A trivial conclusion is that

$$-\frac{\delta_{z_{k+1}}}{\delta_{y_{k+1}}} < -\frac{\delta_{z_k}}{\delta_{y_k}}$$

which justifies the following geometric characterization of the set of minimal solutions of any homogeneous equation in three unknowns.

Corollary 4.0.1. *Let $S = \{s_i\}_{0 \leq i \leq L}$ be the set of the minimal solutions of $a \cdot x = b \cdot y + c \cdot z$, $a, b, c, x, y, z \in \mathbb{N}$ in z-increasing order. Let p_i denote the projection of s_i onto the YZ-plane. The convex hull of the points $\{p_i\}_{0 \leq i \leq L}$ is the polygon whose edges are obtained by linking p_i to p_{i+1}, $0 \leq i < L$ and p_L to p_0. No minimal solution lies strictly inside the polygon. Only minimal solutions lie on the edges $\overline{p_i p_{i+1}}$. All the vertices are minimal solutions.*

```
Input: The positive coefficients a, b and c.
Output: The set S of minimal solutions.

gb := gcd(a,b); gc := gcd(a,c); G := gcd(gb,c);
ymax := a/gb; zmax := a/gc;
dz := gb/G; dy := (c*multiplier(b,a)/G) mod ymax;
y := ymax-dy; z := dz;
S := {(b/gb, ymax, 0), (c/gc, 0, zmax), ((b*y+c*z)/a, y, z)};
while dy > 0 do begin
    while y > dy do begin
        y := y-dy; z := z+dz;
        S := S ∪ {((b*y+c*z)/a, y, z)}
    end;
    f := dy/y; dy := dy mod y; dz := f*z+dz
end
```

Figure 3. The Slopes Algorithm on $a \cdot x = b \cdot y + c \cdot z$

Moreover if s_i, s_j, and s_k are minimal solutions in z-increasing order, then the slope of the line linking p_i to p_j is not less than that of the line linking p_j to p_k.

The Basic Slopes Algorithm, shown in Figure 3, computes directly all the minimal solutions. It starts from the two minimal solutions $(y_{\max}, 0)$, $(y_{\max} - \delta_{y_1}, \delta_{z_1})$ and follows the minimum spacing until there is a (minimal) solution (y_1, z_1) with $y_1 < \delta_{y_1}$. Theorem 4.1 can then be applied resulting a new minimal solution (y_1', z_1') whose difference to (y_1, z_1) defines a new m-spacing. The same procedure is used for this pair and the new m-spacing. The algorithm stops when a minimal solution with null y-component is found.

4.3. WHEN $v \neq 0$

When $v \neq 0$, we already know (by Property 4.4) how to compute the starting solution. Given a minimal solution $s = (x_s, y_s, z_s)$, the question is what m-spacing must be added to obtain the next minimal solution. Now, since z is to be strictly increasing, y must decrease. Thus, the m-spacing must be of the form $(-\delta_y, \delta_z)$ and

(i) $\delta_z > 0$ must be as small as possible, $\delta_y > 0$ must be as large as possible,
(ii) $y_s - \delta_y \geq 0$ and $x_s + (-b \cdot \delta_y + c \cdot \delta_z)/a \geq 0$.

Note that $(-b \cdot \delta_y + c \cdot \delta_z)/a$ is the spacing in x corresponding to $(-\delta_y, \delta_z)$, which we denote in the sequel by δ_x. On the other hand, by Property 4.3, any spacing is of the form

$$(-\delta_y, \delta_z) = (-\delta_{y_1} \cdot k + y_{\max} \cdot t, \delta_{z_1} \cdot k)$$

for some integers k and t. In fact, $k > 0$ if minimal solutions are to be computed in z-increasing order. Equivalently we may write

$$\delta_y \equiv \delta_{y_1} \cdot k \pmod{y_{\max}} \qquad \delta_z = \delta_{z_1} \cdot k. \tag{4.4}$$

Because δ_z and k are in a one-to-one correspondence, we denote the solution $(\delta_{y_i}, k_i, \delta_{z_i})$ of (4.4) corresponding to the ith minimal solution of the congruence $\delta_y \equiv \delta_{y_1} \cdot k$

(mod $y_{\max}$) by $(\delta_{y_i}, \delta_{z_i})$. Moreover, in the sequel, we refer to $(\delta_{y_i}, \delta_{z_i})$ as the minimal solution of (4.4). Equation (4.4) is equivalent to $\delta_{y_1} \cdot k + (y_{\max} - 1) \cdot \delta_y \equiv 0 \pmod{y_{\max}}$, and its solution set can be obtained by Theorem 4.1 in k-strictly increasing (δ_y-strictly decreasing) order. Note that if (δ_{y_i}, k_i) is minimal then $(\delta_{y_i}, \delta_{z_i})$ is minimal, for δ_z strictly increases with k.

Now we are going to show the result we mentioned in section 3: any m-spacing $(-\delta_y, \delta_z)$ required to solve any problem in the family $\{a \cdot x = b \cdot y + c \cdot z + v\}_{v \in \mathbb{Z}}$ is given as

$$(-\delta_y, \delta_z) = t \cdot (0, z_{\max}) + (-\delta'_y, \delta'_z) + r \cdot (-y_{\max}, 0)$$

for some (possibly none) positive minimal solution (δ'_y, δ'_z) of (4.4), and some $t \geq 0$, $k \geq 0$.

The following Lemma says how to compute t. It gives the smallest $t \geq 0$ such that some spacing $\Delta_i = (\delta_{x_i}, -\delta_{y_i}, \delta_{z_i})$ associated to a solution of (4.4), can be added to $(x_s, y_s, z_s) + t \cdot \Delta_L$, where $\Delta_L = (c \cdot z_{\max}/a, 0, z_{\max})$ is the spacing determined by the last (i.e. the Lth) solution of (4.4). "Can be added" meaning that the solution obtained is non-negative and not comparable with s, that is, $0 < \delta_{y_i} \leq y_s$ and $x_s + t \cdot \delta_{x_L} + \delta_{x_i} \geq 0$. Moreover, the Lemma also says which is the first spacing that can be applied.

LEMMA 4.1. *Let $\{(\delta_{y_i}, \delta_{z_i})\}_{0 \leq i \leq L}$ be the set of minimal solutions of (4.4) in δ_z-increasing order. Let $s = (x_s, y_s, z_s)$ be a minimal solution of $a \cdot x = b \cdot y + c \cdot z + v$, for a fixed $v \in \mathbb{Z}$, being $y_s \geq \delta_{y_{L-1}} = min\{\delta_{y_i} \mid \delta_{y_i} \neq 0\}$. Then, the minimum t such that some spacing $\Delta_i = (\delta_{x_i}, -\delta_{y_i}, \delta_{z_i})$ with $i < L$ can be added to $s + t \cdot \Delta_L$ is given by*

$$t_0 = max\left(0, \left\lceil -\frac{x_s + \delta_{x_{L-1}}}{\delta_{x_L}} \right\rceil\right).$$

Moreover, $\Delta_q = (\delta_{x_q}, -\delta_{y_q}, \delta_{z_q})$ is the first spacing that can be added, where

$$q = min\{i \mid \delta_{y_i} \leq y_s, \quad x_s + t_0 \cdot \delta_{x_L} + \delta_{x_i} \geq 0\}$$

PROOF. The sequence $\{\delta_{x_i}\}$ is strictly increasing. Indeed, from $\delta_{y_{i+1}} < \delta_{y_i}$ and $\delta_{z_{i+1}} > \delta_{z_i}$, it follows $a \cdot \delta_{x_{i+1}} = b \cdot (-\delta_{y_{i+1}}) + c \cdot \delta_{z_{i+1}} > b \cdot (-\delta_{y_i}) + c \cdot \delta_{z_i} = a \cdot \delta_{x_i}$. Hence, $\delta_{x_{L-1}} > \delta_{x_i}$ for all $i < L$, and as a result Δ_i can be added to $s + t_0 \cdot \Delta_L$ only if also Δ_{L-1} can be. □

PROPOSITION 4.1. *The m-spacings required to solve the family of problems*

$$\{a \cdot x = b \cdot y + c \cdot z + v\}_{v \in \mathbb{Z}}$$

are obtained from the minimal solutions of (4.4) as follows. Let $\{(\delta_{y_i}, \delta_{z_i})\}_{0 \leq i \leq L}$ be that set of minimal solutions in δ_z-increasing order, and $\Delta_i = (\delta_{x_i}, -\delta_{y_i}, \delta_{z_i})$ be the spacing determined by the ith solution. Let $s = (x_s, y_s, z_s)$ be a minimal solution of $a \cdot x = b \cdot y + c \cdot z + v$, for a fixed $v \in \mathbb{Z}$. If $y_s \geq \delta_{y_{L-1}}$ then

$$t_0 \cdot \Delta_L + \Delta_q + r_0 \cdot \Delta_0$$

is the m-spacing that should be added to s to obtain the next minimal solution, where q and t_0 are as defined in Lemma 4.1, and $r_0 = min\{ \lfloor (y_s - \delta_{y_q})/y_{\max} \rfloor, \lfloor (x_s + t_0 \cdot \delta_{x_L} + \delta_{x_q})/(-\delta_{x_0}) \rfloor \}$. If $y_s < \delta_{y_{L-1}}$, there are no more minimal solutions.

PROOF. If $y_s < \delta_{y_{L-1}}$, there is no minimal solution following s because $\delta_{y_{L-1}}$ is the minimum positive spacing in y between integral solutions.

If $y_s \geq \delta_{y_{L-1}}$ then there exists $k \geq 0$ such that $s + \Delta_{L-1} + k \cdot \Delta_L$ is a positive solution, that is not comparable with s. Thus, s is not the last minimal solution. Then, let $s' = (x', y', z')$ be the minimal solution following immediately s, and let $(\delta_x, -\delta_y, \delta_z)$ be the spacing applied, that is $(\delta_x, -\delta_y, \delta_z) = (x', y', z') - (x_s, y_s, z_s)$. Clearly, we may write $(\delta_x, -\delta_y, \delta_z)$ in terms of $\Delta_L = (\delta_{x_L}, 0, z_{\max})$ as follows

$$(\delta_x, -\delta_y, \delta_z) = \left\lfloor \frac{\delta_z}{z_{\max}} \right\rfloor \cdot (\delta_{x_L}, 0, z_{\max}) + (\delta'_x, -\delta_y, \delta_z \bmod z_{\max})$$

where $\delta'_x = \delta_x - \lfloor \delta_z / z_{\max} \rfloor \cdot \delta_{x_L}$. We are going to show that

$$\lfloor \delta_z / z_{\max} \rfloor = t_0 \tag{4.5}$$

and that

$$(\delta'_x, -\delta_y, \delta_z \bmod z_{\max}) = \Delta_q + r_0 \cdot \Delta_0. \tag{4.6}$$

Because $(\delta_y, \delta_z \bmod z_{\max})$ is a solution of (4.4) there are integers $\alpha_i \geq 0$ such that

$$(\delta'_x, -\delta_y, \delta_z \bmod z_{\max}) = \sum_{i=0}^{L-1} \alpha_i \cdot \Delta_i. \tag{4.7}$$

By definition of t_0, we have $t_0 \neq 0$ only if $\delta_{x_i} < 0$ for all $i < L$. Hence, from (4.7) if $t_0 \neq 0$ then $\delta'_x \leq \delta_{x_{L-1}} < 0$. Moreover $\delta_y \geq \delta_{y_{L-1}}$. Then, as $(\delta'_x, -\delta_y, \delta_z \bmod z_{\max})$ can be added to $s + \lfloor \delta_z / z_{\max} \rfloor \cdot \Delta_L$, so does Δ_{L-1}. This implies that $\lfloor \delta_z / z_{\max} \rfloor \geq t_0$, and thus $\lfloor \delta_z / z_{\max} \rfloor = t_0$. Otherwise, $s + t_0 \cdot \Delta_L + \Delta_{L-1}$ is a solution that contradicts the assumption that s' immediately follows s.

On the other hand, $t_0 = 0$ implies that at least Δ_{L-1} can be added to s. If $\lfloor \delta_z / z_{\max} \rfloor \neq 0 = t_0$ then $s + \Delta_{L-1}$ is a solution that contradicts s' immediately following s for $\delta_{z_{L-1}} < z_{\max}$. Therefore (4.5) holds.

Now we prove (4.6) using the fact that (4.5) holds. From (4.5) and the definition of q given by Lemma 4.1 it follows that $\delta_z \bmod z_{\max} = \delta_{z_q}$. Otherwise, there would be a positive solution whose z component, say z'', verifies $z_s < z'' = z_s + t_0 \cdot z_{\max} + \delta_{z_q} < z'$ contradicting the assumption that s' immediately follows s.

As a result, we may write

$$(\delta'_x, -\delta_y, \delta_z \bmod z_{\max}) = (\delta_{x_q}, -\delta_{y_q}, \delta_{z_q}) + (\delta'_x - \delta_{x_q}, -\delta_y + \delta_{y_q}, 0),$$

. Replacing δ'_x by its definition, and as $(-\delta_y + \delta_{y_q}, 0)$ must be of the form $r \cdot (y_{\max}, 0)$, it is not difficult now to conclude that (4.6) holds. □

It may be worth noting that this result is just the one mentioned previously because we have not required $q \neq 0$ but just $q < L$. If $q = 0$, the m-spacing results from both (and only) the positive minimal solutions of (4.4). If $q \neq 0$, then $(\delta_{y_q}, \delta_{z_q})$ is a positive minimal solution.

When $v \geq 0$, the definition of q can be simplified. Remark that in this case, $a \cdot x = b \cdot (y_s - \delta_{y_i}) + c \cdot (z_s + \delta_{z_i}) + v \geq 0$, if $y_s \geq \delta_{y_i}$. Besides, note that in particular the following corollary justifies that all what has been said about the geometric characterization of the set of minimal solutions when $v = 0$ (see Corollary 4.0.1 in section 4.2) holds in fact when $v \geq 0$. It may not be true however when $v < 0$ as in the example of Figure 1 above.

COROLLARY 4.1.1. *The m-spacings required to solve the family of problems*

$$\{a \cdot x = b \cdot y + c \cdot z + v\}_{v \geq 0}$$

are in a one-to-one correspondence with the positive *minimal solutions of (4.4). Let* $\{(\delta_{y_i}, \delta_{z_i})\}_i$ *be that set of minimal solutions in* δ_z*-increasing order. Given* $s = (x_s, y_s, z_s)$, *a minimal solution of* $a \cdot x = b \cdot y + c \cdot z + v$, *for a fixed* v, *let* $q = min\{i \mid \delta_{y_i} \leq y_s\}$. *The next minimal solution is obtained by adding* $\Delta_q = (\delta_{x_q}, -\delta_{y_q}, \delta_{z_q})$ *to* s, *provided* $\delta_{y_q} \neq 0$. *Otherwise, there are no more minimal solutions.*

Even when $v < 0$, may Proposition 4.1 be simplified. Basically, it can be proved that provided $L \geq 2$, either $\Delta_L = (\delta_{x_L}, 0, z_{\max})$ or $\Delta_0 = (\delta_{x_0}, -y_{\max}, 0)$ has never to be added, and hence the formula given in Proposition 4.1 may be simplified.

COROLLARY 4.1.2. *Under the conditions of Proposition 4.1, and provided* $L \geq 2$,

(i) if $\delta_{x_i} \geq 0$ *for some* $i < L$ *then* Δ_L *is never added, that is* $t_0 = 0$ *for all* s;
(ii) if $\delta_{x_1} < 0$ *then* Δ_0 *is never added, that is* $q \neq 0$ *and* $r_0 = 0$ *for all* s.

The first statement follows trivially from the definition of t_0. As to the second, note that if the solutions were computed in y-increasing order the spacings would just be the symmetrics of the ones we are using now. In that case $-\Delta_0$ would have the same role as Δ_L is having now, and therefore the second statement is just a reformulation of the first.

5. The general case

In this section we describe in more detail one of the possible ways of using the previous algorithms in solving the general case of an equation

$$\sum_i^N a_i \cdot x_i = \sum_j^M b_j \cdot y_j \qquad a_i, b_j, x_i, y_j \in \mathbb{N}. \tag{5.1}$$

5.1. SUB-PROBLEMS

It would be possible to select three unknowns (not all on the same side of the equation) and to reduce (5.1) to an (non-homogeneous) equation in three unknowns by enumerating all the others from zero. There are some advantages of considering instead different sub-problems by giving some of the unknowns the value zero and forcing all the others to have values ≥ 1.

Hence, for each sub-problem we find its minimal *positive* solutions by enumerating all but three unknowns from 1 and using the algorithms described above. Each such solution corresponds to a solution (obtained by "inserting" a zero for each null unknown) of (5.1) that we refer to as a *candidate solution*. In general, a candidate solution must be compared with other solutions of (5.1) to check whether it is minimal or not.

There are two advantages in considering sub-problems in this way:

(i) for each sub-problem it is easy to choose the more appropriate algorithm, by considering the numbers of (positive) unknowns in each side;

(ii) it becomes possible to separate sets of sub-problems giving origin to solutions of (5.1) which are not comparable (for instance, solutions with the same number of null components but with different null components are certainly not comparable). This decreases the number of comparisons that must be made and leads to an algorithm for solving the general equation which is highly suited for parallel implementation. This latter issue is currently being addressed by our team.

We choose a particular order for solving all the sub-problems so that the solution set of (5.1) is computed monotonically, i.e. each candidate solution is minimal if it is not less than any minimal solution already in the set.

5.2. ENUMERATION AND BOUNDS

We have used a lexicographic enumeration procedure with the property that each valuation is always smaller or not comparable (in a component-wise ordering) with any valuation that is obtained later. We profit from this when solving each sub-problem in order to obtain its solution set in a monotonic way.

The enumeration of values for the unknowns is controlled by using the bounds described in Huet (1978) and Lambert (1987), and also by bounds dynamically determined from *quasi-Boolean* solutions, i.e., solutions whose non-null unknowns all but one have value 1. On the other hand, solutions whose non-null unknowns are all 1 (*Boolean* solutions) are used to avoid dealing with sub-problems for which no minimal solution will be generated.

If the problem being solved has a single coefficient in one side of the equation then the minimality of a solution can be checked by inspecting only the components in the other side. Hence we extend the notions of Boolean and quasi-Boolean solutions in this case by disregarding the value of the isolated component.

From several tests with different versions of our algorithms we may conclude that when dealing with problems having small coefficients (≤ 3) the use of the Lambert's bounds and of the information from quasi-Boolean and Boolean solutions is usually a cause of inefficiency.

6. Implementation

We have a sequential implemention in C of our method, using streamlined forms of the algorithms presented before. For a fast access to the information concerning Boolean solutions and for speeding up the comparison between minimal solutions and candidate solutions, we identify each configuration of non-null components by a bit mask. We also keep, for each sub-problem, a list (built using lazy-evaluation) of the minimal solutions that should be compared with candidate solutions. In one of the versions of the implementation the coefficients are internally ordered so that they are decreasing in the left-hand side and increasing in the right-hand side of the equation, this in most cases speeding up the algorithm.

The analysis of profiling data obtained with the Unix `profile` utility, suggested a couple of changes that increased the overall efficiency.

Two parallel implementations, for shared-memory and distributed-memory architectures, are presently being developed. The former is based on the p-System (Lopes & Silva, 1994), that provides a parallel extension to the C programming language, while the latter uses PVM (Geist *et al.*, 1993).

7. Efficiency comparisons

We made an extensive comparison of our sequential implementation with two other algorithms for solving a single equation[†], namely, those described in Elliott (1903) and Clausen and Fortenbacher (1989). The latter (to which we will refer by the initials "C&F") was generally believed to be the fastest method available, according to the results of comparisons (with algorithms by Huet, but improved with the bounds of Lambert, by Fortenbacher, and by Lankford) in Clausen and Fortenbacher (1989). The results below show clearly that our methods are faster than the other two, and therefore, at present, the fastest available, for a certain class of equations. Typically these equations will have a small number of unknowns in one side, and the maximum value for the coefficients will be greater than 3.

All the algorithms under comparison were implemented in the same language (C, using the standard Unix cc with optimization flag `-O3`) and the same machine, a Sun Sparc-Center with 8 processors (that makes possible running much larger problems than those that we used for comparisons in previous papers). Execution times below are CPU-times obtained by making calls to the appropriate Unix system routines. For some problems and in order to have measurable times, the computation of the solutions was repeated 100 times and the average CPU-time was taken.

We adopted a statistical approach in making the comparisons. A set of problems were generated at random (by a program using the standard UNIX `rand` routine) and submitted to the different programs. The problems were classified according to the number of unknowns in each side of the equation and the maximum value for the coefficients. The behaviour of the algorithms for problems in the same class is very irregular, as the following examples show:

	Exec. times (msec.)		Min. sols.
Coefficients	C&F	Slopes	
(1 2 3)(7 5 6 4)	2.7	2.8	38
(1 7 4)(2 5 6 3)	11.3	4.5	79
(107 57)(77 101 46 55 63)	13.4e3	21.3e3	1040
(14 107)(89 95 93 22 31)	120.2e3	69.3e3	2286

The fact that for a set of 1230 equations (evenly distributed in 138 different classes), the standard deviations of the execution times are 6.7 (C&F) and 19 (Slopes) times bigger than the respective averages also shows that the behaviour of the algorithms is indeed unpredictable. Dividing the execution time by the number of solutions does not help in finding a more regular data set. Note that this is mainly due to the big changes, in both the number and the spatial distribution of solutions, caused by small differences in the number or the values of the coefficients. This behaviour makes any comparison based on averages of execution times completely meaningless.

It is also worth noting that when the number of coefficients and its maximum increase, "pathological" cases are likely to occur either with too many minimal solutions (e.g.

[†] Algorithms for solving systems of equations are much more inefficient. However, it would be unfair to compare algorithms that address different kinds of problems.

(654)(7 8 9 23 24 26) has 197,942 and is solved in 7.7 hours by Slopes), or in which the algorithms spend huge amounts of time — e.g. Slopes needs about 6.2 days to find the 1543 solutions of (503 324 84)(152 84 144 424 270). This imposes practical limitations on which and how many tests can be done. For instance, as Elliott's algorithm becomes too slow for harder problems, the set of problems used in comparing it to Slopes is a subset of the one used in comparing C&F and Slopes.

If comparing averages of execution times has no meaning, a direct comparison of execution times for a small set of equations is obviously also completely meaningless. This discredits not only the comparisons in our previous papers (based on a set of 45 equations), but also those in papers by other authors (namely Clausen & Fortenbacher, 1989) where times for only 8 equations (in different classes) are used.

We resorted instead to another technique: we compared the number of cases in which each algorithm is better than the other one for a small sample of (normally) 10 equations in each class. We only consider as meaningful the results for classes having 80% or more cases favouring one of the algorithms. We are allowed to do so by the following reason: the probability of having a repartition of 80% cases in favour of one algorithm when a sample of 10 equations is taken assuming that the two algorithms have the same speed is very low. This probability can be computed as

$$p_s(8|0.5) = \frac{\binom{0.5P}{8}\binom{0.5P}{2}}{\binom{P}{10}}$$

where P is the number of possible problems in the class. This number is less than (because of simmetries)

$$\binom{N+A-1}{N}\binom{M+A-1}{M}$$

where N and M are the number of unknowns in the left- and right-hand sides of the equation and A is the maximum of the coefficients. The following table gives an idea of the sizes of the "populations" for different classes.

	A	5	13	1021
N	M			
1	2	75	1183	532e6
1	9	3575	3.8e6	3.5e24
5	5	15876	38.3e6	87.2e24

The probability $p_s(8|0.5)$ takes the values 4.3% for $P = 1000$ and 4.4% for values of P up to 100e24 (these computations were made using infinite precision software[†]). When 9 in 10, or 10 in 10 cases favour one algorithm the probabilities of the same speed are much lower: $p_s(9|0.5)$ ranges from 0.95% for $P = 1000$ to 0.98% for $P = 100e24$, while $p_s(10|0.5)$ ranges from 0.09% to 0.1% for the same values of P. We are then in a position to reject the hypothesis that the two algorithms have equal speed in these extreme situations (i.e.

[†] Namely, GAP, a system for computational group theory developed at RTWH Aachen, and the UNIX standard bc.

8 or more cases for one of the algorithms), the results being significant at the 5% level. However we are not allowed any conclusion when the situation is not so clear-cut.

We are now ready to present the results of the comparisons. In the tables below for *each class (fixed by N, M and A, respectively the numbers of unknowns in each side of* the equation and the maximum coefficient) we list an entry $k_1 : k_2$ where k_1 (k_2) is the number of times the first (the second) algorithm was faster than the second (the first). When there were draws, their number was equally distributed over k_1 and k_2 (this is common practice in applied statistics).

We start by the comparison with Elliott's algorithm which we implemented from scratch using the descriptions in Elliott (1903) and MacMahon (1918) and also Stanley (1973). The latter has a description that is said to be "sketchy" by the author and that omits conditions that are essential for termination; the termination proof is based on the strict monotonicity of a function which is not always strictly monotonic even under the conditions given by Elliott. From the basic implementation an enhanced one was made by introducing filtering by Boolean solutions and taking advantage of the fact (noticed by Eric Domenjoud — personal communication, 1992) that about one third of the work done by the algorithm is superfluous. More details on this algorithm and its version for solving systems of equations may be found in Filgueiras and Tomás (1992*c*). The results of the comparison of Slopes and the enhanced version of Elliott's are in Table ??, for which 1121 problems were considered.

Table 4. Comparing Elliott's and Slopes algorithms

	A	2	3	5	13	29	39	107
N	M							
1	2	3 : 2	5 : 3	4 : 5	4 : 5	1 : 9	2 : 8	1 : 9
1	3	3 : 1	5 : 4	5 : 4	2 : 7	1 : 9	0 : 10	2 : 8
1	4	4 : 4	4 : 6	2 : 7	0 : 10	0 : 9	1 : 9	1 : 9
1	5	6 : 2	4 : 6	2 : 8	0 : 10	2 : 8	0 : 9	0 : 10
1	6	4 : 5	3 : 6	4 : 6	0 : 10	1 : 9	0 : 10	0 : 10
1	7	6 : 4	2 : 8	1 : 8	1 : 9	0 : 10	0 : 9	
1	8	5 : 4	4 : 6	3 : 7	0 : 10			
1	9	5 : 5	0 : 10	1 : 8				
2	2	4 : 2	5 : 4	1 : 8	1 : 9	2 : 8	0 : 10	2 : 7
2	3	3 : 5	2 : 7	0 : 9	0 : 10	1 : 9	1 : 8	1 : 9
2	4	5 : 5	1 : 9	0 : 10	0 : 10	0 : 10	0 : 10	2 : 8
2	5	5 : 4	0 : 9	0 : 10	0 : 10	0 : 10	0 : 10	0 : 10
2	6	*6 : 3*	*0 : 10*	*0 : 10*	*0 : 10*			
2	7	2 : 8	0 : 10	0 : 10	0 : 10			
2	8	4 : 5	0 : 10	0 : 10				
3	3	3 : 5	0 : 9	0 : 10	0 : 10	0 : 10	0 : 10	3 : 7
3	4	0 : 10	0 : 9	0 : 10	0 : 10	0 : 10	0 : 10	2 : 8
3	5	0 : 10	0 : 10	0 : 10	0 : 10			
3	6	0 : 10	0 : 10	0 : 10				
4	4	0 : 8	0 : 10	0 : 10	0 : 10			
4	5	0 : 10	0 : 10	0 : 10				

It can be seen that in 82 out of 115 classes we are allowed to say that Slopes is faster. These 82 classes correspond to the harder problems, with more unknowns and/or bigger coefficients. On the other hand there is no class for which we can say that Elliott's is faster.

In order to compare C&F and Slopes, we translated the Pascal program given in Clausen and Fortenbacher (1989) to C and changed the graph initialization procedure so that the maximum coefficient in the equation being solved is taken into account (otherwise a constant time is lost in the initialization phase). The results of a comparison with Slopes for 1564 problems are given in Table 5.

Table 5. Comparing C&F to Slopes

	A	2	3	5	13	29	39	107	503	1021
N	M									
1	2	4 : 2	4 : 4	1 : 9	0 : 9	0 : 10	0 : 10	0 : 10	0 : 10	0 : 10
1	3	3 : 1	4 : 5	2 : 7	0 : 10	0 : 10	0 : 10	0 : 9	0 : 10	0 : 10
1	4	6 : 3	5 : 5	1 : 8	0 : 10	0 : 10	0 : 10	0 : 10	0 : 10	0 : 10
1	5	8 : 1	5 : 5	2 : 8	0 : 10	0 : 10	0 : 10	0 : 10	0 : 10	0 : 10
1	6	7 : 2	8 : 2	4 : 6	0 : 10	0 : 9	0 : 10	0 : 10	0 : 10	
1	7	9 : 1	9 : 1	1 : 8	1 : 9	0 : 9	0 : 9	0 : 9	0 : 10	
1	8	9 : 1	9 : 1	4 : 5	0 : 10	1 : 8	0 : 10	0 : 8		
1	9	10 : 0	8 : 2	6 : 3	0 : 10	0 : 10	0 : 10			
2	2	4 : 1	5 : 4	0 : 8	0 : 10	0 : 10	0 : 10	0 : 10	0 : 10	0 : 10
2	3	6 : 2	5 : 4	0 : 9	0 : 10	0 : 10	0 : 9	0 : 10	3 : 7	4 : 6
2	4	9 : 1	8 : 2	0 : 9	0 : 10	0 : 10	2 : 8	4 : 6	7 : 3	10 : 0
2	5	9 : 0	7 : 3	1 : 8	0 : 10	2 : 8	0 : 10	5 : 5	9 : 0	
2	6	10 : 0	10 : 0	1 : 8	0 : 10	2 : 8	3 : 7	8 : 2		
2	7	10 : 0	10 : 0	2 : 7	1 : 9	2 : 8	4 : 6			
2	8	9 : 0	10 : 0	5 : 5	0 : 9	4 : 6	2 : 8			
3	3	8 : 0	6 : 3	0 : 9	0 : 10	0 : 10	0 : 10	3 : 7	10 : 0	
3	4	10 : 0	8 : 2	0 : 10	0 : 10	1 : 8	2 : 8	7 : 3		
3	5	10 : 0	10 : 0	0 : 10	0 : 10	0 : 10	3 : 6	9 : 1		
3	6	10 : 0	10 : 0	1 : 9	0 : 10	4 : 6	3 : 7			
4	4	8 : 0	10 : 0	1 : 9	0 : 10	3 : 7	5 : 5	6 : 3		
4	5	10 : 0	9 : 1	3 : 7	0 : 10	1 : 9	5 : 5			

It can be seen that in 88 out of 160 classes we are allowed to say that Slopes is faster, while in 33 classes we are allowed to conclude the opposite. It is clear that most of these 33 classes (more precisely, 28) correspond to coefficients of at most 3. This is in line with our previous statements about the unsuitability of our algorithm for this kind of problem — in fact, the time Slopes spends in computing all the gcd's it needs will normally be greater than the time taken by any brute-force enumeration technique to solve the problem.

It also seems to be the case that when the coefficients become large (≥ 107) and the numbers of unknowns in each side are not too different, Slopes is slower than C&F. This is easily explained by the use of enumeration by Slopes and the lack of good bounds for the values of the unknowns. In the Rectangles algorithm we try to circumvent this problem by computing bounds dynamically by a technique that has relations to those of constraint propagation.

8. Conclusions

We presented a new algorithm, Slopes, for finding the basis of minimal solutions of linear Diophantine equations on the natural numbers. It combines an enumeration process for some of the components with linear algorithms for solving equations in three unknowns. These algorithms are based on a geometric characterization of the set of minimal solutions, which we are trying to extend to the case of equations with more unknowns.

The fact that at each stage Slopes splits the problem into several independent subproblems makes it highly suitable for parallel implementation. This is being explored both for shared- and distributed-memory architectures.

The results presented in the previous section clearly show that Slopes is the fastest algorithm when the equation has coefficients greater than 3 and the number of unknowns in one side is small.

Other points that we are addressing or will address in the near future are:

(i) the possible application of the ideas underlying our methods in obtaining methods for solving systems. In particular, it can be shown that the algorithm for solving equations in three unknowns applies directly to systems of $m \times (m+2)$ equations of rank m since they are equivalent to a system of congruences;

(ii) further study of the Elliott algorithm which has a simple formulation and seems to have potentialities not yet explored;

(iii) integration of our methods in Constraint Logic Programming systems dealing with finite domains and naturals.

Acknowledgements

We would like to express our gratitude to the referees of previous versions of this paper for their constructive comments. Discussions on this subject with Alexander Herold, Claude Kirchner, Eric Domenjoud, Evelyne Contejean, Jean-Pierre Jouannaud, and Loic Pottier were also very helpful. These discussions as well as our visits to LRI in Orsay were partly funded by the Commission of the European Communities through the HCM SOL Network. Our thanks go also to Margarida Brito who suggested the statistical foundations for our comparison method.

References

Abdulrab, H. & Pécuchet, J.-P. (1986). Solving systems of linear Diophantine equations and word equations. In (N. Dershowitz, ed.) *Proceedings of the 3rd International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, 355, Springer-Verlag, pp. 530–532.

Bachem, A. & Kannan, R. (1984). Lattices and The Basis Reduction Algorithm. Tech. Rep. CMU-CS-84-11, Carnegie-Mellon University.

Chou, T. J. & Collins, G. E. (1982). Algorithms for the solution of systems of linear Diophantine equations. *SIAM J. Comput.*, **11**(4), 687–708.

Clausen, M. & Fortenbacher, A. (1989). Efficient solution of linear Diophantine equations. *J. Symbolic Computation* **8**, 201–216.

Contejean, E. (1993). Solving linear Diophantine constraints incrementally. In (D. S. Warren, ed.) *Proceedings of the 10th International Conference on Logic Programming*, MIT Press, pp. 532–549.

Domenjoud, E. (1991). *Outils pour la Déduction Automatique dans les Théories Associatives-Commutatives*. Thése de doctorat, Université de Nancy I.

Elliott, E. B. (1903). On linear homogenous Diophantine equations. *Quart. J. Pure Appl. Math.* **34**, 348–377.

Filgueiras, M. & Tomás, A. P. (1992*a*). A Congruence-based Method with Slope Information for Solving Linear Constraints over Natural Numbers. Presented at the Workshop on Constraint Logic Programming '92, Marseille.

Filgueiras, M. & Tomás, A. P. (1992*b*).*Solving Linear Diophantine Equations: The Slopes Algorithm*. Centro de Informática da Universidade do Porto.

Filgueiras, M. & Tomás, A. P. (1992*c*).*A Note on the Implementation of the MacMahon-Elliott Algorithm*. Centro de Informática da Universidade do Porto.

Filgueiras, M. & Tomás, A. P. (1993). Fast Methods for Solving Linear Diophantine Equations. In (M. Filgueiras, L. Damas, eds.) *Progress in Artificial Intelligence — 6th Portuguese Conference on Artificial Intelligence*, Lecture Notes in Artificial Intelligence 727, Springer-Verlag, pp. 297–306.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. & Sunderam, V. (1993). PVM3 User's Guide and Reference Manual. Oak Ridge National Laboratory.

Guckenbiehl, T. & Herold, A. (1985). Solving Linear Diophantine Equations. Memo SEKI-85-IV-KL, Universität Kaiserslautern.

Huet, G. (1978). An algorithm to generate the basis of solutions to homogeneous linear Diophantine equations. *Information Processing Letters*, 7(3). AU PAGES?

Jaffar, J., Lassez, J.-L. & Maher, M. (1986). Logic Programming language scheme. In (D. DeGroot and G. Lindstrom, eds.), *Logic Programming: Functions, Relations, and Equations*, Prentice-Hall.

Jaffar, J. & Lassez, J.-L. (1987). Constraint Logic Programming. In *Proceedings of the 14th POPL Conference*.

Lambert, J.-L. (1987). Une borne pour les générateurs des solutions entières positives d'une équation diophantienne linéaire. *Comptes Rendus de l'Académie des Sciences de Paris*, t. 305, série I, 39–40.

Lopes, L. B. & Silva, F. (1994). Scheduling algorithms performance with the p-System parallel programming environment. To appear in *Proceedings of the 1994 Parallel Architectures and Languages Europe (PARLE'94)*, Lecture Notes in Computer Science, Springer-Verlag.

MacMahon, P. (1918). *Combinatory Anclysis*, 2. Chelsea Publishing Co..

Stanley, R. (1973). Linear homogeneous Diophantine equations and magic labelings of graphs. *Duke Math. J.*, **40**, 607–632.

Stickel, M. E. (1981). A unification algorithm for associative-commutative functions. *JACM*, **28**(3).

Tomás, A. P. & Filgueiras, M. (1991*a*). A new method for solving linear constraints on the natural numbers. In (P. Barahona, L. Moniz Pereira, A. Porto, eds.) *Proceedings of the 5th Portuguese Conference on Artificial Intelligence*, Lecture Notes in Artificial Intelligence 541, Springer-Verlag, pp. 30–44.

Tomás, A. P., & Filgueiras, M. (1991*b*). *A Congruence-based Method for Finding the Basis of Solutions to Linear Diophantine Equations*. Centro de Informática da Universidade do Porto.