Fast Methods for Solving Linear Diophantine Equations

Miguel Filgueiras and Ana Paula Tomás

LIACC, Universidade do Porto, R. do Campo Alegre 823, 4100 Porto, Portugal
email: mig@ncc.up.pt, apt@ncc.up.pt

Abstract. We present some recent results from our research on methods for finding the minimal solutions to linear Diophantine equations over the naturals. We give an overview of a family of methods we developed and describe two of them, called Slopes algorithm and Rectangles algorithm. From empirical evidence obtained by directly comparing our methods with others, and which is partly presented here, we are convinced that ours are the fastest known to date when the equation coefficients are not too small (ie., greater than 2 or 3).

1 Introduction

We present some recent results from our research on methods for finding the minimal solutions to linear Diophantine equations over the naturals. Such methods will be useful in the implementation of programming languages or systems that solve constraints on the naturals or on finite domains, as well as in term rewriting systems, namely in unification algorithms of terms with associative-commutative function symbols (AC-unification).

We have developed a family of methods all based on the idea that the minimal solutions of an equation may be obtained by using very efficient algorithms that solve equations having a small number of unknowns in combination with an enumeration procedure for the values of the other unknowns. Previously we have reported on methods centered on solving equations on 3 unknowns (Tomás and Filgueiras, 1991a; Tomás and Filgueiras, 1991b; Filgueiras and Tomás, 1992a; Filgueiras and Tomás, 1992b). In this paper we give an overview of the family of methods and describe two of them, called Slopes Algorithm and Rectangles Algorithm. From empirical evidence obtained by directly comparing our methods with others we are convinced that ours are the fastest known to date when the equation coefficients are not too small (ie., greater than 2 or 3).

2 A Family of Methods

Several algorithms have been put forth to find the basis of non-negative solutions to linear Diophantine equations or systems of such equations. The oldest method we know of is due to Elliott (1903) in the context of Number Theory. With the exception of Elliott's and of our methods, all others emerged from

work on AC-unification algorithms — see (Domenjoud, 1991) for an overview. The one by Huet (1978) uses lexicographic enumeration (a technique originating from dynamic programming) of values for the unknowns and filters out non-minimal solutions by comparison with the minimal solutions already obtained. Our family of methods may be seen as a refinement of Huet's algorithm: instead of enumerating values for all the unknowns we choose a small number of them (tipically 3 or 4) and use enumeration[1] for the other unknowns. For each valuation of the enumerated unknowns we have to solve an equation in a small number of unknowns and this can be done very efficiently. Moreover, we use some solutions to obtain finer bounds for the unknowns than those proposed by Huet (1978) and Lambert (1987).

In more formal terms let us consider the Equation (1).

$$\sum_{i}^{N} a_i \cdot x_i = \sum_{j}^{M} b_j \cdot y_j \qquad a_i, b_j, x_i, y_j \in \mathbb{N} \tag{1}$$

By selecting, for instance, x_1, y_1, and y_2 we get

$$a_1 \cdot x_1 + \sum_{2}^{N} a_i \cdot x_i = b_1 \cdot y_1 + b_2 \cdot y_2 + \sum_{3}^{M} b_j \cdot y_j$$

which, for each particular valuation of the unknowns in the sums, can be written as

$$a \cdot x = b \cdot y + c \cdot z + v \qquad v \in \mathbb{Z} \tag{2}$$

to which very fast methods can be applied.

The methods in the family differ in the number of selected unknowns and/or in the algorithms used for solving the reduced equations.

The problem of solving (1) is split into several sub-problems according to the number and pattern of non-null unknowns. We define the *order* of a sub-problem as being the number of non-null unknowns in it. We first solve all order 2 sub-problems, then all the problems with only one non-null unknown in one side of the equation and this in increasing order, and finally all the others in increasing order. In this way and by using a suitable enumeration procedure minimal solutions are generated incrementally: a candidate solution may be accepted as minimal if it is not greater than a minimal solution already obtained. The fact that the solutions of sub-problems of the same order for different patterns of non-null unknowns are never comparable may be used for a highly parallel implementation (although the implementations we have are sequential).

The enumeration of values for the unknowns is controlled by the bounds described in (Huet, 1978; Lambert, 1987), and also by bounds dynamically determined from *quasi-Boolean* solutions, ie., 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. Furthermore, in the Rectangles

[1] Not a lexicographic enumeration, see Section 5.

Algorithm not only Boolean and quasi-Boolean solutions but all the solutions computed so far are used to dynamically derive bounds.

In the next sections the basic algorithms that are at the core of the Slopes Algorithm and the Rectangles Algorithm are presented.

3 The Basic Slopes Algorithm

The Basic Slopes Algorithm, which solves directly Equation (2) for $v \geq 0$, is an improvement of the basic Congruence-Based Algorithm — CBA for short (Tomás and Filgueiras, 1991a). It results from the identification of all possible *minimal spacings*[2] which is a consequence of the following theorem.

Theorem 1. *Let $s_2 = (y_k, z_k)$ and $s_1 = (y_k + dy_k, z_k - dz_k)$ be two minimal solutions of (2), with $v = 0$, such that $0 < y_k < dy_k$ and there is no minimal solution with y-component in $]y_k, y_k + dy_k[$. Then, taking $F_k = \lceil dy_k / y_k \rceil$, the minimal solution with maximum y-component less than y_k is*

$$(y', z') = (F_k \cdot y_k - dy_k, F_k \cdot z_k + dz_k)$$

No superfluous (ie., non-minimal) candidate solution is generated. The algorithm starts from the two minimal solutions $(y_{\max}, 0)$, $(y_{\max} - dy_1, dz_1)$, where (Tomás and Filgueiras, 1991b) $y_{\max} = a/\gcd(a, b)$, the *minimum spacing* (which depends only on a, b, and c) is

$$dy_1 = (m_{\mathrm{b}} \cdot \frac{c}{\gcd(a, b, c)}) \bmod y_{\max} \qquad dz_1 = \frac{\gcd(a, b)}{\gcd(a, b, c)}$$

and m_{b} is an integer such that $\gcd(a, b) = m_{\mathrm{a}} \cdot a + m_{\mathrm{b}} \cdot b$. The minimum slope line is followed until there is a (minimal) solution (y_1, z_1) with $y_1 < dy_1$. Theorem 1 can then be applied resulting a new minimal solution (y_1', z_1') which, together with (y_1, z_1), defines a new spacing for minimal solutions and a new slope. The same procedure is used for this pair and the new spacing. The algorithm stops when a minimal solution with null y-component is found.

When $v > 0$ the starting solution is given by the following formula[3]

$$z_0 = \frac{-v \cdot M_{\mathrm{c}}}{\gcd(a, b, c)} \bmod dz_1 \qquad y_0 = \frac{(-v - z_0 \cdot c) \cdot m_{\mathrm{b}}}{\gcd(a, b)} \bmod y_{\max}$$

where $b \cdot M_{\mathrm{b}} + c \cdot M_{\mathrm{c}} + a \cdot M_{\mathrm{a}} = \gcd(a, b, c)$ and $y_{\max}$ and m_{b} are defined as above (Tomás and Filgueiras, 1991b). Now the problem is to find the minimal spacing to be applied next.

From CBA we have that any minimal spacing is of the form

$$(dy_1 \cdot k - y_{\max} \cdot t, k \cdot dz_1)$$

[2] A minimal spacing is the difference between two consecutive (in y-decreasing order) minimal solutions of (2), $v \geq 0$.

[3] $\gcd(a, b, c)$ divides v, otherwise no solution exists.

with minimum k. So, when starting from a minimal solution with $y = y_S$ the minimal spacing is the solution of the *inequality problem*: minimize k, subject to $0 < k \leq y_{max}$, $0 < y_S < y_{max}$, and

$$dy_1 \cdot k < y_S \pmod{y_{max}} \tag{3}$$

in the sense of $0 \leq dy_1 \cdot k - y_{max} \cdot t < y_S$, for some $t \geq 0$ with $k > 0$, $y_S > 0$ and $min\ \ t,\ k$. This can be written as $y_{max} \cdot t \leq dy_1 \cdot k < y_S + y_{max} \cdot t$. The question turns out to be that of finding just one multiple of dy_1 in the interval. The problem may be solved by using congruences: find the set of minimal solutions of

$$(y_{max} \bmod dy_1) \cdot t + y_S \equiv 0 \pmod{dy_1}$$

or, equivalently, find the set of minimal solutions of

$$dy_1 \cdot k - y_S \equiv 0 \pmod{y_{max}} \tag{4}$$

Equation (4) is equivalent to $dy_1 \cdot k + (y_{max} - 1) \cdot y_S \equiv 0 \pmod{y_{max}}$, and its solution set can be obtained by Theorem 1 in decreasing y-component order. If (k_i, y_i) is the i^{th} solution in the set, then k_i solves (3) for $y_i < y_S \leq y_{i-1}$.

Since $dy_1 \cdot k_i \equiv y_i \pmod{y_{max}}$, the spacing sought is $(-y_i, dz_1 \cdot k_i)$. An implementation detail that is worth noting is that we are interested in solving a family of problems like that of Equation (2) only differing in v. They determine the same inequality problem, which we solve only once.

The method for solving Equation (2) with $v < 0$ is the same as in CBA until a minimal solution with $y < y_{max}$ is found, after what the above method is applied.

4 The Basic Rectangles Algorithm

This new algorithm results from an effort to speed up the enumeration procedure and the tests for minimality. The idea is to solve (5)

$$a \cdot x = b \cdot y + c \cdot z + d \cdot w + v, \quad v \geq 0 \tag{5}$$

by taking x and w as dependent unknowns and selecting dynamically either y or z as a free unknown to have its values enumerated. Which of y or z is selected depends on a worst-case estimation of the number of values remaining to be given.

Suppose that z was the one selected, and that the search space was

$$]y_0, y_M[\times]z_0, z_M[\times]w_0, w_M[\ .$$

For a given value z_1 an equation in 3 unknowns is solved as in the Slopes Algorithm and minimal solutions are computed in w-decreasing order, with $y_0 < y < y_M$. Any solution (not minimal) with $w \leq w_0$ establishes a new proper upper bound y_M^1 for y, and may narrow substantially the search space

rectangle. In this case, the choice of the free variable is reviewed, but now for a smaller rectangle $]y_0, y^1_{\mathrm{M}}[\times]z_1, z_{\mathrm{M}}[$.

In order to shorten the search, this new algorithm makes a better use of the minimal solutions computed at each step, and takes advantage of the empirical evidence that minimal solutions occur close to the axes. One of the main ideas has been that of identifying what minimal solutions are relevant in rejecting candidate solutions, and what upper bounds, for the values of y, z, and w, they introduce.

Relevant solutions are kept in an ordered list, so-called *filter*, that is updated whenever a minimal solution is found. Actually, not one but two filters are kept because of the possible selection of either y or z as free variables. During the process, filters are adjusted to the current rectangle, providing a dynamical upper bound for w. When this upper bound becomes w_0 or less, the search is stopped. Also, as filters are kept ordered, only a small segment of the active filter must actually be used to test each candidate.

The Rectangles Algorithm generates much less candidate solutions that are not minimal, and is to some extent less sensitive to the order of coefficients than the Slopes Algorithm. Obviously, it is not symmetric since dynamic selection of the free variable only applies to z and y. Proof of completeness of this method follows trivially from the completeness of the Slopes Algorithm.

5 Implementation

For a fast access to the information concerning Boolean solutions and for speeding up the comparison between minimal solutions and candidates, we attach to each configuration of non-zero components a bit mask. Each minimal solution is stored along with the mask for the current subproblem. We keep track (in a lazy way) of the minimal solutions having a mask compatible with the current one, in order not to loose time searching for them.

The enumeration algorithm implemented is iterative and increases the value of one unknown at a time, until this value reaches its upper bound. Components are incremented from right to left, and the algorithm ensures that the values of the unknowns placed to the right of the one being incremented are 1.

In the current implementation, the Rectangles Algorithm applies to Equation (1), when $N = 1$. The values of all but three unknowns in the right-hand side are enumerated. Bounds for this enumeration are derived from minimal solutions already computed. For each valuation, bounds for the other three unknowns are also derived and used in solving the subproblem of the type (5) associated to it. How to speed-up the derivation of bounds and the construction of filters has been a major concern of our research. We have tried out several approaches with distinct levels of lazy evaluation of bounds and filters. This was motivated by empirical evidence that many candidates are filtered out just by the bounds of the search space, and that normally, when changing a valuation by increasing the value of one unknown, a decrease in these bounds does not happen.

6 Comparisons

When presenting CBA (Tomás and Filgueiras, 1991a) we noted that it compared favourably, in terms of speed, with the algorithms of Huet and of Fortenbacher — described in (Guckenbiehl and Herold, 1985).

In this section we give some results of an extensive comparison of our new algorithms with CBA[4] and the algorithms by Elliott (1903) and by Clausen and Fortenbacher (1989). The latter was generally believed to be the fastest method available, according to the results of comparisons[5] in (Clausen and Fortenbacher, 1989). The results below show clearly that our methods are faster than the other two, and therefore, for the moment being, the fastest available when the problems to solve are not too small.

All the algorithms under comparison were implemented in the same language (C, using the standard Unix *cc* with optimization flag $-O$) and the same machine (an old Sun 3/60). Execution times in the Tables below are CPU-times obtained by making calls to the appropriate Unix system routines. For problems that take less than 1/100 of a minute, the computation of the solutions was repeated 100 times and the average CPU-time spent is given. The values shown are the lowest CPU-times obtained in at least 3 runs.

We have adopted a sample of about 45 equations for comparisons, and we choose for presentation an illustrative subset of that sample (cf. Table 1). The first seven equations were taken from (Guckenbiehl and Herold, 1985), while the last four show the behaviour of the algorithms when the coefficients are larger. Note that the size of the problems was kept relatively small because of the inefficiency of the algorithms we are running. Larger problems will be treated when comparing our own algorithms between them.

Table 1. Equations used in the benchmarks.

	Coefficients	Min. sols.
E1	(2 1)(1 1 1 2 2)	13
E2	(2 1)(1 1 2 10)	13
E3	(9 5 2)(3 7 8)	65
E4	(9 5 2)(1 2 3 7 8)	119
E5	(3 3 3 2 2 2)(2 2 2 3 3 3)	138
E6	(8 7 3 2 1)(1 2 2 5 9)	345
E7	(10 2 1)(1 1 1 2 2)	349
E8	(653)(235 784 212)	37
E9	(29 13 8)(42 15 22)	133
E10	(29 13 8)(44 14 23)	216
E11	(23 21 15)(19 17 12 11)	303

[4] More specifically, CBA-V5, Slopes-V5i, Rect-LS1, and Rect-LG.

[5] With algorithms by Huet (but improved with the bounds of Lambert), by Fortenbacher, and by Lankford.

We start by comparing the Slopes algorithm with the algorithm by Elliott which we implemented from scratch[6]. 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. The results of the comparison are in Table 2. We have also implemented the version for solving systems of equations and compared it with recent methods for solving systems of equations — those of (Domenjoud, 1991; Boudet *et al.*, 1990) — with favourable results for the latter. The details of these comparisons may be found in (Filgueiras and Tomás, 1992c).

Table 2. Timing (ms) Slopes and Elliott algorithms.

Eq.	Slopes	Elliott basic	Elliott enhanced	Ell.(enh.) /Slopes
E1	3.5	18.7	1.7	0.5
E2	2.8	34.8	6.5	2.3
E3	17.7	1216.7	350.0	19.8
E4	44.3	8.4e3	1.6e3	36.1
E5	133.3	36.2e3	7.0e3	52.5
E6	200.0	431.8e3	60.7e3	303.5
E7	166.7	247.2e3	54.6e3	327.5
E8	16.7	19.7e3	1.5e3	89.8
E9	50.0	14.7e3	2.6e3	52.0
E10	533.3	75.3e3	15.9e3	29.8
E11	450.0	0.50e6	0.11e6	244.4

In order to compare the algorithm of Clausen and Fortenbacher we translated the Pascal program given in their paper to C and changed the graph initialization procedure so that the maximum coefficient in the equation being solved is taken into account. The results of a direct comparison with the Slopes algorithm are illustrated in Table 3. The conclusion is that our method is faster when the maximum coefficient is greater than about 3.

Finally we compare our algorithms between them. For the Rectangles algorithm we present the results for two of its implementations named Rect-LS1 and Rect-LG, the latter determining finer bounds for all unknowns. For the previous set of equations we get the results in Table 4. We may conclude that there is no major difference for small problems, although Slopes is faster than CBA when the coefficients increase, and the Rectangles algorithm beats by large the other two when solving equation E8, for obvious reasons.

[6] Using the descriptions in (Elliott, 1903; MacMahon, 1918; Stanley, 1973) — the latter however has some errors in its description and a wrong termination proof; see (Filgueiras and Tomás, 1992c).

Table 3. Timing (ms) Slopes and Clausen & Fortenbacher algorithms.

Eq.	Slopes	CF	CF/Slopes
E1	3.5	2.5	0.7
E2	2.8	11.0	3.9
E3	17.7	49.0	2.8
E4	44.3	300.0	6.8
E5	133.3	50.0	0.4
E6	200.0	1.55e3	7.8
E7	166.7	516.7	3.1
E8	16.7	1.75e3	104.8
E9	50.0	500.0	10.0
E10	533.3	1.02e3	1.9
E11	450.0	2.13e3	4.7

Table 4. Timing (ms) CBA, Slopes, Rect-LS1 and Rect-LG.

Eq.	CBA	Slopes	R-LS1	R-LG
E1	3.7	3.5	3.5	3.7
E2	2.7	2.8	3.0	3.0
E3	17.2	17.7	16.7	16.7
E4	42.5	44.3	33.3	33.3
E5	133.3	133.3	133.3	150.0
E6	233.3	200.0	233.3	283.3
E7	216.7	166.7	133.3	150.0
E8	33.3	16.7	7.0	7.0
E9	66.7	50.0	66.7	33.3
E10	583.3	533.3	550.0	766.7
E11	516.7	450.0	466.7	433.3

The differences between the algorithms become more visible for larger problems, as those in Table 5 for which we obtained the results in Table 6.

When there are many coefficients not too different (first two examples) Slopes appears to be faster than the other methods. In the next three examples one of the coefficients is much larger than the others, and the Rectangles algorithm is faster. The last three examples (equations with a single left-hand side coefficient and three on the left-hand) show the behaviour of the algorithms when the size of the left-hand side coefficient increases. The Rectangles algorithm is clearly better.

7 Conclusions

The results presented in the previous section clearly show that our methods are faster than the other methods available. This claim may be not true however when the coefficients are too small (2 or 3).

Table 5. A set of larger problems.

	Coefficients	Min. sols.
L1	(15 14 13 12 11)(11 11 12 13 14)	1093
L2	(25 24 23 21)(21 21 22 23)	3167
L3	(5021)(9 13 19)	5759
L4	(3229 95)(9 13 19)	2473
L5	(753 57)(9 11 13 19)	3337
L6	(653)(122 200 235)	53
L7	(1653)(122 200 235)	75
L8	(11653)(122 200 235)	151

Table 6. Timing (ms) the larger problems.

Eq.	CBA	Slopes	R-LS1	R-LG
L1	2.97e3	1.52e3	1.65e3	1.80e3
L2	27.72e3	17.85e3	19.62e3	32.00e3
L3	0.56e6	0.48e6	950.0	950.0
L4	49.78e3	42.42e3	1.78e3	516.7
L5	49.28e3	43.83e3	12.56e3	11.65e3
L6	33.3	16.7	10.2	10.5
L7	133.3	66.7	17.2	17.3
L8	12.38e3	1.87e3	33.3	33.3

From the work done so far there are some directions of research that appear as promising. One of them has to do with introducing more laziness in the evaluation of bounds. Another concerns extensions of the basic Rectangles algorithm for equations with two unknowns in each side and with more than four unknowns.

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

1. the possible application of the ideas underlying our methods in obtaining methods for solving systems. A method for solving systems under particular forms has been already designed and will be implemented soon,
2. further study of the Elliott algorithm which has a simple formulation and seems to have potentialities not yet explored,
3. integration of our methods in Constraint Logic Programming systems dealing with finite domains and naturals.

References

Boudet, A., Contejean E., and Devie, H.: A new AC Unification algorithm with an algorithm for solving systems of Diophantine equations. In Proceedings of the 5th Conference on Logic and Computer Science, IEEE, 289–299, 1990.

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

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

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

Filgueiras, M. and Tomás, A. P.: A Congruence-based Method with Slope Information for Solving Linear Constraints over Natural Numbers. Presented at the Workshop on Constraint Logic Programming '92, Marseille. Also as internal report, Centro de Informática da Universidade do Porto, 1992a.

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

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

Guckenbiehl, T. and Herold, A.: Solving Linear Diophantine Equations. Memo SEKI-85-IV-KL, Universität Kaiserslautern, 1985.

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

Lambert, J.-L.: 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, 1987.

MacMahon, P.: *Combinatory Analysis*, 2. Chelsea Publishing Co., 1918.

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

Tomás, A. P. and Filgueiras, M.: 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, 30–44, 1991a.

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