An Algorithm for Solving Systems of Linear Diophantine Equations in Naturals

Ana Paula Tomás and Miguel Filgueiras

LIACC & DCC, Universidade do Porto, Portugal
R. do Campo Alegre 823, 4150 Porto, Portugal
email: {apt,mig}@ncc.up.pt

Abstract. A new algorithm for finding the minimal solutions of systems of linear Diophantine equations has recently been published. In its description the emphasis was put on the mathematical aspects of the algorithm. In complement to that, in this paper another presentation of the algorithm is given which may be of use for anyone wanting to implement it.

1 Introduction

It is known since about 1900 that the monoid $\mathcal{M}$ of nonnegative integral solutions of the system of linear Diophantine equations (1)

$$AX = 0, \quad A \text{ a } m \times n \text{ integral matrix}, X \in \mathbb{N}^n \tag{1}$$

is finitely generated, there existing a unique finite subset of $\mathcal{M}$ such that $\mathcal{M}$ is the set of linear nonnegative integral combinations of the solutions in that subset, which we shall denote by $\mathcal{H}(\mathcal{M})$. The solutions in $\mathcal{H}(\mathcal{M})$ are the nonnull solutions of (1) that are minimal in the component-wise ordering given by $(u_1, \ldots, u_n) \preceq (v_1, \ldots, v_n)$ iff $u_i \leq v_i$ for all i. Problems as (1) arise in several different contexts, e.g., Constraint Programming, Integer Programming, Combinatorics, Rewriting Systems, and Abstract Interpretation. In some situations, solving means finding just one solution (as for instance, an optimal solution for some cost function). The problem of finding all the minimal solutions of (1) has been investigated by Elliott [4] and MacMahon [9] in the beginning of this century, and more recently by several other researchers ([7, 8, 2, 1, 11, 10, 3, 6, 5]) when it was found to be related to areas such as AC-unification, word problems, Petri Nets.

It should be noted that the problem belongs to the NP-complete complexity class, although it may be checked in polynomial time whether it is satisfiable. The algorithm presented here is polynomial when the solution space is planar. However, in the general case, the number of minimal solutions is exponential in the size of the system matrix. This number may be so large that a complete representation of the solution set by the set of minimal solutions will be of no practical interest. In term rewriting applications, and more specifically in some associative-commutative unification algorithms based on solving systems of Diophantine equations, a system with about twenty minimal solutions may render the unification problem intractable. In fact, solving an AC-problem may

involve solving more than one system of equations, and furthermore, to consider, at intermediate steps, subsets of the set of minimal solutions found.

The existing algorithms follow rather distinct approaches (see [15] for a brief survey). For instance, in general terms, the algorithm by Contejean and Devie [1], proceeds by increasing components one by one checking whether some solution is reached. In order to ensure termination, the components that can be incremented at a given step are selected following a criterion whose fundamental idea is that a move aims at improving, in some sense, the residue of a node. The algorithm by Domenjoud [3] uses the fact that the solutions of (1) in nonnegative real numbers define a polyhedral cone, $\mathcal{H}(\mathcal{M})$ being a subset of the finite set of $\mathbb{N}$-solutions that are rational nonnegative combinations with coefficients ≤ 1 of maximal subsets of linearly independent smallest $\mathbb{N}$-solutions generating the extreme rays of the cone. It uses a procedure for finding those combinations that give $\mathbb{N}$-solutions.

In our view the Slopes Algorithm, described here, is closer to the nature of the problem being solved. Its mathematical foundations give a better insight into the structure of the problems. This may be useful in identifying classes of problems for which one of the algorithms is more appropriate than another, as well as in improving the search.

The paper is organized as follows. We first recall some of the mathematical concepts involved, then give an outline of the Slopes Algorithm and proceed to see in detail each of its main steps.

2 Some background

Given $A \in \mathbb{Z}^{m\times n}$, the set of the nonnegative real solutions of $AX = 0$ is a pointed convex polyhedral cone $\mathcal{C}$, which we call *the solution cone*. When $\mathcal{C}$ is non-degenerated, $\mathcal{C}$ is the convex hull of its extreme rays (i.e., 1-dimensional faces) which are finitely many. Each extreme ray is given by $\{\alpha \mathtt{r}_i \mid \alpha \in \mathbb{R}_0^+\}$ where $0 \neq \mathtt{r}_i \in \mathbb{N}^n$ is some minimal solution of minimal support of (1). The support of x is supp $x = \{i \mid x_i \neq 0\}$, the x_i's being the coordinates of x wrt the canonical basis of $\mathbb{R}^n$. By definition, supp $X = \cup_{x\in X}$ supp x, for all $X \subseteq \mathbb{R}^n$. It is known that the faces of $\mathcal{C}$, together with the improper face $\mathcal{C}$, form a lattice under inclusion, the so-called face lattice. This lattice is isomorphic to the lattice of the supports of the faces under inclusion. Each face $\mathcal{F}$ of $\mathcal{C}$ is a pointed convex polyhedral cone whose extreme rays are extreme rays of $\mathcal{C}$. That is, if $\mathcal{R}_\mathcal{C} = \{\mathtt{r}_1, \ldots, \mathtt{r}_p\}$ is the set of minimal solutions of minimal support, then $\mathcal{C} = \mathit{cone}\mathcal{R}_\mathcal{C} = \{\sum_{i=1}^p \alpha_i \mathtt{r}_i \mid \alpha_i \in \mathbb{R}_0^+\}$, and for each face $\mathcal{F} \subset \mathcal{C}$ we have $\mathcal{F} = \mathit{cone}\mathcal{R}_\mathcal{F}$ for some $\mathcal{R}_\mathcal{F} \subset \mathcal{R}_\mathcal{C}$. The dimension of $\mathcal{F}$, denoted by dim $\mathcal{F}$, is the dimension of the linear subspace generated by $\mathcal{R}_\mathcal{F}$. A cone $\mathcal{F}$ is called *simplicial* if its extreme rays are linearly independent. If $\mathcal{R}_\mathcal{F}$ is some subset of $\mathcal{R}_\mathcal{C}$, then a necessary and sufficient condition for $\mathit{cone}\mathcal{R}_\mathcal{F}$ to be a face of $\mathcal{C}$ is that there exists no $r \in \mathcal{R}_\mathcal{C} \backslash \mathcal{R}_\mathcal{F}$ such that supp$\mathcal{R}_\mathcal{F}$ = supp$(\mathcal{R}_\mathcal{F} \cup \{r\})$. Moreover, if $\mathcal{F}'$ is a face of $\mathcal{F}$, and $\mathcal{F}'$ precedes immediately $\mathcal{F}$, then $\mathcal{F}'$ is called a facet of $\mathcal{F}$ and dim $\mathcal{F}$ = dim $\mathcal{F}' + 1$.

Algorithms that compute the solutions of minimal support and the face lattice are given in sections 4 and 5, respectively.

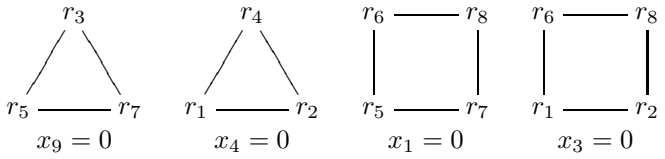
Example 1 The solution cone $\mathcal{C}$ of $AX = 0$ where

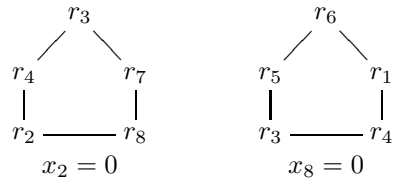
$$A = \begin{bmatrix} -1 & 1 & 0 & 1 & 1 & -2 & 1 & 0 & 2 \\ 2 & -1 & 0 & 1 & -2 & -1 & 2 & 1 & 2 \\ 0 & -2 & -2 & 2 & 1 & -1 & 2 & 0 & 0 \\ 0 & -2 & -1 & 1 & 0 & 1 & 0 & -2 & 2 \\ 0 & 0 & 0 & -2 & 0 & -1 & 2 & 2 & -2 \end{bmatrix}$$

is 4-dimensional with eight extreme rays defined by the minimal support solutions $r_1, \ldots, r_8$.

$$\begin{array}{ll} r_1 = (& 20\ 16\ 0\ 0\ 22\ 22\ 16\ 0\ 5\ ) \\ r_2 = (& 4\ 0\ 0\ 0\ 14\ 14\ 0\ 16\ 9\ ) \\ r_3 = (& 7\ 0\ 20\ 6\ 16\ 14\ 13\ 0\ 0\ ) \\ r_4 = (& 20\ 0\ 16\ 0\ 26\ 10\ 8\ 0\ 3\ ) \\ r_5 = (& 0\ 7\ 5\ 5\ 4\ 14\ 12\ 0\ 0\ ) \\ r_6 = (& 0\ 96\ 0\ 40\ 22\ 142\ 116\ 0\ 5\ ) \\ r_7 = (& 0\ 0\ 7\ 3\ 4\ 6\ 5\ 1\ 0\ ) \\ r_8 = (& 0\ 0\ 0\ 8\ 62\ 86\ 4\ 96\ 49\ ) \end{array}$$

The cross sections of the six 3-dimensional faces (i.e., facets) of $\mathcal{C}$ are schematically as follows, the two on the top-left being simplicial.

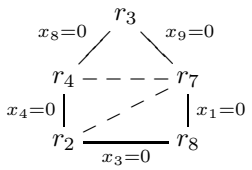




Each pointed convex polyhedral cone $\mathcal{F}$ may be decomposed into simplicial cones of dimension $\dim \mathcal{F}$ and such that the intersection of any two cones in the decomposition is a common face of both. We refer to such a decomposition as a *triangulation* of $\mathcal{F}$. It is possible to triangulate $\mathcal{F}$ in such a way that the extreme rays of the cones in the triangulation are extreme rays of $\mathcal{F}$ (e.g., [13]). We shall be considering only triangulations as such, and in section 6 an algorithm to compute such a triangulation is presented. From the characterization of the monoid $\mathcal{M}$ given by Stanley in [13], it may be deduced that if $s \in \mathcal{M}$ is a minimal solution lying in the interior of a simplicial face $\mathcal{F} = cone\{\mathtt{r}_1, \ldots, \mathtt{r}_k\}$ then

$s = \sum_{i=1}^{k} \alpha_i \mathbf{r}_i$ with $\alpha_i \in \mathbb{Q}$ and $0 < \alpha_i < 1$ (this result was stated independently by Domenjoud in [3]). When $\mathcal{F}$ is not simplicial, the same can be said about minimal solutions lying in the interior of each cone in a triangulation of $\mathcal{F}$. In particular, this result is useful to deduce upper bounds on the components of minimal solutions in the interior of a given face $\mathcal{F}$ from the minimal solutions of minimal support that generate the extreme rays of $\mathcal{F}$, i.e., from $\mathcal{R}_{\mathcal{F}}$.

Example 2 (Example 1 continued.) From the latter remarks, and the triangulation schematically shown below on the left, it follows that any minimal solution in the interior of $\mathcal{F} = cone\{r_2, r_3, r_4, r_7, r_8\}$ lies in the interior of either $cone\{r_2, r_4, r_7\}$ or $cone\{r_2, r_7, r_8\}$, since $x_8 < 1$ for minimal solutions in the interior of $cone\{r_3, r_4, r_7\}$. Thus, for any minimal solution in the interior of $\mathcal{F}$, a strict upper bound on x_i is $max\{(r_2 + r_4 + r_7)_i, (r_2 + r_7 + r_8)_i\}$. In particular, we see that $x_4 < 11$. The label on each 2-dimensional face indicates the component that is null for solutions lying in that face, either than x_2.



Bounds deduced for each subcone

$r_3 + r_4 + r_7 = (\ 27\ 0\ 43\ \ 9\ 46\ \ 30\ 26\ \ 1\ \ 3\ )$
$r_2 + r_4 + r_7 = (\ 24\ 0\ 23\ \ 3\ 44\ \ 30\ 13\ \ 17\ 12\ )$
$r_2 + r_7 + r_8 = (\ \ 4\ 0\ \ 7\ 11\ 80\ 106\ \ 9\ 113\ 58\ )$

The following 4-dimensional simplicial cones make a triangulation of $\mathcal{C}$:

$cone\{r_1, r_2, r_4, r_7\}$, $cone\{r_2, r_6, r_7, r_8\}$, $cone\{r_1, r_2, r_6, r_7\}$,
$cone\{r_1, r_5, r_6, r_7\}$, $cone\{r_3, r_4, r_5, r_7\}$, $cone\{r_1, r_4, r_5, r_7\}$.

3 The Slopes Algorithm

The general idea of the Slopes algorithm [15] is to visit all faces of the solution cone $\mathcal{C}$, searching for minimal solutions lying in the interior of each face (including $\mathcal{C}$, which is the top of the face lattice). The faces are explored in increasing order of dimension. The solutions in the interior of a face $\mathcal{F}$ are the solutions in *positive* integers (i.e., positive solutions) of the subsystem $A_{\mathcal{F}} X_{\mathcal{F}} = 0$ where $A_{\mathcal{F}}$ is obtained from A by keeping just the columns A^i such that $i \in \text{supp}\mathcal{F}$, and $X_{\mathcal{F}}$ resulting from X in a similar way. To compute these solutions, some of the variables in $X_{\mathcal{F}}$ are given values through enumeration. For each tuple of values, a problem of the form (2) is solved to find values for the remaining variables.

$$aX = By + Cz + V, \quad a > 0, \ \ B, C \in \mathbb{N}^r, \ V \in \mathbb{Z}^r \tag{2}$$

The minimal (X, y, z) in $\mathbb{N}^{r+2}$ satisfying (2) are well characterized [15], there existing efficient algorithms to compute them.

We shall now explain how (2) is obtained from $A_{\mathcal{F}}X_{\mathcal{F}} = 0$. It is shown in [15] that any system of the form $AX = 0$ with a *positive* solution and whose solution cone is at least two dimensional, is equivalent to (3),

$$\begin{cases} aX_0 = \mathcal{A}_{11}X_1 + \mathcal{A}_{12}X_2 \\ a'X_3 = \mathcal{A}_{22}X_2 \end{cases} \tag{3}$$

where all the coefficents are integers, $a > 0$, $\mathcal{A}_{11} \in \mathbb{N}^{r\times p}$, X is partitioned into X_0, X_1, X_2, X_3 (possibly, X_2 or X_3 are empty). In fact, suppose $A \in \mathbb{Z}^{m\times n}$, and let $\mathcal{C}$ be the solution cone. Always, rank$A = n - \dim\mathcal{C}$. To give the system form (3), consider some simplicial face $\mathcal{F}$ of $\mathcal{C}$ (e.g., all 2-dimensional faces are simplicial). Given supp$\mathcal{F}$, $p = \dim\mathcal{F} \geq 2$, and $\mathcal{R}_{\mathcal{F}} = \{\mathtt{r}_1, \ldots, \mathtt{r}_p\}$, identify X_1 with $\{x_{i_k} \mid 1{\leq}k{\leq}p\}$ where $i_k \in$ supp $\mathtt{r}_k$ and $i_k \notin$ supp$(\mathcal{R}_{\mathcal{F}}\backslash\{\mathtt{r}_k\})$, and X_0 with $X_{\mathcal{F}}\backslash X_1$. Let A_{X_0} be the submatrix of A consisting of the columns that correspond to X_0. This submatrix is full column rank, and $r = \text{rank}A_{\mathcal{F}} = \text{rank}A_{X_0}$. Now, either $r = \text{rank}A$ and the system is given the form $aX_0 = \mathcal{A}_{11}X_1 + \mathcal{A}_{12}X_2$, by diagonalization of A_{X_0}. Or, $r < \text{rank}A$, and rank$A - r$ variables from $X\backslash X_{\mathcal{F}}$ are selected so that rank$(A_{X_0}A_{X_3}) = \text{rank}A$, and $(A_{X_0}A_{X_3})$ is given a diagonal form to obtain (3).

By enumerating the values of variables in X_2 and of all those in X_1 but two (which we denote by y and z), the values of X_0, y, and z may be found by solving a problem as (2). We denote by W all the variables in $(X_1\backslash\{y, z\}) \cup X_2$.

Example 3 Consider again the previous example. For solutions in $cone\{r_2, r_3, r_4, r_7, r_8\}$ the value of x_2 is null, and the subsystem may be written, wrt $\mathcal{F} = cone\{r_2, r_4\}$, as

$$\begin{cases} 16x_1 = 20x_3 + 4x_8 - 48x_4 \\ 16x_5 = 26x_3 + 14x_8 - 44x_4 \\ 16x_6 = 10x_3 + 14x_8 + 4x_4 \\ 16x_7 = 8x_3 + 0x_8 + 8x_4 \\ 16x_9 = 3x_3 + 9x_8 - 10x_4 \end{cases}$$

For each value of x_4, with $1 \leq x_4 < 11$, the problem reduces to (2).

The method developed in [15] for solving $AX = 0$ is:

```
compute sols of minimal support;
compute the face lattice;
find minimal sols in 2-dimensional faces;
for all k with 2 < k <= dim(Top)
  for each k-dimensional face F
    triangulate F to deduce bounds on sols;
    give the subsystem the form  aX = By + Cz + DW;
    compute the basic spacings;
    enumerate values of all variables in W and
      for each tuple solve aX = By + Cz + V  where V = DW.
```

We shall now go on to explain each of the steps of the algorithm in more detail. In the sequel all gcd's are taken to be positive.

4 The Solutions of Minimal Support

The computation of the solutions of minimal support can be made in several ways. In our implementation we used an algebraic method that is fast enough for the examples we have. It is based on the fact that any such solution is a solution of some subsystem involving `M+1` columns of A, which is assumed of full row rank with `M` rows.

```
for each combination C of M+1 matrix columns do
  compute s[1], ..., s[M+1] with
    s[k] = det(Ck)  where Ck is C leaving out column k;
  if not all s[k] are null and all the components of
    (s[1],-s[2], ... (-1)^M*s[M+1]) are of same sign Sgn
  then Sgn*(s[1],..., (-1)^M*s[M+1])/gcd(s[1],..., s[M+1])
     are the components of a solution of minimal support
     corresponding to the columns of the matrix in C,
     the other components being null.
```

In the implementation we first compute all the determinants for combinations of M columns of the given matrix. Such combinations can be represented in a compact way by bit masks; moreover, with matrix dimensions less than 20, there are less than $2^{20} = 2048$ possible combinations and it is feasible to have an array of determinant values directly indexed by the masks. For accessing the matrix elements we keep an array of column indices.

5 Computing the Lattice of Faces

The maximal face (top node) in the lattice is described by the set of all extreme rays (solutions of minimal support). Each extreme ray is a 1-dimensional face. These facts are used to start the computation of the lattice, in which we do not include the bottom element. We call *layer* a set of faces of same dimension.

If there is only one extreme ray, the top node of the lattice is the only one to be created and contains it. If this is not the case, the algorithm starts by creating the top node of the lattice and faces for the extreme rays setting their dimension to 1. Then it proceeds by forming new faces through combining an already created face with each extreme ray. During this process a dimension is assigned to the new face and adjustments to the dimensions of other faces may have to be made. When no new face can be formed, each face (different from the maximal one) is inserted in the layer corresponding to its dimension. Finally, each layer is visited and the faces in it are linked to their subfaces in the next (lower) layer, and the top node is linked to the faces in the upper layer.

This algorithm can be described as follows:

```
set Top to a new face containing all extreme rays;
if there is only one extreme ray stop;
for each extreme ray create a new face of dimension 1;
combine each extreme ray with each created face f =/= Top
  if resulting face is Top or is f do nothing
  else if resulting face exists then
    if its dimension < dim(f)+1
    then set its dimension to  dim(f)+1
    else do nothing;
  else (resulting face is new)
    set F to a new face;
    set dim(F) to  1+max { dim(F') | F' is subface of F };
    for each face F' with F a subface of F'
      set dim(F') to  max(dim(F)+1, dim(F'));
set all layers to empty;
for each face f =/= Top
  insert f into Layer[dim(f)];
set dim(Top) to 1+maximum dimension found;
for each i>1, i<dim(Top)
  for each face f in Layer[i]
    for each face f' in Layer[i-1]
      if f' is subface of f
      then link f to f';
for each face f in upper layer  link Top to f.
```

6 Triangulating a Face

The result of the computation is a set of cones, each being represented by the set of its extreme rays. The algorithm has two parameters: the face to triangulate and a set of selected rays (empty on the first call). It begins by a test on whether the given face is simplicial, in which case a new cone, given by the union of the current set of extreme rays with the extreme rays in the face, obtains. Otherwise, an extreme ray in the face is selected (as described below) and added to the set of rays, and recursive calls are made on each facet of the given face such that it does not contain the selected ray.

There are no constraints on the selection of an extreme ray. However, choosing different rays may lead to different triangulations and possibly to different bounds (derived from them) on the values of variables. We use as heuristics the selection of the ray with least sum of components. For representing sets of extreme rays we use bit masks, so that set operations are very efficient.

```
set SC to empty;
call triang(Face, {}),
```

```
where triang(F, Sel_Rays) is
  if F is simplicial
  then  set SC to  SC U {Sel_Rays U rays_of(F)}
  else
    select a vertex R of F;
    for all f, f a facet of F, R not in f
      call triang(f, {R} U Sel_Rays).
```

7 Solving $aX = By + Cz + V$

Solving $aX = By + Cz + V$, where $B, C \in \mathbb{N}^m$, $a \in \mathbb{N}\backslash\{0\}$, is related to solving $By + Cz + V \equiv 0 \pmod{a}$ with $By + Cz + V \in \mathbb{N}^m$. So, reduce $(B\ C)$ to $\begin{pmatrix} t_{11}\,0 & 0\ldots 0 \\ t_{12}\,t_{22}\,0\ldots 0 \end{pmatrix}^t$, which we denote by $\begin{pmatrix} T \\ 0 \end{pmatrix}$, by performing elementary row transformations (similar to the computation of Hermite normal form [12]) and reducing the elements modulo a along the process. Let $U \in \mathbb{Z}^{m\times m}$ be the matrix of transformations, i.e., $U(B\ \ C) \equiv \begin{pmatrix} T \\ 0 \end{pmatrix} \pmod{a}$.

7.1 When $V = 0$

Since $(X, y, z) \in \mathbb{N}^{m+2}$ is a minimal solution of $aX = By + Cz$ iff $(y, w) \in \mathbb{N}^2$ is a minimal nonnull solution of

$$t_{11}y + a/\gcd(a, t_{22})t_{12}w \equiv 0 \pmod{a} \tag{4}$$

being $z = a/\gcd(a, t_{22})w$, and $X = 1/a(By + Cz)$, the algorithm solves (4).

Example 4 In the example above, the minimal solutions in $cone\{r_2, r_4\}$ correspond to the minimal $(x_3, x_8) \in \mathbb{N}^2$ satisfying $8x_8 \equiv 0 \pmod{16} \wedge x_3 + 3x_8 \equiv 0 \pmod{16}$, or equivalently to the minimal (x_3, w) in $\mathbb{N}^2$ such that $x_3 + 6w \equiv 0 \pmod{16} \wedge x_8 = 2w$.

The minimal nonnull $(y, z) \in \mathbb{N}^2$ satisfying $by + cz \equiv 0 \pmod{a}$, $a > 0$, $b, c \geq 0$, may be found efficiently by the following algorithm, where `mod` gives a nonnegative remainder, and `multiplier(b,a)` computes $m_b > 0$ such that $m_b b + m_a a = \gcd(a, b)$.

```
ymax = a/gcd(a,b);  zmax = a/gcd(a,c);
dz = gcd(a,b)/gcd(a,b,c);
dy = (c*multiplier(b,a)/gcd(a,b,c)) mod ymax;
S =  {(ymax, 0), (0, zmax)};
if  dy = 0 return;
y = ymax-dy;   z = dz;  S = S U {(y, z)};
while  dy > 0
  while  y > dy
    y = y-dy;  z = z+dz;  S = S U {(y, z)};
  aux = dy/y;   dy = dy mod y;   dz = aux*z+dz.
```

The minimal solutions in 2-dimensional faces are computed by this method, since if $\dim \mathcal{F} = 2$, then $A_{\mathcal{F}} X_{\mathcal{F}} = 0$ is equivalent to some system of the form $aX = By + Cz$, which is obtained by applying the transformation in Section 3. The algorithm computes just minimal solutions and no superfluous candidates.

7.2 When $V \neq 0$

The general idea is to compute a starting solution, and to move from a given minimal solution s to the next one (in increasing order of z) by adding a suitable spacing to s. Note that, if some solution (X, y, z) found by the method has some null component, (X, y, z, W) is not checked for (global) minimality since it is not in the *interior* of the face being explored.

Computing the basic spacings Rename t_{11} and $t_{12}a/\gcd(a, t_{22})$ to b and c respectively. The so-called set of basic spacings `BS` is computed by the following method.

```
ymax = a/gcd(a,b);
dw1 = gcd(a,b)/gcd(a,b,c);
dy1 = (c*multiplier(b,a)/gcd(a,b,c)) mod ymax;
compute the set Sp of minimal (dy,u) such that
  (ymax-1)*dy+dy1*u equiv 0 (mod ymax);
dz1 = a/gcd(a,t22)*dw1;
BS = {(1/a*(-dy*B+dz*C),-dy,dz)) | (dy,u) in Sp, dz=dz1*u}.
```

The algorithm described in the previous section is used yielding the elements of `Sp` in increasing order of u. Thus, `BS` is assumed to be in increasing order of δ_z (i.e., variation in z).

The starting solution The starting solution is the solution with the least z. Provided $U_i V \equiv 0 \pmod{a}$ for all $3 \leq i \leq m$, and $U_2 V \equiv 0 \pmod{\gcd(a, t_{22})}$, otherwise the problem is unsatisfiable, let

$$z_0 = -U_2 V/\gcd(a, t_{22}) \text{multiplier}(t_{22}, a) \mod a/\gcd(a, t_{22}).$$

Here, U_i stands for the ith row of the matrix of transformations U. Now, the problem is satisfiable iff $\gcd(a, t_{11}, t_{12}a/\gcd(a, t_{22}))$ divides $-U_1 V - t_{12} z_0$. Replace z_0 by $z_0 + a/\gcd(a, t_{22}) max(0, \lceil (z_{\min} - z_0)\gcd(a, t_{22})/a \rceil)$, where $z_{\min} = max(0, \{\lceil -v_i/c_i \rceil \mid b_i = 0 \wedge c_i \neq 0\})$, being $B = (b_i)_i$, $C = (c_i)_i$, $V = (v_i)_i$. Let $b = t_{11}$, $c = t_{12}a/\gcd(a, t_{22})$, and $v = -U_1 V - t_{12} z_0$, and compute

$$w_0 = \frac{v M_c}{\gcd(a, b, c)} \bmod \delta_w^1 \qquad y_0 = \frac{(v - w_0 c) m_b}{\gcd(a, b)} \bmod y_{\max}. \tag{5}$$

where $y_{\max} = a/\gcd(a, b)$, $\delta_w^1 = \gcd(a, b)/\gcd(a, b, c)$, m_b is multiplier(b, a), and M_c is any integer satisfying $bM_b + cM_c + aM_a = \gcd(a, b, c)$, for some integers M_a, and M_b. The *starting solution* is given by

$$(y_0 + k_0 y_{\max},\ z_0 + w_0 a/\gcd(a, t_{22}))$$

with $k_0 = max(0, \{\lceil(-b_i y_0 - c_i(z_0 + w_0 a/\gcd(a, t_{22})) - v_i)/(b_i y_{\max})\rceil \mid b_i \neq 0\})$. Clearly, for the starting solution,

$$X = 1/a(B(y_0 + k_0 y_{\max}) + C(z_0 + w_0 a/\gcd(a, t_{22})) + V).$$

Note that, $k_0 = z_{\min} = y_{\min} = 0$ when $V \succeq 0$, where $y_{\min}$ is defined as $max(0, \{\lceil -v_i/b_i \rceil \mid b_i \neq 0 \wedge c_i = 0\})$.

Finding the minimal solutions when $V \succeq 0$

```
zmax = a/gcd(a,t12,t22);
dymin = the least nonnull dy;
find starting solution (X,y,z);
check global minimality of the solution found;
while y > dymin and z < zmax
  add the first spacing in BS with dy < y to (X,y,z);
  check global minimality of the solution found.
```

The least positive `dy` is given by the last but one element in `BS`. Note that, if the kth spacing was used, then the spacings that can be used in the subsequent steps are either the kth or the following ones in `BS`.

Finding the minimal solutions when $V \not\succeq 0$

```
Boundz = upper bound on z;
dymin = the least nonnull dy;
dXL1 = dX for the last but one spacing in BS;
dXL  = dX for the last spacing in BS;
find starting solution (X,y,z);
check global minimality of the solution found;
while y >= dymin+ymin and z < Boundz
  if no spacing in BS but the last can be added to (X,y,z);
    t0 = mininum t with  X + t*dXL1 + dXL  nonnegative;
    add t0*(dXL,0,zmax) to (X,y,z);
  else
    add the 1st spacing with dy<=y-ymin, X+dX>=0 to (X,y,z);
    check global minimality of the solution found.
```

Here `Boundz` is some known upper bound on the z-component for the minimal solutions in the interior of the face $\mathcal{F}$, which can be deduced, for instance, from a triangulation.

8 Checking Global Minimality

Global minimality of candidate solutions must be checked by making sure that known bounds for the components were not exceeded and by comparison with minimal solutions computed before. Only solutions with support included in the support of the candidate need to be compared. The isomorphism between the lattice of supports and the lattice of faces is used here: we keep information on which face the minimal solutions belong to and when checking minimality for a candidate for a given face we just compare it with the solutions in the face and in its descendants in the lattice. The representation of the set of solutions for a face in the lattice consists of a pair of indices in the global array of solutions, as the algorithm computes all the solutions for each face consecutively. This leads to the following procedure:

```
if bounds for y or X are violated  return;
for each solution s in current face
  if each component of candidate > component of s  return;
visit recursively, once, each descendant of current face
  and return if there is a solution smaller than candidate;
add candidate as new minimal solution.
```

Candidate solutions found in a given face for the same W are not comparable, so `s` above can be taken as some solution found for a different W.

9 Conclusion

We have presented, in algorithmic terms, the method for solving systems of linear Diophantine equations described, in mathematical terms, in [15, 14]. This presentation may be useful for those interested in implementing it without delving into the mathematical details.

We have implemented our algorithm in C and compared it with other similar algorithms by applying the statistical method of [6]. The results of the comparison are given in [16] and the conclusion is that for most cases our algorithm is faster.

Our current implementation incorporates some improvements (see [16] for the details), among which

- when the solution cone is simplicial, the system is equivalent to a system of congruences, as deduced from the transformation described in Section 3, and can be solved in a faster way. The minimal solutions may be used to prune effectively the search space.
- rather than reducing a subsystem to the form $aX = By + Cz + DW$ and enumerating W for all possible values, it can be studied whether it is advantageous to change the enumerated variables by performing again a reduction wrt another subface.

References

1. 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.
2. Clausen, M., and Fortenbacher, A.: Efficient solution of linear Diophantine equations. *J. Symbolic Computation*, 8, 201–216, 1989.
3. Domenjoud, E.: *Outils pour la Déduction Automatique dans les Théories Associatives-Commutatives*. Thése de doctorat, Université de Nancy I, 1991.
4. Elliott, E. B.: On linear homogenous Diophantine equations. *Quart. J. Pure Appl. Math.*, 34, 348–377, 1903.
5. Filgueiras, M., and Tomás, A. P.: 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, 297–306, 1993.
6. Filgueiras, M., and Tomás, A. P.: A Fast Method for Finding the Basis of Non-negative Solutions to a Linear Diophantine Equation. *J. Symbolic Computation*, 19, 507–526, 1995.
7. Huet, G.: An algorithm to generate the basis of solutions to homogeneous linear Diophantine equations. *Information Processing Letters*, 7(3), 1978.
8. 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.
9. MacMahon, P.: *Combinatory Analysis*, 2. Chelsea Publishing Co., 1918.
10. Moulinet-Ossola, C.: *Algorithmique des Réseaux et des Systèmes Diophantiens Linéaires*. Thése de doctorat, Université de Nice Sophia-Antipolis, 1995.
11. Pottier, L.: Minimal solutions of linear diophantine systems: bounds and algorithms. In R. V. Book (ed.), *Proceedings of the 4th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 488, Springer-Verlag, 162–173, 1991.
12. A. Schrijver, *Theory of Linear and Integer Programming*, Wiley-Interscience, 1986.
13. Stanley, R.P.: *Enumerative Combinatorics*, Vol I, The Wadsworth & Brooks/Cole Mathematics Series, 1986.
14. Tomás, A. P., Filgueiras, M.: Solving linear Diophantine equations using the geometric structure of the solution space. In H. Common (ed.), *Rewriting Techniques and Applications*. Proceedings, Lecture Notes in Computer Science 1232, Springer-Verlag, 269–283, 1997.
15. Tomás, A. P.: *On Solving Linear Diophantine Constraints*. Tese de Doutoramento, Faculdade de Ciências da Universidade do Porto, 1997.
16. Tomás, A. P., Filgueiras, M.: Exploiting the Geometric Structure of the Solution Space in Solving Linear Diophantine Equations. Internal Report, DCC & LIACC, Universidade do Porto, 1997.

This article was processed using the LaTeX macro package with LLNCS style.