On Defining and Computing "Good" Conservation Laws

François Lemaire, Alexandre Temperville

On Defining and Computing "Good" Conservation Laws

François Lemaire[1] and Alexandre Temperville[1]

Université Lille 1, LIFL, UMR CNRS 8022, Computer Algebra Group,
francois.lemaire@univ-lille1.fr,a.temperville@ed.univ-lille1.fr

Abstract. Conservation laws are a key-tool to study systems of chemical reactions in biology. We address the problem of defining and computing "good" sets of conservation laws. In this article, we chose to focus on sparsest sets of conservation laws. We present a greedy algorithm computing a sparsest set of conservation laws equivalent to a given set of conservation laws. Benchmarks over a subset of the curated models taken from the BioModels database are given.

Keywords: conservation analysis, sparse conservation laws, biological models, sparse null space, greedy algorithm.

1 Introduction

Many biological processes can be modelled by systems of chemical reactions. In order to study such systems, one usually computes its (linear) conservation laws (*i.e.* linear combinations of number of species) which have the property of being constant along the time. In this article, we only consider (linear) conservation laws. For a given chemical reaction system, a complete set of conservation laws is easily computed by computing a basis of the kernel of the transpose of the stoichiometry matrix of the system [10].

This paper tries to answer to the difficult question: "what is a good conservation law ?". Consider for example the well known enzymatic degradation given by the reactions $E + S \leftrightarrow C$ and $C \rightarrow E + P$. It admits for example $E + C$ and $S + C + P$ as conservation laws. One could as well consider their sum (*i.e.* $E + 2C + S + P$), their difference (*i.e.* $E - S - P$), ... On the example, the laws $E + C$ and $S + C + P$ seem less artificial and closer to the physics of the system, than the two laws $E + 2C + S + P$ and $E - S - P$. Indeed the law $E + C$ corresponds to the conservation of the enzyme E, and $S + C + P$ corresponds to the conservation of the substract S (which is either in the form S, C or P).

In an attempt to define "good conservation laws", we think that a good conservation law should have many zero coefficients (*i.e.* sparse laws) and many positive coefficients. Concerning the sparse property, we think that a practitioner would understand better sparse laws than dense laws, since sparse laws are shorter and thus easier to read. Moreover, a sparse conservation law can also be useful when doing substitutions in differential equations to preserve the

sparsity of the differential equations. For example, if one has a sparse system $\dot{X} = F(X)$ (where X is a vector of species X_1, X_2, ...), one can use a conservation law involving X_1 (say $X_1 + X_5 - X_8 = c_0$) to discard the variable X_1 by substituting X_1 by an expression in the other species ($c_0 - X_5 + X_8$ on the example). Consequently, a sparse conservation law will more likely preserve the sparsity of the differential equations. Concerning positive coefficients, we think that conservation laws with positive coefficients are more likely to represent a conservation of matter.

Those two criteria (sparsity and positiveness) are sometimes impossible to satisfy at the same time. For example, if we have a basis of two conservation laws $X_1 + X_2 + X_3$ and $X_2 + X_3 + X_4$, then the difference $X_1 - X_4$ is sparser than any of the two laws but involves a negative coefficient. Moreover, in some particular examples, there are no conservation laws with positive coefficients only (like in $A + B \to \emptyset$ which only admits $A - B$ as a conservation law).

In this paper, we have chosen to compute a sparsest basis of conservation laws, leaving the positivity argument for a further work. As a consequence, our approach differs from computing minimal semi-positive P-invariants (*i.e.* conservation laws with non-negative coefficients with minimal support [13]).

Our approach corresponds to the well known Nullspace Sparsest Problem (NSP) which is proven to be NP-hard in [3]. NSP consists in finding a matrix with the fewest nonzeros, whose columns span the null space of a given matrix. Approximate algorithms to solve NSP are given in [3, 4].

We chose to develop our method by testing it on the Biomodels database [1]. Our hope was that biological models might have special properties and might be solved easily even if the problem is NP-hard. Even if we could not exhibit special properties of the biological models, our method computes the sparsest basis of conservation laws for most curated models of the Biomodels database (see Section 5), thus validating our approach.

Some usual linear algebra algorithms can sometimes produce a sparser basis, with no guarantee it is a sparsest one. The Hermite normal form is such a technique, the (reduced or not) row echelon form is another. In the context of $\mathbb{Z}$-lattices, [5] introduces and computes "short" (pseudo-) bases using the LLL algorithm [7] and a variant of the Hermite normal form. In a numerical context (*i.e.* using floating point coefficients), there are methods to compute sparser basis (as in [2] where the turnback algorithm computes a sparse and banded null basis of a large and sparse matrix).

[14, 10] present method based on numerical computations (QR decomposition, SVD,...) to compute exact conservation laws of large systems. Finally, we wonder if the techniques used in the extreme pathways [12, 11] could be used to compute sparse conservation laws.

The paper is organized in the following way. Section 2 presents, on an example, the idea of our algorithm ComputeSparsestBasis(B), which computes a sparsest basis equivalent to B. Section 3 presents the algorithm ComputeSparsestBasis and its sub-algorithms, and Appendix A details their proofs. Section 4 details the implementation and improvements. Finally, Section 5 shows our benchmarks.

2 A Worked Out Example

We illustrate our method on the model number 361 (BIOMD0000000361) of the BioModels database [1, 8]. For clarity reasons, one renames the species in the following way : $VIIa\_TF \to V$, $VIIa\_TF\_X \to VX$, $VIIa\_TF\_Xa \to VX_a$, $TFPI \to T$, $Xa\_TFPI \to X_aT$, $Xa\_TFPI\_VIIa\_TF \to X_aTV$, $Xa \to X_a$.

The model contains the five following chemical reactions:

$$X + V \leftrightarrow VX, \qquad VX \to VX_a, \qquad VX_a \leftrightarrow X_a + V$$
$$X_a + T \leftrightarrow X_aT, \qquad V + X_aT \leftrightarrow X_aTV.$$

By choosing the vector of species ${}^t\big(X_aTV, X_a, X, VX_a, VX, T, V, X_aT\big)$, one can compute the stoichiometry matrix M and a basis of conservation laws (written row by row) $B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & -1 & -1 & -1 & -1 & 1 & 0 & 0 \end{pmatrix}$ by computing a basis of the nullspace of the transpose of M (denoted ${}^tM$). Thus, the matrix B represents the three conservation laws:

1. $X_aTV + X_a + X + VX_a + VX + X_aT$
2. $X_aTV + VX_a + VX + V$
3. $-X_a - X - VX_a - VX + T$

Our method for decreasing the number of nonzeros consists in finding a linear combination $w = {}^tvB$ of the rows of B such that w contains less nonzeros than one row B_i of B. If one can find such a combination w, it feels natural to replace the row B_i by w in order to decrease the total number of nonzeros in B. Repeating this process until no such linear combination can be found, one obtains a sparsest basis in terms of nonzeros. This approach is greedy and is justified in Section 3.

However, replacing a row of B by w should only be done if one maintains a basis. This last requirement is fulfilled by replacing the row B_i of B by w only if $v_i \neq 0$ (which loosely speaking means that the information in the row B_i has been kept).

Consider the linear combination $w = {}^tvB$ with $v = (\alpha, \beta, \gamma)$, so one has $w = (\alpha+\beta, \alpha-\gamma, \alpha-\gamma, \alpha+\beta-\gamma, \alpha+\beta-\gamma, \gamma, \beta, \alpha)$. The number of nonzeros of w clearly depends on the values of α, β and γ. In order to compute the number of nonzeros of w, one considers all possible cases, corresponding to the cancellation or the non cancellation of each element of w. In theory, if w has n components, one has 2^n cases to consider. For example, if we request w to have the form $(\neq 0, 0, 0, 0, 0, \neq 0, 0, \neq 0)$ one considers the following system of equations and inequations:

$$\begin{cases} \alpha + \beta \neq 0 & \text{(column 1)} \\ \alpha - \gamma = 0 & \text{(columns 2 and 3)} \\ \alpha + \beta - \gamma = 0 & \text{(columns 4 and 5)} \\ \gamma \neq 0 & \text{(column 6)} \\ \beta = 0 & \text{(column 7)} \\ \alpha \neq 0 & \text{(column 8)} \end{cases}$$

This last system admits for example the solution $\alpha = \gamma = 1$ and $\beta = 0$. The corresponding linear combination $w = {}^t vB$ is $w = B_1 + B_3 = (1, 0, 0, 0, 0, 1, 0, 1)$ which contains 3 nonzeros, and is thus better than the rows B_1, B_2 and B_3 (which respectively involves 6, 4, and 5 nonzeros). Since w involves the rows B_1 and B_3 (*i.e.* $\alpha \neq 0$ and $\beta \neq 0$), one can replace B_1 or B_3 by w. Note that replacing B_2 by w would lead to a matrix of rank 2, meaning that our basis has been lost. For example, replacing B_1 by w, one obtains an equivalent basis:

$$B' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & -1 & -1 & -1 & -1 & 1 & 0 & 0 \end{pmatrix}.$$

In practice, one does not enumerate all the possible patterns of zeros and nonzeros for the vector w. Instead, one considers the columns of B one by one from left to right, and one builds systems of equations (corresponding to the zeros in w) and inequations (corresponding to the nonzeros in w). Since each column of B yields two cases, one builds a binary tree of systems of equations and inequations. By doing this, many branches are hopefully cut before all the columns of B have been treated. For example, if one tries to cancel the first five columns of w, one gets the system of equations $\begin{cases} \alpha + \beta = 0 & (1^{st} \text{ column of } B) \\ \alpha - \gamma = 0 & (2^{nd}, 3^{rd} \text{ columns of } B) \\ \alpha + \beta - \gamma = 0 & (4^{th}, 5^{th} \text{ columns of } B) \end{cases}$. which only admits the useless solution $\alpha = \beta = \gamma = 0$.

Let us continue the improvement of B'. Following the same ideas as above, one can find the following linear combinations of the rows of B':

$$w' = B'_1 - B'_2 - B'_3 = (0, 1, 1, 0, 0, 0, -1, 1)$$

which has less nonzeros than B'_3. By replacing B'_3 by w, one gets the basis

$$B'' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

Finally, further computations would show that there does not exist a linear combination of the rows allowing to improve our basis B''. In that case, one knows that our basis is a sparsest one. If one replaces the third line of B'' by $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$, then one gets three conservation laws with a total of 13 non-negative coefficients (instead of 11 nonzero coefficients for B''). One can show that computing the minimal semi-positive P-invariants (as done in [13]) would retrieve these three conservation laws with 13 nonzeros.

To summarize, our method adopts a greedy approach by successively improving the initial basis (each time by only changing one row), until it reaches a sparsest basis. Each improvement consists in a binary tree exploration where each node is a system of equations and inequations.

3 The Algorithm ComputeSparsestBasis

3.1 Sparsest Basis of Conservation Laws

As mentioned in the introduction, we have chosen to compute sparsest basis of conservation laws. We define that notion precisely in this part.

Let B be a matrix of dimensions $m \times n$ over $\mathbb{Q}$, with $m \leq n$. The matrix B is called a *basis matrix* if B is a full rank matrix *i.e.* $\text{Rank}(B) = m$. In this paper, a basis matrix contains a basis of conservation laws written row by row. Let B and B' be basis matrices of same dimensions. B and B' are *equivalent* if and only if there exists an invertible matrix Q such that $B = QB'$. Let M (resp. v) be a matrix (resp. vector), we denote $\mathcal{N}(M)$ (resp. $\mathcal{N}(v)$) the number of nonzero coefficients of M (resp. v). Let v be a vector, we denote $\mathcal{N}_k(v)$ the number of nonzero coefficients in the first k coefficients of v. A basis matrix B' is a *sparsest basis matrix* if and only if for any basis matrix B equivalent to B, one has $\mathcal{N}(B') \leq \mathcal{N}(B)$.

For any basis matrix B, it is clear that there exists a sparsest matrix B' equivalent to B. Indeed, consider the set of all equivalent matrices to B, and pick one matrix B' in that set such that $\mathcal{N}(B')$ is minimal.

3.2 A Greedy Approach

Our method follows a greedy approach. Given a basis matrix B, one looks for a vector v and an index i such that $\mathcal{N}({}^tvB) < \mathcal{N}(B_i)$ and $v_i \neq 0$. If such v and i exists, one can decrease the number of nonzeros of B by replacing the row B_i by ${}^tvB$. Moreover, the rank of B does not drop since one has $v_i \neq 0$. When such suitable v and i do no exist, our method stops and claims that our basis has become a sparsest one. This last claim is not obvious, since one could have fallen in a local minimum. The following theorem justifies our greedy approach.

Theorem 1. *A basis matrix B is not a sparsest one if and only if there exist a vector v and an index j such that $\mathcal{N}({}^tvB) < \mathcal{N}(B_j)$ and $v_j \neq 0$.*

Proof. $\Leftarrow$: Taking $B' = B$ and replacing the row B'_j by ${}^tvB$, one gets a matrix B' equivalent to B such that $\mathcal{N}(B') < \mathcal{N}(B)$, which proves that B is not a sparsest one.

$\Rightarrow$: Assume B has dimensions $m \times n$. There exists B' equivalent to B such that $\mathcal{N}(B') < \mathcal{N}(B)$. By permuting the rows of B and the rows of B', one can assume $\mathcal{N}(B_1) \geq \mathcal{N}(B_2) \geq \cdots \geq \mathcal{N}(B_m)$ and $\mathcal{N}(B'_1) \geq \mathcal{N}(B'_2) \geq \cdots \geq \mathcal{N}(B'_m)$. As $\mathcal{N}(B) = \sum_{i=1}^{m} \mathcal{N}(B_i) > \sum_{i=1}^{m} \mathcal{N}(B'_i) = \mathcal{N}(B')$, there exists an index k such that $\mathcal{N}(B_k) > \mathcal{N}(B'_k)$. Since B and B' are equivalent, each row of B' is a linear combination of rows of B. If all the $m-k+1$ rows $B'_k, B'_{k+1}, \cdots, B'_m$ were linear combinations of the $m-k$ rows $B_{k+1}, \ldots, B_m$, then B' would not be a full rank matrix. Thus, there exist a vector v and indices j, l with $j \leq k \leq l$ such that $B'_l = {}^tvB$ with $v_j \neq 0$. Since $\mathcal{N}(B'_l) \leq \mathcal{N}(B'_k) < \mathcal{N}(B_k) \leq \mathcal{N}(B_j)$, one has $\mathcal{N}({}^tvB) = \mathcal{N}(B'_l) < \mathcal{N}(B_j)$ with $v_j \neq 0$. □

3.3 Description of a Task

As explained in Section 2, our method builds a (binary) tree of systems of equations and inequations. In practice, one only stores the leaves of the tree in construction. One introduces the notion of *task* which basically represents one leaf of the tree. In order to cut useless branches as soon as possible, one also requires a task to satisfy the extra properties **LCP** and **IZP** of Definition 1. Let v be a vector of dimension n. The notation $v \not\equiv 0$ means that $\forall i \in [\![1, n]\!]$, $v_i \neq 0$.

Definition 1. *Let B be a basis matrix. Let A, Λ be matrices with m columns. Let $(\mathcal{S}) : \begin{cases} Ax = 0 \\ \Lambda x \not\equiv 0 \end{cases}$ be a system in the variable x. Let c be the number of rows of Λ. Let k be the sum of the number of rows of A and Λ. A **task** $t = \text{TASK}[A, \Lambda, c, k]$, stemming from B, is defined as follow :*

- *the union of the rows of A and Λ coincides with the first k columns of B (up to the order),*
- *(A, Λ) satisfies the so-called **LCP** property (Linear Combination Property)* i.e. *there exists a solution v of $(\mathcal{S})$ with at least two nonzero coefficients,*
- *(A, Λ, c, k) satisfies the so-called **IZP** property (Increase Zeros Property)* i.e. *there exist a solution v of $(\mathcal{S})$ and an index j such that $v_j \neq 0$ and $\mathcal{N}_k({}^tvB) < \mathcal{N}(B_j)$.*

Proposition 1. *Consider a task $t = \text{TASK}[A, \Lambda, c, k]$ stemming from a basis matrix B, and the system $(\mathcal{S})$, as defined in Definition 1. Consider $\mathcal{U} = \{i \in [\![1, m]\!], c < \mathcal{N}(B_i)\}$. Then one has the following properties:*

1. *$A \in \mathbb{Q}^{(k-c)\times m}$ and $\Lambda \in \mathbb{Q}^{c\times m}$ with $0 \leq c \leq k \leq n$,*
2. *For each nonzero solution v of $(\mathcal{S})$, $\mathcal{N}_k({}^tvB) = c$* i.e. *$c$ is the number of nonzeros in the first k coefficients of any solution of $(\mathcal{S})$,*
3. *There exist a solution v of $(\mathcal{S})$ and $j \in \mathcal{U}$ such that $v_j \neq 0$ and $c < \mathcal{N}(B_j)$.*

Proof. 1. Trivial.

2. Take a nonzero solution v of $(\mathcal{S})$ and consider $w = {}^tvB$. Consider a column j of B with $j \leq k$. Then the transpose of this column is either a row of A or a row of Λ. If it is a row A_i of A (resp. a row Λ_i of Λ), then the j^{th} coefficient of w equals zero (resp. is nonzero) since $A_iv = 0$ (resp. $\Lambda_iv \neq 0$). Consequently, the number of nonzero elements among the first k coefficients of $w = {}^tvB$ equals c (*i.e.* the number of rows of Λ).
3. It is a consequence of the **IZP** property and $\mathcal{N}_k({}^tvB) = c$. □

The task $t_0 = \text{TASK}[\text{the } 0 \times m \text{ matrix}, \text{the } 0 \times m \text{ matrix}, 0, 0]$ is called the *initial task*. A task $t = \text{TASK}[A, \Lambda, c, k]$ stemming from a basis matrix B of dimensions $m \times n$ is called a *solved task* if $k = n$.

Using Proposition 1, a solved task ensures the existence of a vector v and an index i such that $\mathcal{N}({}^tvB) < \mathcal{N}(B_i)$ and $v_i \neq 0$, allowing the improvement of B.

3.4 The Algorithms

In this section, one presents the pseudo codes of the algorithms, and gives some hints on the way they work. The rigorous proofs of the algorithms are given in Appendix A.

Algorithm ComputeSparsestBasis(B). It is the main algorithm. It takes a basis B as an input and returns a sparsest basis B' equivalent to B. It relies on the algorithm EnhanceBasis(B) which either returns an equivalent basis B' with $\mathcal{N}(B') < \mathcal{N}(B)$ or proves that B was a sparsest basis. Thus, ComputeSparsestBasis(B) iterates calls to EnhanceBasis(B) until the basis is a sparsest one.

```
   Input: B a basis matrix of dimensions m × n
   Output: B', a sparsest basis matrix equivalent to B
1  begin
2      B' ← B ; a ← true ;
3      while a do
4          a, B' ← EnhanceBasis(B') ;
5      return B' ;
```

Algorithm 1: ComputeSparsestBasis(B)

Algorithm EnhanceBasis(B). It relies on the algorithms BasisToSolvedTask and EnhanceBasisUsingSolvedTask. The algorithm BasisToSolvedTask(B) builds a solved task stemming from B if it exists, or returns the empty set if no such solved task exists. If such a solved task can be computed, Algorithm EnhanceBasisUsingSolvedTask is used to improve the basis B.

```
   Input: B a basis matrix of dimensions m × n
   Output: One of the two cases: false and B if B is a sparsest basis matrix ;
           true and a basis matrix B' equivalent to B such that N(B') < N(B)
           otherwise
1  begin
2      t ← BasisToSolvedTask(B) ;
3      if t ≠ ∅ then
4          B' ← EnhanceBasisUsingSolvedTask(t, B) ;
5          return true, B' ;
6      else
7          return false, B ;
```

Algorithm 2: EnhanceBasis(B)

Algorithm BasisToSolvedTask(B). It looks for a solved task by exploring a binary tree. It makes use of a stack, initially filled with the initial task. At each step of the loop, a task t (where k columns of B have been processed) is popped and two new candidate tasks t_1 and t_2 are built by processing the column $k+1$ of B. These candidates are pushed onto the stack if they are actually tasks (which is checked by Algorithm IsTask).

Input: a basis matrix B
Output: a solved task t, stemming from B, or $\emptyset$ if no such solved task exists (*i.e.* B is a sparsest basis)

```
1  begin
2      Let S_t be an empty stack ;
3      Push the initial task t_0 onto S_t ;
4      while S_t ≠ ∅ do
5          Pop a task t = TASK[A, Λ, c, k] from S_t ;
6          if k < n then
7              // The task t is not solved
8              Let w be the transpose of the (k+1)^th column of B ;
9              A' ← (A / w) ; Λ' ← (Λ / w) ;
10             t_1 ← [A', Λ, c, k+1] ;                    // t_1 may be a task
11             if IsTask(t_1, B) then  Push t_1 onto S_t ;
12             ;
13             t_2 ← [A, Λ', c+1, k+1] ;                  // t_2 may be a task
14             if IsTask(t_2, B) then  Push t_2 onto S_t ;
15             ;
16         else
17             return t ;
18     return ∅ ;
```

Algorithm 3: BasisToSolvedTask(B)

Algorithm EnhanceBasisUsingSolvedTask(t, B). It basically finds a vector v and an index i such that $\mathcal{N}({}^tvB) < \mathcal{N}(B_i)$ and $v_i \neq 0$, which necessarily exist since t is a solved task. It then builds an improved basis B' by making a copy of B and replacing the row B'_i by ${}^tvB$.

Algorithm IsTask(t). It checks where a candidate task t is indeed a task, by checking if t satisfies the **LCP** and **IZP** properties. The goal of this function is to detect as soon as possible useless tasks (*i.e.* tasks that will not help improving our basis).

Algorithm NextVector(u). The goal of NextVector is to iterate the p-tuples of $\mathbb{Z}^p$. This is needed in Algorithm EnhanceBasisUsingSolvedTask to obtain a solution v of the system $(\mathcal{S})$, which is composed of equations (*i.e.* $Ax = 0$) and inequations (*i.e.* $\Lambda x \not\equiv 0$). Indeed, the only way we have found to obtain solutions

Input: a solved task $t = \text{TASK}[A, \Lambda, c, n]$, stemming from B of size $m \times n$
Output: B' a basis matrix equivalent to B such that B and B' only differ by one row and $\mathcal{N}(B') < \mathcal{N}(B)$

begin
  $B' \leftarrow B$;
  Compute a basis of $\text{Ker}(A)$ and store it columnwise in the matrix K of dimensions $m \times p$, where $p = m - \text{Rank}(A)$;
  Compute $\mathcal{U} = \{i \in [\![1, m]\!], c < \mathcal{N}(B_i)\}$;
  if $p = 1$ **then**
    $v \leftarrow$ the unique column of K ;
    Choose $i \in \mathcal{U}$ such that $v_i \neq 0$;
  else
    $i \leftarrow 0$; $u \leftarrow 0$; *// u is the zero vector of dimension p*
    while $i = 0$ **do**
      $u \leftarrow \mathsf{NextVector}(u)$;
      $v \leftarrow Ku$;
      if $\Lambda v \not\equiv 0$ **then**
        // v is a nonzero solution of $(\mathcal{S})$
        Choose $i \in \mathcal{U}$ such that $v_i \neq 0$ if it exists ;
  $B'_i \leftarrow {}^t vB$;
  Multiply B'_i by the LCM of the denominators of the elements of B'_i ;
  Divide B'_i by the GCD of the elements of B'_i ;
  return B' ;

Algorithm 4: $\mathsf{EnhanceBasisUsingSolvedTask}(t, B)$

Input: $t = [A, \Lambda, c, k]$, satisfying all conditions of a task stemming from a basis matrix B of dimensions $m \times n$, except **LCP** and **IZP** properties
Output: **true** if t satisfies **LCP** and **IZP** (*i.e.* t is a task), **false** otherwise

begin
  ***// LCP** (resp. **IZP**) is true if the tests lines 6 (resp. 11) and 3 are false*
  if $\left(\exists i \in [\![1, c]\!], \text{Rank}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \text{Rank}(A)\right)$ *or* $(\text{Rank}(A) = m)$ **then**
    return false ;
  // One has $\left(\forall i, \text{Rank}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \text{Rank}(A) + 1\right)$ *and* $(\text{Rank}(A) \leq m - 1)$
  if $(\text{Rank}(A) = m - 1)$ *and A contains at least one zero column* **then**
    return false ;
  // One has $(\text{Rank}(A) \leq m - 2)$ *or A does not have any zero column*
  Compute A_{RREF}, the RREF form of A ;
  Compute $\mathcal{U} = \{i \in [\![1, m]\!], c < \mathcal{N}(B_i)\}$;
  if $\forall j \in \mathcal{U}$, *$A_{RREF}$ is row-unit of index j* **then**
    return false ;
  return true ;

Algorithm 5: $\mathsf{IsTask}(t, B)$

of $Ax = 0$ also satisfying $\Lambda x \not\equiv 0$ consists in iterating some solutions of $Ax = 0$ until $\Lambda x \not\equiv 0$ is satisfied.

Input: an integer vector u of dimension p
Output: the vector following u for some fixed ordering on $\mathbb{Z}^p$

Algorithm 6: NextVector(u)

4 Complexity and Improvements

4.1 Complexity

The bottleneck of ComputeSparsestBasis is located in BasisToSolvedTask. Indeed, the number of while loops performed in BasisToSolvedTask can be close to $2^{n+1} - 1$ in the worst case (*i.e.* when the binary tree is almost completely explored). It is the only place where an exponential complexity occurs, since all other operations rely on linear algebra operations (such as computing a nullspace, a RREF, ...).

However, many branches are cut thanks to the line 11 in the algorithm IsTask. The sparser the matrix B is, the more branches are cut. Indeed, let us denote $d = \max\{\mathcal{N}(B_i), i \in [\![1, m]\!]\}$. Suppose that, in our binary tree, the left (resp. right) child corresponds to adding an equation (resp. an inequation). If the number of inequations c of some task is greater than or equal to d, the set $\mathcal{U}$ at line 10 in IsTask is empty, thus IsTask returns false. This implies that only the branches starting from the root and going through the right children at most d times will be explored. The number of processed nodes at depth k is equal to $\sum_{i=0}^{d} \binom{k}{i}$. Thus the total number of processed nodes is bounded by $\sum_{k=0}^{n}\sum_{i=0}^{d} \binom{k}{i} = \sum_{i=0}^{d}\sum_{k=i}^{n} \binom{k}{i} = \sum_{i=0}^{d} \binom{n+1}{i+1}$. Easy computations show that $\sum_{i=0}^{d} \binom{n+1}{i+1} \leq 2(n+1)^{d+1}$ which can be much smaller than $2^{n+1} - 1$ (for example when d is much smaller than n). Experimentally, we observed that models with small values d were easily solved.

Finally, the number of calls to EnhanceBasis is bounded by the number of nonzeros of the initial basis B, which is bounded by nd (since the number of nonzeros decreases at least by 1 at each call of EnhanceBasis, except for the last call).

4.2 Implementation

We chose to implement algorithms given in Section 3 using the Computer Algebra software Maple, which natively handles long integers and contains many linear algebra routines with exact coefficients. With no surprise, those algorithms

can be improved because many useless computations are performed. For example, many useless rank computations are done in Algorithm IsTask. The next section describes the improvements of the algorithms given in Section 3.

4.3 Improvements

Computation and Choice of the Solved Task Algorithm BasisToSolvedTask stops when it first encounters a solved task. This solved task may change if one pushes t_2 before t_1 in the Algorithm BasisToSolvedTask. This change has no real impact since it speeds up some examples, and slows down others.

We have experimented another strategy consisting in computing the set of all the solved tasks stemming from B instead of stopping at the first solved task. Once this set is computed, one can choose the solved task that leads to the biggest decrease of the number of nonzeros, and only keep solved tasks with $\mathrm{Rank}(A) = m-1$ if one encounters a solved task with $\mathrm{Rank}(A) = m-1$ at some point. Indeed, solved tasks with $\mathrm{Rank}(A) = m-1$ lead to easy computations in Algorithm EnhanceBasisUsingSolvedTask since $p = 1$.

It is not clear whether the strategy of computing all the solved tasks is better or worst than stopping on the first solved task. Indeed, searching all solved tasks is obviously more costly, but choosing a suitable solved task might decrease the number of subsequent calls to Algorithm EnhanceBasis.

Using Reduced Row Echelon Forms It is possible to request more properties for a task $\textsc{task}[A, \Lambda, c, k]$. For example, one can request A to be in reduced row echelon form with no zero rows. Moreover, one can request the rows of Λ to be reduced w.r.t. the matrix A in the following sense: a row b is reduced w.r.t. A if the row contains a zero at the location of the pivots of A (reducing a row Λ_i by A can be done by subtracting multiple of rows of A to Λ_i to get zeros at the positions of the pivots of A).

Those requirements have several advantages, especially in Algorithms 4 and 5. In Algorithm 4, the computation of $\mathrm{Rank}(A)$ is immediate since it equals the number of rows of A. Moreover the condition $\mathrm{Rank}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \mathrm{Rank}(A)+1$ can be checked immediately: indeed, since Λ_i is reduced w.r.t. A, the condition is true if and only if the row Λ_i is not the zero row. In Algorithm 5, the computation of the matrix K is immediate and can be done by simply rearranging the entries of A in a new matrix K.

Finally, since the rows Λ_i are reduced w.r.t. A, it is easier to detect that two rows Λ_i and Λ_j are equal modulo a linear combination of lines of A. Indeed, if this is the case, both lines are necessarily proportional (and then one can discard one of them).

5 Benchmarks

We have tested our methods on some models taken from the BioModels database [1] (accessed on June 16th, 2014). Among the curated models, we have selected

all the models involving only one compartment, with rational and integer stoichiometric coefficients. After rejecting models without conservation laws, we ended up with 214 models.

After computing a basis of linear conservation laws for each of the 214 models, our method detected that 141 bases were already sparsest ones. In the rest of the section, one only considers the remaining 73 models.

In this section, the *CSB* version denotes the non-improved version of the algorithm ComputeSparsestBasis, as described in Section 3, and the *CSB'* version denotes the version based on RREF computations as described in Section 4.3. Timings were measured on a Pentium Xeon 3.40GHz with 32Gb of memory.

Model	**Size of** B	d	**Time (in s)**		**Model**	**Size of** B	d	**Time (in s)**	
			CSB	**CSB'**				**CSB**	**CSB'**
068	3×8	4	0.15	0.06	475	7×23	14	309.8	10.59
064	4×21	18	28.4	1.99	014	8×86	45	> 3000	235.9
183	4×67	61	505.1	36.50	478	11×33	11	419.4	1.85
086	5×17	12	15.99	3.12	153	11×75	38	> 3000	964.6
336	5×18	7	3.68	0.30	152	11×64	32	> 3000	97.46
237	6×26	17	57.93	0.91	334	13×73	50	> 3000	132.6
431	6×27	15	70.68	10.93	019	15×61	13	> 3000	24.36

Fig. 1. Timing comparison between the basic version and the improved version (with B basis matrices of the models and $d = \max\{\mathcal{N}(B_i), i \in [\![1, m]\!]\}$ as described Section 4.1)

Checking that the 141 bases are indeed sparsest ones takes at most 1s for each model. The improved *CSB'* version takes less than 3000s for 69 models out of the 73 models. The remaining 4 models involve heavier computations: model 332 takes around 4000s, model 175 takes around 1 day, computation of models 205 and 457 were stopped after 2 days. These 4 models involve between 15 and 40 conservation laws (*i.e.* B has between 15 and 40 rows), and between 50 and 200 species (*i.e.* B has between 50 and 200 columns). As shown in [3], finding a sparsest basis for the null space of a matrix is NP-hard, so it is not surprising that some models are challenging.

The *CSB'* version is faster than the basic version, usually by a factor of at least 10. Figure 1 shows some timings for a sample of the 69 models. Note that the timings can be different between models with bases of similar sizes (like models 153 and 152).

For each basis B, one can define the ratio $x = \frac{\mathcal{N}(B')}{\mathcal{N}(B)}$ where B' is a sparsest basis equivalent to B. For a non sparsest basis B, this ratio satisfies $0 < x < 1$. Figure 2 represents the frequency of the 69 models, plus the models 332 and 175 involving heavier computations, as a function of the ratio x.

Appendix B shows a comparison between the number of nonzeros obtained after some usual linear algebra methods and our algorithm.

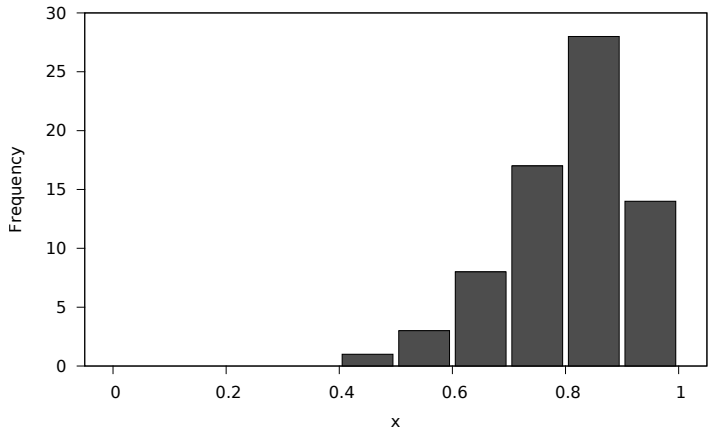

Fig. 2. Number of bases given some proportion $x = \frac{\mathcal{N}(B')}{\mathcal{N}(B)}$

6 Conclusion

We plan to implement our method in Python, for different reasons: it is a free software, it handles large integers natively, and it is easy to interface with other software. One could also implements our method in C (by using the multiprecision arithmetic library GMP [6]) for performance reasons.

Looking back to the problem of getting "good" sets of conservation laws we discussed in Section 1, it is not always possible to have a basis with the less possible negative values and a sparsest basis at the same time. There probably exists a compromise between these two properties, which is left for further work.

Acknowledgements. We would like to thank the reviewers for their helpful comments and Sylvain Soliman for his help on the Nicotine software (`http://contraintes.inria.fr/~soliman/nicotine.html`).

References

1. Biomodels database. `http://www.ebi.ac.uk/biomodels-main/publmodels`. [Online; accessed June 16th, 2014].
2. M.W. Berry, M.T. Heath, I. Kaneko, M. Lawo, R.J. Plemmons, and R.C. Ward. An algorithm to compute a sparse basis of the null space. *Numerische Mathematik*, 47(4):483–504, 1985.
3. Thomas F Coleman and Alex Pothen. The null space problem i. complexity. *SIAM J. Algebraic Discrete Methods*, 7(4):527–537, October 1986.
4. Thomas F Coleman and Alex Pothen. The null space problem ii. algorithms. *SIAM Journal on Algebraic Discrete Methods*, 8(4):544–563, 1987.
5. Claus Fieker and Damien Stehlé. Short bases of lattices over number fields. In Guillaume Hanrot, François Morain, and Emmanuel Thomé, editors, *Algorithmic*

Number Theory, volume 6197 of *Lecture Notes in Computer Science*, pages 157–173. Springer Berlin Heidelberg, 2010.
6. Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. `http://gmplib.org/`.
7. A.K. Lenstra, H.W. Lenstra, and L. Lovász. *Factoring Polynomials with Rational Coefficients*. Afdeling Informatica: IW. Mathematisch Centrum, Afdeling Informatica, 1982.
8. Mikhail A. Panteleev, Mikhail V. Ovanesov, Dmitrii A. Kireev, Aleksei M. Shibeko, Elena I. Sinauridze, Natalya M. Ananyeva, Andrey A. Butylin, Evgueni L. Saenko, and Fazoil I. Ataullakhanov. Spatial propagation and localization of blood coagulation are regulated by intrinsic and protein c pathways, respectively. *Biophysical Journal*, 90(5):1489 – 1500, 2006.
9. S. Roman. *Advanced Linear Algebra*. Graduate Texts in Mathematics. Springer, 2007.
10. Herbert M. Sauro and Brian Ingalls. Conservation analysis in biochemical networks: computational issues for software writers. *Biophysical Chemistry*, 109(1):1–15, 2004.
11. Christophe H. Schilling, David Letscher, and Bernhard Ø. Palsson. Theory for the Systemic Definition of Metabolic Pathways and their use in Interpreting Metabolic Function from a Pathway-Oriented Perspective. *Journal of Theoretical Biology*, 203(3):229–248, April 2000.
12. Stefan Schuster and Claus Hilgetag. On elementary flux modes in biochemical reaction systems at steady state. *Journal of Biological Systems*, 2(2):165–182, 1994.
13. Sylvain Soliman. Invariants and Other Structural Properties of Biochemical Models as a Constraint Satisfaction Problem. *Algorithms for Molecular Biology*, 7(1):15, 2012.
14. Ravishankar Rao Vallabhajosyula, Vijay Chickarmane, and Herbert M. Sauro. Conservation analysis of large biochemical networks. *Bioinformatics*, 22(3):346–353, 2006.

A Proof of Algorithms

Each subsection of this section proves an algorithm, by proving it stops and returns the correct result. When needed, some mathematical properties are proved at the beginning of the subsections.

A.1 ComputeSparsestBasis(B)

Halting The algorithm halts because EnhanceBasis returns either **false**, B or **true**, B' and this last case cannot happen indefinitely as the number of nonzeros in B' is strictly decreasing and bounded by 0.

Correction The algorithm halts when $a =$ **false**, *i.e.* when EnhanceBasis detects that B' is a sparsest basis.

A.2 EnhanceBasis(B)

Halting Trivial.

Correction BasisToSolvedTask returns either a solved task or $\emptyset$ if no such solved task exists. If $t \neq \emptyset$, EnhanceBasisUsingSolvedTask computes a new basis matrix B' with $\mathcal{N}(B') < \mathcal{N}(B)$, otherwise no solved task exists, which proves that B is a sparsest basis.

A.3 EnhanceBasisUsingSolvedTask(t, B)

Lemma 1 (Finite union of vector subspaces). *Let $\mathbb{K}$ be an infinite field. If E is a $\mathbb{K}$-vector subspace, then every finite union of proper subspaces of E is strictly included in E.*

Proof. See [9]. □

Theorem 2 (Characteristic theorem of solutions of $(\mathcal{S})$). *Consider $A \in \mathbb{Q}^{r\times m}, \Lambda \in \mathbb{Q}^{c\times m}$, $K \in \mathbb{Q}^{m\times q}$ a matrix representing a basis of* $\mathrm{Ker}(A)$ *stored columnwise, and the system $(\mathcal{S}) : \begin{cases} Ax = 0 \\ \Lambda x \neq 0 \end{cases}$.*

The following assertions are equivalent:

1. *$(\mathcal{S})$ has nonzero solutions in $\mathbb{Q}^m$,*
2. *$\left(\forall i \in [\![1, c]\!], \mathrm{Rank}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \mathrm{Rank}(A) + 1\right)$ and $(\mathrm{Rank}(A) \leq m - 1)$,*
3. *$\exists u \in \mathbb{Z}^q \setminus \{0\}, \Lambda K u \not\equiv 0$.*

Proof. (1) ⇒ (3): Take a nonzero solution v in $\mathbb{Q}^m$ of $(\mathcal{S})$. Thus, there exists u' in $\mathbb{Q}^q \setminus \{0\}$ such that $v = Ku'$ (since the columns of K are a basis of $\mathrm{Ker}(A)$). Taking $u = \lambda u'$ (with a suitable integer λ such that u belongs to $\mathbb{Z}^q \setminus \{0\}$), one has $Ku = \lambda v$. Since λv is also a nonzero solution of $(\mathcal{S})$, one has $\Lambda K u \not\equiv 0$.

(3) ⇒ (1): Take $v = Ku$. Then $Av = 0$ and $\Lambda v \not\equiv 0$, so v is a nonzero solution of $(\mathcal{S})$.

(1) ⇒ (2): Consider a nonzero solution v of $(\mathcal{S})$. Since v is nonzero and satisfies $Av = 0$, one has $\mathrm{Ker}(A) \neq \{0\}$ thus $\mathrm{Rank}(A) \leq m - 1$ (according to the rank-nullity theorem). Let us suppose that $\mathrm{Rank}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \mathrm{Rank}(A)$ for some i. Then, Λ_i is a linear combination of rows of A so $\Lambda_i v = 0$, hence v is not a solution of $(\mathcal{S})$, contradiction. We conclude that $\forall i \in [\![1, c]\!]$, $\mathrm{Rank}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \mathrm{Rank}(A) + 1$ and $\mathrm{Rank}(A) \leq m - 1$.

(2) ⇒ (1): The set of solutions of $(\mathcal{S})$ is $V = \mathrm{Ker}(A) \setminus \bigcup_{i=1}^{c} \mathrm{Ker}(\Lambda_i) = \mathrm{Ker}(A) \setminus \bigcup_{i=1}^{c} \mathrm{Ker}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix}$. For any $i \in [\![1, c]\!]$, $\mathrm{Ker}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix}$ is a proper vector subspace of $\mathrm{Ker}(A)$ as $\mathrm{Rank}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \mathrm{Rank}(A) + 1$. According to Lemma 1, $\bigcup_{i=1}^{c} \mathrm{Ker}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix}$ is strictly included in $\mathrm{Ker}(A)$. Moreover, since $\mathrm{Rank}(A) \leq m-1$, one has $\mathrm{Ker}(A) \neq \{0\}$ which implies $V \setminus \{0\} \neq \emptyset$. □

Halting If $p = 1$, it is trivial that the algorithm halts. If $p \neq 1$, one needs to check that the while loop halts:

- As t is a solved task, there exists a nonzero solution v in $\mathbb{Q}^m$ of $(\mathcal{S})$ and $j \in \mathcal{U}$ such that $v_j \neq 0$ and $c < \mathcal{N}(B_j)$ according to Proposition 1.
- Using the third point of Theorem 2, there exists a vector u in $\mathbb{Z}^q$ such that the vector $v = Ku$ is a nonzero solution of $(\mathcal{S})$. Hence, the while loop will eventually reach such a u since NextVector enumerates all elements of $\mathbb{Z}^q$.

Correction At line 2, $B' \leftarrow B$. At the end of the algorithm, B'_i is modified and contains more zeros than B_i. Indeed, $\mathcal{N}(B'_i) = c$ and since $i \in \mathcal{U}$, $c < \mathcal{N}(B_i)$. Finally, B' is also a basis matrix since $B'_i = {}^t vB$ with $v_i \neq 0$.

A.4 NextVector

There is some freedom in coding this algorithm (which is not given in this paper), in particular the ordering in which the p-tuples are output. However, to ensure the termination of Algorithm EnhanceBasisUsingSolvedTask, one requires the Algorithm NextVector(u) to iterate all the p-tuples of $\mathbb{Z}^p$. This can be achieved for example by starting from the zero tuple, and enumerating the tuples of $\mathbb{Z}^p$ by increasing 1-norm (where the 1-norm of a tuple is the sum of the absolute values), and by lexicographic order for tuples of the same 1-norm. For example, when $p = 2$, the p-tuples can be enumerated in the following way: $(0, 0)$, $(-1, 0)$, $(0, -1)$, $(0, 1)$, $(1, 0)$, $(-2, 0)$, $(-1, -1)$, $(-1, 1)$, $(0, -2)$, $(0, 2)$, ...

One could also rely on a random number generator, provided it has the property to eventually generate any p-tuple with a nonzero probability (in order to ensure the halting of Algorithm EnhanceBasisUsingSolvedTask).

A.5 BasisToSolvedTask(B)

Halting Consider a current task t. The algorithm stops if t is solved (*i.e.* $k = n$). Otherwise, t is not a solved task and generates new tasks, obtained from t, with $k + 1$ columns processed (instead of k). This last case cannot occur indefinitely.

Correction Consider a non solved task t inside the while loop. One creates the object t_1 (resp. t_2) corresponding to the cancellation (resp. the non cancellation) of the coefficient number $k + 1$ of the linear combinations of the lines of B. The objects t_1 and t_2 are possibly discarded thanks to the function IsTask if they are not tasks (*i.e.* if they cannot be used to increase the number of zeros). Consequently, all cases are considered and the function will return $\emptyset$ if and only if there does not exist any solved task stemming from B.

A.6 IsTask(t, B)

Proposition 2. *Let M be a matrix. Then M has a zero column if and only if* $\mathrm{Ker}(M)$ *contains at least one vector with exactly one nonzero coefficient.*

Proof. Trivial □

Corollary 1. *Let M be a matrix. Then M does not have any zero column if and only if* $\mathrm{Ker}(M) \setminus \{0\}$ *only contains vectors with at least two nonzero coefficients.*

Proposition 3. *Take $A \in \mathbb{Q}^{r\times m}, \Lambda \in \mathbb{Q}^{c\times m}$ and the system $(\mathcal{S}) : \begin{cases} Ax = 0 \\ \Lambda x \not\equiv 0 \end{cases}$. If $(\mathcal{S})$ has nonzero solutions and* $\mathrm{Rank}(A) \leq m-2$*, then $(\mathcal{S})$ has nonzero solutions with at least two nonzero coefficients.*

Proof. $\mathrm{Rank}(A) \leq m-2 \Leftrightarrow \mathrm{Dim}(\mathrm{Ker}(A)) \geq 2$ with the rank-nullity theorem. Consequently, there exist at least two independent nonzero vectors v_1 and v_2 solutions of $Ax = 0$. Consider a nonzero solution v of $(\mathcal{S})$. By a topology argument the vector $v+\varepsilon_1 v_1+\varepsilon_2 v_2$ is also solution of $(\mathcal{S})$ for any ε_1 and ε_2 satisfying $|\varepsilon_1| < \varepsilon$ and $|\varepsilon_2| < \varepsilon$ (for a suitable small fixed $\varepsilon > 0$). Suppose that $v + \varepsilon_1 v_1 + \varepsilon_2 v_2$ has exactly one nonzero coefficient for any $|\varepsilon_1| < \varepsilon$ and $|\varepsilon_2| < \varepsilon$. That would imply that both v_1 and v_2 have exactly one nonzero coefficient at the same position, which is impossible since v_1 and v_2 are linearly independent. Consequently, V contains at least one nonzero vector with two nonzero coefficients.

Theorem 3. *Consider $A \in \mathbb{Q}^{r\times m}, \Lambda \in \mathbb{Q}^{c\times m}$ and the system $(\mathcal{S}) : \begin{cases} Ax = 0 \\ \Lambda x \not\equiv 0 \end{cases}$.*

The following assertions are equivalent :

1. *$(\mathcal{S})$ has nonzero solutions with at least two nonzero coefficients,*
2. *$(\mathcal{S})$ has nonzero solutions and at least one of the two following conditions is true:*
   - $\mathrm{Rank}(A) \leq m-2$*,*
   - *A does not have any zero column.*

Proof. (1) ⇒ (2): Since $(\mathcal{S})$ has nonzero solutions, then $\mathrm{Rank}(A) \leq m-1$. If the condition $\mathrm{Rank}(A) \leq m-2$ is not true, then one has $\mathrm{Rank}(A) = m-1$. Consequently, $\mathrm{Ker}(A)$ is generated by one vector containing at least two nonzero coefficients. According to Corollary 1, A does not have any zero column.

(2) ⇒ (1): Direct consequence of Corollary 1 if $\mathrm{Rank}(A) = m-1$, or Proposition 3 if $\mathrm{Rank}(A) \leq m-2$. □

Definition 2. *A matrix A is **row-unit of index** j if there exists a row of A with only one nonzero coefficient, which is at position j.*

Theorem 4. *Consider a matrix A' in $\mathbb{Q}^{r\times m}$ under reduced row echelon form. Let K be a matrix containing a basis of* $\mathrm{Ker}(A')$ *stored columnwise. Suppose that* $\mathrm{Ker}(A') \neq \{0\}$*. For any index j, the following assertions are equivalent:*

1. *A' is row-unit of index j,*

2. $K_j = 0$.

Proof. (1) $\Rightarrow$ (2): There exists a row $A'_i = \begin{pmatrix}0 \cdots 0\ 1\ 0 \cdots 0\end{pmatrix}$ where the 1 is at position j, for some i in $[\![1, r]\!]$. Thus, any solution v (in particular all elements of the basis K) of $A'v = 0$ must satisfy $v_j = 0$, hence $K_j = 0$.

(2) $\Rightarrow$ (1): From $A'K = 0$, one has ${}^tK{}^tA' = 0$. Since $K_j = 0$, the column j of ${}^tK$ is zero, which implies that the canonical vector e_j belongs to $\mathrm{Ker}({}^tK)$ using Proposition 2. Hence, the row $l = {}^te_j = \begin{pmatrix}0 \cdots 0\ 1\ 0 \cdots 0\end{pmatrix}$, where the 1 is at position j, is a linear combination of the rows of A'. Since A' is in reduced row echelon form, if the combination l of rows of A' involved strictly more than one row of A', l would at least involve two nonzero coefficients (corresponding to the pivots). Thus, l is a row of A' and A' is row-unit of index j. □

Halting Trivial.

Correction

1. If the first condition (line 3) is true, then $(\mathcal{S})$ has no nonzero solutions in $\mathbb{Q}^m$, so t does not satisfy condition **LCP**, and one returns **false**. Otherwise, one has $(\forall i \in [\![1, c]\!], \mathrm{Rank}\begin{pmatrix}A\\ \Lambda_i\end{pmatrix} = \mathrm{Rank}(A) + 1)$ and $(\mathrm{Rank}(A) \leq m - 1)$ so $(\mathcal{S})$ has nonzero solutions according to Theorem 2.
2. If the second condition (line 6) is true, since $(\mathcal{S})$ admits nonzero solutions and thanks to Theorem 3, $(\mathcal{S})$ does not have solutions with at least two nonzero coefficients, so t does not satisfy **LCP**, and one returns false. Otherwise, t satisfies **LCP**.
3. If the third condition (line 11) is true, then for any $j \in \mathcal{U}$, one has $K_j = 0$ thanks to Theorem 4. Consequently, any solution v of $(\mathcal{S})$ satisfies $v_j = 0$ for all $j \in \mathcal{U}$. Therefore, t does not satisfy **IZP** and one returns false.
   Otherwise, there exists a $j \in \mathcal{U}$ such that $K_j \neq 0$ thanks to Theorem 4, so there exists a column w of K such that $w_j \neq 0$. It may happen that w_j is not solution of $(\mathcal{S})$. In that case, consider a nonzero solution u of $(\mathcal{S})$ with at least two nonzero coefficients. By a topology argument, the vector $\bar{u} = u + \varepsilon_1 w$ is also solution of $(\mathcal{S})$ for any rational ε_1 satisfying $|\varepsilon_1| < \varepsilon$ (where ε is a small rational). Finally, one can choose a suitable ε_1 to obtain the condition $\bar{u}_j \neq 0$, and $(\mathcal{S})$ satisfies **IZP**. One then returns **true**.

B Comparison with some Matrix Algorithms

We consider a sample of 10 models taken from the 61 models where computations end in less than 3000s. We present here the number of nonzeros of the initial bases and of the bases obtained after using our algorithm (CSB), the Reduced Row Echelon Form (RREF), the LLL algorithm [7] and the Hermite Normal Form (HNF). When comparing our algorithm with the usual linear algebra algorithms we used, one sees that the sparsest bases are not always reached by these linear algebra algorithms and that they sometimes make it worst.

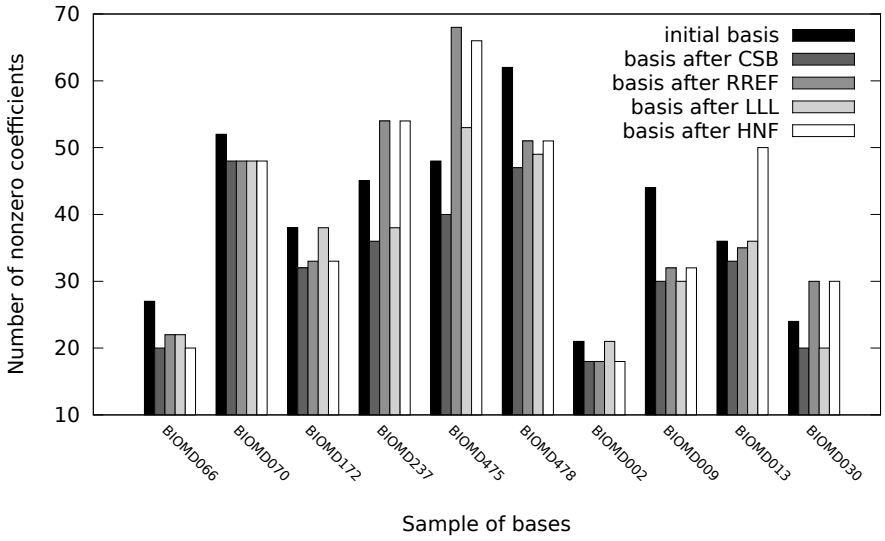

Fig. 3. Number of nonzeros after some matrix algorithms