Degroebnerization and its applications: a new approach for data modelling.

EXTENDED ABSTRACT FOR MEGA 2021

Michela Ceria, Teo Mora, Andrea Visconti

Abstract

In this work, we propose a new Computer Algebra-based method to give a polynomial model for data. In particular, we refer to the problem of reverse engineering for gene regulatory networks, giving an alternative to the method proposed by Laubenbacher and Stigler. We show that, in order to give the required model, which is composed by polynomials in normal form with respect to the vanishing ideal of the given data points, it is not necessary to compute Gröbner bases or performing any step of Buchberger reduction. It is possible to recover the required polynomials by means of linear algebra and combinatorics.

1 Introduction

Gröbner bases' theory, besides being used in the framework of polynomials systems' solving, has found applications also in the data modelling framework.
As an example, in [13], the authors propose a Computer Algebra-based approach to study gene regulatory networks as dynamical systems, starting from measured data, which are essentially states of n given genes in the networks at different times. Any state is considered as a point in $(\mathbb{Z}_p)^n$, where p is a prime. Let $\mathbf{X} = \{P_1, ..., P_N\}$ the set of such points. In order to model their data, they want to construct polynomial functions that are *in normal form*, namely reduced modulo the ideal $I(\mathbf{X})$ whose variety is given by the points.
They first compute a separator family for $\mathbf{X}$, so they have one polynomial for each point, say Q_i, $1 \leq i \leq N$, such that for $i, j \in \{1, ..., N\}$, $Q_i(P_j) = 1$ if $i = j$ and $Q_i(P_j) = 0$ otherwise. Then, they reduce modulo a Gröbner basis of the ideal of points $I(\mathbf{X})$ some polynomials they find as linear combinations of separator polynomials.
Using separator polynomials to model data, is something that can be found also in other disciplines. One for all is Algebraic Statistics. In [11], the authors define the *indicator function* to represent fractions of full factorial desings. Such a function is again a combination of separator polynomials.

Recently, a new approach in Computer Algebra has been proposed, that is *Degroebnerization* [18, 14, 19], which has been explicitly expressed and endorsed in [17, Vol.3, 40.12,41.15]. Degroebnerization means avoiding Gröbner basis computation and Buchberger's reduction as much as possible, leaving their use to the only cases in which it is really necessary. This is due to the computational intensity of Gröbner bases' computation. Such a new approach consists then in finding new ways to solve practical problems that have been originally solved using Gröbner basis computation and Buchberger's reduction. Usually, the "new ways" that can be found consist in using linear algebra and combinatorial methods.
This work places itself in this framework. Indeed, here we show a new, degroebnerized approach to solve the problem proposed in [13].
After some notation (Section 2) and preliminaries on Laubenbacher-Stigler's work (Section 3), we recall a combinatorial method to compute squarefree separator polynomials for a finite set of points (Section 4). Being squarefree, the separators we find avoid redundant multiplicative factors that present, instead, while using the other separators' formulas present in literature [12, 14].
Then, in Section 5, we recall Lundqvist's formula for normal forms (Proposition 7) and an efficient combinatorial method to compute a basis for the quotient of the polynomial ring modulo the ideal of points $I(\mathbf{X})$, referring to [5]. Unlike [13] where all data are intentionally expressed through polynomials, our method completely and intentionally avoids the computation of any polynomial, depending only on simple comparisons among the coordinates of the points. For efficiency reasons, it relies on two data structures, namely the *point trie* [12], to encode information about points and the Bar Code [1, 3], to encode information on the quotient algebra basis. Using the quotient algebra basis and the given set $\mathbf{X}$, it is possible to compute the normal form of any polynomial modulo the ideal of points, only via evaluations and matrix multiplication.
In Section 6, we show how to implement the above combinatorial method and some testing activity.
Finally, in Section 7, we present some perspectives.

2 Notation

In this paper, we mainly rely on the notation of [17].
In particular, we denote by $\mathcal{P} := \mathbf{k}[x_1, ..., x_n]$ the polynomial ring over the field $\mathbf{k}$, on the n variables $\{x_1, ..., x_n\}$. The *semigroup of terms* in the same variables is

$$\mathcal{T} := \{x_1^{\gamma_1} \cdots x_n^{\gamma_n} | \, \gamma_1, ..., \gamma_n \in \mathbb{N}\}.$$

For each element $t \in \mathcal{T}$:

- $\deg(t) = \sum_{i=1}^{n} \gamma_i$ is the *degree* of t;
- $\deg_i(t) = \gamma_i$, $1 \leq i \leq n$ is the i-degree, namely its degree in the variable x_i.

A *semigroup ordering* is a total ordering over $\mathcal{T}$ such that it respects multiplication, namely if $s, t_1, t_2 \in \mathcal{T}$, it holds

$$t_1 < t_2 \Rightarrow st_1 < st_2.$$

If such an ordering is also a well ordering, then we have a *term ordering*. We are interested in a specific term ordering, namely the *lexicographical ordering* (Lex for short) $<$, induced by $x_1 < ... < x_n$, defined as follows. Given $t = x_1^{\gamma_1} \cdots x_n^{\gamma_n}, s = x_1^{\delta_1} \cdots x_n^{\delta_n} \in \mathcal{T}$, we say that $t < s$ if and only if there is a $j \in \{1, ..., n\}$ such that $\gamma_j < \delta_j$ and, moreover, $\gamma_i = \delta_i$, for each $j \leq i \leq n$. Once fixed a term ordering on $\mathcal{T}$, if $f \in \mathcal{P}$ is a polynomial, we can define (and identify in a constructive way) its *leading term* $\mathsf{T}(f)$, namely its maximal term with respect to the given ordering.
We call *semigroup ideal* a subset $J \subseteq \mathcal{T}$ that is closed under multiplication by any element of $\mathcal{T}$, namely if $t \in J$ then $st \in J$, for each $s \in \mathcal{T}$. On the other hands, a subset $\mathsf{N} \subseteq \mathcal{T}$ is an *order ideal if* it is closed under division by terms, namely if $t \in \mathsf{N}$ then $s \in \mathsf{N}$, for each divisor s of t.
Given an ideal I of $\mathcal{P}$, we can define the set $\mathsf{T}\{I\} := \{\mathsf{T}(g), \, g \in I\}$ and the semigroup ideal of leading terms $\mathsf{T}(I) := \{t\mathsf{T}(g) | \, t \in \mathcal{T}, g \in I\}$. The minimal generating set for $\mathsf{T}(I) = \mathsf{T}\{I\}$ is the *monomial basis* of I and it is denoted by $\mathsf{G}(I)$. Finally the order ideal $\mathsf{N}(I) := \mathcal{T} \setminus \mathsf{T}(I)$ is called *Gröbner escalier* of the ideal I. If $f \in I$ is a polynomial, we call *normal form* of f with respect to I the unique polynomial $\mathrm{Nf}(f) := \sum_{t \in A} c_t t$, such that A is a basis of $\mathcal{P}/I$ as $\mathbf{k}$-vector space and $f - \mathrm{Nf}(f) \in I$. In particular, in this paper, our basis for $\mathcal{P}/I$ as $\mathbf{k}$-vector space will be $A := \mathsf{N}(I)$, with respect to the lexicographical ordering.

Given a finite set of simple points $\mathbf{X} = \{P_1, ..., P_N\} \subset \mathbf{k}^n$, let us denote by $P_i = (a_{1i}, ..., a_{ni})$, $1 \leq i \leq N$, the coordinates of such points and by $I(\mathbf{X})$ the ideal of all polynomials vanishing on $\mathbf{X}$. For $1 \leq i, j \leq N$, let

$$c_{i,j} := \begin{cases} 0, & \text{if } i = j \\ \min\{h : \, 1 \leq h \leq n, \, a_{hi} \neq a_{hj}\} & \text{otherwise} \end{cases}$$

We define the following Lagrange-like polynomials, that will be building factors for separator polynomials (see [14])

$$p_{i,j}^{[c_{i,j}]} = \frac{x_{c_{i,j}} - a_{c_{i,j},j}}{a_{c_{i,j},i} - a_{c_{i,j},j}}.$$

We continue recalling some definitions from Graph Theory, following the notation of [12].

Definition 1. *We call* tree *a connected acyclic graph. A* rooted tree *is a tree where a special* vertex *(or* node*) called* root *is singled out.*

We call *level* of a vertex the number of edges between the root and such vertex. If v is a vertex different from the root, and u is the vertex preceding v on the path from the root, then u is the *parent* of v and v is a *child* of u. Two vertices with the same parent are called *siblings*. The *leaves* are all vertices u that have no children and a *branch* in the tree is a path from the root to a leaf. We consider only trees whose branches have all the same length, namely the same number of edges composing them.

Definition 2. *A* trie *is a rooted tree in which there is a symbol written on every edge from a fixed alphabet.*

We conclude this section with a very quick recap on Bar Codes, basing on [1, 3].
Bar Codes are bidimensional diagrams that can be used to encode some properties of monomial ideals. They have been proposed for the first time at MEGA 2017 [1], with an application to enumerative combinatorics on stable strongly stable ideals. In [2, 3, 4, 5] other applications of Bar Codes have been highlighted.

Definition 3. *A Bar Code* B *is a picture composed by segments, called* bars*, superimposed in horizontal rows, which satisfies conditions a., b. below. Denote by*

- $\mathsf{B}_j^{(i)}$ *the j-th bar (from left to right) of the i-th row (from top to bottom),* $1 \leq i \leq n$*, i.e. the* j-th i-bar *(or the j-th bar at* level *i);*

- $\mu(i)$ *the number of bars of the i-th row*
- $l_1(\mathsf{B}_j^{(1)}) := 1$, $\forall j \in \{1, 2, ..., \mu(1)\}$ *the* 1−length *of the* 1*-bars;*
- $l_i(\mathsf{B}_j^{(k)})$, $2 \leq k \leq n$, $1 \leq i \leq k-1$, $1 \leq j \leq \mu(k)$ *the* i-length *of* $\mathsf{B}_j^{(k)}$*, i.e. the number of i-bars lying over* $\mathsf{B}_j^{(k)}$

a. $\forall i, j$, $1 \leq i \leq n-1$, $1 \leq j \leq \mu(i)$, $\exists! \bar{j} \in \{1, ..., \mu(i+1)\}$ *s.t.* $\mathsf{B}_{\bar{j}}^{(i+1)}$ *lies under* $\mathsf{B}_j^{(i)}$

b. $\forall i_1, i_2 \in \{1, ..., n\}$, $\sum_{j_1=1}^{\mu(i_1)} l_1(\mathsf{B}_{j_1}^{(i_1)}) = \sum_{j_2=1}^{\mu(i_2)} l_1(\mathsf{B}_{j_2}^{(i_2)})$*; we will then say that* all the rows have the same length.

Given a finite set of terms $M \subset \mathcal{T}$ of cardinality m, it is easy to associate a Bar Code B_M to M.
Let us first consider a term $t = x_1^{\alpha_1} \cdots x_n^{\alpha_n} \in \mathcal{T}$ and define, for each $1 \leq i \leq n$, the term $\pi^i(t) = x_i^{\alpha_i} \cdots x_n^{\alpha_n} \in \mathcal{T}$. Of course, this can be done for all the elements of M, getting n sets

$$M^{[i]} := \pi^i(M) := \{\pi^i(t) | t \in M\}, \text{ for } 1 \leq i \leq n.$$

We order M and $M^{[i]}$, $1 \leq i \leq n$ in increasing order with respect to the lexicographical ordering, getting the lists $\overline{M}$ and $\overline{M}^{[i]}$, $1 \leq i \leq n$. Finally, we define the $n \times m$ matrix $\mathcal{M}$, whose rows are exactly the ordered lists $\overline{M}^{[i]}$, $1 \leq i \leq n$.

Definition 4. *The* Bar Code diagram B *associated to M (or, equivalently, to* $\overline{M}$*) is a* $n \times m$ *diagram, made by segments s.t. the i-th row of* B*,* $1 \leq i \leq n$ *is constructed as follows:*

1. *take the i-th row of* $\mathcal{M}$*, i.e.* $\overline{M}^{[i]}$
2. *consider all the sublists of repeated terms, i.e.* $[\pi^i(t_{j_1}), \pi^i(t_{j_1+1}), ..., \pi^i(t_{j_1+h})]$ *s.t.* $\pi^i(t_{j_1}) = \pi^i(t_{j_1+1}) = ... = \pi^i(t_{j_1+h})$*, noticing that* $0 \leq h < m$
3. *underline each sublist with a segment*
4. *delete the terms of* $\overline{M}^{[i]}$*, leaving only the segments (i.e. the* i-bars*).*

We usually label each 1*-bar* $\mathsf{B}_j^{(1)}$*,* $j \in \{1, ..., \mu(1)\}$ *with the term* $t_j \in \overline{M}$*.*

The Bar Code diagram we obtain is the exactly B_M, the Bar Code we were looking for.
Conversely, now we see how to associate a set of terms to a Bar Code. In order to do that, it is enough to perform the two steps below:

$\mathfrak{B}1$ take the n-th row, composed by the bars $B_1^{(n)}, ..., B_{\mu(n)}^{(n)}$. Let $l_1(B_j^{(n)}) = \ell_j^{(n)}$, for $j \in \{1, ..., \mu(n)\}$. Label each bar $B_j^{(n)}$ with $\ell_j^{(n)}$ copies of x_n^{j-1}.

$\mathfrak{B}2$ For each $i = 1, ..., n-1$, $1 \leq j \leq \mu(n-i+1)$ consider the bar $B_j^{(n-i+1)}$ and suppose that it has been labelled by $\ell_j^{(n-i+1)}$ copies of a term t. Consider all the $(n-i)$-bars $B_{\bar{j}}^{(n-i)}, ..., B_{\bar{j}+h}^{(n-i)}$ lying immediately above $B_j^{(n-i+1)}$; note that h satisfies $0 \leq h \leq \mu(n-i) - \bar{j}$. Denote the 1-lengths of $B_{\bar{j}}^{(n-i)}, ..., B_{\bar{j}+h}^{(n-i)}$ by $l_1(B_{\bar{j}}^{(n-i)}) = \ell_{\bar{j}}^{(n-i)}$,..., $l_1(B_{\bar{j}+h}^{(n-i)}) = \ell_{\bar{j}+h}^{(n-i)}$. For each $0 \leq k \leq h$, label $B_{\bar{j}+k}^{(n-i)}$ with $\ell_{\bar{j}+k}^{(n-i)}$ copies of $t x_{n-i}^k$.

In this work, we focus only on *admissible Bar Codes*, namely those such that the set M obtained by applying $\mathfrak{B}1$ and $\mathfrak{B}2$ to B is an order ideal.

3 Reverse engineering: the previous approach

In this section, we recall the main ideas from [13], on how Computer Algebra methods have been used in reverse engineering for gene regulatory networks.
The paper [13] studies the dynamics of a gene regulatory networks. The authors *adopt the modeling framework of time-discrete multi-state dynamical systems* [13].
In particular, the system is composed by states (that are our points) whose nodes (the variables) represent the genes. The states of each node, so the values each variable can take are the elements in $\mathbb{Z}_p$, with p prime. The dynamical system is represented by the function $F : (\mathbb{Z}_p)^n \to (\mathbb{Z}_p)^n$, sending each point to the next one, so representing the transitions among the states[1].
More precisely, if $\mathbf{X} = \{P_1, ..., P_N\} \subseteq (\mathbb{Z}_p)^n$ are our points, it holds $F(P_i) = P_{i+1}$, for $i = 1, ..., N-1$. We can describe F via the coordinate functions $f_j : (\mathbb{Z}_p)^n \to \mathbb{Z}_p$, $1 \leq j \leq n$, so that $F(P_i) = (f_1(P_i), ..., f_n(P_i))$, for $i = 1, ..., N-1$. Such functions

[1]The values of the prime p indicate how that state changes at each transition; for instance for $p = 3$, -1 means decreases, 0 means stability and $+1$means increases.

are actually *polynomial functions*. What is done in [13], is to compute separately the functions f_j, $1 \leq j \leq n$, from which they reconstruct F. In particular, the polynomials they want to find should be *in normal form*, namely in reduced form modulo the ideal $I(\mathbf{X})$.
The proposed algorithm is composed of three steps:

- compute a particular solution for f_j, $1 \leq j \leq n$;
- compute a Gröbner basis $\mathcal{G}$ of $I(\mathbf{X})$, via Buchberger-Möller algorithm [16], which interpolates on $\mathbf{X}$;
- perform Buchberger reduction on the particular solutions, modulo $\mathcal{G}$.

The particular solution is defined as

$$\sum_{i=1}^{N-1} a_{j,i+2} r_i(x_1, ..., x_n),$$

where $r_i(x_1, ..., x_n) = \prod_{k=1}^{N-1} p_{ik}^{[c_{i,k}]}$. We can easily note that the polynomials r_i form a separator family for our points. Therefore, we can conclude that what it is needed to be computed is *the normal form of separator polynomials*.
In the paper [13], the authors also propose their complexity analysis, stating that

> *In summary, the complexity of the algorithm is*
>
> $$O(n^2N^2) + O((N^3 + N)(\log(p))^2 + N^2n^2) + O(n(N-1))2^{cN+N-1}).$$
>
> *It is quadratic in the number n of variables and exponential in the number N of time points.*

4 Squarefree separator polynomials

Let us consider a finite set of simple points $\mathbf{X} := \{P_1, ..., P_N\} \subset \mathbf{k}^n$. A family of *separator polynomials* for $\mathbf{X}$ is a set $\mathcal{Q} := \{Q_1, .., Q_N\} \subset \mathcal{P}$, such that, for $i, j \in \{1, ..., N\}$, $Q_i(P_j) = 1$ if $i = j$ and $Q_i(P_j) = 0$ otherwise. Such polynomials are quite important for data modelling, since they are the "fundamental bricks" for polynomial interpolation. There are some other approaches to compute them, like Möller algorithm [16, 15] and the formulas in [12, 14]. Möller algorithm has the drawback of involving the whole Gröbner basis computation, while the formulas in [12, 14] compute separators polynomials with useless repeated factors. In [6], we propose a new algorithm for computing separator polynomials without repeated factors.

We recall that (see Section 2), given a finite set of simple points $\mathbf{X} = \{P_1, ..., P_N\} \subset \mathbf{k}^n$, with each point denoted by $P_i = (a_{1i}, ..., a_{ni})$, $1 \leq i \leq N$, we can define, for $1 \leq i, j \leq N$,

$$c_{i,j} := \begin{cases} 0, & \text{if } i = j \\ \min\{h : \ 1 \leq h \leq n, \ a_{hi} \neq a_{hj}\} & \text{otherwise} \end{cases}$$

The building factors for separator polynomials are

$$p_{i,j}^{[c_{i,j}]} = \frac{x_{c_{i,j}} - a_{c_{i,j},j}}{a_{c_{i,j},i} - a_{c_{i,j},j}},$$

which have been also used in [14] to compute his version of the separators.
In order to avoid repetitions of factors of the formula in [14], we first construct the *point trie* associated to $\mathbf{X}$ [12, 6]. This is a rooted tree representing the reciprocal relations among the coordinates of the points. It has as many branches (and so as many leaves) as the points; its nodes are labelled with the indices of the points in $\mathbf{X}$, while the edges are labelled by the coordinates of the points, with the additional rule that two points share the same path from the root to level i ($1 \leq i \leq n$) if and only if they share the same coordinates, from the first one to the i-th one.
Note that the point trie can be constructed *iteratively* on the points, by adding one branch at a time.

Example 5. For the set $\mathbf{X} := \{P_1 = (0, 0), P_2 = (1, 0), P_3 = (0, 1), P_4 = (1, 1)\}$, the point trie is constructed as shown in Figures 1, 2, 3 and 4.

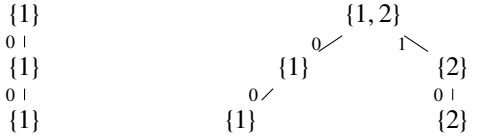


Figure 1: Start with P_1.

Figure 2: Add P_2.

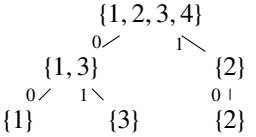


Figure 3: Add P_3.

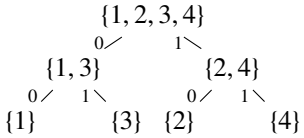


Figure 4: Add P_4.

♢

The trie is the crucial object allowing us to avoid repetitions. Indeed, via this trie we can know how many coordinates are shared by two points and so whether some polynomial already vanishes on some point, thus allowing us to avoid a useless multiplicative factor.
Therefore, in order to construct the separator polynomials we proceed as follows. In the case we have only one point (and so the point trie has only one branch), the desired polynomial is $Q_1 = 1$ and this is also the basis for our procedure. Indeed, suppose now to know the separator family $\{Q_1, ..., Q_{N-1}\}$ for the set $\{P_1, ..., P_{N-1}\} \subset \mathbf{X}$ and let us see how to construct such kind of family for the whole $\mathbf{X}$.
We first add to the trie associated to $\{P_1, ..., P_{N-1}\}$ the new branch associated to P_N and we set $Q_N = 1$. Then we read the new branch from the root to the leaf so, in particular, we consider each level j from the first to the n-th and we do this way:

- let v the node at level j whose label contains N;
- for each sibling node w of v
  1. pick an index i in the label of w;
  2. update Q_N as $Q_N \cdot p_{N,i}^{[j]}$;
- if v is labelled only by the index of N (that is, no points in $\{P_1, ..., P_{N-1}\}$ share the first j coordinates with P_N) then, for each sibling node w and each index i labelling such a node, update Q_i as $Q_i \cdot p_{i,N}^{[j]}$.

The family of separators associated to $\mathbf{X}$ is given by $\{Q_1, ..., Q_N\}$, where the polynomials Q_i not modified by the above steps are simply kept from the family associated to $\{P_1, ..., P_{N-1}\}$.

Example 6. For the set $\mathbf{X} := \{P_1 = (0,0), P_2 = (1,0), P_3 = (0,1), P_4 = (1,1)\}$ of Example 5, the algorithm to compute squarefree separator polynomials performs this way:

P_1 $Q_1 = 1$;

P_2 $Q_2 = p_{2,1}^{[1]}$, $Q_1 = p_{1,2}^{[1]}$;

P_3 $Q_3 = p_{32}^{[1]} p_{3,1}^{[2]}$, $Q_1 = p_{1,2}^{[1]} p_{1,3}^{[2]}$

P_4 $Q_4 = p_{4,1}^{[1]} p_{4,2}^{[2]}$, $Q_2 = p_{2,1}^{[1]} p_{2,4}^{[2]}$.

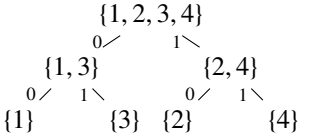


Figure 5: The point trie.

In conclusion, $Q_1 = p_{1,2}^{[1]} p_{1,3}^{[2]} = (x-1)(y-1)$, $Q_2 = p_{2,1}^{[1]} p_{2,4}^{[2]} = -x(y-1)$, $Q_3 = p_{32}^{[1]} p_{3,1}^{[2]} = -y(x-1)$, $Q_4 = p_{4,1}^{[1]} p_{4,2}^{[2]} = xy$. ◊

As explained in [6], the trie can be represented via a n-ary tree structure, which is implemented using pointers. This way, it can be updated iteratively on the points and the needed number of pointers is minimized.
In our implementation, instead of multiplying the building factors and get a longer polynomial, we limit ourselves at finding out what are the factors appearing in each Q_i. The reason is that, for our application, only evaluating such polynomials will be needed, so the factors are enough.
The testing activity described in [6, Table 1] has been expandend with new experiments. We report here the average lower bound (0.01 seconds, for 1024 points in three coordinates) and the average upper bound (6 minutes for 65,536 points in four coordinates). The variance is approximately zero, our computations are very reliable.

5 Degroebnerization and the quotient algebra basis: a new approach for normal forms

As shown in Section 3, for applications, the normal form of the separator polynomials modulo the ideal of the given points is needed.
Generally, what it is done is to take the set $\mathbf{X}$, compute a separator family $\mathcal{Q}$ for it, find a Gröbner basis $\mathcal{G}$ for the ideal of points $I(\mathbf{X})$ and perform Buchberger reduction on the elements in $\mathcal{Q}$ modulo $\mathcal{G}$.
We show now how to get the same result without the need of computing $\mathcal{G}$, nor of performing any step of Buchberger reduction. Our aim is to use Lundqvist's interpolation approach, taking advantage of point tries and Bar Codes to speed up the computation. Let us consider the following proposition.

Proposition 7 ([14])**.** *Let* $\mathbf{X} = \{P_1, ..., P_N\}$ *be a finite set of simple points,* $I := I(\mathbf{X}) \triangleleft \mathbf{k}[x_1, ..., x_n]$ *the ideal of points and* $\mathsf{N} = \{t_1, ..., t_N\} \subset \mathbf{k}[x_1, ..., x_n]$ *such that* $[\mathsf{N}] = \{[t_1], ..., [t_N]\}$ *is a basis for* $A := \mathbf{k}[x_1, ..., x_n]/I$*. Then, for each* $f \in \mathbf{k}[x_1, ..., x_n]$ *we have*

$$\mathrm{Nf}(f, \mathsf{N}) = (t_1, ..., t_N)(\mathsf{N}[\mathbf{X}]^{-1})^t (f(P_1), ..., f(P_N))^t,$$

where $\mathsf{N}[\mathbf{X}]$ *is the matrix whose rows are the evaluations of* N *at the elements of* $\mathbf{X}$ *and* $Nf(f, \mathsf{N})$ *is the normal form of* f *w.r.t.* $I(\mathbf{X})$*.*

The above Proposition 7 shows that, as soon as we have $\mathbf{X}$ and a basis of the quotient algebra $\mathcal{P}/I(\mathbf{X})$, computing the normal form of a polynomial modulo $I(\mathbf{X})$ is only a matter of *evaluations* and *linear algebra*.
Clearly, if one has a Gröbner basis for $I(\mathbf{X})$ with respect to some term ordering, finding a basis for $\mathcal{P}/I(\mathbf{X})$ is trivial, since one such a basis is the Gröbner escalier of $I(\mathbf{X})$.
Anyway, it is possible to get the lexicographical Gröbner escalier $\mathsf{N}(\mathbf{X})$ of $I(\mathbf{X})$ also in a "purely combinatorial" way, that is, only using comparisons among the coordinates of the points and *without needing to compute any polynomial*.
The first result in this framework has been given by Cerlienco and Mureddu [7, 8, 9], who proved a bijection between $\mathbf{X}$ and $\mathsf{N}(\mathbf{X})$, providing an algorithm to compute $\mathsf{N}(\mathbf{X})$. Such algorithm is iterative on $\mathbf{X}$, but it needs recursion on the variables, leading to a bad complexity: $O\left(n^2N^2\right)$, where, as usual N is the number of points and n the number of variables.
An improved alternative has been given by the Lex Game [12], which, making a large use of tries (not only the point trie described above, but also another trie related to the terms'exponents), improves the complexity to $O\left(nN + N\min(N, nr)\right)$, which depends also on r, the maximal number of children of a node in the point trie of $\mathbf{X}$. This last algorithm, despite being fast, has a crucial drawback: it is not iterative on the points, thing that for applications to data modelling is a great disadvantage, due to the dynamicity of experiments, which could produce data in different times.
In [5], we present a new algorithm (called *Iterative Lex Game*, Iter LG for short) that, employing the point trie and a Bar Code to dynamically store the terms, is more performant than Cerlienco-Mureddu's algortithm but it is iterative on $\mathbf{X}$. In particular, the complexity turns out to be $O(N^2 n \log(N))$ and we think it is the best which can be done, keeping iterativity on the points. We sketch now the main idea behind the algortithm, referring to [5] for further details.
The term in the escalier corresponding to the first point, $P_1 \in \mathbf{X}$, is surely $t_1 = 1$. Indeed the ideal associated to P_1 is the maximal ideal $I(\{P_1\}) = (x_1 - a_{11}, ..., x_n - a_{n1})$, this implying that the only term that is never the leading term of any polynomial in $I(\{P_1\})$ is $t_1 = 1$. As a basis for our algorithm, we construct the point trie associated to P_1 (which is a trie with only one leaf) and the Bar Code associated to $\{t_1\}$ (which is formed by n superimposed bars, one for each variable).
Suppose now to know $\mathsf{N}(\{P_1, ..., P_{N-1}\}) = \{t_1, ..., t_{N-1}\}$; we look for the term t_N associated to P_N in order to get the whole lexicographical Gröbner escalier $\mathsf{N}(\mathbf{X})$.
We first update the point trie associated to $\{P_1, ..., P_{N-1}\}$, adding the branch associated to P_N and keeping track of the level f, $1 \leq f \leq n$, where such branch forks from all the others, namely the first level such that the node labelled by the index N is not labelled by any other index. Let us call v such node. The value f is crucial since it already tells us that $x_f \mid t_N$, while $x_{f+1}, ..., x_n \nmid t_N$. The fact that $x_{f+1}, ..., x_n \nmid t_N$ is translated on the Bar Code by saying that at levels $f + 1, ..., n$, t_N is over the first bar.
The second step is to understand what is the exponent of x_f in t_N. For it, we need to pick an *antecedent point*. To find it, we look at the rightmost sibling node w of v and pick the first label l of w. This gives us a point, $P_l \in \{P_1, ..., P_{N-1}\}$, whose corresponding term t_l has already been computed: the exponent of x_f in t_N, is one more the exponent of x_f in t_l. Therefore, we look at the bar under t_l at level f and we know that we have to place t_N over a bar at level f, just on the right of that of t_l, but, of course, remaining over the first bar at levels $f + 1, ..., n$. If such a bar does not exist, we construct it, and this gives us t_N as a pure power of x_f. Otherwise, we restrict to P_N and the points whose corresponding terms are over such bar and we repeat the procedure, taking care of looking at the new forking level for P_N, reading the trie from level f up and repeating the above steps. In this case t_N is not a pure power but it is divided also by some variables smaller than x_f and t_N can be retrieved in a finite number of steps.

Example 8. For the set $\mathbf{X} := \{P_1 = (0, 0), P_2 = (1, 0), P_3 = (0, 1), P_4 = (1, 1)\}$ of Examples 5 (from which you can read the point trie for each point) and 6, we compute the basis of the quotient algebra $\mathbf{k}[x, y]/I(\mathbf{X})$ via our algorithm, as the lexicographical Gröbner escalier obtained by considering $x < y$, and finally we get the normal forms of the separator polynomials we computed in Example 6.

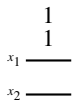


Figure 6: Bar Code of P_1.

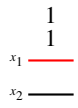


Figure 7: Red bar for P_2.

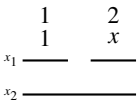


Figure 8: Bar Code of P_2.

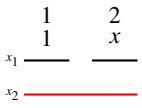


Figure 9: Red bar for P_3.

As usual $t_1 = 1$, so the Bar Code is as in Figure 6.

For $P_2 = (1, 0)$, we have $f = 1$ and $l = 1$. We consider then the red bar in Figure 7 and we see that there is no bar on the right, we construct it getting $t_2 = x$ (see Figure 8).
For $P_3 = (0, 1)$, we have $f = 2$ and $l = 1$; we look at the red bar in Figure 9 and we see that there is no bar on the right. Constructing it we get $t_3 = y$ (see Figure 10).

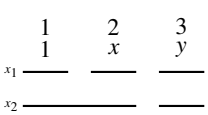


Figure 10: Bar Code for P_3.

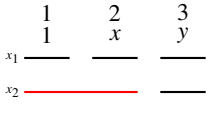


Figure 11: Red bar for P_4 (1).

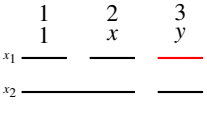


Figure 12: Red bar for P_4 (2).

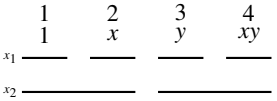


Figure 13: Bar Code for P_4.

Finally, for $P_4 = (1, 1)$, $f = 2$ and $l = 2$; this time there is a bar on the right of the red one (see Figure 11). We then restrict to $\{P_3, P_4\}$, so we get the new forking level to be $f = 1$ and the new antecedent to be P_3. This time, looking at the red bar

in Figure 12, we see that there is no bar on the right, so we construct it and we get $t_4 = xy$ as in Figure 13.
We can easily note that the separators polynomials are already in normal form, but we check it via Lundqvist's formula. First of all, note that $\mathsf{N}[\mathbf{X}] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and so $(\mathsf{N}[\mathbf{X}]^{-1})^t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix}$. We compute

$$(1, x, y, xy)(\mathsf{N}[\mathbf{X}]^{-1})^t = (1 - x - y + xy, x - xy, y - xy, xy)$$

and note that

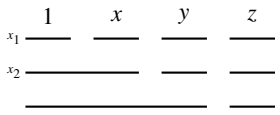
$$(1 - x - y + xy, x - xy, y - xy, xy) = (Q_1, Q_2, Q_3, Q_4),$$

proving that the separator polynomials we found are already in normal form.

♢

6 Implementation and timings for the Gröbner escalier

In this section, we go through the implementation of our algorithm to compute the Gröbner escalier. In particular, we need to construct the data structure used for implementing the algorithm described more precisely in Section 5.
First of all, a Bar Code can be implemented in C using concatenated objects, implemented using a linked list of data structures. We need three different lists, namely one containing the terms, one related to the single bars and one containing the different levels of the Bar Code.
The list related to the single bars (so containing some structures called barNode as elements) contains in its structure the exponent of the variable related to that bar, a pointer to the bar under it (which is of course NULL if we are considering the n-th bars), a pointer to the next bar to the right, a pointer to the first term of the bar and a pointer to the last one. The list related to a level contains the data structures barRow as elements, which, in turn, contain the index of such level, a pointer to the first bar of this level and a pointer to the following level. A graphical representation of this idea can be found in Figure 15, which is the implementation of the Bar Code in Figure 14, referring to the set of terms $\{1, x, y, z\}$.



We can easily note that, for implementation's convenience, the Bar Code is reversed (the bars on the top correspond to the maximal variable, while those on the bottom correspond to the minimal one).

Figure 14: Bar Code for $\{1, x, y, z\}$.

The circles represent the levels, while the rectangles are the bars. The former ones take note - for each term - of the exponents of the related variable (the variable x_i where $i = n - j$ and j is the index of the level we are considering). The arrows represent the pointers.

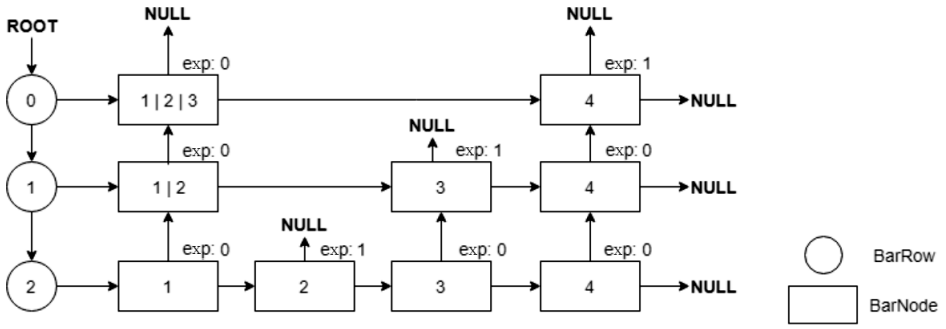


Figure 15: How the Bar Code of Figure 14 is implemented.

In addition, we have implemented also an auxiliary structure to keep track of forking levels. It is again a concatenated list, whose elements are structs called forkNode. Each such struct contains the information about the presence or absence of a fork, a pointer to the next element of the list, a pointer to the list of all points labelling the same node of the trie and a pointer to the labels' lists of the sibling nodes.

Tesing activity has been executed on a machine with an Intel(R) Core(TM) i7-8550U processor 64-bit architecture, CPU @ 1.80GHz equipped with 8 GB of RAM. The operating system installed on this machine was Linux Mint 19 (Tara) 64 bits.
In our knowledge, the only function computing the Gröbner escalier associated to a finite set of finite points in a combinatorial way, is the function *nonMonomials* of the library *pointid.lib* in the Software Singular [10], which implements Cerlienco-Mureddu's algorithm. Figure 16 indicates the obtained timings, highlighting the advantages, compared to the aforementioned *nonMonomials* function.

$GF(2^m)$	Points	Coord	Singular	Iter LG
2^4	256	4	1.68s	0.13s
2^4	256	4	5.01s	0.11s
2^5	1024	3	16.77s	0.13s
2^6	4096	3	5m 27.04s	2.18s
2^5	1024	3	23.06s	0.18s
2^6	4096	3	6m 9.29s	2.04s
2^8	65536	3	27h 42m	49m 27s
2^{20}	4096	2	-	37.40s

Figure 16: Testing activity for the Iterative Lex Game.

7 Conclusions

In this work, we have proposed a new method to solve the problem proposed in [13], *without needing Gröbner bases' computation*.
We aim to continue this work in order to show the advantages of Degroebnerization in a more precise way, by making a fine complexity analysis comparison between the approach in [13] and ours. Indeed, we think that Laubenbacher-Stigler's analysis should be corrected.
Moreover, we aim to use our method also to compute the indicator function as in [11], and compare the performances.

References

[1] Ceria, M., *Bar Code for monomial ideals*, Journal of Symbolic Computation, Volume 91, March - April 2019, DOI: 10.1016/*j.jsc*.2018.06.012, 30-56.

[2] Ceria, M. *Bar Code and Janet-like division*, available in arxiv as arXiv:1910.03572 [math.CO].

[3] Ceria, M., *Bar code: a visual representation for finite set of terms and its applications.*, Mathematics in Computer Science, 14(2), 497-513 (2020), online in 2019 doi:10.1007/s11786-019-00425-4

[4] Ceria, M., *Bar Code vs Janet tree*, Atti della Accademia Peloritana dei Pericolanti, Classe di Scienze Fisiche, Matematiche e Naturali, VOL 97, NO 2 (2019) Doi: 10.1478/AAPP.972A6

[5] Ceria M., Mora T. *Combinatorics of ideals of points: a Cerlienco-Mureddu-like approach for an iterative lex game*, available in arxiv as arXiv:1805.09165 [math.AC].

[6] Ceria, M., Mora, T., Visconti, A., *Efficient computation of squarefree separator polynomials*, In International Congress on Mathematical Software (pp. 98-104). Springer, Cham, (2018).

[7] Cerlienco L., Mureddu M., *Algoritmi combinatori per l'interpolazione polinomiale in dimensione* ≥ 2, preprint (1990).

[8] Cerlienco L., Mureddu M., *From algebraic sets to monomial linear bases by means of combinatorial algorithms*, Discrete Math. 139, 73 − 87.

[9] Cerlienco L., Mureddu M., *Multivariate Interpolation and Standard Bases for Macaulay Modules*, J. Algebra 251 (2002), 686 − 726.

[10] Decker, W.; Greuel, G.-M.; Pfister, G.; Schönemann, H.: SINGULAR 4-2-0 — A computer algebra system for polynomial computations. http://www.singular.uni-kl.de (2019).

[11] Pistone, G., Rogantin, M. P., *Indicator function and complex coding for mixed fractional factorial designs*, Journal of Statistical Planning and Inference, 138(3), 787-802,(2008).

[12] Felszeghy, B., Rth, B., Rnyai, L., *The lex game and some applications*, Journal of Symbolic Computation 41(6), 663–681 (2006)

[13] Laubenbacher, R., Stigler, B., *A computational algebra approach to the reverse engineering of gene regulatory networks*, Journal of theoretical biology, 229, 4, 523-537, Academic Press.

[14] Lundqvist, S., *Vector space bases associated to vanishing ideals of points*, Journal of Pure and Applied Algebra 214(4), 309–321 (2010)

[15] Marinari, M.G., Möller, H.M., Mora, T., *Gröbner bases of ideals defined by functionals with an application to ideals of projective points*, Applicable Algebra in Engineering, Communication and Computing 4(2), 103145 (1993)

[16] Möller, H.M., Buchberger, B., *The construction of multivariate polynomials with preassigned zeros*, In: European Computer Algebra Conference. pp. 2431. Springer (1982)

[17] Mora T., *Solving Polynomial Equation Systems* 4 Vols., Cambridge University Press, I DOI: 10.1017/*CBO*9780511542831 (2003), II DOI: 10.1017/*CBO*9781107340954 (2005), III DOI: 10.1017/*CBO*9781139015998 (2015), IV DOI:10.1017/*CBO*9781316271902 (2016).

[18] Mora, T. *An FGLM-like algorithm for computing the radical of a zero-dimensional ideal.* Journal of Algebra and Its Applications, 17(01) (2018).

[19] Mourrain B. A New Criterion for Normal Form Algorithms. In: Fossorier M., Imai H., Lin S., Poli A. (eds) Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. AAECC 1999. Lecture Notes in Computer Science, vol 1719. Springer, Berlin, Heidelberg (1999)