.

Università di Pisa
Dipartimento di Informatica

Technical Report: TR-01-17

Inversion of two level circulant matrices over $\mathbf{Z}_p$

Carlo J. Accettella Gianna M. Del Corso Giovanni Manzini

ADDRESS: Corso Italia 40, 56125 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

Inversion of two level circulant matrices over $\mathbf{Z}_p$

Carlo J. Accettella[*] Gianna M. Del Corso[†] Giovanni Manzini[‡]

Abstract

We consider the problem of inverting block circulant with circulant blocks (BCCB) matrices with entries over the field $\mathbf{Z}_p$. This problem arises in the study of of two-dimensional linear cellular automata. Since the standard reduction to diagonal form by means of FFT has some drawbacks when working over $\mathbf{Z}_p$, we solve this problem by transforming it into the equivalent problem of inverting a circulant matrix with entries over a suitable ring $\mathbf{R}$. We show that a BCCB matrix of size mn can be inverted in $O(mn\, c(m,n))$ operations in $\mathbf{Z}_p$, where c is a low degree polynomial in $\log m$ and $\log n$.

1 Introduction

In this paper we consider the problem of inverting a block circulant with circulant blocks (BCCB) matrix with entries over the field $\mathbf{Z}_p$. In addition to its own interest as a linear algebra problem, the inversion of these matrices plays an important role in the theory of two-dimensional linear cellular automata (see for example [1, 4, 7]).

We denote by BCCB(m,n) the class of matrices which have an $m \times m$ circulant block structure, each block being an $n \times n$ circulant matrix. It is well known that a BCCB(m,n) matrix over $\mathbf{C}$ can be inverted in $O(mn \log(mn))$ operations. The standard inversion algorithm works by reducing the matrix to diagonal form by means of FFT's of order n and m. Unfortunately, this approach does not generalize to BCCB(m,n) matrices over $\mathbf{Z}_p$. If $\gcd(p,n) > 1$ no extension field of $\mathbf{Z}_p$ contains a primitive n-th root of unity and $n \times n$ circulant matrices over $\mathbf{Z}_p$ are not diagonalizable. If $\gcd(p,n) = 1$ a primitive n-th root of unity exists in a suitable extension of $\mathbf{Z}_p$. However, the approach based on the FFT still poses some problems. In fact, working in an extension of $\mathbf{Z}_p$ requires that we find a suitable irreducible polynomial $q(x)$ and every operation in the field involves manipulation of polynomials of degree up to $\deg(q(x)) - 1$.

In this paper we show how to invert BCCB matrices using a different approach. We observe that the problem of inverting a BCCB(m,n) matrix over $\mathbf{Z}_p$ is equivalent to the problem of inverting a $n \times n$ circulant matrix with entries over the ring $\mathbf{R} = \mathbf{Z}_p[y]/(y^m - 1)$. For this reason we study the general problem of inverting a circulant matrix with entries over a ring of the form $\mathbf{R} = \mathbf{Z}_p[y]/(a(y))$ where $a(y)$ is a generic polynomial. We describe two different algorithms for solving this problem. The first one assumes that the factorization of $a(y)$ is known, whereas the second one makes no assumptions on $a(y)$. Both these algorithms can be used to invert a BCCB(m,n) matrix over $\mathbf{Z}_p$

[*]Dipartimento di Informatica, Pisa, Italy. Email: accettel@cli.di.unipi.it.
[†]Dipartimento di Informatica, Pisa, Italy. Email: delcorso@di.unipi.it.
[‡]Dip. Scienze e Tecnologie Avanzate, Università del Piemonte Orientale, Alessandria, and IMC-CNR, Pisa, Italy. Email: manzini@mfn.unipmn.it.

using a number of operations of the form $O(mn\,c(m,n))$ where c is a low degree polynomial in $\log m$ and $\log n$. Finally, we describe a "fast" algorithm for inverting a BCCB(m,n) matrix when m or n is a power of two, and an even faster algorithm for the case in which both m and n are powers of two.

2 Preliminaries

Let U_n denote the $n \times n$ cyclic shift matrix whose entries are $(U_n)_{ij} = 1$ if $j - i \equiv 1 \pmod{n}$, and 0 otherwise. A circulant matrix C over a ring $\mathbf{R}$ can be written as $C = \sum_{i=0}^{n-1} c_i U_n^i$, where $c_i \in \mathbf{R}$. It is natural to associate to C the polynomial (over the ring $\mathbf{R}[x]$) $f(x) = \sum_{i=0}^{n-1} c_i x^i$. Since $U_n^n = I$, there is a natural isomorphism which maps circulant matrices into polynomials over $\mathbf{R}[x]$ taken modulo $x^n - 1$. Us usual, we denote the ring of polynomials modulo $x^n - 1$ as $\mathbf{R}[x]/(x^n - 1)$. In order to find the inverse of C over $\mathbf{R}$ one can equivalently find the polynomial $g(x)$ over $\mathbf{R}[x]$ such that

$$f(x)g(x) \equiv 1 \mod (x^n - 1).$$

Notice that $g(x)$ can be seen as the multiplicative inverse of $f(x)$ in the ring $\mathbf{R}[x]/(x^n - 1)$.

We say that A is a (m,n) block-circulant with circulant blocks (that is, A belongs to the class BCCB(m,n)), if it has the following structure

$$A = \begin{pmatrix} C_0 & C_1 & \cdots & C_{m-1} \\ C_{m-1} & C_0 & \ddots & \ddots \\ \vdots & \ddots & \ddots & C_1 \\ C_1 & \cdots & C_{m-1} & C_0 \end{pmatrix},$$

and each block C_i is an $n \times n$ circulant matrix. It is well known [3] that A can be rewritten as a sum of Kronecker's products, that is

$$A = \sum_{i=0}^{m-1} U_m^i \otimes C_i. \tag{1}$$

Since each C_i is a circulant matrix we can write it as $C_i = \sum_{j=0}^{n-1} a_{ij} U_n^j$. Observing that the Kroneker product is distributive over the sum, we can rewrite A as

$$A = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} (U_m^i \otimes U_n^j).$$

Since $(U_m^i \otimes U_n^j)(U_m^k \otimes U_n^\ell) = (U_m^{i+k} \otimes U_n^{j+\ell})$ and $U_m^m = U_n^n = I$, it is natural to associate to A the bivariate polynomial

$$f(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} x^j y^i,$$

modulo $x^n - 1$ and $y^m - 1$. The problem of computing the inverse of a BCCB(m,n) matrix A is therefore equivalent to the problem of computing a polynomial $g(x,y)$ such that

$$f(x,y)g(x,y) \equiv 1 \mod (x^n - 1,\, y^m - 1). \tag{2}$$

In other words, the problem of inverting a BCCB(m,n) matrix A is equivalent to the inversion of a polynomial modulo x^n-1, y^m-1. In view of the equivalence between circulant matrices and polynomials, in the rest of the paper we will often use the more compact polynomial notation.

The following elementary lemma shows that the problem of inverting a BCCB matrix with entries over the field $\mathbf{Z}_p$ is equivalent to the problem of inverting a circulant matrix over a suitable ring $\mathbf{R}$.

Lemma 2.1 *Let A be a BCCB(m,n) matrix over $\mathbf{Z}_p$. The inversion of A is equivalent to the inversion of a circulant matrix C with entries over the ring $\mathbf{R}=\mathbf{Z}_p[y]/(y^m-1)$.*

Proof. We use the equivalence between circulant matrices and polynomials described above. Let

$$f(x,y)=\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} a_{ij}x^iy^j$$

denote the polynomial associated to A. We can rewrite $f(x,y)$ as

$$f^{(y)}(x)=\sum_{i=0}^{n-1} a_i(y)x^i$$

where $a_i(y)=\sum_{j=0}^{m-1} a_{ij}y^j$ for $i=0,1,\ldots,n-1$, and each $a_i(y)$ belongs to $\mathbf{Z}_p[y]/(y^m-1)$. Assume f is invertible and let $g(x,y)=\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} b_{ij}x^iy^j$ the inverse modulo x^n-1, y^m-1. Construct $g^{(y)}(x)$ as done for f. Then,

$$g^{(y)}(x)f^{(y)}(x)\equiv 1 \pmod{x^n-1},$$

that is, $g^{(y)}(x)$ is the inverse of $f^{(y)}(x)$ modulo x^n-1 over the ring $\mathbf{Z}_p[y]/(y^m-1)$. This means that the $n\times n$ circulant matrices associated to $f^{(y)}$ and $g^{(y)}$ are one inverse of the other. ∎

In view of Lemma 2.1, in Sect. 3 and 4 we consider the general problem of inverting a circulant matrix with entries over a ring of the form $\mathbf{R}=\mathbf{Z}_p[y]/(a(y))$. The algorithms for the solution of this general problem will be used in Sect. 5 for the inversion of BCCB matrices over $\mathbf{Z}_p$.

The problem of inverting a circulant matrix over $\mathbf{R}=\mathbf{Z}_p[y]/(a(y))$ has some similarities with the problem of inverting a circulant matrix over $\mathbf{Z}_m$ which has been studied in [1]. As we will see, the case in which the factorization of $a(y)$ is known can be solved using the same techniques (Newton-Hensel lifting and Chinese remaindering) which has been used in [1] for the case in which the factorization of m is known. However, when the factorization of $a(y)$ is not known the techniques described in [1] cannot be used since they lead to a very inefficient algorithm. In order to get an efficient algorithm we will make use of techniques (such as the squarefree decomposition) which are specific to polynomials.

2.1 Review of complexity results

The cost of each algorithm in this paper will be given in terms of number of operations (sums and products) over the field $\mathbf{Z}_p$.

Operations in $\mathbf{Z}_p[x]$. It is well known (see [9, Chap. 8]) that the product of two degree n polynomials over $\mathbf{Z}_p$ can be done $\mathsf{M}(n)=O(n\log n\log\log n)$ arithmetic operations[1] in $\mathbf{Z}_p$.

[1]In alternative one can use the Lempel, Seroussi, Winograd method [6] especially studied for multiplying polynomial over finite fields. That method requires an almost linear number of operations over the field but it needs an expensive preprocessing phase.

Given two degree n polynomials $a(x), b(x) \in \mathbf{Z}_p[x]$, we can compute $d(x) = \gcd(a(x), b(x))$ using the Fast Extended Euclidean algorithm which takes $O(\mathsf{M}(n)\log n)$ arithmetic operations in $\mathbf{Z}_p$ (see [9, Chap. 11]). The same algorithm returns also the polynomials $s(x)$ and $t(x)$ such that $a(x)s(x) + b(x)t(x) = d(x)$.

Operations in $\mathbf{R}[x]$ with $\mathbf{R} = \mathbf{Z}_p[y]/(r(y))$. Let $r(y) \in \mathbf{Z}_p[y]$ be a degree d polynomial, and let $\mathbf{R} = \mathbf{Z}_p[y]/(r(y))$. Each element $z \in \mathbf{R}$ can be represented by a polynomial $p_z(y)$ of degree at most $d-1$. The element z is invertible in $\mathbf{R}$ iff $\gcd(p_z(y), r(y)) = 1$. If z is invertible, its inverse can be computed using the Extended Euclidean algorithm which, as we have just recalled, takes $O(\mathsf{M}(d)\log d)$ operations over $\mathbf{Z}_p$.

The product of two polynomials in $\mathbf{R}[x]$ can be computed by the algorithm reported in [2, Theorem 1.7.1], which requires $n \log n$ multiplications and $n \log n \log\log n$ sums between degree d polynomials over $\mathbf{Z}_p[y]$, that is $O(\mathsf{M}^*(n,d))$ operations, where

$$\mathsf{M}^*(n,d) = \mathsf{M}(d) n \log n + dn \log n \log\log n.$$

If $r(y)$ is irreducible, then $\mathbf{R} = \mathbf{Z}_p[y]/(r(y))$ is a field and given two polynomials in $\mathbf{R}[x]$ we can compute their gcd using the again the Fast Extended Euclidean algorithm. It easy to see that the computation of the gcd of two degree n polynomials in $\mathbf{R}[x]$ takes $O(\Gamma^*(n,d))$ operations in $\mathbf{Z}_p$ where

$$\Gamma^*(n,d) = \mathsf{M}^*(n,d)\log n + n\mathsf{M}(d)\log d. \tag{3}$$

3 Inversion of circulant matrices over $\mathbf{R} = \mathbf{Z}_p[y]/(a(y))$. Factorization of $a(y)$ known

In this section we consider the problem of inverting an $n \times n$ circulant matrix over $\mathbf{R} = \mathbf{Z}_p[y]/(a(y))$ when the factorization $a(y) = a_1^{t_1}(y) a_2^{t_2}(y) \cdots a_h^{t_h}(y)$ of the modulus is known. As we will see this problem can be solved combining three well known techniques: the Extended Euclidean algorithm, Newton-Hensel lifting, and the Chinese Remaindering theorem. In the following we denote $f^{(y)}(x) \in \mathbf{R}[x]$ the polynomial associated to the circulant matrix C we are trying to invert. We write $f^{(y)}$ since the coefficients of $f^{(y)}(x)$ are polynomials in y of degree at most $\deg(a(y)) - 1$. If the inverse of C exists we denote by $g^{(y)}(x)$ the corresponding polynomial. From our previous discussion we know that $f^{(y)}, g^{(y)}$ satisfy the following relation:

$$f^{(y)}(x) g^{(y)}(x) \equiv 1 \qquad \bmod x^n - 1. \tag{4}$$

In the following we use m to denote the degree of the modulus $a(y)$.

3.1 Case 1: $a(y)$ irreducible

In this section we assume that $a(y)$ is an irreducible polynomial over $\mathbf{Z}_p[y]$ of degree m, so that $\mathbf{R} = \mathbf{Z}_p[y]/(a(y))$ is a field with $q = p^m$ elements. It is well known that in this case $f^{(y)}$ (and the associated circulant matrix) is invertible iff $\gcd(f^{(y)}(x), x^n - 1) = 1$. The polynomial $g^{(y)}$ which satisfies (4) can be found using the Extended Euclidean algorithm whose cost is given by (3). Summing up, when $a(y)$ is invertible the cost of inverting $f^{(y)}$ is

$$T_1(n,m) = O(\Gamma^*(n,m)) = O(\mathsf{M}^*(n,m)\log n + n\mathsf{M}(m)\log m)\,. \tag{5}$$

1. Let $h = \lceil \log t \rceil$. Compute the polynomials $b^2(y), b^4(y), \ldots b^{2^h}(y)$.
2. Let $f_h^{(y)}(x) = f^{(y)}(x)$. For $i = h-1, \ldots, 0$, compute the polynomials
$$f_i^{(y)}(x) = f_{i+1}^{(y)}(x) \bmod b^{2^i}(y)$$
3. Compute the inverse $g_0^{(y)}(x)$ of $f_0^{(y)}(x)$ using the algorithm described in Sect. 3.1.
4. Apply Newton-Hensel lifting procedure for computing the inverse $g^{(y)}(x)$ as follows. For $i = 1, 2, \ldots, h$
$$g_i^{(y)}(x) = \left[2g_{i-1}^{(y)}(x) - \left[g_{i-1}^{(y)}(x)\right]^2 f_i^{(y)}(x)\right] \bmod b^{2^i}(y)$$
where all the computations in x are done modulo $x^n - 1$.
5. Return $g_h^{(y)}(x)$.

Figure 1: Inversion of a circulant matrix over the ring $\mathbf{Z}_p[y]/(b^t(y))$, $b(y)$ irreducible

3.2 Case 2: $a(y) = b^t(y)$, $b(y)$ irreducible

In this section we assume that $a(y) = b^t(y)$, where $b(y)$ is an irreducible polynomial over $\mathbf{Z}_p[y]$ of degree $m_1 = m/t$.

Let $f_0^{(y)}(x) = f^{(y)}(x) \bmod b(y)$, that is, $f_0^{(y)}(x)$ is obtained reducing each coefficients of $f^{(y)}(x)$ modulo $b(y)$. If $f^{(y)}$ is invertible we set $g_0^{(y)}(x) = g^{(y)}(x) \bmod b(y)$. From (4) we get
$$f_0^{(y)}(x) g_0^{(y)}(x) \equiv 1 \qquad \bmod x^n - 1,$$
which means that $g_0^{(y)}$ is the inverse of $f_0^{(y)}$ modulo $x^n - 1$ where $f_0^{(y)}(x)$ and $g_0^{(y)}(x)$ are seen as polynomials with coefficients over the field $\mathbf{Z}_p[y]/(b(y))$. This observation suggests that for computing $g^{(y)}$ we can first find $g_0^{(y)}(x)$ and use the Newton-Hensel lifting procedure to recover $g^{(y)}$. Since $f_0^{(y)}(x)$ is a polynomial with coefficients over the field $\mathbf{Z}_p[y]/(b(y))$ we can find $g_0^{(y)}$ as described in Sect. 3.1. The complete algorithm is given in Fig. 1. We point out that at Step 3 it is possible that the inversion of $f_0^{(y)}(x)$ fails (this happens when $\gcd(f_0^{(y)}(x), x^n - 1) \neq 1$). In this case the algorithm should report that $f^{(y)}(x)$ is not invertible since, as it is easy to see, $f^{(y)}$ is invertible iff $f_0^{(y)}$ is invertible.

To prove the correctness of the algorithm in Fig. 1 we need to show that the polynomial $g_h^{(y)}(x)$ returned at Step 5 is indeed the inverse of $f^{(y)}$. To see this it suffices to verify by induction that for $i = 0, \ldots, h$ we have
$$g_i^{(y)}(x) f_i^{(y)}(x) \equiv 1 \qquad \bmod (x^n - 1);$$
where $g_i^{(y)}(x)$ and $f_i^{(y)}(x)$ are seen as polynomials with coefficients over the ring $\mathbf{Z}_p[y]/(b^{2^i}(y))$.

Let us now examine the cost of the above algorithm. Step 1 requires the computation of h products of polynomials over $\mathbf{Z}_p[y]$. The cost for computing $b^{2^i}(y)$ given $b^{2^{i-1}}(y)$ is $\mathsf{M}(2^i d)$ being m_1 the degree of $a(y)$. Hence, the cost Step 1 is $O\left(\sum_{i=1}^h \mathsf{M}(2^i d)\right) = O(\mathsf{M}(m))$.

In Step 2, for $i = h-1, \ldots, 0$, we reduce each coefficient of $f_{i+1}^{(y)}(x)$ modulo $b^{2^i}(y)$. Each reduction (which involves two polynomials over $\mathbf{Z}_p$) takes, asymptotically, the same time as polynomial multiplication (see Sect. 9.1 in [9]). Since each $f_i^{(y)}(x)$ has at most n coefficients the cost of Step 2 is

$$O\Big(n\,\Big[\mathsf{M}(2^h m_1)+\mathsf{M}(2^{h-1}m_1)+\cdots+\mathsf{M}(2m_1)\Big]\Big)=O\Big(n\mathsf{M}(2^h m_1)\Big)=O(n\mathsf{M}(m))\,.$$

As discussed in Sect. 3.1 Step 3 takes $O(\Gamma^*(n,m_1))$ operations. In Step 4, the i-th iteration of Newton-Hensel lifting consists of two multiplications between degree n polynomials with coefficients over $\mathbf{Z}_p[y]/(b^{2^i}(y))$. This takes $O\big(\mathsf{M}^*(n,2^i m_1)\big)$. Summing for $i=1,2,\ldots,h$, we get that the cost of Step 4 is $O(\mathsf{M}^*(n,m))$. Summing up, the cost of the algorithm described in Fig. 1 is

$$T_2(n,m)=O(\Gamma^*(n,m_1)+\mathsf{M}^*(n,m))\,. \tag{6}$$

Note that the above cost is asymptotically equal to the cost of inverting $f_0^{(y)}$ plus the cost a single multiplication between degree n polynomials with coefficients in the ring $\mathbf{R}=\mathbf{Z}_p[y]/(a(y))$.

3.3 Case 3: $a(y)$ arbitrary

In this section we assume that the factorization of $a(y)$ is $a(y)=a_1^{t_1}(y)a_2^{t_2}(y)\cdots a_h^{t_h}(y)$ where each $a_i(y)$ is an irreducible polynomials over $\mathbf{Z}_p$ of degree d_i. Let $m_i=d_it_i$. We have $m=m_1+m_2+\cdots+m_h$.

Our strategy for inverting $f^{(y)}(x)$ consists in computing, for $i=1,2,\ldots h$, the inverse of $f^{(y)}$ over each ring $\mathbf{R}_i=\mathbf{Z}_p[y]/(a_i^{t_i}(y))$. This is done using the algorithm described in Sect. 3.2. Then we use the Chinese Remaindering Theorem to construct the inverse of $f^{(y)}(x)$ over the ring $\mathbf{R}=\mathbf{Z}_p[y]/(a(y))$.

Our algorithm is described in Fig. 2. Notice that, for efficiency purposes, the computation induced by Chinese Remaindering is done according to a tree-like structure. For example, if $h=4$, we first find the four inverses modulo $a_1^{t_1}(y)$, $a_2^{t_2}(y)$, $a_3^{t_3}(y)$, $a_4^{t_4}(y)$. Then we find the two inverses modulo $a_1^{t_1}(y)a_2^{t_2}(y)$, and $a_3^{t_3}(y)a_4^{t_4}(y)$. Finally we find the desired inverse modulo $a(y)=a_1^{t_1}(y)a_2^{t_2}(y)a_3^{t_3}(y)a_4^{t_4}(y)$. This strategy requires three tree-like computations. The first one—done at Step 2—computes the polynomials $b_{i,j}(y)$'s which are the product of 2, 4, 8, …, polynomials $a_i^{t_i}$'s. This tree is shown in Fig. 3 and is traversed from the leaves to the root.

The second tree-like computation is done at Step 3 where we compute the polynomials $f_{i,0}^{(y)}(x)=f^{(y)}(x)\bmod a_i^{t_i}(y)$. This is done traversing the tree of Fig. 3 from the root to the leaves. Finally, at Step 5 we compute the desired inverse $g^{(y)}(x)$ from the inverses modulo $a_i^{t_i}(y)$. This requires the traversing of the tree of Fig. 3 from the leaves to the root. To prove that the polynomial $g_{1,l}^{(y)}(x)$ is the inverse of $f^{(y)}$ it suffices to verify—by induction—that for $j=0,1,\ldots,l$ and $i=1,2,\ldots,2^{l-j}$ we have

$$g_{i,j}^{(y)}(x)f_{i,j}^{(y)}(x)\equiv 1 \qquad \bmod\ (x^n-1);$$

where $g_{i,j}^{(y)}(x)$ and $f_{i,j}^{(y)}(x)$ are seen as polynomials with coefficients over the ring $\mathbf{Z}_p[y]/(b_{i,j}(y))$.

We now examine the cost of the above algorithm. At Step 1, the computation of the polynomial $b_{i,0}(y)$ given $a_i(y)$ and t_i takes $O(\mathsf{M}(m_i))$ operations in $\mathbf{Z}_p$. Summing for $i=1,\ldots,h$ we get that Step 1 takes $O(\mathsf{M}(m))$ operations overall. At Step 2 the computation of each $b_{i,j}(y)$ consists in a product between two polynomials over $\mathbf{Z}_p$. The total cost of the tree's first level is $O(\mathsf{M}(m_1+m_2)+\mathsf{M}(m_3+m_4)+\cdots+\mathsf{M}(m_{h-1}+m_h))=O(\mathsf{M}(m))$ operations. Similarly it is easy to see the cost of every other level is bounded by $O(\mathsf{M}(m))$ operations, so Step 2 takes overall $O(\lceil\log h\rceil\mathsf{M}(m))$ operations.

1. For $i = 1, 2, \ldots, h$ compute $b_{i,0}(y) = a_i^{t_i}(y)$.
2. Let $l = \lceil \log h \rceil$. For $j = 1, 2, \ldots, l$ and $i = 1, 2, \ldots, 2^{l-j}$ compute $b_{i,j}(y) = b_{2i-1,j-1}(y) b_{2i,j-1}(y)$. Note that

$$b_{i,j}(y) = \prod_{\ell=\alpha+1}^{\beta} a_\ell^{t_\ell}(y) \qquad \text{with} \quad \alpha = 2^j(i-1),\ \beta = \min(\alpha + 2^j, h).$$

3. Let $f_{1,l}^{(y)}(x) = f^{(y)}(x)$. For $j = l-1, l-2, \ldots, 0$, and $i = 1, 2, \ldots, 2^{l-j}$, compute $f_{i,j}^{(y)}(x) = f_{2i-1,j+1}^{(y)}(x) \bmod b_{i,j}(y)$. Note that at the end of this step we have computed, for $i = 1, \ldots, h$, the polynomial $f_{i,0}^{(y)}(x) = f^{(y)}(x) \bmod b_{i,0}(y)$.
4. Compute the inverse $g_{i,0}^{(y)}(x)$ of each $f_{i,0}^{(y)}(x)$ over $\mathbf{R}_i = \mathbf{Z}_p[y]/b_{i,0}(y)$ modulo $x^n - 1$ using the algorithm described in Sect. 3.2 (recall that $b_{i,0}(y) = a_i^{t_i}(y)$ with $a_i(y)$ irreducible).
5. For $j = 1, 2, \ldots, l$ and $i = 1, 2, \ldots, 2^{l-j}$ compute

$$g_{i,j}^{(y)}(x) = [s(y)\, b_{2i,j-1}(y)]\, g_{2i-1,j-1}^{(y)}(x) \ + \ [t(y)\, b_{2i-1,j-1}(y)]\, g_{2i,j-1}^{(y)}(x),$$

where $s(y), t(y)$ are such that $s(y) b_{2i,j-1}(y) + t(y) b_{2i-1,j-1}(y) = 1$ and are computed using the extended gcd algorithm.
6. Return $g_{1,l}^{(y)}(x)$.

Figure 2: Inversion of a circulant matrix over the ring $\mathbf{R} = \mathbf{Z}_p[y]/(a(y))$, $a(y)$ arbitrary.

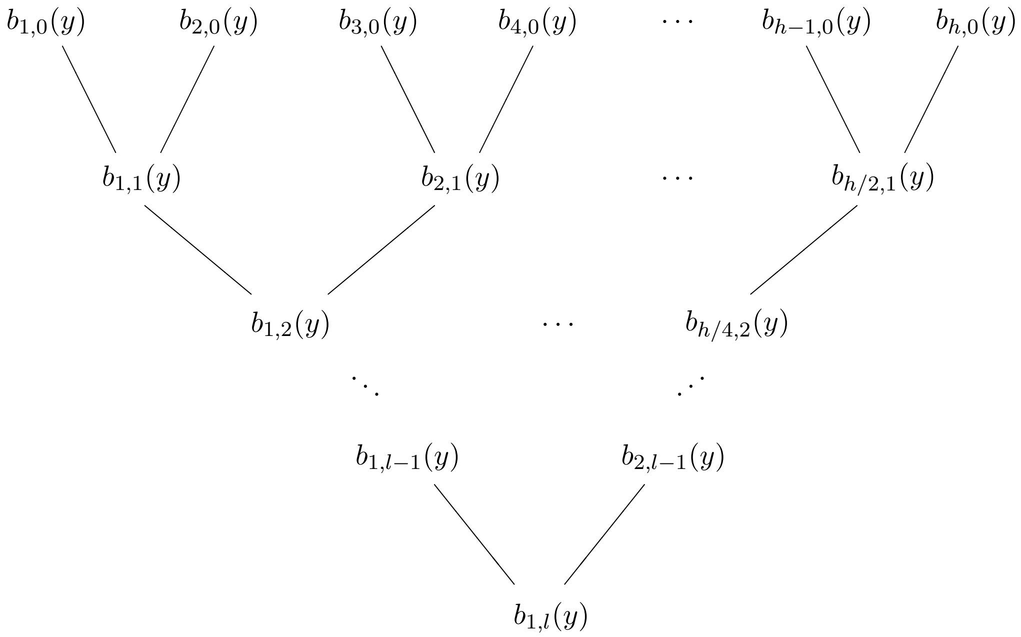

Figure 3: Computation tree for the application of the Chinese Remaindering theorem. The leaves of the tree are associated to the polynomials $b_{i,0}(y) = a_i^{t_i}(y)$. Each internal node is associated to the polynomial which is the product of all leaves above it. The root is associated to the polynomial $b_{i,l}(y) = a(y) = a_1^{t_1}(y)a_2^{t_2}(y)\cdots a_h^{t_h}(y)$.

At Step 3, we compute each $f_{i,j}^{(y)}(x)$ by reducing modulo $b_{i,j}(y)$ each coefficients of $f_{2i-1,j+1}^{(y)}(x)$. Since there are at most n coefficients and modulo computation has the same asymptotic cost as polynomial multiplication we have that Step 3 takes overall $O(n\lceil\log h\rceil \mathsf{M}(m))$ operations. Step 4 consists in the inversion of each $f_{i,1}^{(y)}(x)$ over $\mathbf{Z}_p[y]/(b_{i,0}(y))$. Using (6) we get that this takes

$$O\left(\sum_{i=1}^{h}(\Gamma^*(n,d_i)+\mathsf{M}^*(n,m_i))\right)$$

operations. At Step 5, we compute each $g_{i,j}^{(y)}(x)$ by first computing an extended gcd, and then doing two polynomial multiplication for each coefficient of $g_{i,j}^{(y)}(x)$. An easy calculation shows that the cost of each level of the tree is $O(\mathsf{M}(m)\log m + n\mathsf{M}(m))$. Hence, Step 5 overall takes $O(\lceil\log h\rceil \mathsf{M}(m)\log m + n\lceil\log h\rceil \mathsf{M}(m))$ operations. Summing up, the cost of the algorithm in Fig. 2 is

$$O\left(\lceil\log h\rceil \mathsf{M}(m)\,(n+\log m)+\sum_{i=1}^{h}(\Gamma^*(n,d_i)+\mathsf{M}^*(n,m_i))\right). \tag{7}$$

Notice that

$$\sum_{i=1}^{h}\mathsf{M}^*(n,m_i)\le \mathsf{M}^*(n,m), \qquad \text{and} \qquad \sum_{i=1}^{h}\Gamma^*(n,d_i)\le\Gamma^*(n,m),$$

where we used the inequality $\sum_{i=1}^{h} d_i \le m$. Moreover, since $h\le m$, we rewrite the cost (7) as

$$T_3(n,m)=O\big(\mathsf{M}(m)\left(n\log m+\log^2 m\right)+\mathsf{M}^*(n,m)\log n\big). \tag{8}$$

The following theorem summarizes the results of this section.

Theorem 3.1 *An $n\times n$ circulant matrix over $\mathbf{R}=\mathbf{Z}_p[y]/(a(y))$, with $\deg(a(y))=m$, can be inverted in $O\big(\mathsf{M}(m)\left(n\log m+\log^2 m\right)+\mathsf{M}^*(n,m)\log n\big)$ operations in $\mathbf{Z}_p$, provided that the factorization of $a(y)$ is known.* ∎

4 Inversion of circulant matrices over $\mathbf{R}=\mathbf{Z}_p[y]/(a(y))$. Factorization of $a(y)$ unknown

In this section we consider the problem of inverting an $n\times n$ circulant matrix over $\mathbf{R}=\mathbf{Z}_p[y]/(a(y))$ when we do not know the factorization of $a(y)$. A fist solution consists in computing the factorization of $a(y)$ over $\mathbf{Z}_p[y]$ and then apply the results of the previous section. Unfortunately, there are no polynomial time *deterministic* algorithms for polynomial factorization over finite fields, and the existence of such algorithms is indeed a central open problem in the theory of polynomials over finite fields.

The situation is different if we consider probabilistic factorization algorithms. There are several probabilistic algorithms[2] for polynomial factorization which have a polynomial *expected* running time and work very well in practice. For example in [9, Sect. 14.4] is described an algorithm which factors

[2]These factorization algorithms are probabilistic "Las Vegas" algorithms that is, they always return the correct factorization.

a degree m polynomial over $\mathbf{Z}_p$ in an expected number of $O(m\mathsf{M}(m)\log(mp))$ operations over $\mathbf{Z}_p$. We see that the expected cost of this probabilistic algorithms can be dominated by (8) when m and p are "small" compared to n.

We now describe a (deterministic) algorithm for inverting a circulant matrix over $\mathbf{R} = \mathbf{Z}_p[y]/(a(y))$ which does not require the knowledge of the factorization of $a(y)$. As we will see, this algorithm is by a factor at most $O(\log n)$ slower than the algorithm described in Sect. 3.

Our starting point is the so-called *pretend field technique*. If $\mathbf{R}$ were a field to invert $f^{(y)}(x)$ modulo $x^n - 1$ we would simply run the Extended Euclidean algorithm with input $f^{(y)}(x)$ and $x^n - 1$. We could use for example the recursive algorithm FastEEA shown in Fig. 4 which is a slightly simplified version of the Fast Extended Euclidean Algorithm described in [9, Chap 11]. Now suppose we call FastEEA with input the triplet $(r_0(x), r_1(x), \deg(r_0))$ where $r_0(x), r_1(x) \in \mathbf{R}[x]$. Since $\mathbf{R}$ is not really a field the algorithm FastEEA may fail. However, we can make two important observations:

1. The algorithm may fail only at Step 6 when we compute ρ^{-1} (the multiplicative inverse of ρ may not exist in $\mathbf{R}$, since $\mathbf{R}$ is a ring and not a field);
2. If the algorithm does not fail, that is every time we execute Step 6 the element ρ is invertible in $\mathbf{R}$, then the algorithm returns a monic polynomial $d(x) \in \mathbf{R}[x]$ and two polynomials $s(x), t(x)) \in \mathbf{R}[x]$ such that
$$r_0(x)s(x) + r_1(x)t(x) = d(x).$$
With a little abuse of notation we say that the algorithm returns an *extended gcd* of the pair $r_0(x), r_1(x)$. Notice that, if $d(x) = 1$ the polynomial $s(x)$ is the inverse of $r_0(x)$ over $\mathbf{R}[x]/(r_1(x))$. Vice versa if $\deg(d(x)) > 0$ then $r_0(x)$ is not invertible over $\mathbf{R}[x]/(r_1(x))$.

In the following we say that the triplet $\langle r_0(x), r_1(x), \mathbf{R}\rangle$ is gcd-safe if $r_0(x), r_1(x) \in \mathbf{R}[x]$ and the FastEEA algorithm with input $r_0(x), r_1(x), \deg(r_0)$ does not fail and returns an extended gcd of $r_0(x), r_1(x)$. Clearly, if the triplet $\left\langle x^n - 1, f^{(y)}(x), \mathbf{R}\right\rangle$ is gcd-safe then a single call to FastEEA computes the inverse of $f^{(y)}(x)$ over $\mathbf{R}[x]/(x^n - 1)$ or establishes that $f^{(y)}(x)$ is not invertible.

Unfortunately, in most cases the triplet $\left\langle x^n - 1, f^{(y)}(x), \mathbf{R}\right\rangle$ will not be gcd-safe. Our solution for inverting $f^{(y)}(x)$ consists in computing some polynomials $b_1(y), b_2(y), \ldots, b_\ell(y) \in \mathbf{Z}_p[y]$ such that:

1. $a(y) = b_1^{k_1}(y) \cdots b_\ell^{k_\ell}(y)$ for some integers $k_1, \ldots, k_\ell$;
2. $\gcd(b_i(y), b_j(y)) = 1$ for $1 \le i, j \le \ell$, $i \ne j$;
3. For $i = 1, \ldots, \ell$ the triplet
$$\left\langle x^n - 1, f^{(y)}(x) \bmod b_i(y), \mathbf{R}_i \right\rangle, \text{ where } \mathbf{R}_i = \mathbf{Z}_p[y]/(b_i(y))$$
is gcd-safe.

We call $b_1(y), \ldots, b_\ell(y)$ pseudo-factors and $b_1^{k_1}(y) \cdots b_\ell^{k_\ell}(y)$ a pseudo-factorization of $a(y)$. The reason is that the b_i's are pairwise coprime divisors of $a(y)$ but instead of requiring that they are irreducible, we ask the weaker condition that the triplets $\left\langle x^n - 1, f^{(y)}(x) \bmod b_i(y), \mathbf{R}_i\right\rangle$ are gcd-safe (this is a weaker condition since if $c(y)$ is irreducible every triplet $\langle r_0(x), r_1(x), \mathbf{Z}_p[y]/(c(y))\rangle$ is gcd-safe since $\mathbf{Z}_p[y]/(c(y))$ is a field).

We are interested in pseudo-factors for the following reason. Because of Property 3 we are able to compute, for $i = 1, \ldots, \ell$, the inverse $g_i^{(y)}$ of $f^{(y)}(x)$ in $\mathbf{R}_i[x]/(x^n - 1)$ (or to prove that such inverse

Algorithm FASTEEA$(r_0(x), r_1(x), k)$

1. **if** $r_1(x) = 0$ or $k < \deg(r_0) - \deg(r_1)$ **return** $\binom{1\,0}{0\,1}$.
2. $d \leftarrow \lfloor k/2 \rfloor$;
3. $R \leftarrow$ FASTEEA $(r_0(x) \upharpoonright 2d, r_1(x) \upharpoonright [2d - (\deg(r_0) - \deg(r_1))], d)$;
4. $\begin{pmatrix} s_0(x) \\ s_1(x) \end{pmatrix} \leftarrow R \begin{pmatrix} r_0(x) \\ r_1(x) \end{pmatrix}$;
5. **if** $s_1(x) = 0$ **or** $k < \deg(r_0) - \deg(s_1)$ **return** R;
6. $q(x) \leftarrow s_0(x)$ quo $s_1(x)$, $\quad \tilde{s}_2(x) \leftarrow s_0(x)$ rem $s_1(x)$, $\quad \rho \leftarrow \mathrm{lc}(\tilde{s}_2(x))$, $\quad s_2(x) \leftarrow \rho^{-1}(\tilde{s}_2(x))$;
7. $d^* \leftarrow k - \deg(r_0) - \deg(s_1)$;
8. $S \leftarrow$ FASTEEA $(r_1(x) \upharpoonright 2d^*, s_2(x) \upharpoonright [2d^* - (\deg(s_1) - \deg(s_2))], d^*)$;
9. **return** $S \begin{pmatrix} 0 & 1 \\ \rho^{-1} & -q(x)\rho^{-1} \end{pmatrix} R$.

Figure 4: Fast Extended Euclidean Algorithm for two monic polynomials $r_0(x), r_1(x)$ over an arbitrary field $\mathbf{F}$ with $\deg(r_0) > \deg(r_1)$. The procedure FASTEEA(r_0, r_1, k) returns a matrix $R \in \mathbf{F}[x]^{2\times 2}$ with the following property. Let $\binom{s_0}{s_1} = R\binom{r_0}{r_1}$, the polynomials s_0, s_1 are consecutive remainders in the Euclidean (monic) remainder sequence; they are related to the parameter k by the inequalities $\deg(s_1) < \deg(r_0) - k \leq \deg(s_0)$. If we set $k = \deg(r_0)$ we have that $s_0 = \gcd(r_0, r_1)$ and the first row of R contains the Bézout coefficients. In the above algorithm, rem and quo denote, as usual, the remainder and quotient of polynomial division. $\mathrm{lc}(f(x))$ denotes the leading coefficients of $f(x)$ or 1 if $f(x) = 0$. The notation $f(x) \upharpoonright k$ denotes the degree k polynomial whose coefficients coincides with the $k + 1$ highest coefficients of $f(x)$. See [9, Chap. 11] for details.

does not exists). If $g_i^{(y)}$ exists for all i, we can use Newton-Hensel lifting and Chinese Remaindering to compute the inverse of $f^{(y)}(x) \bmod b_i(y)$ in $\mathbf{R}[x]/(x^n - 1)$ proceeding exactly like in Sect. 3.

We now show how to compute a pseudo-factorization for $a(y)$. Without loss of generality in the following we assume that $a(y)$ is a monic polynomial.

Definition 1 Given a monic polynomial $f(y)$ over an arbitrary field the *squarefree decomposition* of $f(y)$ is the unique sequence of monic squarefree coprime polynomials $(g_1(y), \dots, g_k(y))$ such that

$$f(y) = g_1(y)\, g_2^2(y)\, g_3^3(y) \cdots g_k^k(y).$$

■

For example, the squarefree decomposition of $(y-1)(y+1)^3y^3$ is $(y-1, 1, y^2+y)$. In [9, Chap. 14] it is shown that the squarefree decomposition of a degree m polynomial over a finite field $\mathbf{F}$ can be computed in $O(\mathsf{M}(m)\log m)$ operations in $\mathbf{F}$. Now assume that $g_i(y) = b_{i,1}(y) \cdots b_{i,\ell_i}$ is a pseudo-factorization of $g_i(y)$. Then it is easy to see that

$$a(y) = b_{1,1}(y) \cdots b_{1,\ell_1}(y)\, b_{2,1}^2(y) \cdots b_{2,\ell_2}^2(y) \;\cdots\; b_{k,1}^k(y) \cdots b_{k,\ell_k}^k(y)$$

is a pseudo factorization of $a(y)$. This observation tells us that we only need to compute a pseudo-factorization for the monic squarefree polynomials g_i's.

The idea behind our algorithm for computing the pseudo-factors of a generic squarefree polynomial $p(y)$ is the following. Suppose we execute the algorithm of Fig. 4 with input $x^n - 1$ and $f^{(y)}(x)$ working in $\mathbf{R} = \mathbf{Z}_p[y]/(p(y))$. Assume for a moment that the recursive call at Step 3 goes through (that is, during the recursive call we are lucky and each time we need to invert an element in $\mathbf{R}$ it turns out to be invertible). When we reach Step 6 it possible that the value $\rho = \mathrm{lc}(\tilde{s}_2)$ is not invertible in $\mathbf{R} = \mathbf{Z}_p[y]/(p(y))$. If this happens, we split $p(y)$ as $p(y) = p_1(y) \cdots p_\ell(y)$ so that, for $i = 1, \dots, \ell$, the polynomial $\tilde{s}_2(x) \bmod p_i(y)$ is either 0 or has a leading coefficient invertible in $\mathbf{Z}_p[y]/(p_i(y))$. A fundamental observation is that if we execute the algorithm FastEEA with input $x^n - 1$ and $f^{(y)}(x) \bmod p_i(y)$ working in $\mathbf{Z}_p[y]/(p_i(y))$, then everything would go through up to Step 6. The polynomials computed at each step would be the same as before with each coefficient reduced modulo $p_i(y)$. At Step 6 we would have to invert the value $\rho = \mathrm{lc}(\tilde{s}_2(x) \bmod p_i(y))$ but the choice of the $p_i(y)$'s guarantees that such element is invertible in $\mathbf{Z}_p[y]/(p_i(y))$.

The bottom line is: if working modulo $p(y)$ the algorithm fails at Step 6 we find a splitting $p(y) = p_1(y) \cdots p_\ell(y)$ such that working modulo each $p_i(y)$ the algorithm does not fail either before or at Step 6. The following lemma shows that the splitting of the modulus we have just described is indeed possible.

Lemma 4.1 *Let $p(y) \in \mathbf{Z}_p[y]$ be a squarefree polynomial. Let $f^{(y)}(x)$ denote a polynomial with coefficients over $\mathbf{Z}_p[y]/(p(y))$. The algorithm* Split *of Fig. 5 returns a sequence of pairwise coprime polynomials $p_1(y), \dots, p_\ell(y)$ such that $p(y) = p_1(y) \cdots p_\ell(y)$, and for $i = 1, \dots, \ell$ $\mathrm{lc}(f^{(y)}(x) \bmod p_i(y))$ is invertible in $\mathbf{Z}[y]/(p_i(y))$. If $n = \deg(f^{(y)}(x))$ and $m = \deg(p(y))$ algorithm* Split *takes $O(n\mathsf{M}(m)\log m)$ operations in $\mathbf{Z}_p$.*

Proof. Recall that we have defined $\mathrm{lc}(f(x))$ to be the leading coefficient of $f(x)$ or 1 if $f(x) = 0$. Therefore we start by proving that, for all i, $f^{(y)}(x) \bmod p_i(y)$ is either 0 or a polynomial whose leading coefficient is coprime with $p_i(y)$.

Algorithm $\mathsf{Split}(f^{(y)}(x), p(y))$

1. $\mathsf{Output} \leftarrow \{\}, \quad n \leftarrow \deg(f^{(y)}(x)), \quad t(y) \leftarrow p(y);$
2. **for** $j = n$ **down to** 0 **do**
3. $\quad c_j(y) \leftarrow \mathrm{coef}_j(f^{(y)}(x));$
4. $\quad d_j(y) \leftarrow \gcd(c_j(y), t(y));$
5. $\quad$ **if** $d_j(y) = 1$ **then return** $\mathsf{Output} \cup \{t(y)\};$
6. $\quad$ **if** $d_j(y) \neq t(y)$ **then** $\mathsf{Output} \leftarrow \mathsf{Output} \cup \{t(y)/d_j(y)\},\ t(y) = d_j(y);$
7. **return** $\mathsf{Output} \cup \{t(y)\};$

Figure 5: Algorithm for splitting $p(y)$ as $p(y) = p_1(y) \cdots p_\ell(y)$ so that, for $i = 1, \ldots, \ell$, the polynomial $\mathrm{lc}(f^{(y)}(x) \bmod p_i(y))$ is invertible in $\mathbf{Z}_p[y]/(p_i(y))$. Note that at Step 3 we write $\mathrm{coef}_j(f^{(y)})$ to denote the coefficient of x^j in the polynomial $f^{(y)}(x)$.

A simple inductive argument shows that at the beginning of each **for** iteration $t(y)$ divides the coefficients $c_{j+1}(y), c_{j+2}(y), \ldots, c_n(y)$ of $f^{(y)}(x)$. Suppose $p_i(y)$ is added to Output at Step 6 of the jth iteration, that is, $p_i(y) = t(y)/d_j(y)$. Since $p_i(y)|t(y)$ it divides $c_{j+1}(y), c_{j+2}(y), \ldots, c_n(y)$. We now show that $\gcd(p_i(y), c_j(y)) = 1$ which proves our claim about $f^{(y)}(x) \bmod p_i(y)$. By the property of the gcd we known that $p_i(y) = t(y)/d_j(y)$ is coprime with $c_j(y)/d_j(y)$. Since $t(y)$ is squarefree, $p_i(y)$ is coprime with $d_j(y)$. Putting together the last two statements we get that $p_i(y)$ is coprime with $c_j(y)$ as claimed. Similarly, if $p_i(y)$ is added to Output at Step 5 we have that $p_i(y)$ divides $c_{j+1}(y), c_{j+2}(y), \ldots, c_n(y)$ and $\gcd(p_i(y), c_j(y)) = 1$. Finally, if $p_i(y)$ is added to Output at Step 7 then $p_i(y)$ divides $c_0(y), c_1(y), \ldots, c_n(y)$ and $f^{(y)}(x) \bmod p_i(y) = 0$.

We now prove that $p(y) = p_1(y) \cdots p_\ell(y)$. It is easy to see that at the beginning of each iteration $p(y)$ equals $t(y)$ times the product of the polynomials in Output. Since before returning the algorithm adds $t(y)$ to Output (Steps 5 and 7) the thesis follows. Notice that, since $p(y)$ is squarefree, the polynomials $p_1(y), \ldots, p_\ell(y)$ are pairwise coprime.

The running time of the algorithm is dominated by the gcd computation at each iteration of the **for** loop. Since each gcd involves polynomials of degree at most m the total cost is $O(n\mathsf{M}(m)\log m)$ operations in $\mathbf{Z}_p$ ■

We are now ready to describe a complete algorithm for computing the pseudo-factorization of a squarefree polynomial $p(y) \in \mathbf{Z}_p[y]$. The idea is to modify algorithm FastEEA so that every time Step 6 is executed we call algorithm Split to split the current modulus so that we can complete Step 6 for each one of the new moduli. This simple modification makes it possible to derive a new algorithm which takes as input two polynomials $r_0(x), r_1(x)$ with coefficients over $\mathbf{Z}_p[y]/(p(y))$ and returns a pseudo-factorization of $p(y)$ *and* the gcd with respect to the new moduli. More precisely, when called with $k = \deg(r_0)$, the algorithm PseudoFactEEA shown in Fig. 6 returns a sequence of pairs $\{\langle R_1, p_1(y)\rangle, \ldots, \langle R_\ell, p_\ell(y)\rangle\}$ such that

1. $p(y) = p_1(y) \cdots p_\ell(y)$ is a pseudo-factorization of $p(y)$;

2. For $i = 1, \ldots, \ell$, let

$$\begin{pmatrix} r_{0,i}(x) \\ r_{1,i}(x) \end{pmatrix} = \begin{pmatrix} r_0(x) \bmod p_i(y) \\ r_1(x) \bmod p_i(y) \end{pmatrix}, \qquad \begin{pmatrix} d_i(x) \\ e_i(x) \end{pmatrix} = R_i \begin{pmatrix} r_{0,i}(x) \\ r_{1,i}(x) \end{pmatrix}$$

   then $d_i(x) = \gcd(r_{0,i}(x), r_{1,i}(x))$, and the first row of R_i contains the corresponding Bézout coefficients.

We point out that the only conceptual difference between FastEEA and PseudoFactEEA is that the latter needs to call the procedure Split and therefore needs to maintain a list of moduli. Since also each recursive call (Steps 3 and 12 of PseudoFactEEA) returns a list of moduli the pseudo-code of Fig. 6 may appear quite complex. However, apart for the handling of the lists of moduli, which is done by the **for** loops at Steps 4, 9, and 13, the working of algorithm PseudoFactEEA is identical to algorithm FastEEA. As a consequence, the proof of the correctness of PseudoFactEEA can be obtained repeating step by step the proof given in [9, Chap. 11] for algorithm FastEEA. For what concerns the running time of PseudoFactEEA we have the following result.

Lemma 4.2 *The call* PseudoFactEEA$(r_0(x), r_1(x), n, p(y))$ *with* $n = \deg(r_0)$, $m = \deg(p(y))$ *takes* $O(\mathsf{M}(m) n \log n \log m + \mathsf{M}^*(n, m) \log n)$ *operations in* $\mathbf{Z}_p$.

Proof. We denote by $T(k, m)$ the number of operations over $\mathbf{Z}_p$ executed by PseudoFactEEA when called with parameters $r_0(x), r_1(x), k, p(y)$ with $m = \deg(p(y))$. As in the analysis of algorithm FastEEA given in [9, Chap. 11], we can assume that $\deg(r_1) < \deg(r_0) \le 2k$.

The recursive call at Step 3 takes $T(\lfloor k/2 \rfloor, m)$ operations. Let $m_j = \deg(p_j(y))$. We have $\sum_j m_j = m$ since $\prod_j p_j(y) = p(y)$. For the **for** loop starting at Step 4 we now compute the total cost of each instruction, that is, we sum the contribution of all iterations. At Step 5 we need to reduce each coefficient of r_0 and r_1 modulo $p_j(y)$ for any $p_j(y)$ in List1. With a tree like computation (see Sect. 3.3) reducing a single coefficient modulo every $p_j(y)$ takes $O(\mathsf{M}(m) \log m)$ operations, hence reducing both r_0 and r_1 takes overall $O(k\mathsf{M}(m) \log m)$ operations. Computing the pairs s_0, s_1 takes overall $\sum_j \mathsf{M}^*(2k, m_j) = O(\mathsf{M}^*(k, m))$ operations. Step 7 takes overall $\sum_j \mathsf{M}^*(2k, m_j) = O(\mathsf{M}^*(k, m))$ operations. By Lemma 4.1 we get that Step 8 takes overall $O(\sum_j k\mathsf{M}(m_j) \log m_j) = O(k\mathsf{M}(m) \log m)$ operations.

Let $m_{jz} = \deg(p_j^{(z)}(y))$. We have $\sum_{jz} m_{jz} = m$. Steps 10 and 11 takes overall $O(k\mathsf{M}(m) \log m)$ operations and Step 12 takes $\sum_{jz} T(\lfloor k/2 \rfloor, m_{jz})$ operations. At Step 14 the reductions modulo the $t_i(y)$'s take overall $O(k\mathsf{M}(m) \log m)$ operations. Similarly, at Step 15 the multiplication of the 2×2 matrices takes overall $\sum_{j,z,i} \mathsf{M}^*(k, \deg(t_i)) = O(\mathsf{M}^*(k, m))$ operations.

Summing up, we can find two constants c_1, c_2 such that

$$T(k, m) \le T(\lfloor k/2 \rfloor, m) + c_1 k\mathsf{M}(m) \log m + c_2 \mathsf{M}^*(k, m) + \textstyle\sum_{jz} T(\lfloor k/2 \rfloor, m_{jz}).$$

It is easy to see that $\sum_{jz} T(\lfloor k/2 \rfloor, m_{jz}) \le T(\lfloor k/2 \rfloor, m)$. This yields a simple recurrence for $T(k, m)$ whose solution is

$$T(k, m) = O(k \log k \mathsf{M}(m) \log m + \log k \mathsf{M}^*(k, m))$$

and the thesis follows. ∎

The following theorem establishes the time bound for our algorithm inverting a circulant matrix over $\mathbf{R} = \mathbf{Z}_p[y]/(a(y))$ when the factorization of $a(y)$ is unknown.

Algorithm $\textsc{PseudoFactEEA}(r_0(x), r_1(x), k, p(y))$

1. **if** $r_1(x) = 0$ **or** $k < \deg(r_0) - \deg(r_1)$ **return** $\left\{\left\langle \left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right), p(y)\right\rangle\right\}$;
2. $d \leftarrow \lfloor k/2 \rfloor$, $\quad \mathsf{Output} \leftarrow \{\}$;
3. $\mathsf{List1} \leftarrow \textsc{PseudoFactEEA}\left(r_0(x) \upharpoonright 2d, r_1(x) \upharpoonright [2d - (\deg(r_0) - \deg(r_1))], d, p(y)\right)$;
4. **for each pair** $\langle R_j, p_j(y)\rangle$ **in** $\mathsf{List1}$ **do**
5. $\quad \begin{pmatrix} s_0(x) \\ s_1(x) \end{pmatrix} \leftarrow R_j \begin{pmatrix} r_0(x) \bmod p_j(y) \\ r_1(x) \bmod p_j(y) \end{pmatrix}$;
6. $\quad$ **if** $s_1(x) = 0$ **or** $k < \deg(r_0) - \deg(s_1)$
   $\qquad \mathsf{Output} \leftarrow \mathsf{Output} \cup \{\langle R_j, p_j(y)\rangle\}$; **goto** Step 4
7. $\quad \tilde{s}_2(x) \leftarrow s_0(x) \operatorname{rem} s_1(x)$, $\quad d^* \leftarrow k - \deg(r_0) - \deg(s_1)$;
8. $\quad \left\{p_j^{(1)}(y), \ldots, p_j^{(\ell_j)}(y)\right\} \leftarrow \mathsf{Split}(\tilde{s}_2(x), p_j(y))$;
9. $\quad$ **for** $z = 1, \ldots, \ell_j$ **do**
10. $\qquad s_0^{(z)}(x) \leftarrow s_0(x) \bmod p_j^{(z)}(y)$, $s_1^{(z)}(x) \leftarrow s_1(x) \bmod p_j^{(z)}(y)$, $q^{(z)}(x) \leftarrow s_0^{(z)}(x) \operatorname{quo} s_1^{(z)}(x)$;
11. $\qquad \tilde{s}_2^{(z)}(x) \leftarrow \tilde{s}_2(x) \bmod p_j^{(z)}(y)$, $\quad \rho = \operatorname{lc}(\tilde{s}_2^{(z)}(x))$, $\quad s_2^{(z)}(x) = \rho^{-1}\tilde{s}_2^{(z)}(x)$;
12. $\qquad \mathsf{List2} \leftarrow \textsc{PseudoFactEEA}\left(s_1^{(z)} \upharpoonright 2d^*, s_2^{(z)} \upharpoonright [2d^* - (\deg(s_1) - \deg(s_2))], d^*, p_j^{(z)}(y)\right)$;
13. $\qquad$ **for each pair** $\langle S_i, t_i(y)\rangle$ **in** $\mathsf{List2}$ **do**
14. $\qquad\quad q_i(x) \leftarrow q^{(z)}(x) \bmod t_i(y)$, $\quad R_{ij} \leftarrow R_j \bmod t_i(y)$, $\quad \rho_i = \rho \bmod t_i(y)$;
15. $\qquad\quad \mathsf{Output} \leftarrow \mathsf{Output} \cup \left\langle S_i \left(\begin{smallmatrix} 0 & 1 \\ \rho_i^{-1} & q_i\rho_i^{-1} \end{smallmatrix}\right) R_{ij}, t_i(y) \right\rangle$;
16. **return** Output.

Figure 6: Algorithm for computing a pseudo-factorization $p(y) = p_1(y) \cdots p_\ell(y)$ and the extended gcd between $(r_0 \bmod p_i(y))$ and $(r_1 \bmod p_i(y))$, for $i = 1, \ldots, \ell$. See Fig. 4 for comments and notation.

Theorem 4.3 *An $n \times n$ circulant matrix over $\mathbf{R} = \mathbf{Z}_p[y]/(a(y))$, with $\deg(a(y)) = m$, can be inverted in $O\big(\mathsf{M}(m)\,\big(n\log n\log m + \log^2 m\big) + \mathsf{M}^*(n,m)\log n\big)$ operations in $\mathbf{Z}_p$.*

Proof. Our algorithm first computes the squarefree decomposition $a(y) = g_1(y)\,g_2^2(y)\cdots g_k^k(y)$. Then it computes a pseudo-factorization of each g_i using algorithm PseudoFactEEA. Finally we invert $f^{(y)}$ using Chinese remaindering and Newton-Hensel lifting as in Sect. 3.3.

Computing the squarefree decomposition takes $O(\mathsf{M}(m)\log m)$ operations in $\mathbf{Z}_p$. If $m_i = \deg(g_i)$, computing the pseudo-factorization of all g_i's takes $O(\sum_i n\log n\mathsf{M}(m_i)\log m_i + \mathsf{M}^*(n,m_i)\log n)$ which is $O(\mathsf{M}(m)n\log n\log m + \mathsf{M}^*(n,m)\log n)$ operations. The cost of inverting $f^{(y)}$ using Chinese remaindering and Newton-Hensel lifting is bounded by (8) and the theorem follows. ∎

5 Inversion of a BCCB(m,n) matrix over $\mathbf{Z}_p$

We are now ready to discuss the problem of inverting a BCCB(m,n) matrix A with coefficients in $\mathbf{Z}_p$. Let $f(x,y)$ denote the polynomial associated to A. In view of Lemma 2.1, the problem of inverting A is equivalent to the problem of inverting, modulo x^n-1, the polynomial $f^{(y)}(x) \in \mathbf{R}[x]$ with $\mathbf{R} = \mathbf{Z}_p[y]/(y^m-1)$.

If the factorization of y^m-1 is unknown, we use Theorem 4.3 and we get that we can invert A in

$$T_4(n,m) = O\big(\mathsf{M}(m)\,\big(n\log n\log m + \log^2 m\big) + \mathsf{M}^*(n,m)\log n\big) \tag{9}$$

operations in $\mathbf{Z}_p$. Notice that when we consider the problem of inverting $f(x,y)$ we can interchange the two variables and consider the problem of inverting $f^{(x)}(y) \in \mathbf{R}'[y]$ with $\mathbf{R}' = \mathbf{Z}_p[x]/(x^n-1)$. Therefore, the cost of inverting a BCCB(m,n) matrix is $O(\min(T_4(m,n), T_4(n,m)))$ operations in $\mathbf{Z}_p$. Note that if we take $m \le n$ a simple computation shows that the bound (9) becomes $O(\mathsf{M}^*(n,m)\log n)$. This means that the cost of inverting a BCCB(m,n) matrix is bounded asymptotically by the cost of computing $\log n$ products of bivariate polynomials in $\mathbf{Z}_p[x,y]$ with degree at most n in x and at most m in y. If we know the factorization of x^n-1 or y^m-1 then we can use Theorem 3.1 and get a slightly smaller operation count.

5.1 Inversion of a BCCB(m,n) matrix over $\mathbf{Z}_p$, when $n = 2^k$

Let A denote a BCCB(m,n) matrix with coefficients over $\mathbf{Z}_p$. We now show that if either m or n is a power of 2, we can transform the problem of inverting A into a problem of half the initial size. The technique described in this section is an extention of the algorithm proposed in [1] for the inversion of $n \times n$ circulant matrices over $\mathbf{Z}_m$ when n is a power of 2 and is inspired by the Graeffe method for the approximation of polynomial zeros [5, 8].

Assume for example that $n = 2^k$. The following lemma is the basic tool for reducing the problem of inverting a BCCB(m,n) matrix to the problem of inverting a BCCB$(m,n/2)$ matrix. The proof is identical to the proof of Lemma 5.1 in [1].

Lemma 5.1 *Let $f(x,y) \in \mathbf{Z}_p[x,y]/(x^n-1, y^m-1)$ and $n = 2^k$. If $f(x,y)$ is invertible over $\mathbf{Z}_p[x,y]/(x^n-1, y^m-1)$ then $f(-x,y)$ is invertible as well and the product $f(-x,y)f(x,y)$ does not contain odd power terms in the variable x.* ∎

The above lemma suggests that we can halve the size of the original problem by splitting each polynomial into its even and odd powers of x. Let $F(x^2, y) = f(x, y)f(-x, y) \bmod x^n - 1, y^m - 1$. By Lemma 5.1, if $f(x, y)$ is invertible the inverse $g(x, y)$ satisfies

$$F(x^2, y)g(x, y) \equiv f(-x, y) \pmod{x^n - 1,\ y^m - 1}. \tag{10}$$

Now we split $g(x, y)$ and $f(-x, y)$ in their odd and even x powers. We obtain

$$g(x, y) = T_e(x^2, y) + xT_o(x^2, y), \qquad f(-x, y) = S_e(x^2, y) + xS_o(x^2, y).$$

From (10) we have

$$F(x^2, y)[T_e(x^2, y) + xT_o(x^2, y)] \equiv S_e(x^2, y) + xS_o(x^2, y) \pmod{x^n - 1,\ y^m - 1}.$$

If $f(x, y)$ is invertible, $F(x^2, y)$ is invertible as well, its inverse being $g(x, y)g(-x, y)$. We can therefore retrieve $T_e(x^2, y)$ and $T_o(x^2, y)$ by solving the two subproblems

$$\begin{cases} F(x^2, y)T_e(x^2, y) \equiv S_e(x^2, y) \pmod{x^n - 1,\ y^m - 1}, \\ F(x^2, y)T_o(x^2, y) \equiv S_o(x^2, y) \pmod{x^n - 1,\ y^m - 1}. \end{cases}$$

Hence, to find $g(x, y)$ it suffices to compute the inverse of $F(x^2, y)$ and to execute two polynomial multiplications to retrieve T_e and T_o. The fundamental observation is that by setting $z = x^2$ we reduce the inversion of $F(x^2, y)$ to an inversion modulo $x^{(n/2)} - 1$, $y^m - 1$. Since $n/2$ is still a power of two we can apply the same approach recursively. The recursion stops when $n = 1$; at that point we have to to invert a (univariate) polynomial $F(y)$ modulo $y^m - 1$. This latter inversion can be done using the Extended Euclidean algorithm which takes $O(\mathsf{M}(m) \log m)$ operations in $\mathbf{Z}_p$.

Summing up, if we denote by $T(n, m)$ the cost of the above recursive algorithm, the following recurrence holds:

$$T(n, m) = \begin{cases} \mathsf{M}(m) \log m, & \text{if } n = 1; \\ \mathsf{M}^*(n, m) + T(n/2, m) + \mathsf{M}^*(n/2, m), & \text{otherwise.} \end{cases}$$

Unfolding the recurrence we get $T(n, m) = O(\mathsf{M}(m) \log m + \mathsf{M}^*(n, m))$. As a final comment we note that if m is a power of 2 as well, we can invert $F(y)$ modulo $y^m - 1$ using the algorithm described in [1, Sect. 5] which takes $O(\mathsf{M}(m))$ operations in $\mathbf{Z}_p$. As a result, the inversion of a BCCB(m, n) matrix when both dimensions are powers of 2 can be done in $O(\mathsf{M}^*(n, m))$ operations in $\mathbf{Z}_p$ (note that this is the same asymptotic cost of a single bivariate polynomial multiplication).

The results of this section are summarized by the following theorem.

Theorem 5.2 *If $n = 2^k$, we can invert a BCCB(m, n) matrix over $\mathbf{Z}_p$ in $O(\mathsf{M}(m) \log m + \mathsf{M}^*(n, m))$ operations in $\mathbf{Z}_p$. If m is a power of 2 as well, the cost drops to $O(\mathsf{M}^*(n, m))$ operations in $\mathbf{Z}_p$.* ∎

References

[1] D. Bini, G. M. Del Corso, G. Manzini, and L. Margara. Inversion of circulant matrices over $\mathbf{Z}_m$. *Mathematics of Computations*, 70:1169–1182, 2001.

[2] D. Bini and V. Y. Pan. *Polynomial and Matrix Computations, Fundamental Algorithms*, volume 1. Birkhäuser, 1994.

[3] P. J. Davis. *Circulant Matrices.* John Wiley & Sons, New York, 1979.

[4] P. Guan and Y. He. Exacts results for deterministic cellular automata with additive rules. *Jour. Stat. Physics*, 43:463–478, 1986.

[5] A. S. Householder. *The Numerical Treatment of a Single Nonliear Equation.* McGraw-Hill, New York, 1970.

[6] A. Lempel, G. Seroussi, and S. Winograd. On the complexity of multiplication in finite fields. *Theoretical Computer Science*, 22(3):285–296, February 1983.

[7] G. Manzini and L. Margara. Invertible linear cellular automata over Z_m: Algorithmic and dynamical aspects. *Journal of Computer and System Sciences*, 56:60–67, 1998.

[8] A. M. Ostrowski. Recherches sur la méthode de Graeffe et les zéros des polynomes et des series de laurent. *Acta Math.*, 72:99–257, 1940.

[9] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 1999.