Nearly Sparse Linear Algebra and application to Discrete Logarithms Computations

Antoine Joux, Cécile Pierrot

Nearly Sparse Linear Algebra

Antoine Joux[1,2,4] and Cécile Pierrot[3,4]

[1] CryptoExperts, France
[2] Chaire de Cryptologie de la Fondation de l'UPMC
[3] CNRS and Direction Générale de l'Armement
[4] Laboratoire d'Informatique de Paris 6, UMR 7606
Sorbonne Universités, UPMC Univ Paris 06,
4 place Jussieu 75005 Paris.
antoine.joux@m4x.org, Cecile.Pierrot@lip6.fr

Abstract. In this article, we propose a method to perform linear algebra on a matrix with nearly sparse properties. More precisely, although we require the main part of the matrix to be sparse, we allow some dense columns with possibly large coefficients. We modify Block Wiedemann algorithm and show that the contribution of these heavy columns can be made negligible compared to the one of the sparse part of the matrix. In particular, this eases the computation of discrete logarithms in medium and high characteristic finite fields, where *nearly sparse matrices* naturally appear.

Keywords. Sparse Linear Algebra. Block Wiedemann Algorithm. Discrete Logarithm. Finite Fields.

1 Introduction

Linear algebra is a widely used tool in both mathematics and computer science. At the boundary of these two disciplines, cryptography is no exception to this rule. Yet, one notable difference is that cryptographers mostly consider linear algebra over finite fields, bringing both drawbacks – the notion of convergence is no longer available – and advantages – no stability problems can occur. As in combinatory analysis or in the course of solving partial differential equations, cryptography also presents the specificity of frequently dealing with sparse matrices. For instance, sparse linear systems over finite fields appeared in cryptography in the late 70s when the first sub-exponential algorithm to solve the discrete logarithm problem in finite fields with prime order was designed [Adl79]. Nowadays, every algorithm belonging to the Index Calculus family deals with a sparse matrix [JOP14, Section 3.4]. Hence, since both Frobenius Representation Algorithms (for small characteristic finite fields) and discrete logarithm variants of the Number Field Sieve (for medium and high characteristics) belong to this Index Calculus family, all recent discrete logarithm records on finite fields need to find a solution of a sparse system of linear equations modulo a large integer. Similarly, all recent record-breaking factorizations of composite numbers, which

are based on the Number Field Sieve, need to perform a sparse linear algebra step modulo 2.

A sparse matrix is a matrix containing a relatively small number of coefficients that are not equal to zero. It often takes the form of a matrix in which each row (or column) only has a small number of non-zero entries, compared to the dimension of the matrix. With sparse matrices, it is possible to represent in computer memory much larger matrices, by giving for each row (or column) the list of positions containing a non-zero coefficient, together with its value. When dealing with a sparse linear system of equations, using plain Gaussian Elimination is often a bad idea, since it does not consider nor preserve the sparsity of the input matrix. Indeed, each pivoting step during Gaussian Elimination increases the number of entries in the matrix and, after a relatively small number of steps, it overflows the available memory.

Thus, in order to deal with sparse systems, a different approach is required. Three main families of algorithms have been devised: the first one adapts the ordinary Gaussian Elimination in order to choose pivots that minimize the loss of sparsity and is generally used to reduce the initial problem to a smaller slightly less sparse problem. The two other algorithm families work in a totally different way. Namely, they do not try to modify the input matrix but aim at directly finding a solution of the sparse linear system by computing only matrix-by-vector multiplications. One of these families consists of Krylov Subspace methods, adapted from numerical analysis, and constructs sequences of mutually orthogonal vectors. For instance, this family contains the Lanczos and Conjugate Gradient algorithms, adapted for the first time to finite fields in 1986 [COS86].

Throughout this article, we focus on the third family that contains Wiedemann algorithm and its generalizations. Instead of computing an orthogonal family of vectors, D. Wiedemann proposed in 1986 [Wie86] to reconstruct the minimal polynomial of the considered matrix. This algorithm computes a sequence of scalars of the form ${}^twA^iv$ where v and w are two vectors and A the sparse matrix of the linear algebra problem. It tries then to extract a recursive relationship that holds for this sequence. In 1994, to achieve computations in a realistic time, D. Coppersmith [Cop94] adapted Wiedemann algorithm over the finite field $\mathbb{F}_2$ for parallelization and even distributed computations. This was then generalized to arbitrary finite fields by E. Kaltofen [Kal95]. The main idea of Coppersmith's Block Wiedemann algorithm is to compute a sequence of matrices of the form ${}^tWA^iV$ where V and W are not vectors as previously but *blocks* of vectors. This step is parallelized by distributing the vectors of the block V to several processors or CPUs – let us say c. The asymptotic complexity of extracting the recursive relationships within the sequence of small matrices is in $\tilde{O}(cN^2)$ where N is the largest dimension of the input matrix. An other subquadratic algorithm was presented by B. Beckerman and G. Labahn for the same task in 1994 [BL94]. Finally, a further improvement was proposed by E. Thomé [Tho02] in 2002: he reduced the complexity of finding the recursive relationships to $\tilde{O}(c^2N)$.

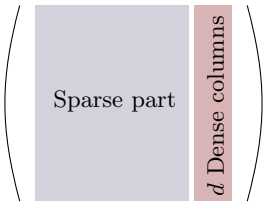


Fig. 1. A nearly sparse matrix

Note that both Krylov Subspace methods and Wiedemann algorithms cost a number of matrix-by-vector multiplications equal to a small multiple of the matrix dimension: for a matrix containing λ entries per row in average, the cost of these matrix-by-vector multiplications is $O(\lambda N^2)$. With Block Wiedemann, it is possible to distribute the cost of these products on c machines. In this case, the search for recursive relationships adds an extra cost of the form $\tilde{O}(c^2 N)$. For a *nearly sparse matrix*, which includes d dense columns in addition to its sparse part, the cost of matrix-by-vector multiplications increases. As a consequence, the total complexity becomes $O((\lambda + d)N^2)$ with an extra cost of $\tilde{O}(c^2 N)$ for Block Wiedemann. Figure 1 provides a quick overview of the structure of such nearly sparse matrices.

In this article, our aim is to adapt the Coppersmith's Block Wiedemann algorithm to improve the cost of linear algebra on matrices that have nearly sparse properties and reduce it to $O(\lambda N^2) + \tilde{O}(\max(c, d)^2 N)$. In particular, when the number of dense columns is lower than the number of processors we use for the matrix-by-vector steps, we show that the presence of these unwelcome columns does not affect the complexity of solving linear systems associated to these matrices. In practice, this result precisely applies to the discrete logarithm problem. Indeed, nearly sparse matrices appear in both medium and high characteristic finite fields discrete logarithm computations. To illustrate this claim, we recall the latest record [BGI+14] announced in June 2014 for the computation of discrete logarithms in a prime field $\mathbb{F}_p$, where p is a 180 digit prime number. It uses a matrix containing 7.28M rows and columns with an average weight of 150 non-zero coefficients per row and also presents 4 dense Schirokauer maps columns. These columns precisely give to the matrix the nearly sparse structure we study in the sequel.

Outline. Section 2 makes a short recap on Coppersmith's Block Wiedemann algorithm, which is the currently best known algorithm to perform algebra on sparse linear systems while tolerating some amount of distributed computation. We propose in Section 3 the definition of a *nearly sparse matrix* and present then an algorithm to solve linear algebra problems associated to these special matrices. In Section 3.5 we give a comparison of our method with preexisting linear algebra techniques and show that it is competitive even with a large number of dense columns. Section 4 ends by a practical application of this new result: it

explains how nearly sparse linear algebra eases discrete logarithm computations in medium and high characteristic finite fields.

2 Coppersmith's Block Wiedemann Algorithm

This section first presents the classical problems of linear algebra that are encountered when dealing with sparse matrices. We then explain how the considered matrix is preconditioned into a square matrix. Section 2.2 draws the outline of the algorithm proposed by Wiedemann to solve linear systems given by a square matrix whereas Section 2.3 presents the parallelized variant due to Coppersmith. More precisely, our goal is to solve:

Problem 1. Let $\mathbb{K} = \mathbb{Z}/p\mathbb{Z}$ be a prime finite field and $S \in \mathcal{M}_{n\times N}(\mathbb{K})$ be a (non necessarily square) sparse matrix with at most λ non-zero coefficients per row. Let v be a vector with n coefficients. The problem is to find a vector x with N coefficients such that $S \cdot x = v$ or, alternatively, a non-zero vector x such that $S \cdot x = 0$.

In practice, this problem can be generalized to rings $\mathbb{Z}/\mathcal{N}\mathbb{Z}$ for a modulus $\mathcal{N}$ of unknown factorization. However, for simplicity of exposition, we prefer to only present the prime field case.

2.1 Preconditioning: from a sparse matrix to a square matrix

In order to be able to compute sequences of matrix-by-vector products of the form $(A^i y)_{i>0}$, both Wiedemann and Block Wiedemann algorithms need to work with a square matrix. Indeed, powers are only defined for square matrices. Consequently, if $N \neq n$, the first preconditioning stage is to create a random sparse matrix $R \in \mathcal{M}_{N\times n}(\mathcal{R})$ with at most λ non-zero coefficients per row, and transform afterwards the two problems into finding a vector x such that $(RS)x = Rv$ or, alternatively, such that $(RS)x = 0$. Setting $A = RS$ and $y = Rv$, we can rewrite Problem 1 as finding a vector x such that:

$$A \cdot x = y$$

or such that:

$$A \cdot x = 0$$

depending on the initial problem. The choice of R needs to be constrained to guarantee that left-multiplication by R does not change the set of solutions. Generically, this property holds for a random matrix. Moreover, since it is always possible and easy to check the obtained solution, we can safely neglect the very low probability of selecting a wrong R. Note that the square matrix A of dimension $N \times N$ is no longer sparse. Yet, considering the fact that it is defined as the product of two sparse matrices, it is still possible to efficiently compute a product of A by any vector of the right size. Namely, considering first the

product of S by the corresponding vector and then the product of R with the resulting vector, it can be done in $O(\lambda \cdot \max(n, N))$ operations.

An alternative would be to precondition by multiplying by a matrix R on the right. However, since S has more rows than columns in discrete logarithm applications, the resulting square matrix A would be (somewhat) larger. This explains why we prefer here to precondition on the left.

2.2 Wiedemann algorithm

Let us now consider a square matrix A of size $N \times N$ and denote m_A the number of operations required to compute the product of a vector of $\mathbb{K}^N$ by A. Wiedemann algorithm works by finding a non-trivial sequence of coefficients $(a_i)_{0 \le i \le N}$ such that:

$$\sum_{i=0}^{N} a_i A^i = 0. \tag{1}$$

Solving $Ax = y$. If A is invertible, then we can assume that $a_0 \neq 0$. Indeed, if $a_0 = 0$ we can rewrite $0 = \sum_{i=1}^{N} a_i A^i = A(\sum_{i=1}^{N} a_i A^{i-1})$. Multiplying by A^{-1} it leads to $\sum_{i=0}^{N-1} a_{i+1} A^i = 0$. So, shifting the coefficients until we find the first non-zero one allows to write $a_0 \neq 0$. Let us apply Equation (1) to the vector x we are seeking. It yields $-a_0 x = \sum_{i=1}^{N} a_i A^i x = \sum_{i=1}^{N} a_i A^{i-1}(Ax)$. Finally we recover $x = -(1/a_0) \sum_{i=1}^{N} a_i A^{i-1} y$. This last sum can be computed using N sequential multiplications of the initial vector y by the matrix A. The total cost to compute x as this sum is $O(N \cdot m_A)$ operations.

Solving $Ax = 0$. Assuming that there exists a non-trivial element of the kernel of A, we deduce that $a_0 = 0$. Thus, for any vector r we have $0 = \sum_{i=1}^{N} a_i A^i r = A(\sum_{i=1}^{N} a_i A^{i-1} r)$. Computing $\sum_{i=1}^{N} a_i A^{i-1} r$ gives a random (usually non-zero) element of the kernel of A in $O(N \cdot m_A)$ operations as well.

Algorithm 1 Wiedemann algorithm

Require: a matrix A of size $N \times N$, y a vector with N coefficients
Ensure: x such that $A \cdot x = y$ or $A \cdot x = 0$.

Computing a sequence of scalars

1: $v_0 \leftarrow_\in \mathbb{K}^N$, $w \leftarrow_\in \mathbb{K}^N$ two random vectors
2: **for** $i = 0, \cdots, 2N$ **do**
3: $\quad \lambda_i \leftarrow {}^t w v_i$
4: $\quad v_{i+1} \leftarrow A v_i$
5: **end for**

Berlekamp-Massey algorithm

6: From $\lambda_0, \cdots, \lambda_{2N}$ recover coefficients $(a_i)_{0 \le i \le N}$ such that $\sum_{i=0}^{N} a_i A^i = 0$.

Resolution

7: **return** $\sum_{i=1}^{N} a_i A^{i-1} r$ with r a random vector to solve $A \cdot x = 0$, or $-(1/a_0) \sum_{i=1}^{N-1} a_{i+1} A^i y$ to solve $A \cdot x = y$.

How to find coefficients a_i verifying Equation (1). Cayley-Hamilton theorem testifies that the polynomial defined as $P = \det(A - X \cdot \mathrm{Id})$ annihilates the matrix A, *i.e.* $P(A) = 0$. So we know that there exists a polynomial of degree at most N whose coefficients satisfy Equation (1). Yet, directly computing such a polynomial would be too costly. The idea is, in fact, to process by necessary conditions.

Let $(a_i)_{0 \le i \le N}$ be such that $\sum_{i=0}^{N} a_i A^i = 0$. Then, for any arbitrary vector v we obtain $\sum_{i=0}^{N} a_i A^i v = 0$. Again, for any arbitrary vector w and for any integer j we can write $\sum_{i=0}^{N} a_i {}^t w A^{i+j} v = 0$. Conversely, if $\sum_{i=0}^{N} a_i {}^t w A^{i+j} v = 0$ for any random vectors v and w and for any $0 \le j \le N$, then the probability to obtain coefficients verifying Equation (1) is close to 1 when $\mathbb{K}$ is large. Thus, Wiedemann algorithm seeks coefficients a_i that annihilate the sequence of scalars ${}^t w A^i v$. To do so, we use the classical Berlekamp-Massey algorithm [Ber68,Mas69] that finds the minimal polynomial of a recursive linear sequence in an arbitrary field. In a nutshell, the idea is to consider the generating function f of the sequence ${}^t wv, {}^t wAv, {}^t wA^2 v, \cdots, {}^t wA^{2N} v$ and to find afterwards two polynomials g and h such that $f = g/h \mod X^{2N}$. Alternatively, the Berlekamp-Massey algorithm can be replaced by a half extended Euclidean algorithm, yielding a quasi-linear algorithm in the size of the matrix A.

2.3 Coppersmith's Block Wiedemann algorithm

Due to Don Coppersmith, the Block Wiedemann algorithm is a parallelization of the previous Wiedemann algorithm. Let us consider that we have l processors. Instead of seeking for coefficients satisfying Equation (1), we assume in this case that we are able, given l linearly independent vectors $v_1, \cdots, v_l$, to finds coefficients a_{ij} such that:

$$\sum_{j=1}^{l} \sum_{i=0}^{\lfloor N/l \rfloor + 1} a_{ij} A^i v_j = 0 \tag{2}$$

Note that we are looking for approximately as many coefficients as in the previous algorithm.

Solving $Ax = y$. We suppose in this case that A is invertible. Let us set $v_1 = y$ and choose for $2 \le i \le l$ the vectors $v_i = Ar_i$, where r_i is a random vector of the right size. From Equation (2) we derive $0 = \sum_{i=0}^{\lfloor N/l \rfloor + 1} a_{i1} A^i y + \sum_{j=2}^{l} \sum_{i=0}^{\lfloor N/l \rfloor + 1} a_{ij} A^{i+1} r_j$, and, then, multiplying by the inverse of A, we obtain: $0 = \sum_{i=0}^{\lfloor N/l \rfloor + 1} a_{i1} A^i x + \sum_{j=2}^{l} \sum_{i=0}^{\lfloor N/l \rfloor + 1} a_{ij} A^i r_j$. Thus, we can recover x by computing:

$$(-1/a_{01}) \cdot \left(\sum_{i=1}^{\lfloor N/l \rfloor + 1} a_{i1} A^{i-1} y + \sum_{j=2}^{l} \sum_{i=0}^{\lfloor N/l \rfloor + 1} a_{ij} A^i r_j \right).$$

This can be done with a total cost of $O(N \cdot m_A)$ operations parallelized over the l processors: each one is given a vector r and deals with the computation of one sequence of matrix-by-vector products of the form $A^i r$, for a cost of $O(N \cdot m_A / l)$ operations per sequence.

Solving $Ax = 0$. Again, we choose l random vectors $r_1, \cdots, r_l$ and set $v_i = Ar_i$. Equation (2) gives $\sum_{j=1}^{l} \sum_{i=0}^{\lfloor N/l \rfloor+1} a_{ij} A^{i+1} r_j = 0$, *i.e.* $A(\sum_{j=1}^{l} \sum_{i=0}^{\lfloor N/l \rfloor+1} a_{ij} A^i r_j) = 0$. Finally, $\sum_{j=1}^{l} \sum_{i=0}^{\lfloor N/l \rfloor+1} a_{ij} A^i r_j$ is a usually non-trivial element of the kernel of A.

Algorithm 2 Block Wiedemann algorithm

Require: a matrix A of size $N \times N$, y a vector with N coefficients
Ensure: x such that $A \cdot x = y$ or $A \cdot x = 0$.

Computing a sequence of matrices

$r_1 \leftarrow\in \mathbb{K}^N, \cdots, r_l \leftarrow\in \mathbb{K}^N$ and $w_1 \leftarrow\in \mathbb{K}^N, \cdots, w_l \leftarrow\in \mathbb{K}^N$
$v_1 \leftarrow y$ to solve $A \cdot x = y$ or $v_1 \leftarrow Ar_1$ to solve $A \cdot x = 0$
$v_2 \leftarrow Ar_2, \cdots, v_l \leftarrow Ar_l$
for any of the l processors indexed by j **do**
  $u_0 \leftarrow v_j$
  for $i = 0, \cdots, 2(\lfloor N/l \rfloor + 1)$ **do**
    for $k = 1, \cdots, l$ **do**
      $\lambda_{i,j,k} \leftarrow {}^t w_k u_i$
      $u_{i+1} \leftarrow Au_i$
    end for
  end for
end for
for $i = 0, \cdots, 2(\lfloor N/l \rfloor + 1)$ **do**
  $M_i \leftarrow (\lambda_{i,j,k})$ the $l \times l$ matrix containing all the products of the form ${}^t w A^i v$
end for

Thomé algorithm

From $M_0, \cdots, M_{2(\lfloor N/l \rfloor+1)}$ recover coefficients a_{ij} such that $\sum_{j=1}^{l} \sum_{i=0}^{\lfloor N/l \rfloor+1} a_{ij} A^i v_j = 0$.

Resolution

return $(-1/a_{01}) \cdot (\sum_{i=1}^{\lfloor N/l \rfloor+1} a_{i1} A^{i-1} y + \sum_{j=2}^{l} \sum_{i=0}^{\lfloor N/l \rfloor+1} a_{ij} A^i r_j)$ to solve $A \cdot x = y$ or $\sum_{j=1}^{l} \sum_{i=0}^{\lfloor N/l \rfloor+1} a_{ij} A^i r_j$ to solve $A \cdot x = 0$.

How to find coefficients a_i *verifying Equation* (2). Let $v_1, \cdots, v_l$ be l vectors and let consider the $l(\lfloor N/l \rfloor + 1)$ elements obtained by the matrix-by-vector products of the form $A^i v_j$ that appear in the sum of Equation (2). Since $l(\lfloor N/l \rfloor + 1) > N$, all these vectors cannot be independent, so there exist coefficients satisfying (2). As for Wiedemann algorithm, we process by necessary conditions. More precisely, let $w_1, \ldots, w_l$ be l vectors. Assume that for any $0 \le \kappa \le \lfloor N/l \rfloor + 1$ and $1 \le k \le l$ we have $\sum_{j=1}^{l} \sum_{i=0}^{\lfloor N/l \rfloor+1} a_{ij} {}^t w_k A^{i+\kappa} v_j = 0$, then the probability that the coefficients a_{ij} verify Equation (2) is close to 1 when $\mathbb{K}$ is large[5]. So Block Wiedemann algorithm looks for coefficients that annihilate the sequence of $2\lfloor N/l \rfloor + 1$ small matrices of dimension $l \times l$ computed as $\left({}^t w_k A^\kappa v_j\right)$. Here, $0 \le \kappa \le 2\lfloor N/l \rfloor$ numbers the matrices, while k and j respectively denote the column and row numbers within each matrix. Without going into tricky details, we recall that

[5] When $\mathbb{K}$ is small, it is easy to make the probability close to 1 by increasing the number of vectors w beyond l in the analysis as done in [Cop94].

it is possible to compute the coefficients a_{ij} in subquadratic time. For instance, in [Tho02] Thomé gives an efficient method with complexity $\tilde{O}(l^2N)$. A bird's eye view of the procedure is given in Algorithm 2.

As a consequence, putting together the matrix-by-vector products and the search for coefficients, the overall complexity is expressed as $O(N \cdot m_A) + \tilde{O}(l^2N)$. Where the $O(N \cdot m_A)$ part can be distributed on up to l processors and the $\tilde{O}(l^2N)$ part is computed sequentially.

Remark 1. In this section, we assumed that the number of sequences l is equal to the number of processors c. This is the most natural choice in the classical application of Block Wiedemann, since increasing l beyond the number of processors can only degrades the overall performance. More precisely, the change leaves the $O(N \cdot m_A)$ contribution unaffected but increases $\tilde{O}(l^2N)$. However, since values of l larger than c are considered in Section 3, it is useful to know that this can be achieved by sequentially computing several independent sequences on each processor. In this case, it is a good idea in practice to make the number of sequences a multiple of the number of processors, in order to minimize the wall clock running time of the matrix-by-vector multiplications.

3 Nearly Sparse Linear Algebra

3.1 Nearly sparse matrices

We now aim at adapting the Block Wiedemann algorithm to improve the resolution of some linear algebra problems that are not exactly sparse but close enough to be treated similarly. In the sequel we focus on linear systems given as follows:

Problem 2. Let M be a matrix of size $N \times (s + d)$ with coefficients in a ring $\mathbb{K}$. We assume that there exist two smaller matrices $M_s \in \mathcal{M}_{N\times s}(\mathbb{K})$ and $M_d \in \mathcal{M}_{N\times d}(\mathbb{K})$ such that :

1. $M = M_s|M_d$, where $|$ is the concatenation of matrices.
2. M_s is sparse. Let us assume it has at most λ non-zero coefficients per row.

If y is a given vector with N coefficients, the problem is to find a vector x with $s + d$ coefficients such that:

$$M \cdot x = y$$

or, alternatively, such that:

$$M \cdot x = 0.$$

Such a matrix M is said to be *d-nearly sparse*, or as a shortcut, simply *nearly sparse* when d is implied by context.[6]

[6] Note that there is no theoretical restriction on the number of dense columns that appear in the matrix !

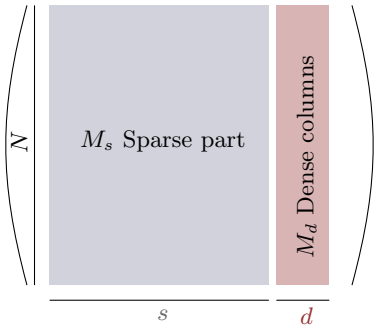


Fig. 2. Parameters of the nearly spare linear algebra problem

Interestingly, the two variants that appear in Problem 2 are much more related to each other than their counterparts in Problem 1. Indeed, it is easy to transform the resolution of $M \cdot x = y$ into the resolution of $M' \cdot x' = 0$ for a nearly sparse matrix M' closely related to M. It suffices to set $M' = M|y$ the matrix obtained by concatenating one additional dense column equal to y to the right of M. We now see that x is a solution of $M \cdot x = y$ if and only if $(x| - 1)$ is a solution of $M' \cdot x' = 0$. Keeping this transformation in mind, in the sequel we only explain how to find a non-trivial element of the kernel of a nearly sparse matrix.

With such a nearly sparse matrix M, it is of course possible to directly apply the usual Block Wiedemann algorithm. However, in this case, the cost of multiplying a vector by the matrix M becomes larger, of the order of $(\lambda + d)N$ operations. As a consequence, the complete cost of the usual Block Wiedemann algorithm becomes $O((\lambda + d) \cdot N^2) + \tilde{O}(l^2 N)$ when using l processors.

The cornerstone of our method consists in working with the sparse part M_s of the matrix while forcing the initial vectors of the sequences computed by Block Wiedemann algorithm to be derived from the dense columns of M_d, instead of choosing random initial vectors. In the rest of this section, we describe this idea in details.

3.2 Preconditioning: from a nearly sparse matrix to a square matrix

Unfortunately we do not know how to use preconditioning on the left in our new context. As a consequence, we consider preconditioning on the right. In order to find an element of the kernel of M, we first search for an element x of the kernel of MR for some random matrix R. When x is in the kernel of MR then Rx is in the kernel of M. However, we want to ensure that Rx is not systematically the zero vector. As long as a non-zero element of the kernel of M is in the image of R, the dimension of the kernel of MR is larger that the dimension of the kernel of R. Thus, for a random element x in the kernel of MR it is unlikely that Rx is zero. In particular, if R is surjective, this is always true.

To precondition our nearly sparse matrix $M = M_s|M_d$ of dimension $N \times (s+d)$ we assume that $N \geq s+d$, *i.e.* that M has more rows than columns. We construct the matrix R as a block matrix given by:

$$R = \left(\begin{array}{c|c} R_s & 0 \\ \hline 0 & I_d \end{array}\right),$$

where R_s is a randomly chosen sparse matrix of dimension $s \times N$ and I_d is the $d \times d$ identity matrix. We see that R is surjective, if and only if R_s is surjective. Moreover, using matrix block-multiplication, we find that:

$$MR = (M_s R_s | M_d),$$

where $M_s R_s$ is a square matrix of dimension N.

We now write $\delta_1, \cdots, \delta_d$ the d columns of M_d and remark that finding an element in the kernel of MR is equivalent to finding a reduced vector x and d coefficients $\varkappa_1, \ldots, \varkappa_d$ such that:

$$Ax + \sum_{i=1}^{d} \varkappa_i \, \delta_i = 0, \tag{3}$$

where $A = M_s R_s$. Indeed, ${}^t(x|\varkappa_1|\cdots|\varkappa_d)$ is then in the kernel of MR.

After this preconditioning step, the idea is to apply the Block Wiedemann algorithm to A with these vectors δ_i as inputs. Remember that δ_i are precisely the dense columns of M.

3.3 Linear algebra on M_s gives an element of $\ker(M)$

We consider now the matrix A and the vectors $\delta_1, \cdots, \delta_d$. Let $l \geq d$ be the smallest multiple of the number of available processors larger than d. We are going to use Block Wiedemann algorithm as presented in Section 2 with l sequences. The first d initial sequences are initialized using $v_i = \delta_i$ and the $l - d$ remaining sequences are initialized using $v_i = A \cdot r_i$ where $r_{d+1}, \ldots, r_l$ are randomly chosen.

As usual, we are able to recover coefficients a_{ij} satisfying Equation (2). Namely we obtain:

$$\sum_{j=1}^{l} \sum_{i=0}^{\lfloor N/l \rfloor + 1} a_{ij} A^i v_j = 0.$$

In other words, we have found coefficients such that:

$$\sum_{j=1}^{d} \sum_{i=0}^{\lfloor N/l \rfloor + 1} a_{ij} A^i \delta_j + \sum_{j=d+1}^{l} \sum_{i=0}^{\lfloor N/l \rfloor + 1} a_{ij} A^{i+1} r_j = 0.$$

Rewritting this last equation as:

$$A\left(\sum_{j=1}^{d} \sum_{i=1}^{\lfloor N/l \rfloor + 1} a_{ij} A^{i-1} \delta_j + \sum_{j=d+1}^{l} \sum_{i=0}^{\lfloor N/l \rfloor + 1} a_{ij} A^i r_j\right) + \sum_{j=1}^{d} a_{0j} \delta_j = 0$$

we obtain the desired solution of Equation (3).

Remark 2. A simple and interesting case occurs when $c = d$. Indeed, in this case, we can choose $l = c = d$ and all sequences can be initialized from δ_i vectors, without any need to add random vectors. Moreover, the final equation then simplifies to:

$$A\left(\sum_{j=1}^{d}\sum_{i=1}^{\lfloor N/d\rfloor+1} a_{ij}A^{i-1}\delta_j\right) + \sum_{j=1}^{d} a_{0j}\delta_j = 0.$$

3.4 Complexity analysis

The total cost of our method contains two parts. One part is the complexity of the matrix-by-vector products whose sequential cost is $O(\lambda N^2)$ including the preparation of the sequence of $l \times l$ matrices and the final computation of the kernel element. It can easily be distributed on several processors by setting the number of sequences l to be the smallest multiple of the number of processors such that $l \geq d$. This choice minimizes the wall clock time of the matrix-by-vector phases at $O(\lambda N^2/c)$. Moreover, the phase that recovers the coefficients a_{ij} has complexity $\tilde{O}(l^2 N)$ using Thomé's algorithm. Since we need $l \geq d$ to be able to perform our algorithm and since we require $l \geq c$ to minimize the wall clock time, we can finally express our complexity as a function of the number d of dense columns and of the number c of processors:

$$O(\lambda N^2) + \tilde{O}(\max(c,d)^2 N).$$

This has to be compared with the previous $O((\lambda + d)N^2) + \tilde{O}(c^2 N)$ obtained when combining Block Wiedeman algorithm with Thomé variant to solve the same problem. Note that the wall clock time decreases in the meantime from $O((\lambda + d)N^2/c) + \tilde{O}(c^2 N)$ to $O(\lambda N^2/c) + \tilde{O}(\max(c,d)^2 N)$.

3.5 How many dense columns could we have ?

Let us fix c the number of processors.

If $d \leq c$ then the complexity of the variant we propose is clearly in $O(\lambda N^2) + \tilde{O}(c^2 N)$, which is exactly the complexity obtained when combining Block Wiedeman algorithm with Thomé variant to solve a linear algebra problem on a (truly) sparse matrix of the same size. In a nutshell, our variant is not only better than previous algorithms but, **when parallelizing on c processors, it is possible to tackle up to c dense columns for free.**

If $c < d \leq N^{1-\epsilon}$ with $\epsilon > 0$, then our complexity becomes $O(\lambda N^2) + \tilde{O}(d^2 N)$. Thus, assuming that $\tilde{O}(c^2 N)$ is negligible compared to $O(dN^2)$, the question is now to compare $\tilde{O}(d^2 N)$ (coming from our algorithm) with $O(dN^2)$ (coming from Thomé variant of Block Wiedemann). From $d \leq N^{1-\epsilon}$ we can write $d = N^{1-\epsilon'}$ with $\epsilon' > 0$. So there exists $\alpha > 0$ such that $\tilde{O}(d^2 N) = O(N^{3-2\epsilon'}(\log N)^\alpha)$ and $O(dN^2) = O(N^{3-\epsilon'})$. Since asymptotically $(\log N)^\alpha$ is negligible compared

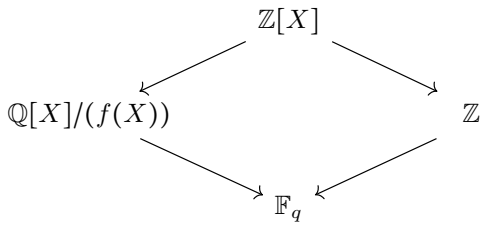
to $N^{\epsilon'}$, we conclude that, **as soon as $d \leq N^{1-\epsilon}$, the variant we propose to deal with nearly sparse matrices has a lower asymptotic complexity than previous variants of Block Wiedemann algorithm applied on the same matrices.**

Of course, when d becomes that large, it would make more sense to compare with dense methods, *i.e.* to compare our complexity with N^{ω} for some $\omega \in]2;3]$ depending on the dense algorithm we are considering. **In this case, we see that our algorithm remains competitive up to $d = N^{\omega-2-\epsilon}$.**

4 Application to Discrete Logarithm Computations in Medium and High Characteristic Finite Fields

In this section, we discuss the application of our adaptation of Block Wiedemann to discrete logarithm computations using the Number Field Sieve (NFS) algorithm [LHWL93,JLSV06,Pie15], which applies to medium and high characteristic finite fields $\mathbb{F}_q$.

NFS contains several phases. First, a preparation phase constructs a commutative diagram of the form:



when using a rational side. Even if there also exists a generalization with a number field on each side of the diagram, for the sake of simplicity, we only sketch the description of the rational-side case.

The second phase builds multiplicative relations between the images in $\mathbb{F}_q$ of products of ideals of small prime norms in the number field $\mathbb{Q}[X]/(f(X))$ and products of small primes. These relations are then transformed into linear relations between virtual logarithms of ideals and logarithms of primes modulo the multiplicative order of $\mathbb{F}_q^*$. Writing down these linear relations requires to get rid of a number of technical obstructions. In practice, this means that each relation is completed using a few extra unknowns in the linear system whose coefficients are computed from the so-called Schirokauer's maps [Sch93]. Essentially, these maps represent the contribution of units from the number field in the equations. Due to the way they are computed, each of these maps introduces a dense column in the linear system of equations. The total number of such columns is upper-bounded by the degree of f (or the sum of the degrees when there are two number fields in the diagram).

The third phase is simply the resolution of the above linear system. In a final phase which we do not describe, NFS computes individual logarithms of field elements. An optimal candidate to apply our adaptation of Coppersmith's Block

Wiedemann algorithm precisely lies in this third sparse linear algebra phase. Indeed, the number of dense columns is small enough to be smaller than the number of processors that one would expect to use in such a computation. Typically, in [BGI+14], the degree of the number field was 5, whereas the number of maps (so the number of dense columns) was 4, and the number of processors 12. Asymptotically, we know that in the range of application of NFS, the degree of the polynomials defining the number fields are at most $O((\log q/\log\log q)^{2/3})$. This is negligible compared to the size of the linear system, which is about $L_q(1/3) = \exp(O((\log q)^{1/3}(\log\log q)^{2/3}))$.

Thus, our new adaptation of Coppersmith's Block Wiedemann algorithm completely removes the difficulty of taking care of the dense columns that appear in this context. It is worth noting that these dense columns were a real practical worry and that other, less efficient, approaches have been tried to lighten the associated cost. For instance, in [BGGM14], the construction of the commutative diagram was replaced by a sophisticated method based on automorphisms to reduce the number of maps required in the computation.

Moreover, since we generally have some extra processors in practice, it is even possible to consider the columns corresponding to very small primes or to ideals of very small norms as part of the dense part of the matrix and further reduce the cost of the linear algebra.

References

[Adl79] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Foundations of Computer Science, 20th Annual Symposium on*, pages 55–60. IEEE, 1979.

[Ber68] Elwyn R. Berlekamp. Nonbinary BCH decoding (abstr.). *IEEE Transactions on Information Theory*, 14(2):242, 1968.

[BGGM14] Razvan Barbulescu, Pierrick Gaudry, Aurore Guillevic, and François Morain. Improvements to the number field sieve for non-prime finite fields. INRIA Hal Archive, Report 01052449, 2014.

[BGI+14] Cyril Bouvier, Pierrick Gaudry, Laurent Imbert, Hamza Jeljeli, and Emmanuel Thomé. Discrete logarithms in GF(p) – 180 digits, June 2014. Announcement to the NMBRTHRY list, item 003161.

[BL94] Bernhard Beckermann and George Labahn. A uniform approach for the fast computation of matrix-type Padé approximants. *SIAM Journal on Matrix Analysis and Applications*, 15(3):804–823, 1994.

[Cop94] Don Coppersmith. Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. *Mathematics of Computation*, 62:333–350, 1994.

[COS86] Don Coppersmith, Andrew M. Odlyzko, and Richard Schroeppel. Discrete logarithms in GF(p). *Algorithmica*, 1(1):1–15, 1986.

[JLSV06] Antoine Joux, Reynald Lercier, Nigel Smart, and Frederik Vercauteren. The number field sieve in the medium prime case. In *Advances in Cryptology-CRYPTO 2006*, pages 326–344. Springer, 2006.

[JOP14] Antoine Joux, Andrew Odlyzko, and Cécile Pierrot. The past, evolving present and future of discrete logarithm. In *Open Problems in Mathematics and Computational Sciences*, pages 5–36. C. K. Koc, ed. Springer, 2014.

[Kal95] Erich Kaltofen. Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems. *Mathematics of Computation*, pages 777–806, 1995.

[LHWL93] Arjen K. Lenstra and Jr. Hendrik W. Lenstra, editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1993.

[Mas69] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969.

[Pie15] Cécile Pierrot. The multiple number field sieve with conjugation and generalized Joux-Lercier methods. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 156–170, 2015.

[Sch93] Oliver Schirokauer. Discrete logarithm and local units. *Philosophical Transactions of the Royal Society of London*, pages 409–423, 1993.

[Tho02] Emmanuel Thomé. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *J. Symb. Comput.*, 33(5):757–775, 2002.

[Wie86] Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, 1986.