HAL
open science

On finding multiplicities of characteristic polynomial factors of black-box matrices

Jean-Guillaume Dumas, Clément Pernet, B. David Saunders

To cite this version:

Jean-Guillaume Dumas, Clément Pernet, B. David Saunders. On finding multiplicities of characteristic polynomial factors of black-box matrices. ISSAC 2009 - International Symposium on Symbolic and Algebraic Computation, Jul 2009, Séoul, South Korea. pp.135-142, 10.1145/1576702.1576723 . hal-00357262v2

HAL Id: hal-00357262

https://hal.science/hal-00357262v2

Submitted on 18 May 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On finding multiplicities of characteristic polynomial factors of black-box matrices.*

Jean-Guillaume Dumas† Clément Pernet‡

B. David Saunders§

May 18, 2009

Abstract

We present algorithms and heuristics to compute the characteristic polynomial of a matrix given its minimal polynomial. The matrix is represented as a black-box, i.e., by a function to compute its matrix-vector product. The methods apply to matrices either over the integers or over a large enough finite field. Experiments show that these methods perform efficiently in practice. Combined in an adaptive strategy, these algorithms reach significant speedups in practice for some integer matrices arising in an application from graph theory.

Keywords: Characteristic polynomial ; black-box matrix ; finite field.

1 Introduction

Computing the characteristic polynomial of an integer matrix is a classical mathematical problem. It is closely related to the computation of the Frobenius normal form which can be used to test two matrices for similarity, or computing invariant subspaces under the action of the matrix. Although the Frobenius normal form contains more information on the matrix than the characteristic polynomial, most algorithms to compute it are based on computations of characteristic polynomials (see for example [25, §9.7]).

Several matrix representations are used in computational linear algebra. In the dense representation, a $m \times n$ matrix is considered as the array of all the $m \times n$ coefficients. The sparse representation only considers non-zero coefficients using different possible data structures. In the black-box representation, the matrix

*Saunders supported by National Science Foundation Grants CCF-0515197, CCF-0830130.

†Laboratoire J. Kuntzmann, Université de Grenoble. 51, rue des Mathématiques, umr CNRS 5224, bp 53X, F38041 Grenoble, France, `Jean-Guillaume.Dumas@imag.fr`

‡Laboratoire LIG, Université de Grenoble. umr CNRS, F38330 Montbonnot, France. `Clement.Pernet@imag.fr`

§University of Delaware, Computer and Information Science Department. Newark / DE / 19716, USA. `saunders@cis.udel.edu`

is viewed as a linear operator, and no other operation than the application to a vector is allowed. Though constraining, this limitation preserves the structure or sparsity of the matrix and is therefore especially well suited for very large sparse or structured matrices.

Computation of the characteristic polynomial of *dense* matrices has already been well studied both in theory and practice: over a finite field, [23, 24] set the best complexity (using respectively a deterministic and a probabilistic algorithm), and [8, 24] propose efficient implementations. Over the integers, the best complexity is achieved in [22], but currently the most efficient implementations rely on the Chinese remainder algorithm [8].

In the latter article, a competitive approach is introduced that limits the use of the Chinese remainder algorithm to the computation of the minimal polynomial. The characteristic polynomial is then recovered by determining the multiplicities of its irreducible factors. This task is done using the deterministic algorithm for the characteristic polynomial over a randomly chosen prime field.

In the black-box model, the minimal polynomial is used as a building block for many algorithms over a finite field. Adapted from the iterative numerical methods (Lanczos, Krylov), the Wiedemann minimal polynomial algorithm [29, 21] has excellent asymptotic complexity and is used in efficient black-box linear algebra software such as `LinBox`[1].

However less is known concerning the characteristic polynomial of black-box matrices. It is an open problem [20, Open Problem 3] to compute the characteristic polynomial as efficiently as the minimal polynomial, using the Wiedemann method. The latter uses $\mathcal{O}(n)$ products of a square $n \times n$ matrix by a vector and $\mathcal{O}(n^2(\log n)^{\mathcal{O}(1)})$ additional arithmetic operations with $\mathcal{O}(n)$ extra memory storage. Eberly gives an algorithm using $\mathcal{O}(n)$ matrix vector products, and $\mathcal{O}(\phi n^2)$ additional operations, where ϕ is the number of invariant factors of the matrix [12]. In the worst case, $\phi = \mathcal{O}(n)$ and the algorithm does not improve on the complexity of dense algorithms. Villard proposed in [28] a black box algorithm to compute the Frobenius normal form and therefore the characteristic polynomial in $\mathcal{O}(\sqrt{n}\log(n))$ computations of minimal polynomials and $\mathcal{O}(n^{2.5}(\log n)^2 \log\log n)$ additional field operations.

Instead, we propose here several algorithms and heuristics focusing on efficiency in practice. The general strategy is to compute the minimal polynomial using Wiedemann's algorithm and decompose it into irreducible factors. There only remains to determine to which multiplicity each of these factors appear in the characteristic polynomial. In section 2 we propose several methods to determine these multiplicities. Adaptive combination of them is discussed in section 3. Under a conjectured hypothesis the latter is shown to require $\mathcal{O}\ (n\sqrt{n})$ matrix vector products which improves by a logarithmic factor on the complexity of Villard's algorithm.

Lastly, an algorithm for the computation over the ring of integers is derived in section 4. It is based on the multifactor Hensel lifting of a gcd-free basis, following Storjohann [26]. The benefit of this approach is verified by experiments

[1] `www.linalg.org`

presented in section 5. Several sparse matrices are considered, including a set of adjacency matrices of strongly regular graphs, coming from an application in graph theory.

2 Three methods for computing multiplicities

In this section we consider a matrix A over a finite field $K = \mathrm{GF}(q)$. Let $P^A_{\min} = \prod_{i=1}^k P_i^{e_i}$ be the decomposition of the minimal polynomial of A in irreducible monic factors. The characteristic polynomial is then

$$P^A_{\mathrm{char}} = \prod_{i=1}^{k} P_i^{m_i} \tag{1}$$

for some $m_i \geq e_i$. We also denote by d_i the degrees of the factors: $d_i = \deg(P_i)$.

To recover the multiplicities m_i, we will present three techniques, based on black-box computations with the matrix A: the nullity method (§2.1) uses the rank of $P_i(A)$ to reveal information on the multiplicity m_i, the combinatorial search (§2.2) is a branch and bound technique to solve the total degree equation whose integral unknowns are the multiplicities and the index calculus technique (§2.3) uses a linear system solving based on the discrete logarithm of equation 1 evaluated in random values.

2.1 The nullity method

Definiton 2.1. *The nullity $\nu(A)$ of a matrix A is the dimension of its nullspace.*

We also recall the following definitions:

The *companion matrix* of the monic polynomial $P = X^d + \sum_{i=0}^{d-1} a_i X^i$ is the matrix $C_P = \begin{bmatrix} 0 & & & -a_0 \\ 1 & 0 & & -a_1 \\ & \ddots & \ddots & \vdots \\ & & 1 & -a_{d-1} \end{bmatrix}$. Its minimal polynomial and its characteristic polynomial are equal to P.

The *block Jordan matrix* of an irreducible polynomial P of degree d to a power k is the $kd \times kd$ matrix J_{P^k} of the form $J_{P^k} = \begin{bmatrix} C_P & B & & \\ & \ddots & \ddots & \\ & & C_P & B \\ & & & C_P \end{bmatrix}$ where the $d \times d$ matrix B is filled with zeros except for $B_{d,1} = 1$. Its minimal polynomial and its characteristic polynomial are equal to P^k. This definition extends the usual notion of Jordan blocks for $d = 1$.

The *Frobenius normal form* of a Matrix A is the unique block diagonal matrix $F = \mathrm{Diag}(C_{f_0}, C_{f_1}, \dots)$ such that $A = U^{-1}FU$ for a nonsingular matrix U. The polynomials f_i are the invariant factors of A and satisfy $f_0 = P^A_{\min}$ and f_{i+1} divides f_i for all $i \geq 0$.

The *primary form* of a Matrix A (also called the second Frobenius form in [15]) is a further decomposition of the Frobenius normal form where each

companion block C_{f_i} is replaced by a block diagonal matrix $\mathrm{Diag}(J_{g_1^{k_1}}, J_{g_2^{k_2}}, \dots)$. The g_j are the irreducible factors of f_i, with the respective multiplicities k_j. The primary form is unique up to the order of the blocks.

Example 2.2. *Consider the matrix in Frobenius normal form*

$$A = Diag(C_{X^5-6X^4+14X^3-16X^2+9X-2}, C_{X^2-2X+1})$$

over $GF(5)$. The corresponding primary form is the matrix

$$B = Diag(J_{(X^2-2X-1)^2}, J_{X-2}, J_{X^2-2X-1}).$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & & & \\ 1 & 0 & 0 & 0 & -9 & & & \\ & 1 & 0 & 0 & 16 & & & \\ & & 1 & 0 & -14 & & & \\ & & & 1 & 6 & & & \\ & & & & & 0 & -1 \\ & & & & & 1 & 2 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 & -1 & & & \\ 1 & 0 & 0 & 4 & & & \\ & 1 & 0 & -6 & & & \\ & & 1 & 4 & & & \\ & & & & 2 & & \\ & & & & & 0 & -1 \\ & & & & & 1 & 2 \end{bmatrix}$$

The method of the nullity is based on the following lemma:

Lemma 2.3. *Let A be a square matrix and let P be an irreducible polynomial of degree d, of multiplicity e in the minimal polynomial of A, and of multiplicity m in the characteristic polynomial of A. Then $\nu(P^e(A)) = md$.*

Proof. Let F be the primary form of A over K: $F = U^{-1}AU$ for a non singular matrix U. F is block diagonal of the form $\mathrm{Diag}(J_{P_j^{e_j}})$. Then $P^e(A) = U^{-1}P^e(F)U = U^{-1}\mathrm{Diag}(P^e(J_{P_j^k}))U$. On one hand P^e annihilates the blocks $J_{P_j^k}$ where $P = P_j$ and $k \leq e$. On the other hand, the rank of $P^e(J_{P_j^k})$ is full for $P \neq P_j$, since P and P_j are relatively prime. Thus the nullity of $P^e(A)$ exactly corresponds to the total dimension of the blocks $J_{P_j^k}$ where $P = P_j$, which is md. □

From this lemma the following algorithm, computing the multiplicity m_i of an irreducible factor P_i is straight-forward:

Algorithm 1: `Nullity`

Data: A: an $n \times n$ matrix over a Field K,
Data: P: an irreducible factor of $P^A_{\min}$,
Data: e: the multiplicity of P in $P^A_{\min}$,
Result: m: the multiplicity of P in P^A_{char}.
begin
  $r = \mathrm{rank}(P^e(A))$
  return $m = (n - r)/\mathrm{degree}(P)$
end

Proposition 2.4. *Algorithm 1 computes the multiplicity of P in the characteristic polynomial of an $n \times n$ matrix A using $\mathcal{O}(edn\Omega)$ field operations, where Ω is the cost of a matrix-vector product with A, d is the degree of P and e is its multiplicity in the minimal polynomial.*

Proof. Using Horner's rule, the matrix $P(A)$ can be written as $a_0 I_n + A(a_1 I_n + A(a_2 I_n + \dots))$. Hence, applying a vector to this blackbox only requires d applications of a vector to the blackbox A, ie. $\mathcal{O}(d\Omega)$ field operations. Thus applying a vector to $P^e(A)$ costs $\mathcal{O}(ed\Omega)$. Lastly, the rank of this matrix can be computed in $\mathcal{O}(edn\Omega)$ field operations, using Wiedemann's algorithm combined with preconditioners [11]. □

This algorithm is therefore suitable for irreducible factors P where the product ed is small.

Now if e is large, the computation of $\mathrm{rank}(P^e(A))$ may be too expensive. Still, some partial knowledge on the multiplicity can be recovered from the rank of the first powers of $P(A)$. This can help to shorten the computation of other algorithms, as will be shown in section 3. We now describe how these partial multiplicities can be recovered.

The multiplicity m of P is formed by the contribution of several blocks of the type J_{P^j} for $j \in [1 \dots e]$ in the primary form of A. Whereas the blocks with small j can be numerous, there must be few blocks with large j, due to the limitation of the total dimension (since e is large).

We denote by $n_{i,j}$ the number of occurrences of $J_{P_i^j}$ in the primary form of A. From the determination of the $n_{i,j}$, we can directly deduce the multiplicity m_i by the relation

$$m_i = \sum_{j=1}^{e} j n_{i,j}. \tag{2}$$

We now show how to compute the $n_{i,j}$ for small j, using algorithm 1.

Lemma 2.5. *Let P be an irreducible polynomial of degree d over a finite field K and k and $e \geq 1$ be two integers. Then $\nu(P^k(J_{P^e})) = min(k, e)d$*

Proof. Let $A = J_{P^e}$ and $B = P^k(A)$. If $k \geq e$, then P^k is a multiple of the minimal polynomial of A. Thus B is the zero matrix, and its nullity equals its dimension: ed.

Now suppose $k < e$. Let $\overline{K}$ be an extension of K such that P splits into d degree one factors P_i over $\overline{K}$. Since any finite field is a perfect field, these factors are distinct.

The minimal polynomial of A over $\overline{K}$ is still P^e. Consequently, the Frobenius normal form of A over $\overline{K}$ is C_{P^e} and its primary form is $F = \mathrm{Diag}(J_{P_i^e})$. More precisely, there exists $U \in M_n(\overline{K})$ such that $A = U^{-1}FU$. We have therefore $B = U^{-1}P^k(F)U = U^{-1}\mathrm{Diag}(P^k(J_{P_i^e}))U$.

First consider the case $k = 1$: the minimal polynomial of each $P_i(J_{P_i^e})$ is X^e and so is the minimal polynomial of each $P(J_{P_i^e})$ (since the P_i are relatively prime). Hence the primary form of $P(J_{P_i^e})$ is J_{X^e}. Therefore there exist $V \in M_n(\overline{K})$ such that

$$B = U^{-1}V^{-1}Diag(\underbrace{J_{X^e}, \dots, J_{X^e}}_{d \text{ times}})VU.$$

Lastly the rank of J_{X^e} being $e-1$, we deduce that $\text{rank}(B) = d(e-1)$. The nullity of B is therefore $\nu(B) = d$.

For the general case, we have

$$B = U^{-1}V^{-1}\text{Diag}(\underbrace{(J_{X^e})^k, \ldots, (J_{X^e})^k}_{d \text{ times}})VU.$$

Now J_{X^e} is $e \times e$ and nilpotent with ones on the super-diagonal so that its k-th power has rank $\max(0, e-k)$. Thus, $\text{rank}(B) = \max(0, e-k)d$ and $\nu(B) = \min(e, k)d$. □

We now apply this result to the irreducible factors of the minimal polynomial and denote the nullity of $P_i^j(A)$ by $\nu_{i,j} = \nu(P_i^j(A))$.

First, the nullity of $P_i(A)$, can be decomposed into the sum of the nullities of each $P_i(J_{P_i^k})$ for every $k \leq e_i$:

$$\nu_{i,1} = \sum_{k=1}^{e_i} n_{i,k} d_i \tag{3}$$

Now applying P_i^j to A, every $P_i^j(J_{P_i^k})$ for $k \leq j$ will be a zero matrix and therefore contribute with kd_i to the nullity. Otherwise, if $k > j$, the contribution to the nullity remains jd_i. Therefore we have:

$$\nu_{i,j} = \sum_{k=1}^{j} n_{i,k} k d_i + \sum_{k=j+1}^{e_i} n_{i,k} j d_i \tag{4}$$

From these two equations, we deduce the $n_{i,j}$: first we have

$$\frac{1}{j-1}\nu_{i,j-1} = \frac{1}{j-1}\sum_{k=1}^{j-1} n_{i,k} k d_i + n_{i,j} d_i + \sum_{k=j+1}^{e_i} n_{i,k} d_i.$$

Now, since: $\sum_{k=j+1}^{e_i} n_{i,k} d_i = \nu_{i,j+1} - \nu_{i,j}$, the number of occurrences directly is:

$$n_{i,j} = \frac{1}{d_i}\left(\frac{1}{j-1}\nu_{i,j-1} + \nu_{i,j} - \nu_{i,j+1}\right) - \frac{1}{j-1}\sum_{k=1}^{j-1} n_{i,k} k.$$

Therefore we obtain corollary 2.6 giving the expression of the $n_{i,j}$:

Corollary 2.6.

$$\begin{aligned}
n_{i,1} &= (2\nu_{i,1} - \nu_{i,2})/d_i \\
n_{i,j} &= \frac{1}{d_i}\left(\frac{1}{j-1}\nu_{i,j-1} + \nu_{i,j} - \nu_{i,j+1}\right) \\
&\quad - \frac{1}{j-1}\sum_{k=1}^{j-1} n_{i,k} k \quad \forall j \in [1 \ldots e_i] \\
n_{i,e_i} &= \frac{\nu_{i,e_i}}{e_i d_i} - \frac{1}{e_i}\sum_{k=1}^{e_i-1} n_{i,k} k
\end{aligned}$$

The last formula for n_{i,e_i} is given for the sake of completeness: in practice, one will never compute every $n_{i,j}$, since one would rather directly compute the nullity of $P_i^{e_i}(A)$ instead, to deduce the multiplicity m_i from algorithm 1.

2.2 The combinatorial search

In the following, we want to determine the values of the unknown $n_{i,j}$. They must satisfy the total degree equation:

$$n = \sum_i d_i \sum_{j=1}^{k_i} j n_{i,j}. \tag{5}$$

We can also discriminate potential candidates using the trace: the degree $n-1$ coefficient of the characteristic polynomial is the negative of the trace of the matrix. Denote by t_i the degree $n-1$ coefficient of an irreducible factor $P_i = X^{d_i} + t_i X^{d_i - 1} + \ldots$. Then the degree $n-1$ coefficient of $\prod_i P_i^{m_i}$ is $\sum_i t_i m_i$. We thus have the trace test:

$$\mathrm{Tr}(A) = -\sum_i t_i m_i = -\sum_i t_i \sum_{j=1}^{k_i} j n_{i,j}. \tag{6}$$

In a pure black-box model, the trace can be computed using n matrix-vector products. For many sparse or structured matrix representations, a faster method is available as well.

Then it suffices to use e.g. a `Branch-and-Cut` algorithm to compute all the integer k-tuples satisfying both equations (5) and (6). Of course, if some of the unknowns $n_{i,j}$ are already computed (e.g. by the nullity method) the set of candidates is accordingly reduced.

The remaining candidates will then be discriminated by evaluations of the characteristic polynomial at random values, i.e. computations of determinants of $\lambda I - A$ matrices. Indeed, we have efficient methods of computing the determinant of a black-box matrix (see e.g. [27, §3.1 Determinant Preserving Preconditioners] and references therein). Algorithm 2 sums up this combinatorial search strategy.

2.3 Index calculus method

Evaluating equation (1) at a point λ leads to an equation over the finite field, where the multiplicities m_i are the unknowns. Inspired by index calculus techniques [4], the idea here is to consider the discrete logarithm of such an equation (with an arbitrary choice of generator), to produce a linear equation in the m_i. Taking several of these equations for different λ_i forms a linear system of equations, with dimension k, the number of unknown multiplicities.

In this discussion the base field is $GF(q)$ and q is sufficiently large with respect to n as discussed below. The characteristic polynomial evaluated at a

Algorithm 2: `Combinatorial-search`

Data: A, an $n \times n$ matrix
Data: $D = (d_i)_i$, the degrees of the irreducible factors P_i of P^A_{char}
Data: M, a set of precomputed $n_{i,j}$
Result: $N = (n_{i,j})$
begin
  `/* using degree and trace constraints */`
  sol = `Branch-and-Cut`(A, D, M)
  while $\#sol > 1$ **do**
    Pick $\lambda \in K$ at random
    $\delta = \det(\lambda I - A)$
    Discard any $N \in$ sol s.t. $\prod (P_i^j)^{n_{i,j}}(\lambda) \neq \delta$
  return $N = sol[1]$
end

given value λ presents this equation in the unknown exponents m_j:

$$\prod_{j=1}^{k} P_j^{m_j}(\lambda) = \det(\lambda I - A). \tag{7}$$

Now, if λ is not a root of the characteristic polynomial, taking the discrete logarithm of these terms for a generator g of the field leads to this equation modulo $q-1$:

$$\sum_{j=1}^{k} m_j \log_g(P_j(\lambda)) \equiv \log_g(\det(\lambda I - A)) \mod q-1, \tag{8}$$

which is linear in the unknowns m_j. We can therefore build a $l \times k$ linear system by randomly choosing l values λ_i. This system is consistent since the multiplicities m_i are a solution vector of this system. If the solution is unique, then it is the vector of multiplicities over $\mathbb{Z}$.

The computation of this vector can either be done by a dense Gaussian elimination over the ring $\mathbb{Z}_{q-1}$ or over a finite field $\mathbb{Z}_p$ where p is a large prime factor of $q-1$ (larger than n). In this last case, the result will be correct as long as the system remains nonsingular modulo p.

Algorithm 3 describes this techniques in more details.

Let k be the number of unknown multiplicities. The first step is to find k values λ_i forming a non singular system. Therefore, we propose, in algorithm 3, to evaluate the system at more than k points. The complexity of forming a row of G costs only $3\sum_{i=1}^{k} d_i$ arithmetic operations using Horner's method. Therefore trying as many as n different values for λ is a negligible cost. Furthermore, one could use fast multi-point evaluation to get blocks of rows simultaneously (up to n rows at a cost essentially linear in n).

Algorithm 3: `Index-calculus`

Data: A, an $n \times n$ matrix over a finite field $K = \mathrm{GF}(q)$
Data: P_i, the irreducible factors of $P^A_{\min}$,
Data: S, the set of indices of the unknown multiplicities,
Data: $Q = \prod_{j \notin S} P_j^{m_j}$, the partial product of the irreducible factors with known multiplicity m_j.
Result: P^A_{char} or "fail"

```
begin
    k = #S; l = 0; H = [ ];
    Choose a generator g of K
    Let p be a prime factor of q
    while rank(H) < k do
        l = l + 1; if l > n then return "fail"
        Choose randomly λ_l ∈ K
        repeat
            α_{l,j} = P_j(λ_l) for all j
            γ_l = Q(λ_l)
        until γ ≠ 0 and α_j ≠ 0, ∀j ∈ S
        /* H is extended to the size l × k                    */
        Stack the row [log_g α_{l,j} mod (q − 1)] mod p to H
    K = { indices of the first k independent rows of H}
    Set B = a k × k nonsingular matrix of these rows
    Compute b = [log_g(det(λ_i I − A)) − log_g(γ_i)]_{i∈K}
    Solve Bx = b
    return P = ∏_{j∈S} P_j^{x_j} Q
end
```

The rank of H is computed all along the process, each new row being incrementally added to the triangular decomposition of the current matrix. This Gaussian elimination (performed by the LQUP algorithm [19] for example) also provides the indices of the first k linearly independent rows, and therefore the indices of the convenient λ_i. Lastly the vector b is formed, using only k determinant computations.

In practice, it appeared, as in index calculus [2, 18], that the number of rows required to get a full rank matrix B is always quite close to k. However, we do not have a proof of this property, and we therefore state it as the conjecture 2.7.

Conjecture 2.7. *Let A be a $n \times n$ matrix over $\mathbb{Z}$ and $P_1 \dots P_k$ be the irreducible factors of its minimal polynomial. Let $p > n$ be a prime chosen randomly in finite set. Let $q = 1 + \lambda p$ of the form r^k where r is a prime number. Let g be a generator of $GF(q)$ and $(\lambda_1, \dots, \lambda_n)$ uniformly chosen at random in $GF(q)$. Let $H = [h_{i,j}]$ where $h_{i,j} = (\log_g(P_j(\lambda_i)) \mod (q-1)) \mod p$. Then rank$(H) = k$ with high probability.*

Informally, in our system the evaluations at the λ_i are independent and can be considered as seeds for the pseudo-random generator of taking the discrete logarithm of the polynomial evaluation. Therefore the entries of the system modulo $q-1$ are at least close to random entries as soon as the polynomials are distinct. Would they be true random values, the singularity/nonsingularity of the matrices would follow the analysis of e.g. [3, Corollary 2.4]: if L is a square matrix of uniformly random entries modulo $q-1$ and p is a prime diving $q-1$, then the probability that $L \mod p$ is singular is of order $\frac{1}{p}$.

Theorem 2.8. *Assuming conj. 2.7, algorithm 3 is correct and its asymptotic complexity is $\mathcal{O}(kn\Omega)$ where $\Omega \geq n$ is the cost of a multiplication of A by a vector.*

Proof. Let l be the number of rows required to get an invertible system, $l \geq k$. Each determinant computation requires $\mathcal{O}(n)$ application of A to a vector [27, §3.1]. Building each row of the matrix requires a Horner like evaluation of a polynomial with total degree less than n, it therefore costs $\mathcal{O}(n)$ operations. Triangularization of G requires $\mathcal{O}(lk^{\omega-1})$ operations. Solving the system $Bx = b$, knowing the triangular decomposition of B requires $\mathcal{O}(k^2)$ operations. The discrete logarithms can be tabulated [6] with $\mathcal{O}(q)$ memory (or to avoid this extra memory, one can compute the whole sequence of powers of a generator of K, sort the matrix and vector entries and find the correspondences with some $\mathcal{O}\tilde{}(lk+q)$ extra field operations). The overall complexity is thus $\mathcal{O}(kn\Omega + ln + lk^{\omega-1})$ which is $O(kn\Omega)$ when $l = \mathcal{O}(n\Omega k^{2-\omega})$. □

The algorithm stops arbitrarily when $l = n+1$. We see here that a larger l is acceptable for the complexity result, but in our experiments a very small l (e.g. $l \approx k$) always suffices. In the following sections, this algorithm will be used with $k < \sqrt{n}$, thus giving an expected $\mathcal{O}(n^{1.5}\Omega)$ complexity.

3 Adaptive black-box algorithm over a finite field

We show in the present section how to combine the ideas of the previous section together with already existing techniques to form an adaptive algorithm computing the characteristic polynomial of a black-box matrix over a finite field. The algorithm is adaptive in the sense of [5], meaning that it chooses the best variant depending on discovered properties of its input.

We first combine the nullity method with the combinatorial search. We then show an algorithm improving on the asymptotic complexity. Finally we give some improvements which are efficient in practice on typical matrices.

3.1 Nullity method and combinatorial search

These two algorithms are complementary: the nullity is efficient for the determination of the multiplicites of factors of small degree, whereas the combinatorial search is adapted to the large degree factors.

More precisely, algorithm 4 sorts the list of the unknown occurences $n_{i,j}$ according to the increasing jd_i. The nullities are then computed until there remain fewer than a fixed number T of unknowns to be determined by combinatorial search.

Algorithm 4: `Nullity-comb-search`

Data: A: an $n \times n$ matrix over a finite field,
Data: P_i: the irreducible factors of $P^A_{\min}$,
Data: T: a static threshold
Result: m_i: the multiplicities of each P_i in P^A_{char}.
begin
  $\mathcal{E} = \{(i,j)/i = 1 \dots k, j = 1 \dots e_i\}$
  Sort $\mathcal{E}$ according to the increasing values of jd_i
  while $(\#\mathcal{E} > T)$ **do**
    Pop (i,j) from $\mathcal{E}$
    Compute $\nu_{i,j} = n - \text{rank}(P_i^j(A))$
  for $i = 1 \dots k$ **do**
    Let j_i be the largest index s.t. ν_{i,j_i} is computed
    if $j_i < e_i$ **then**
      Compute $\nu_{i,j_i+1} = n - \text{rank}(P_i^{j_i+1}(A))$
    for $k = 1 \dots j_i$ **do** Compute $n_{i,k}$ using cor. 2.6
  `Combinatorial-search`$(A, (d_1, \dots, d_k), (j_1, \dots, j_k))$
  for $i = 1 \dots k$ **do** $m_i = \sum_{j=1}^{e_i} jn_{i,j}$
  return $m = (m_1, \dots, m_k)$
end

The combinatorial search has exponential complexity. The threshold T must be small. In experiments, we found that $T = 5$ was the best choice for various matrices. In the case of numerous factors with large degree, this approach is of

reduced effeciveness. We propose in the next section how to combine it with a third algorithm.

3.2 Nullity method and system resolution

The index calculus method also enables the design of a hybrid algorithm. If the multiplicities of some factors have already been computed by another method, we can limit the system to the unknown multiplicities only, thus reducing its dimension.

Suppose that the multiplicities m_i of the factors P_i for $i \in \mathcal{C}$ are already known, then equation (8) reduces to

$$\begin{aligned}\sum_{j \notin \mathcal{C}} m_j \log(P_j(\lambda_i)) \equiv & \log(\det(\lambda_i I - A)) \\ & - \sum_{j \in \mathcal{C}} m_j \log(P_j(\lambda_i)) \mod q-1.\end{aligned} \tag{9}$$

A first simple hybrid approach is the following: the method of the nullity (section 2.1) is applied to every degree one factor with multiplicity one in the minimal polynomial, and the remaining factors are left to the index calculus method.

This approach is always worthy since the computation of the rank of $P_j(A)$, for a degree one polynomial P_i is cheaper than the computation of $det(\lambda_i I - A)$ [11].

A second hybrid approach also introduces a combinatorial search to this algorithm: the nullity method still handles the t degree one factors as previously. The remaining factors P_i are sorted by decreasing degree. For a convenient choice of s, a list of every possible assignment for the multiplicities of the first s factors is determined, using a combinatorial search. Then for each partial assignment, the multiplicities of the remaining factors are determined by the resolution of an index calculus system of the form:

$$\begin{aligned}\sum_{j=s+1}^{k-t} m_j \log(P_j(\lambda_i)) \equiv & \log_g(\det(\lambda_i I - A)) \\ & - \log_g \left(\prod_{j=1}^{s} P_j^{m_j}(\lambda_i) \right) \mod q-1 \ \forall i.\end{aligned} \tag{10}$$

For each partial assignment, the system resolutions share the same matrix B. Therefore the expensive part of it, namely the Gaussian elimination, can be performed only once at cost $\mathcal{O}(m^3)$, where $m = k-s-t$. There only remains to solve two triangular systems (in $\mathcal{O}(m^2)$) for each possible assignment. Lastly, the assignments will be discriminated against each other by a test on the total degree. To sum up, this techniques makes it possible to balance the cost of the computations of determinants, and the cost of the system solving, by reducing the dimension of the system, but increasing the dimension of its right hand side. The most appropriate value for s has to be determined dynamically, according

to the number of possible assignments induced, and using an estimate of the cost function of this algorithm: e.g.

$$2mn\Omega + \frac{2}{3}m^3 + 4m^2\tau_s$$

where τ_s denote the number of possible assignments for a chosen subset of s factors.

3.3 Index calculus and kth invariant

The best known black-box algorithm to compute the Frobenius normal form over a field is given by Villard in [28]. It is proved that computing the kth invariant factor of a matrix reduces to the computation of a minimal polynomial of the input matrix with a rank k additive perturbation. Using a binary search technique, the algorithm only performs $\mu\log(n)$ such computations, where μ is the number of distinct invariant factors of the matrix. Since μ is smaller than $3\sqrt{n}/2$ and an invariant factor can be recovered using $\mathcal{O}(n)$ matrix vector products, this corresponds to a total number of $\mathcal{O}(n^{3/2}\log(n))$ matrix-vector products and an additional cost of $\mathcal{O}(n^{5/2}\log^2(n)\log\log(n))$ arithmetic operations.

We propose in algorithm 5 an alternative approach combining the index calculus method with computations of individual invariant factors.

Algorithm 5: `black-box-charpoly`

Data: A: an $n \times n$ matrix over a finite field K
Result: P^A_{char} or "fail"
begin
  $f_1 =$ `InvFact`(1)
  Factor $f_1 = \prod_{i=1}^{k} P_i^{e_i}$ using Cantor-Zassenhaus
  Set $S = \{P_1, \ldots, P_k\}$ and $j = 2$
  while $(\#S > \sqrt{n})$ **do**
    $f_j =$ `InvFact`(j)
    forall $P_i \in S$ **do**
      Compute α s.t. $\gcd(P_i^{e_i}, f_j) = P_i^{\alpha}$
      if $\alpha = 0$ **then** $S = S \backslash \{P_i\}$
      $m_i \mathrel{+}= \alpha$
  return *`Index-calculus`*$(A, (P_i), S, \prod_{j \notin S} P_j^{m_j})$
end

The idea is to reduce the dimension of the index calculus system to $\sqrt{n}$ by computing a few of the first invariant factors of the matrix.

After each computation of an invariant factor Φ, the multiplicity of each irreducible polynomial P_i is updated, and those P_i that are no longer in Φ are removed from the list of the factors with unknown multiplicity.

The `while` loop is executed at most $\sqrt{n}$ times. Otherwise, there would be more than $\sqrt{n}$ invariant factors having more than $\sqrt{n}$ irreducible factors, and the total degree would be larger than n.

Now the condition of exit for this loop ensures that the order of the linear system will be smaller than $\sqrt{n}$. Therefore only $\sqrt{n}$ determinants will be computed and the overall number of blackbox matrix-vector products is $\mathcal{O}(n\sqrt{n})$

The remaining multiplicities are then determined by the index calculus method described previously, requiring at most $\mathcal{O}(n^{3/2}\Omega)$ applications of the matrix to a vector.

Under the conditions of validity for the index calculus algorithm, this heuristic improves on the computation time of Villard's algorithm by a logarithmic factor.

4 Lifting over the integers

Storjohann gives in [26] a method for the computation of the Frobenius normal form of a black-box integer matrix. It is based on a computation of the minimal polynomial over $\mathbb{Z}$ and on a computation of the Frobenius normal form over a prime field. Then a gcd-free basis for the invariant factors over $\mathbb{Z}_p[X]$ is computed and lifted over $\mathbb{Z}[X]$.

We use the same idea but just for the characteristic polynomial, and not for all the invariant factors. It is thus simpler since we don't need to ensure that the Frobenius form of A modulo p equals the integer Frobenius form reduced modulo p. We just need ensure that the minimal and characteristic polynomials of A modulo p equal the minimal and characteristic polynomials over the integers reduced modulo p.

The goal of the following algorithm 6 is to compute the integer characteristic polynomial from the integer minimal polynomial and the characteristic polynomial modulo some prime p, obtained via the previous sections. Algorithm 6 is just a simplification of that of [26]:

The integer minimal polynomial is computed via [9, Theorem 3.3] with an $\mathcal{O}(sd\Omega)$ probabilistic complexity, where s is the size of its integer coefficients and d, its degree. The characteristic polynomial modulo p is computed via algorithm 5 with an $\mathcal{O}(n^{1.5}\Omega)$ complexity and $\mathcal{O}\tilde{}(n^{2.5})$ extra field operations. Then the squarefree part [17] and the Hensel lifting of the gcd-free basis takes $\mathcal{O}\tilde{}(nk)$ word operations with fast integer and polynomial arithmetic [16, Theorem 15.18].

The size of the coefficients of the integer minimal polynomial is bounded in the worst case by $s \leq \frac{n}{2}(\log_2(n) + \log_2(||A||^2) + 0.212)$ where $||A||$ is the largest entry in absolute value of the matrix A, see [7, Lemma 2.1]. When the matrix entries are of constant size, $s = \mathcal{O}\tilde{}(n)$ and as the degree of the minimal polynomial is bounded by n, the dominant asymptotic cost is that of the integer minimal polynomial computation. This result is already in [26]. In practice, however, the coefficients of the minimal polynomial are often much smaller than the bound and than those of the integer characteristic polynomial. Furthermore, the degree of the minimal polynomial can be extremely small,

Algorithm 6: `Gcd-free lifting of the characteristic polynomial`

Data: A: an $n \times n$ integer matrix, $P^A_{\min}$ its minimal polynomial over $\mathbb{Z}$,
Data: p: a prime number,
Data: $\overline{P^A_{\text{char}}}$ the characteristic polynomial of $A \mod p$,
Result: P^A_{char} the characteristic polynomial of A over $\mathbb{Z}$
begin
Compute S the squarefree part of $P^A_{\min}$
$\overline{S} = S \mod p$, $\overline{P^A_{\min}} = P^A_{\min} \mod p$
Compute a modular gcd-free basis $(\bar{g}_1, \ldots, \bar{g}_\chi)$ of $(\overline{S}, \overline{P^A_{\min}}, \overline{P^A_{\text{char}}})$, together with exponents $(\mu_1, \ldots, \mu_\chi)$ such that $\overline{P^A_{\text{char}}} = \prod \bar{g}_i^{\mu_i}$
Apply Hensel lifting on the basis $(\bar{g}_1, \ldots, \bar{g}_\chi)$ to produce $(g_1, \ldots, g_\chi)$ so that $S \equiv g_1 \ldots g_\chi \mod p^k$ and $(g_1, \ldots, g_\chi) \equiv (\bar{g}_1, \ldots, \bar{g}_\chi) \mod p$
return $P^A_{char} = \prod g_i^{\mu_i}$
end

especially for structured or sparse matrices (see e.g. homology matrices in [9]). In those cases, the dominant cost will be the computation of the characteristic polynomial modulo p. Then, our algorithm enables faster computations since a factor of $\sqrt{n}$ has been gained, as illustrated in table 1.

A supplemental constraint can be introduced by computing $\det(\lambda I - A)$, at random integer λ. Using e.g. [10, Theorem 4.2] with [13], $\mathcal{O}(\sqrt{n})$ of these can be done to speed-up the modular adaptive search, without increased complexity.

5 Experimental comparisons and applications

We have implemented some of the algorithms presented in the previous sections using the `LinBox`[2] library for the black-box computation of minimal polynomials, ranks and determinants. In a first approach, we replaced the computation of the gcd-free basis of algorithm 6 by a factorization into irreducible factors, using Hensel lifting. This algorithm is more expensive in the worst case, but the efficient implementation by `NTL`[3] makes it practicable in numerous cases.

This work was partly motivated by an application from graph theory. For a graph X on n vertices with vertex set $V(X)$ and edge set $E(X)$ the k-th symmetric power X^k is the graph with the $\binom{n}{k}$ k-subsets of $V(X)$ as vertices and with two such k-subsets adjacent if their symmetric difference is in $E(X)$.

Graph theorists are interested in the spectrum of such graphs (defined as the spectrum of their adjacency matrix) since they are closely related to the description of their isomorphism class. More precisely, if a certain power k is found, such that the symmetric kth power of a graph describes its isomorphism

[2] `www.linalg.org`
[3] `www.shoup.net/ntl`

class, this would provide a polynomial algorithm to solve the graph isomorphism problem.

This motivated the team of Audenaert, Godsil, Royle and Rudolph to study in [1] the spectrum of symmetric powers of a class of graphs: the strongly regular graphs. They prove that there exist infinitely many graphs having co-spectral symmetric squares. But concerning the symmetric cubes, no pair of graph is known to have co-spectral symmetric cubes until now.

We helped Royle to investigate further the computation of the characteristic polynomial of the symmetric cubes of strongly regular graphs. He was able to test the first 72 cases corresponding to the graphs with fewer than 29 vertices. Using our implementations available in `LinBox`, we have been able to test the 36 582 graphs with fewer than 36 vertices and check that there is no pair of graphs among them having cospectral symmetric cubes.

We used the matrices of this application to benchmark the implementations of the previously presented algorithms. These matrices are sparse and symmetric, and therefore especially suited to black-box computations. Moreover, several parameters such as the degrees of their minimal polynomials or the average number ω of nonzero elements per row vary among the matrices. The matrices `EX1, EX3, EX5` correspond respectively to the symmetric cubes of the strongly regular graphs with parameters (16,6,2,2), (26,10,3,4) and (35,16,6,8). Their dimensions are respectively $560 = \binom{16}{3}$, $2600 = \binom{26}{3}$ and $6545 = \binom{35}{3}$. The matrices `EX2` and `EX4` correspond to different graphs but with similar parameters as `EX1` and `EX3`.

All the matrices used in the experiments, including adjacency matrices of the symmetric powers, are avaible on-line in the *Sparse Integer Matrices Collection*[4]. In particular we used, in the following tables, matrices from the `SPG`, `Forest`, `Trefethen` and `Homology` sections of the collection. When the tested matrix was not square, we considered the square matrix obtained by padding it with zeroes.

We report in table 1 the computation time of the different modules described in this paper. For each matrix, two computations are compared: they share the computation of the minimal polynomial over $\mathbb{Z}$. Then the determination of the multiplicities is done either by the combination of the nullity algorithm and the combinatorial search (with the threshold T set to 5), or by the index calculus method.

We first note that the determination of the multiplicities may be the dominant operation when the degree of the minimal polynomial is small, as for the matrix `EX1`. This makes the motivation for this study obvious. For this task either method, nullity or resolution of the logarithmic system, can be the most competitive option, depending on the structure of the irreducible factors. This advocates for the adaptive approach of algorithm 5 combining both methods, and the computation of the kth invariant factor.

In order to emphasize the improvement of the black-box determination of the multiplicities over dense methods, we now compare it to the alternative tech-

[4] `ljk.imag.fr/membres/Jean-Guillaume.Dumas/SIMC`

Matrix	EX1	EX2	EX3	EX4	EX5
n: dimension	560	560	2600	2600	6545
d: deg ($P_{\min}$)	54	103	1036	1552	2874
ω: sparsity	15.6	15.6	27.6	27.6	45.2
$\mathbb{Z}$-`Minpoly`	0.11s	0.26s	117s	260s	5002s
$\mathbb{Z}[X]$ factorize	0.02s	0.07s	9.4	18.15	74.09s
Nullity/comb.	**3.37s**	5.33s	**33.2s**	**30.15s**	**289s**
Total	**3.51s**	5.66s	**159.4s**	**308.1s**	**5366s**
Index calc.	3.46s	**4.31s**	64.0s	57.0s	647s
Total	3.59s	**4.64s**	190.4s	336.4s	5641s

Table 1: Computation time for tasks of the integer adaptive algorithm on a Pentium4 (3.2 GHz; 1 Gb)

nique presented in [8, §4.2.2]. This also relies on the computation of the minimal polynomial in $\mathbb{Z}[X]$ and its decomposition into irreducible factors. But the multiplicities are then obtained using one *dense* computation of the characteristic polynomial in a randomly chosen finite field. It is therefore not anymore a black-box algorithm. We will denote it by `dchar`. `Comb` is the nullity-combinatorial search algorithm, and `ind` is the index calculus method. A denotes `08blocks`, B is `ch5-5.b3`, and T is `Tref500` from the Sparse Integer Matrices Collection.

Matrix	n	ω	`dchar`	`null-comb`	`ind.`
A	300	1.9	0.32s	0.08s	**0.07s**
AA^T	300	2.95	0.81s	**0.12s**	**0.12s**
B	600	4	4.4s	**1.52s**	1.97s
BB^T	600	13	**2.15s**	3.96	7.48s
`TF12`	552	7.6	6.8s	**5.53s**	5.75s
`mk9b3`	1260	3	31.25s	**10.51s**	177s
`Tref500`	500	16.9	65.14s	**25.14s**	25.17s

Table 2: Integer black-box approach for multiplicities on an Athlon (1.8 GHz; 2 Gb)

Table 2 shows the improvement of the black-box approach for several matrices coming from different applications. Once again, the structure of the irreducible factors of the minimal polynomials cause various behaviors for each variant. For example the times of `Index-calculus` are similar to those of `Nullity-comb-search`, sometimes better but also sometimes much slower, as for the matrices BB^T and `mk9b3`.

6 Conclusion

We developed several ways to recover the multiplicities of the factors of the characteristic polynomial from a factorization of the minimal polynomial. Over a finite field hybrid heuristics are proposed, that compete with the best theoretical complexity. Over the ring of integers, our approach enables fast computations particularly when the coefficients or degree of the minimal polynomial are small. This is illustrated on a family of strongly regular graphs, in order to verify that there are no symmetric co-spectral cubes.

Further studies on the theoretical complexity remain to be done, and could lead to better implementations in practice. In particular, a recent algorithm for dense matrices [24] might be adapted for black-box matrices. In this regard, extending the block projections of [14] to the case of similarity transformations would play a crucial role.

References

[1] K. Audenart, C. Godsil, G. Royle, and T. Rudolph. Symmetric squares of graphs. *Journal of Combinatorial Theory*, 97(1):74–90, Jan. 2007.

[2] R. L. Bender, and C. Pomerance. Rigorous Discrete Logarithm Computations in Finite Fields via Smooth Polynomials. *Studies in Advanced Mathematics*, American Mathematical Society and International Press, 7, 1998.

[3] J. Blömer, R. Karp, and E. Welzl. The rank of sparse random matrices over finite fields. *RSA: Random Structures & Algorithms*, 10, 1997.

[4] D. Coppersmith, A. M. Odlyzko, and R. Schroeppel. Discrete logarithms in GF(p). *Algorithmica*, 1(1):1–15, 1986.

[5] V.-D. Cung, V. Danjean, J.-G. Dumas, T. Gautier, G. Huard, B. Raffin, C. Rapine, J.-L. Roch, and D. Trystram. Adaptive and hybrid algorithms: classification and illustration on triangular system solving. In *Transgressive Computing 2006*, pages 131–148, Apr. 2006.

[6] J.-G. Dumas. Efficient dot product over finite fields. In *CASC'2004*, pages 139–154, July 2004.

[7] J.-G. Dumas. Bounds on the coefficients of the characteristic and minimal polynomials. *Journal of Inequalities in Pure and Applied Mathematics*, 8(2):art. 31, 6 pp, Apr. 2007.

[8] J.-G. Dumas, C. Pernet, and Z. Wan. Efficient computation of the characteristic polynomial. In *ISSAC'2005*, pages 140–147, July 2005.

[9] J.-G. Dumas, B. D. Saunders, and G. Villard. On efficient sparse integer matrix Smith normal form computations. *Journal of Symbolic Computation*, 32(1/2):71–99, July–Aug. 2001.

[10] J.-G. Dumas and A. Urbańska. An introspective algorithm for the determinant. In *Transgressive Computing 2006*, pages 185–202, Apr. 2006.

[11] J.-G. Dumas and G. Villard. Computing the rank of sparse matrices over finite fields. In *CASC'2002*, pages 47–62, Sept. 2002.

[12] W. Eberly. Black box frobenius decomposition over small fields. In *ISSAC'2000*, Aug. 2000.

[13] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. Solving sparse rational linear systems. In *ISSAC'2006*, pages 63–70, July 2006.

[14] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. Faster inversion and other black box matrix computations using efficient block projections. In *ISSAC'2007*, pages 143–150, Jul. 29 – Aug. 1 2007.

[15] F. R. Gantmacher. *The Theory of Matrices*. Chelsea, New York, 1959.

[16] J. v. Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[17] J. Gerhard. Fast modular algorithms for squarefree factorization and hermite integration. *Applicable Algebra in Engineering Communication and Computing*, 11(3):203–226, 2001.

[18] F. Heß. Computing relations in divisor class groups of algebraic curves over finite fields. Technical report, 2007. `www.math.tu-berlin.de/~hess/personal/dlog.ps.gz`

[19] O. H. Ibarra, S. Moran, and R. Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45–56, Mar. 1982.

[20] E. Kaltofen. Challenges of symbolic computation: My favorite open problems. *Journal of Symbolic Computation*, 29(6):891–919, June 2000.

[21] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In *AAAAECC'91*, volume 539 of *Lecture Notes in Computer Science*, pages 29–38, Oct. 1991.

[22] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 13(3-4):91–130, 2005.

[23] W. Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theoretical computer science*, 36:309–317, 1985.

[24] C. Pernet and A. Storjohann. Faster algorithms for the characteristic polynomial. In *ISSAC'2007*, pages 307–314, Jul. 29 – Aug. 1 2007.

[25] A. Storjohann. *Algorithms for Matrix Canonical Forms*. PhD thesis, ETH, Zürich, Switzerland, Nov. 2000.

[26] A. Storjohann. Computing the frobenius form of a sparse integer matrix. to be submitted, Apr. 2000.

[27] W. J. Turner. *Blackbox linear algebra with the LinBox library*. PhD thesis, North Carolina State University, May 2002.

[28] G. Villard. Computing the Frobenius normal form of a sparse matrix. In *CASC'00*, pages 395–407, Oct. 2000.

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