MATHEMATICS OF COMPUTATION
VOLUME 62, NUMBER 205
JANUARY 1994, PAGES 333–350

SOLVING HOMOGENEOUS LINEAR EQUATIONS OVER $GF(2)$ VIA BLOCK WIEDEMANN ALGORITHM

DON COPPERSMITH

ABSTRACT. We propose a method of solving large sparse systems of homogeneous linear equations over $GF(2)$, the field with two elements. We modify an algorithm due to Wiedemann. A block version of the algorithm allows us to perform 32 matrix-vector operations for the cost of one. The resulting algorithm is competitive with structured Gaussian elimination in terms of time and has much lower space requirements. It may be useful in the last stage of integer factorization.

1. INTRODUCTION

We address here the problem of solving a large sparse system of homogeneous linear equations over $GF(2)$, the field with two elements. One important application, which motivates the present work, arises in integer factorization. During the last stage of most integer factorization algorithms, we are presented with a large sparse integer matrix and are asked to find linear combinations of the columns of this matrix which vanish modulo 2. For example [7], the matrix may have 100,000 columns, with an average of 15 nonzero entries per column. For this application we would like to obtain several solutions, because a given solution will lead to a nontrivial factorization with probability $1/2$; with n independent solutions, our probability of finding a factorization rises to $1 - 2^{-n}$.

Structured Gaussian elimination can be used [7], but as problems get larger, it may become infeasible to store the matrices obtained in the intermediate stages of Gaussian elimination. The Wiedemann algorithm [9, 7] has smaller storage requirements (one need only store a few vectors and an encoding of a sparse matrix, not a dense matrix as occurs in Gaussian elimination after fill-in), and it may have fewer computational steps (since one takes advantage of the sparseness of the matrix). But its efficiency is hampered by the fact that the algorithm acts on only one bit at a time. In the present paper we work with blocks of vectors at a single time. By treating 32 vectors at a time (on a machine with 32-bit words), we can perform 32 matrix-vector products at once, thus considerably decreasing the cost of indexing. This can be viewed as a block Wiedemann algorithm.

The main technical difficulty is in obtaining the correct generalization of the Berlekamp-Massey algorithm to a block version, namely, a multidimensional version of the extended Euclidean algorithm.

Received by the editor November 20, 1991 and, in revised form, July 24, 1992.
1991 *Mathematics Subject Classification.* Primary 15A33, 11Y05, 11-04, 15-04.

If we have N equations in N unknowns, the present method requires $3N/32$ matrix-vector products (sparse matrix times block of vectors); about $2N/32$ block inner products (multiplication of two matrices of sizes at most $32 \times N$ and $N \times 32$, respectively); and $3N/32$ block scalar products (multiplication of two matrices of size 32×64 and $64 \times N$). As an example of running time, a square matrix of size 65,518, with 20 nonzeros per row, arising from integer factorization, was solved in 65 minutes on an IBM 390. The space requirements are minimal; we need to store the matrix B and about five vectors of N words each.

In an earlier, similar paper [2], the author converted the Lanczos algorithm to a block form, with similar computational requirements. The present algorithm is somewhat easier to understand and to program.

The algorithm is probabilistic. It will succeed with high probability unless the matrix B is pathological, that is, unless B has several nonzero eigenvalues of high multiplicity (at least 32).

2. Wiedemann algorithm

Recall the Wiedemann algorithm [9], as applied to a singular square matrix B of size $N \times N$. (We present the algorithm with some modifications, in order to emphasize the connections with the present algorithm.) The goal is to find a nonzero vector $\mathbf{w}$ such that

$$B\mathbf{w} = \mathbf{0}.$$

Begin with random vectors $\mathbf{x}$, $\mathbf{z}$, and set $\mathbf{y} = B\mathbf{z}$. Compute

$$a^{(i)} = \mathbf{x}^{\mathrm{T}} B^i \mathbf{y}, \qquad 0 \le i \le 2N,$$

and let

$$a(\lambda) = \sum_i a^{(i)} \lambda^i.$$

There is also a linear recurrence which would generate $a^{(i)}$. Namely, there are polynomials $f(\lambda)$, $g(\lambda)$ of degree at most N satisfying

$$f(\lambda)a(\lambda) = g(\lambda) \mod \lambda^{2N+1}.$$

In fact, if $\hat{f}$ is the minimal polynomial of B, its reversal

$$\hat{f}^{\mathrm{rev}}(\lambda) \equiv \lambda^{(\deg \hat{f})} \hat{f}(1/\lambda)$$

can be used for $f(\lambda)$.

In the Wiedemann algorithm, one applies the Berlekamp-Massey algorithm to the data $a^{(i)}$ to obtain polynomials f_1, g_1 such that $g_1/f_1 = g/f$; then one obtains $\hat{f}$ as the least common multiple of several versions of f_1^{rev}. Once one has the minimal polynomial $\hat{f}$ of B, one can set $f^-(\lambda) = \hat{f}(\lambda)/\lambda^k$ (dividing by the highest possible power of λ), compute $f^-(B)\mathbf{z}$ for any vector $\mathbf{z}$, and apply powers of B successively until $B^i f^-(B)\mathbf{z} = \mathbf{0}$; then $\mathbf{w} = B^{i-1} f^-(B)\mathbf{z}$ satisfies $B\mathbf{w} = \mathbf{0}$.

The reason one needs about $2N$ data points $\{\mathbf{x}^{\mathrm{T}} B^i \mathbf{y},\ 0 \le i < 2N\}$, rather than only N as one might initially expect, is that one needs to find a linear combination of the vectors $\{B^i \mathbf{y}\}$ which vanishes, and in general this can require about $N+1$ vectors $\{B^i \mathbf{y},\ 0 \le i \le N\}$. One needs to check that this linear combination is orthogonal to the subspace $\{\mathbf{x}^{\mathrm{T}} B^j\}$, and again we need

about N vectors $\{\mathbf{x}^T B^j,\ 0 \le j < N\}$ to span all of $GF(2)^N$, or rather the subspace spanned by $\{\mathbf{x}^T B^j,\ 0 \le j < \infty\}$. Finally, to evaluate the inner products $\{\mathbf{x}^T B^j \times B^i \mathbf{y},\ 0 \le j < N,\ 0 \le i \le N\}$, we need to know the data $\{\mathbf{x}^T B^i \mathbf{y},\ 0 \le i < 2N\}$.

The computation of the data $a^{(i)}$ involves $2N$ applications of the sparse matrix B, and $2N$ inner products of vectors; the Berlekamp-Massey algorithm takes time $O(N^2)$; and the computation of $f^-(B)\mathbf{z}$ takes another N applications of B.

It is useful to view the Berlekamp-Massey algorithm as an extended Euclidean algorithm [3]. If $\deg(a) = 2N$, $\deg(f) = \deg(g) = N$, then the equation

$$f(\lambda)a(\lambda) = g(\lambda) \mod \lambda^{2N+1}$$

can be interpreted as

$$f(\lambda)a(\lambda) = g(\lambda) + e(\lambda)\lambda^{2N+1},$$

with $\deg(e) = N - 1$. Replacing λ by $1/\lambda$ and multiplying by λ^{3N}, we get

$$f^{\text{rev}}(\lambda)a^{\text{rev}}(\lambda) + g^{\text{rev}}(\lambda)\lambda^{2N} = e^{\text{rev}}(\lambda).$$

The extended Eculidean algorithm would take inputs a^{rev} and λ^{2N}. It would produce successive triples of polynomials $(\underline{f}(\lambda), \underline{g}(\lambda), \underline{e}(\lambda))$ satisfying

$$\underline{f}(\lambda)a^{\text{rev}}(\lambda) + \underline{g}(\lambda)\lambda^{2N} = \underline{e}(\lambda),$$

with $\deg(\underline{f})$, $\deg(\underline{g})$ starting at 0 and increasing, and $\deg(\underline{e})$ starting at $2N$ and decreasing. In fact, it would maintain two triples, $(\underline{f}_1, \underline{g}_1, \underline{e}_1)$ and $(\underline{f}_2, \underline{g}_2, \underline{e}_2)$, and update them by taking linear combinations over $GF(2)[\lambda]$:

$$\begin{bmatrix} \underline{f}_1 & \underline{g}_1 & \underline{e}_1 \\ \underline{f}_2 & \underline{g}_2 & \underline{e}_2 \end{bmatrix}^{\text{new}} = \begin{bmatrix} 0 & 1 \\ 1 & q(\lambda) \end{bmatrix} \times \begin{bmatrix} \underline{f}_1 & \underline{g}_1 & \underline{e}_1 \\ \underline{f}_2 & \underline{g}_2 & \underline{e}_2 \end{bmatrix}^{\text{old}},$$

where at each stage $q(\lambda)$ is the unique polynomial which makes

$$\deg(e_2^{\text{new}}) < \deg(e_2^{\text{old}}).$$

Stopping when $\deg(\underline{f}_1) = N$, the midpoint of the extended Euclidean algorithm, and setting $f^{\text{rev}} = \underline{f}_1$, one would obtain the results of the Berlekamp-Massey algorithm.

Remark. The relation between the Berlekamp-Massey algorithm and the extended Euclidean algorithm is masked somewhat by the reversal of the polynomials. We will continue to work with polynomials in the order specified by Berlekamp-Massey rather than the reverse order specified by Euclid, for ease of computation. Specifically, the operation analogous to multiplication of

$$\begin{bmatrix} 0 & 1 \\ 1 & q(\lambda) \end{bmatrix}$$

will be computationally difficult if $q(\lambda)$ has high degree; instead, we need to be able to multiply by only one power of λ at a time. This seems to be facilitated by keeping the polynomials in the Berlekamp-Massey order.

3. Converting the Wiedemann algorithm to block form

We need to convert the Wiedemann algorithm to a block form. We will process 32 vectors at a time, thus reducing the number of matrix-vector products from $3N$ to $3N/32$. We will also need to modify the Berlekamp-Massey

algorithm. This modification resembles the matrix version of the extended Euclidean algorithm described in [1, 5]. It will also be similar in spirit to that described in [8].

Let $\mathbf{z}$ be a random vector block of width n, that is, an $N \times n$ matrix over $GF(2)$, and $\mathbf{x}$ a random vector block of width m. We may imagine $m = n = 32$. It will be especially efficient to make n equal to the word size of the machine, and to store each $N \times n$ matrix as N words of n bits each. For later analysis we will require $m \geq n$, at no loss of efficiency. Compute $\mathbf{y} = B\mathbf{z}$. Compute

$$a^{(i)} = (\mathbf{x}^{\mathrm{T}} B^i \mathbf{y})^{\mathrm{T}}, \qquad 0 \leq i \leq \frac{N}{m} + \frac{N}{n} + O(1),$$
$$a(\lambda) = \sum_i a^{(i)} \lambda^i.$$

The limit $\frac{N}{m} + \frac{N}{n} + O(1)$ plays the role of $2N$ in the original algorithm. We will be taking linear combinations of the $N = (N/n) \times n$ vectors $\{B^i \mathbf{y}_\nu, 0 \leq i < N/n, 1 \leq \nu \leq n\}$ to find a combination which vanishes. We will be guided by orthogonality to the $N = (N/m) \times m$ vectors $\{\mathbf{x}_\mu^{\mathrm{T}} B^j, 0 \leq j < N/m, 1 \leq \mu \leq m\}$. So we will need the data $\{\mathbf{x}^{\mathrm{T}} B^i \mathbf{y}, 0 \leq i < N/n + N/m\}$. The additive term "$+O(1)$" is a safety measure, to account for possible accidental cancellation, a complication brought on by this block version of the algorithm. (In fact N/n will be replaced by $\underline{d}_n \leq N/n$ in later analysis, to more accurately reflect the number of linearly independent vectors $\{B^i \mathbf{y}_\nu, 0 \leq i < N/n, 1 \leq \nu \leq n\}$. Similarly, N/m will be replaced by $\underline{d}_m \leq N/m$.)

The data $a(\lambda)$ provides the input to the modified Berlekamp-Massey algorithm. It is a polynomial in λ whose coefficients are $n \times m$ matrices over $GF(2)$, while in the original Berlekamp-Massey algorithm, $a(\lambda)$ was a polynomial with coefficients $GF(2)$. Alternatively, $a(\lambda)$ may be viewed as an $n \times m$ matrix whose entries are polynomials over $GF(2)$. The indexing is such that

$$a_{\nu,\mu}^{(i)} = \mathbf{x}_\mu^{\mathrm{T}} B^i \mathbf{y}_\nu, \qquad 1 \leq \mu \leq m, \quad 1 \leq \nu \leq n.$$

Our goal is to find linear combinations of the vectors

$$\{B^i \mathbf{y}_\nu, \ 0 \leq i \leq N/n, \ 1 \leq \nu \leq n\}$$

orthogonal to all the vectors

$$\{\mathbf{x}_\mu^{\mathrm{T}} B^i, \ 0 \leq i \leq N/m, \ 1 \leq \mu \leq n\}.$$

On the tth iteration of the modified Berlekamp-Massey algorithm, we will compute the quantity $f^{(t)}(\lambda)$, an $(m+n) \times n$ matrix whose entires are polynomials over $GF(2)$. Again, $f^{(t)}(\lambda)$ can also be viewed as a polynomial in λ whose coefficients are $(m+n) \times n$ matrices over $GF(2)$. The first index, $1 \leq l \leq m+n$, corresponds roughly to the choice between two trial polynomials $\underline{f}_l$, $1 \leq l \leq 2$, in the straight Berlekamp-Massey algorithm; a row of $f^{(t)}(\lambda)$ corresponds to one of these polynomials.

For each row index l and iteration number t, we maintain a *nominal degree*, $\deg_{\mathrm{nom}} f_l^{(t)} \leq t$, as an upper bound to the maximum, over ν, of the degrees of the (l, ν) entries of $f_l^{(t)}$. This is only an upper bound, not the true degree;

in particular, if we add two rows with identical nominal degree, the sum has the same nominal degree, while the true degree may decrease due to accidental cancellation. So $\deg_{\text{nom}}(f+g) = \max(\deg_{\text{nom}} f, \deg_{\text{nom}} g)$, and $\deg_{\text{nom}}(\lambda f) = 1 + \deg_{\text{nom}} f$. The bit $f_{l,\nu}^{(t,k)}$ belongs to column ν, $1 \le \nu \le n$; row l, $1 \le l \le m+n$; coefficient of λ^k, $0 \le k \le \deg_{\text{nom}} f_l^{(t)}$; and iteration t.

Define $\text{Coeff}(j; f_l^{(t)}a)$ as the coefficient of λ^j in $f_l^{(t)}(\lambda)a(\lambda)$. It is an m-bit vector whose μth bit is

$$\text{Coeff}(j; f_l^{(t)}a)_\mu = \sum_{\substack{0 \le k \le j \\ 1 \le \nu \le n}} f_{l,\nu}^{(t,k)} a_{\nu,\mu}^{(j-k)}.$$

The corresponding $(m+n) \times m$ matrix is denoted $\text{Coeff}(j; f^{(t)}a)$.

We will maintain the conditions

(C1) $$(\forall l,\ 1 \le l \le m+n)(\forall j, \deg_{\text{nom}}(f_l^{(t)}) \le j < t) : \text{Coeff}(j; f_l^{(t)}a) = \mathbf{0}^{\mathrm{T}}$$

(C2) $$\text{Rank}(\text{Coeff}(t; f^{(t)}a)) = m.$$

Condition (C1) helps to find vectors orthogonal to the various $\mathbf{x}_\mu^{\mathrm{T}} B^i$. Namely, for each $1 \le l \le n$, $1 \le \mu \le m$, setting $d = \deg_{\text{nom}} f_l^{(t)}$ and letting j satisfy $d \le j < t$, we have

$$0 = \text{Coeff}(j; f_l^{(t)}a)_\mu = \sum_{\nu,k} f_{l,\nu}^{(t,k)} a_{\nu,\mu}^{(j-k)} = (\mathbf{x}_\mu^{\mathrm{T}} B^{j-d}) \left(\sum_{\nu,k} f_{l,\nu}^{(t,k)} B^{d-k} \mathbf{y}_\nu \right),$$

so that the vector $(\sum_{\nu,k} f_{l,\nu}^{(t,k)} B^{d-k} \mathbf{y}_\nu)$ is orthogonal to the $m \times (t-d) = m \times (t - \deg_{\text{nom}} f_l^{(t)})$ vectors $(\mathbf{x}_\mu^{\mathrm{T}} B^{j-d})$, $1 \le \mu \le m$, $d \le j < t$.

Condition (C2) is needed for the inductive step.

We also envision, but do not explicitly compute, an $(m+n) \times m$ matrix $g^{(t)}$ of polynomials over $GF(2)$, with row l given by $g_l^{(t)}(\lambda) = f_l^{(t)}(\lambda)a(\lambda)$ truncated to $\deg_{\text{nom}} f_l^{(t)}$. Put another way, the coefficient of λ^j in $g_l^{(t)}$ is given by

$$\text{Coeff}(j; f_l^{(t)}a), \qquad 0 \le j \le \deg_{\text{nom}} f_l^{(t)}.$$

Then we maintain the property

(P) $$f_l^{(t)}(\lambda)a(\lambda) - g_l^{(t)}(\lambda) = O(\lambda^t).$$

Concatenate the matrices $f^{(t)}$, $g^{(t)}$ horizontally to produce an $(m+n) \times (m+n)$ matrix $h^{(t)} = [f^{(t)} | g^{(t)}]$; again, $h^{(t)}$ is not explicitly computed, but aids in the analysis.

To begin the inductive process, choose $t_0 = \lceil m/n \rceil$. All rows of $f^{(t_0)}$ will have nominal degree t_0. Select the first m rows of $f^{(t_0)}$ to have actual degree $t_0 - 1$, such that these rows already satisfy condition (C2). (In the case $m = n$, $t_0 = 1$, these rows have actual degree 0. This implies that $a^{(1)} = \mathbf{x}^{\mathrm{T}} B \mathbf{y}$ is nonsingular and these first m rows of $f^{(t_0)} = f^{(1)}$ are linearly independent.) (If we are unable to satisfy (C2), we may need to select a different $\mathbf{x}$, or perhaps increase t_0. If this does not work, then in the span of $\mathbf{z}$, $B\mathbf{z}$, $B^2\mathbf{z}$, ..., $B^{(t_0)}\mathbf{z}$, we already have an element of Kernel(B).) The remaining n rows of $f^{(t_0)}$ are chosen as $\lambda^{t_0} I$, a multiple of the $n \times n$ identity matrix. With this choice

of $f^{(t_0)}$, we find that the matrix $h^{(t_0)}(\lambda)$ is nonsingular; in its determinant, the coefficient of the high-order term $\lambda^{(m+n)t_0}$ is seen to be nonzero. (This construction is due to an anonymous referee.) Condition (C1) is vacuous for $t = t_0$.

In the inductive step, we obtain $f^{(t+1)}$ from $f^{(t)}$. We first compute some nonsingular $(m+n) \times (m+n)$ linear transformation $\tau^{(t)}$ over $GF(2)$ such that the first n rows of $\tau^{(t)}\,\mathrm{Coeff}(t; f^{(t)}a)$ are zero (thus the last m rows are linearly independent), and such that the nominal degrees of the rows of $\tau^{(t)}f^{(t)}$ are the same as those of $f_l^{(t)}$ (in some order). To do this, we first order the rows of $\mathrm{Coeff}(t; f^{(t)}a)$ by nominal degree of the corresponding $f_l^{(t)}$ and do row reduction on $\mathrm{Coeff}(t; f^{(t)}a)$ in such a way that a row of $\mathrm{Coeff}(t; f_l^{(t)}a)$ corresponding to an $f_l^{(t)}$ of higher nominal degree is never subtracted from one of lower nominal degree.

Next we multiply the last m rows of $\tau^{(t)}f^{(t)}$ by λ. Equivalently, we multiply on the left by the diagonal matrix

$$D \equiv \mathrm{Diag}\Big(\underbrace{1, 1, \ldots, 1}_{(n)}, \underbrace{\lambda, \lambda, \ldots, \lambda}_{(m)}\Big).$$

We set $f^{(t+1)} = D\tau^{(t)}f^{(t)}$. Along with this, we have

$$g^{(t+1)} = D\tau^{(t)}g^{(t)}, \qquad h^{(t+1)} = D\tau^{(t)}h^{(t)},$$

but we do not compute these quantities. Notice that $D\tau^{(t)}$ is nonsingular, so that $h^{(t+1)}$ remains nonsingular.

The nominal degrees of the last m rows of $f^{(t+1)}$ have each increased by 1. We can check that conditions (C1), (C2) still hold, with $(t+1)$ replacing (t). Regarding (C1), note that the lower bounds on j have increased by 1 for each of the last m rows. The condition at the upper bound of (C1),

$$\mathrm{Coeff}(t; f_l^{(t+1)}a) = \mathbf{0}^{\mathrm{T}},$$

is arranged by $\tau^{(t)}$ for the first n rows and by the multiplication by λ for the last m rows; this condition is not required on the first few iterations, when $\deg_{\mathrm{nom}}(f_l^{(t+1)}) = t+1$. Condition (C2) is maintained because the last m rows of $\tau^{(t)}\,\mathrm{Coeff}(t; f^{(t)}a)$ were linearly independent, and they become the last m rows of $\mathrm{Coeff}(t+1; f^{(t+1)}a)$.

With this construction, the average nominal degree

$$\frac{1}{m+n}\sum_l \deg_{\mathrm{nom}} f_l^{(t)}$$

increases by exactly $m/(m+n)$ as t increases by 1; together with the initial conditions, we find that this average degree is $tm/(m+n) + t_0 n/(m+n) = tm/(m+n) + O(1)$.

We now give an overview of the termination of the procedure and the manner in which solutions are obtained. Probabilistic justification is deferred until §6.

By the time $t = (N/m) + (N/n) + O(1)$, we find that the difference between t and the average nominal degree exceeds N/m. So there are values of l such that $t - \deg_{\mathrm{nom}} f_l^{(t)} > N/m$. For such l, the vector $(\sum_{\nu,k} f_{l,\nu}^{(t,k)} B^{d-k}\mathbf{y}_\nu)$ is orthogonal to the $m \times (t - \deg_{\mathrm{nom}} f_l^{(t)}) > N$ vectors $(\mathbf{x}_\mu^{\mathrm{T}} B^{j-d})$, $1 \le \mu \le m$,

$d \le j < t$. In §6 we will argue that with high probability the space spanned by these vectors already contains $\mathbf{x}_\mu^{\mathrm{T}} B^i$ for all i and also that with high probability a linear combination of $B^i \mathbf{y}_\nu$ orthogonal to that space is already $\mathbf{0}$.

In fact, with high probability, for some $t = (N/m) + (N/n) + O(1)$, instead of (C1) a stronger condition holds on (most of) the first n rows, namely:

(C1′) $\quad (\forall l,\ 1 \le l \le n)(\forall j, \deg_{\mathrm{nom}}(f_l^{(t)}) \le j < \infty)\colon \mathrm{Coeff}(j; f_l^{(t)} a) = \mathbf{0}^{\mathrm{T}}$.

(This may fail for a few values of l.) Here, $a(\lambda)$ is the infinite power series

$$a(\lambda) = \sum_{i=0}^{\infty} (\mathbf{x}^{\mathrm{T}} B^i \mathbf{y})^{\mathrm{T}} \lambda^i,$$

of which we have actually only computed the first $N/m + N/n + O(1)$ terms. (See "Termination" in §6.)

Our first indication of this success is that in the linear transformation $D\tau^{(t)}$ creating $f^{(t+1)}$ from $f^{(t)}$ it happens that

$$f_l^{(t+1)} = f_l^{(t)}, \qquad 1 \le l \le n,$$

implying that

$$\mathrm{Coeff}(t; f_l^{(t)} a) = \mathbf{0}^{\mathrm{T}}, \qquad 1 \le l \le n;$$

that is, condition (C1) held for the value $j = t$ when it was only required for values of j up to $t-1$. When we detect this condition on one or two successive rounds, we terminate this phase of the algorithm.

Suppose (C1′) holds for a given value of l. Note first that $f_l^{(t)}$ is not identically 0, because otherwise row l of $h^{(t)}$ would also be 0, which is impossible since $h^{(t)}$ is nonsingular. Set $d = \deg_{\mathrm{nom}}(f_l^{(t)})$, and suppose $d' \le d$ is the true degree of $f_l^{(t)}$, namely $d' = \sup\{k | \exists \nu, f_{l,\nu}^{(t,k)} \ne 0\}$. (Recall that the nominal degree d can be larger than d' due to initial conditions or cancellation of leading terms.) We will define

$$\tilde{f}_{l,\nu}(\lambda) = \lambda^{-d+d'} f_{l,\nu}^{(t)\,\mathrm{rev}}(\lambda) = \sum_k f_{l,\nu}^{(t,k)} \lambda^{d'-k},$$

and let $\tilde{f}_{l,\nu}^{(0)}$ denote its constant term, the coefficient of λ^0. By construction, for each l there is a ν such that $\tilde{f}_{l,\nu}^{(0)} \ne 0$. (The polynomial reversal $f_{l,\nu}^{(t)\,\mathrm{rev}}(\lambda)$ is with respect to the nominal degree d, not the actual degree d'. The factor $\lambda^{-d+d'}$ accounts for the difference.) As above, we have, for all $j \ge d = \deg_{\mathrm{nom}} f_l^{(t)}$, $1 \le \mu \le m$,

$$0 = \mathrm{Coeff}(j; f_l^{(t)} a)_\mu = \sum_{k,\nu} f_{l,\nu}^{(t,k)} a_{\nu,\mu}^{(j-k)} = \mathbf{x}_\mu^{\mathrm{T}} B^{j-d'} \sum_{\nu,k} f_{l,\nu}^{(t,k)} B^{d'-k} \mathbf{y}_\nu$$

$$= \mathbf{x}_\mu^{\mathrm{T}} B^{j+1-d'} \sum_{\nu,k} f_{l,\nu}^{(t,k)} B^{d'-k} \mathbf{z}_\nu = (\mathbf{x}_\mu^{\mathrm{T}} B^{j+1-d'}) \left(\sum_\nu \tilde{f}_{l,\nu}(B) \mathbf{z}_\nu \right).$$

Then setting $\hat{\mathbf{w}}_l \equiv \sum_\nu \tilde{f}_{l,\nu}(B) \mathbf{z}_\nu$, we find that

$$(\mathbf{x}^{\mathrm{T}} B^i)(B^{1+d-d'} \hat{\mathbf{w}}_l) = 0, \qquad 0 \le i < \infty.$$

Because $B^{1+d-d'}\hat{\mathbf{w}}_l$ is orthogonal to the space spanned by $\mathbf{x}^T$ times all powers of B, there is a high probability that

$$B^{1+d-d'}\hat{\mathbf{w}}_l = \mathbf{0}.$$

(See "Probabilistic Justification.")

If $B^{1+d-d'}\hat{\mathbf{w}}_l = \mathbf{0}$ and $\hat{\mathbf{w}}_l \neq \mathbf{0}$, let i be the largest exponent satisfying $B^i\hat{\mathbf{w}}_l \neq \mathbf{0}$. We have $0 \leq i \leq d - d'$. Setting $\mathbf{w}_l \equiv B^i\hat{\mathbf{w}}_l$, we find that $\mathbf{w}_l \neq \mathbf{0}$, $B\mathbf{w}_l = \mathbf{0}$, so that $\mathbf{w}_l$ is the desired solution.

If everything works as planned, then we have found n elements of Kernel(B), one for each value of l, $1 \leq l \leq n$. In practice, most of these elements have been linearly independent; we may find $(n-1)$ or $(n-2)$ linearly independent elements of Kernel(B).

Technicalities. The following technicalities can be skipped on a first reading.

If $\hat{\mathbf{w}}_l = \mathbf{0}$, then no nontrivial solution is produced corresponding to the index l. This is why we work with $\mathbf{y} = B\mathbf{z}$ during the first part of the computation, and with $\mathbf{z}$ when we try to construct the solution. Fix a value of the index l. The algorithm finds $f^{(t)}$ based only on $\mathbf{x}$ and $\mathbf{y}$, ignoring $\mathbf{z}$. Two random choices of $\mathbf{z}$ differing by elements of the kernel of B would give rise to the same $\mathbf{y}$, and hence the same $f^{(t)}$, and even the same $B\hat{\mathbf{w}}_l$, since this can be computed from $f^{(t)}$ and $\mathbf{y} = B\mathbf{z}$, not using $\mathbf{z}$ itself. But the computed values of $\hat{\mathbf{w}}_l$ would differ by a random element of the kernel of B, since for some value of ν the constant coefficient in $\tilde{f}_{l,\nu}$ is no zero. Loosely speaking, $\hat{\mathbf{w}}_l$ is determined only up to a random element of Kernel(B). So the probability that $\hat{\mathbf{w}}_l = \mathbf{0}$, given that we get that far, is at most $1/|\text{Kernel}(B)|$, which is quite small in our applications, and at any rate bounded by $1/2$.

However, $\mathbf{w}_l$ is not a random element of the kernel, because it is obtained from $\hat{\mathbf{w}}_l$ by multiplication by B^i, and if $i > 0$, the random kernel element is annihilated by this multiplication. Even the various $\hat{\mathbf{w}}_l$ need not be independent as random variables.

There are two ways to try to counteract this dependence of random variables and thereby work towards n linearly independent solutions $\mathbf{w}_l$. These methods amount to removal of factors of B from linear combinations of $\hat{\mathbf{w}}_l$ and of $\mathbf{w}_l$, respectively.

As an example of the first method, suppose $\hat{\mathbf{w}}_1 = \mathbf{z}_1 + B\mathbf{z}_2$ and $\hat{\mathbf{w}}_2 = \mathbf{z}_1 + B\mathbf{z}_3$. Then the linear combination $\hat{\mathbf{w}}_1 - \hat{\mathbf{w}}_2 = B\mathbf{z}_2 - B\mathbf{z}_3$ has an unnecessary factor of B. In this case we can replace $\hat{\mathbf{w}}_2$ by $\hat{\mathbf{w}}_2' = B^{-1}(\hat{\mathbf{w}}_1 - \hat{\mathbf{w}}_2) = \mathbf{z}_2 - \mathbf{z}_3$.

More generally, we should ensure that the $n \times n$ matrix with entries given by the constant coefficients $\tilde{f}_{l,\nu}^{(0)}$ is nonsingular. If it is singular, find a linear combination which vanishes: $\alpha_l \in GF(2)$, $1 \leq l \leq n$, not all zero (say $\alpha_1 \neq 0$), such that

$$\sum_l \alpha_l \tilde{f}_{l,\nu}^{(0)} = 0, \qquad 1 \leq \nu \leq n.$$

Then replace $\tilde{f}_{1,\nu}(\lambda)$ by

$$\left(\sum_l \alpha_l \tilde{f}_{l,\nu}(\lambda)\right)/\lambda.$$

Repeat until the matrix is nonsingular. We have taken a linear combination of $\hat{\mathbf{w}}_l$ and removed an unnecessary factor of B.

Then we can argue as before: $B\hat{\mathbf{w}}_l$ can be computed from $f^{(t)}$ and $\mathbf{y} = B\mathbf{z}$, not using $\mathbf{z}$ itself; changing $\mathbf{z}_\nu$ by a random element of Kernel(B) would not affect $B\hat{\mathbf{w}}_l$, but would affect $\hat{\mathbf{w}}_l$ by $\tilde{f}^{(0)}_{l,\nu}$ times that random element of the kernel; and therefore the $\hat{\mathbf{w}}_l$ are determined only up to independent random elements of the kernel.

But the $\mathbf{w}_l$ are still not necessarily independent, either as random variables or as vectors. The second method attempts to rectify that. As an example, suppose that we have

$$B^2\hat{\mathbf{w}}_1 = \mathbf{0}, \quad B\hat{\mathbf{w}}_1 \neq \mathbf{0}, \quad \mathbf{w}_1 = B\hat{\mathbf{w}}_1,$$
$$B^2\hat{\mathbf{w}}_2 = \mathbf{0}, \quad B\hat{\mathbf{w}}_2 \neq \mathbf{0}, \quad \mathbf{w}_2 = B\hat{\mathbf{w}}_2,$$

but also

$$\mathbf{w}_1 + \mathbf{w}_2 = \mathbf{0},$$

in which case one of our desired solutions has vanished. In this case we can replace $\mathbf{w}_1$ by $\mathbf{w}'_1 = \hat{\mathbf{w}}_1 + \hat{\mathbf{w}}_2$, so that the new value $\mathbf{w}'_1$ satisfies $B\mathbf{w}'_1 = \mathbf{0}$, and hope that $\mathbf{w}'_1$ is a *nonzero* solution. We have taken a linear combination of $\mathbf{w}_l$ and removed an unnecessary factor of B.

These are technicalities which did not come up doing the development and testing of the algorithm, but which might be useful in real-world applications.

4. Minimal polynomial

In the proposed algorithm we do not need to develop $\hat{f}$, the minimal polynomial of B. If we happen to want it for another purpose, it may be available. Let F be the $n \times n$ submatrix formed from the first n rows of $f^{(t)}$ on the final iteration (t), and evaluate $\det(F)^{\text{rev}}$, the reversal of the determinant of F. If $\det(F)^{\text{rev}}$ is nonzero, then for some i, $\lambda^i \det(F)^{\text{rev}}$ will probably be divisible by $\hat{f}$, since $B^i \det(F)^{\text{rev}}(B)$ annihilates the n random vectors $\mathbf{z}_\nu$. The degree of $\det(F)^{\text{rev}}$ will be bounded by about N (in fact, by $n\underline{d}_n$; see later analysis). However, $\det(F)^{\text{rev}}$ can still be too large; it will probably contain multiple copies of any factor of $\hat{f}$ such that $\hat{f}(B)$ has a large kernel. If $\det(F) = 0$, this technique fails, but we consider this unlikely.

To see that $B^i \det(F)^{\text{rev}}(B)$ annihilates $\mathbf{z}_\nu$, consider the case $n = 2$:

$$\begin{aligned} B^i \det(F)^{\text{rev}}(B)\mathbf{z}_1 &= B^i f_{11}^{\text{rev}}(B) f_{22}^{\text{rev}}(B)\mathbf{z}_1 - B^i f_{12}^{\text{rev}}(B) f_{21}^{\text{rev}}(B)\mathbf{z}_1 \\ &= B^i f_{22}^{\text{rev}}(B)[f_{11}^{\text{rev}}(B)\mathbf{z}_1 + f_{12}^{\text{rev}}(B)\mathbf{z}_2] \\ &\quad - B^i f_{12}^{\text{rev}}(B)[f_{21}^{\text{rev}}(B)\mathbf{z}_1 + f_{22}^{\text{rev}}(B)\mathbf{z}_2] \\ &= \mathbf{0} + \mathbf{0} \end{aligned}$$

for i sufficiently large, remembering that polynomials in B commute. Similar computations can be done for larger values of n.

With high probability, the only polynomials in B which annihilate the n randomly chosen vectors $\mathbf{z}_\nu$ are multiplies of the minimal polynomial.

5. Computational cost

In the first phase we compute the vectors $B^i\mathbf{y}$ at a cost of $\frac{N}{m} + \frac{N}{n} + O(1)$ iterations, each involving application of the sparse matrix B to an $N \times n$ matrix of bits, stored as an N-vector of words. If $\mathbf{x}$ is a random $N \times m$ matrix, then

computing $a_{\nu,\mu}^{(i)}$ from $B^i\mathbf{y}$ costs $\frac{N}{m}+\frac{N}{n}+O(1)$ inner products $(\mathbf{x}, B^i\mathbf{y}) \to a^{(i)} = (\mathbf{x}^T B^i \mathbf{y})$. But in practice we let $x_{i,j} = \delta_{i,j}$, so that the computation of $a^{(i)}$ is trivial.

In the second phase we develop the vector polynomials $f_l^{(t)}(\lambda)$. For each of at most $\frac{N}{m}+\frac{N}{n}+O(1)$ iterations $t_0 \le t \le \frac{N}{m}+\frac{N}{n}+O(1)$, this involves multiplying a subarray of $f^{(t)}$ of dimension $n \times n \times (mt/(m+n))$, by a subarray of a of dimension $n \times (mt/(m+n)) \times m$, to produce the $(n \times m)$-bit matrix $\mathrm{Coeff}(t; f_l^{(t)} a)$, $1 \le l \le n$; alternatively, this can be viewed as the multiplication of matrices over $GF(2)$ of dimensions $n \times (nmt/(m+n))$ and $(nmt/(m+n)) \times m$. (We know that the *average* (over l) nominal degree of $f_l^{(t)}$ is $(mt/(m+n))+O(1)$, but in practice this is also the *maximum* nominal degree.) It also involves application of the linear transformation $\tau^{(t)}$, which can be viewed as multiplication of bit matrices of dimensions $n \times (m+n)$ and $(m+n) \times (mnt/(m+n))$, along with copying the other m vectors without change.

The third phase computes $\hat{\mathbf{w}}_l$. This costs about N/n applications of the sparse matrix B to an N-vector of words, plus N/n applications of $n \times n$ linear transformations $f_{*,*}^{(t,k)}$ to $N \times n$ matrices $B^i\mathbf{y}$. All n vectors $\hat{\mathbf{w}}_l$, $1 \le l \le n$, can be computed simultaneously.

The total cost is about $(2N/n)+(N/m)$ matrix-vector products plus a small $O(N^2)$ term for the linear transformations. If $n = m = 32$, the dominant cost is $3N/32$ matrix-vector products; if $n = 32$, $m = 64$, the dominant cost is $(2.5)N/32$ matrix-vector products, but the overhead increases slightly.

A FORTRAN implementation of this algorithm solved a sample matrix from integer factorization, of size 65518 with 20 nonzero entries per row on average, in 65 minutes on the IBM 390, using the parameters $m = n = 32$.

We foresee that a matrix of size 400,000, with an average of 70 nonzeros per row, could be handled in less than a week on a large workstation, such as an IBM RISC System/6000 with 160 megabytes of memory. Such a matrix might arise when factoring the 129-digit RSA challenge integer [7].

Storage. The largest storage requirement is for the matrix B. In addition, to store the arrays f, a, etc., we need about $N \times \max(2m+2n, m+4n)$ bits or $N \times \max(2m+2n, m+4n)/32$ words, as follows. During Stage 1 we need to maintain $B^i\mathbf{y}$ and $B^{i+1}\mathbf{y}$, developing the latter from the former; these require Nn bits each. We also develop the array a, which requires $mn(N/m+N/n) = N(m+n)$ bits. Stage 2 needs the array a and the array $f^{(t)}$ for one value of t at a time. We can overwrite $f^{(t)}$ when computing $f^{(t+1)}$. The array $f^{(t)}$ also requires $N(m+n)$ bits, recalling that $\deg_{\mathrm{nom}}(f_l^{(t)})$ is about $tm/(m+n) \le N/n$. During Stage 3 we need B, $f^{(t)}$, $B^i\mathbf{z}$, $B^{i+1}\mathbf{z}$, and $\hat{\mathbf{w}}$, requiring $N(m+4n)$ bits besides the storage for B.

Implementation details. The speed of the block vector operations depends strongly on the implementation. We show here how two of these operations can be performed efficiently. In this subsection, suppose that $\mathbf{u}$ and $\mathbf{v}$ are both $N \times 32$ bit matrices, stored as N words of 32 bits each; for $1 \le i \le N$, the ith word of $\mathbf{u}$ will be denoted $\mathbf{u}_i$. Suppose also that ω is a 32×32 bit matrix, stored as 32 words; and C is a temporary array of 4×256 words. We show

how to calculate both of the following:

$$\omega \leftarrow \mathbf{u}^{\mathrm{T}}\mathbf{v}, \qquad \mathbf{v} \leftarrow \mathbf{u}\omega .$$

The first calculation, $\omega \leftarrow \mathbf{u}^{\mathrm{T}}\mathbf{v}$, arises when we multiply a subarray of $f^{(t)}$ by a subarray of a. We initialize the array C to 0. For each i, $1 \le i \le N$, we express the word $\mathbf{u}_i$ as the concatenation of four 8-bit indices:

$$\mathbf{u}_i = (j_1 | j_2 | j_3 | j_4) .$$

For each k, $1 \le k \le 4$, we set

$$C(k, j_k) \leftarrow C(k, j_k) \oplus \mathbf{v}_i .$$

Repeat for all i, $1 \le i \le N$. At the end of this stage, the word $C(k, j)$ is the sum (mod 2) of those words in $\mathbf{v}$ corresponding to words in $\mathbf{u}$ whose kth byte (index) had value j. Now for each k, $1 \le k \le 4$, and each l, $1 \le l \le 8$, add (exclusive or) together those words $C(k, j)$ whose index j has a 1 in the l bit, to form word $(8(k-1)+l)$ of the desired product $\omega = \mathbf{u}^{\mathrm{T}}\mathbf{v}$. (This calculation, in turn, can borrow ideas from the Fast Fourier Transform.)

Similar tricks are used to compute $\mathbf{v} \leftarrow \mathbf{u}\omega$, which arises in the application of the linear transformation $\tau^{(t)}$. We preprocess ω to initialize the array C, so that for each j, $0 \le j \le 255$, the word $C(1, j)$ is the product of the 32-bit vector $(j | 0 | 0 | 0)$ with ω (each 0 represents 8 bits):

$$C(1, j) = (j | 0 | 0 | 0) \times \omega, \qquad C(2, j) = (0 | j | 0 | 0) \times \omega,$$
$$C(3, j) = (0 | 0 | j | 0) \times \omega, \qquad C(4, j) = (0 | 0 | 0 | j) \times \omega .$$

(Again this preprocessing can be done in the style of a Fast Fourier Transform.) Now the word $\mathbf{v}_i$ in the product is gotten by expressing $\mathbf{u}_i$ as the concatenation of four 8-bit indices and adding together the corresponding elements of C:

$$\mathbf{u}_i = (j_1 \mid j_2 \mid j_3 \mid j_4),$$
$$\mathbf{v}_i = C(1, j_1) \oplus C(2, j_2) \oplus C(3, j_3) \oplus C(4, j_4) .$$

6. Probabilistic justification

We have given an outline of a block version of the Wiedemann algorithm. Its success depends on several probabilistic events. Here we argue that these events have a high probability of occurring, and that the algorithm will succeed with high probability, unless the matrix B is pathological in a certain way. This section may be viewed as an appendix and can be safely skipped on a first reading.

We define several parameters, $\underline{d}_m$, $\underline{d}_n$, $\underline{c}_m$, $\underline{c}_n$, and $\underline{e}$, depending on B, to help describe the behavior of the algorithm.

For any univariate polynomial $r(\lambda)$ over $GF(2)$, define

$$\underline{d}_m(r) = \deg(r) + \frac{1}{m} \times \mathrm{Rank}(r(B))$$

and set $\underline{d}_m = \min_r \underline{d}_m(r)$. Clearly, any polynomial r achieving this minimum is a divisor of $\hat{f}$, the minimal polynomial of B. (Otherwise, set $\tilde{r} = \gcd(r, \hat{f})$, notice that $\mathrm{Rank}(\tilde{r}(B)) = \mathrm{Rank}(r(B))$, and thus $\underline{d}_m(\tilde{r}) < \underline{d}_m(r)$.)

Setting $r(\lambda) = 1$, we see

$$\underline{d}_m \leq N/m = (\text{number of columns})/m,$$

and setting $r(\lambda) = \lambda$, we have

$$\underline{d}_m \leq 1 + \text{Rank}(B)/m \leq 1 + (\text{number of nonzero rows})/m.$$

For the sort of B arising from integer factorization, this latter bound is usually tight.

Intuitively, $\underline{d}_m$ measures the linear independence of successive vector blocks $\mathbf{x}^T B^i$, in the sense that the number of linearly independent vectors among $\{\mathbf{x}^T B^i,\ 0 \leq i < d\}$ is bounded above by $\min(md, m\underline{d}_m)$, and with high probability this bound is nearly tight; see Lemmas 6.1 and 6.2.

Define

$$\underline{c}_m = 1 + \sum_{r|\hat{f}} 2^{-m(\underline{d}_m(r) - \underline{d}_m)}.$$

We have $\underline{c}_m \geq 2$, and $\underline{c}_m$ will be very close to 2 unless several polynomials r compete for the minimum in the definition of $\underline{d}_m$, that is, unless there are several polynomials with $\dim(\text{Kernel}(r(B))) \simeq m$. A large value of $\underline{c}_m$ will be indicative of the "pathological" matrix B, for which our methods might fail.

The definitions of $\underline{d}_n$, $\underline{c}_n$ are entirely analogous, with n replacing m.

Similarly, define

$$\underline{e} = 1 + \sum_{s|\hat{f}} 2^{-n \deg(s)} + \sum_{rs|\hat{f}} 2^{-(m-n)\deg(r) - n(\underline{d}_n(rs) - \underline{d}_n)}.$$

Assume $m \geq n$. In the first sum consider the term $s = 1$, and in the second sum, $r = 1$ and s corresponding to the maximum defining $\underline{d}_n$; then we find $\underline{e} \geq 3$. Unless B is pathological (has several nonzero eigenvalues with high multiplicity), we will have $\underline{e} \leq 8$. In fact, we take this as a definition:

Definition. B is *pathological* if $\max(\underline{c}_n, \underline{c}_m, \underline{e}) > 8$.

This concept is somewhat unintuitive, and infeasible to verify, but it captures the class of matrices B for which our algorithm is likely to fail. It has been our experience that the matrices arising from integer factorization are not pathological.

From the parameter $\underline{d}_n$ we can compute a tight estimate of the number of rounds needed for Phase 2, namely, $\underline{d}_n(m+n)/m$. The parameters $\underline{c}_m$, $\underline{c}_n$, $\underline{e}$, help determine the probability of success; if all are small, the algorithm will succeed with high probability.

Definition. For $1 \leq d \leq \infty$, define the vector blocks

$$X^T(d) = \{\mathbf{x}_\mu^T B^i | 1 \leq \mu \leq m,\ 0 \leq i < d\},$$
$$Z(d) = \{B^j \mathbf{z}_\nu | 1 \leq \nu \leq n,\ 0 \leq j < d\}.$$

These blocks depend on the random choices $\mathbf{x}$, $\mathbf{z}$. For d_1, $d_2 \geq 1$, define the matrix

$$W(d_1, d_2) = X^T(d_1) \times Z(d_2).$$

This is a $d_1 \times d_2$ block matrix with blocks of size $m \times n$, whose (i, j) block, $0 \leq i < d_1$, $0 \leq j < d_2$, is given by $\mathbf{x}^T B^{i+j} \mathbf{z}$.

Lemma 6.1. *We have* $\mathrm{Rank}(X(d)) \leq \min(md, m\underline{d}_m)$, *and* $\mathrm{Rank}(Z(d)) \leq \min(nd, n\underline{d}_n)$.

Proof. It is immediate that $\mathrm{Rank}(X(d)) \leq md$. To show that $\mathrm{Rank}(X(d)) \leq m\underline{d}_m$, fix a polynomial r. For each $i \geq \deg(r)$, $\mathbf{x}_\mu^{\mathrm{T}} B^i$ differs from a linear combination of $\mathbf{x}_\mu^{\mathrm{T}} B^j$, $j < i$, by a row in the image of $r(B)$. So the $(m \deg(r))$ vectors $\mathbf{x}_\mu^{\mathrm{T}} B^i$, $1 \leq \mu \leq m$, $0 \leq i < \deg(r)$, together with $\mathrm{Rank}(r(B))$ vectors representing a row basis of $r(B)$, span a space which includes $X(d)$. Then the rank of $X(d)$ is bounded above by $m\underline{d}_m(r)$, hence by $m\underline{d}_m$. The second statement follows analogously. □

Lemma 6.2. *Let* $1 \leq d < \infty$. *Let* "Exp" *denote the expected value; here the expected value is over the random choice of* $\mathbf{x}$. *Then*

$$\mathrm{Exp}(2^{-\mathrm{Rank}(X(d))}) \leq \underline{c}_m 2^{-\min(md, m\underline{d}_m)}.$$

Proof. Let $r_\mu(\lambda)$, $1 \leq \mu \leq m$, be polynomials of degree at most $d-1$ over $GF(2)$ such that

$$\sum_\mu \mathbf{x}_\mu^{\mathrm{T}} r_\mu(B) = \mathbf{0}^{\mathrm{T}}.$$

Let $F(d)$ denote the number of choices of such collections of m polynomials. We have $F(d) = 2^{md - \mathrm{Rank}(X(d))}$, so that

$$\mathrm{Exp}(2^{-\mathrm{Rank}(X(d))}) = 2^{-md}\,\mathrm{Exp}(F(d)).$$

The trivial choice of polynomials r_μ is $r_\mu(\lambda) = 0$, $1 \leq \mu \leq m$. For any nontrivial choice, consider the polynomial

$$r(\lambda) = \gcd(\hat{f}(\lambda), \{r_\mu(\lambda) | 1 \leq \mu \leq m\}),$$

where $\hat{f}(\lambda)$ is the minimal polynomial of B. Given r a divisor of $\hat{f}$, the number of possible r_μ such that r divides r_μ is $2^{(d-\deg(r))}$, so the number of choices of all r_μ, $1 \leq \mu \leq m$, for a given r is at most $2^{m(d-\deg(r))}$. (One of these choices is the trivial choice $r_\mu = 0$, and some others may correspond to large gcd and hence a different $r(\lambda)$, so this is only an upper bound.) Given r_μ, the probability that a random collection of vectors $\mathbf{x}_\mu$ will satisfy

$$\sum_\mu \mathbf{x}_\mu^{\mathrm{T}} r_\mu(B) = \mathbf{0}^{\mathrm{T}}$$

is the same as the probability that a random vector $\mathbf{x}_0$ will satisfy $\mathbf{x}_0^{\mathrm{T}} r(B) = \mathbf{0}^{\mathrm{T}}$, and this probability is $2^{-\mathrm{Rank}(r(B))}$. Putting these facts together, interchanging quantifiers, and letting "Exp" denote expected value and "Prob" probability, we get

$$\begin{aligned}\mathrm{Exp}_{\mathbf{x}} \#\left\{ r_\mu \neq 0 \,\middle|\, \sum_\mu \mathbf{x}_\mu^{\mathrm{T}} r_\mu(B) = \mathbf{0}^{\mathrm{T}} \right\} &= \sum_{r_\mu \neq 0} \mathrm{Prob}_{\mathbf{x}} \left\{ \sum_\mu \mathbf{x}_\mu^{\mathrm{T}} r_\mu(B) = \mathbf{0}^{\mathrm{T}} \right\} \\ &\leq \sum_{r|\hat{f}} (2^{m(d-\deg(r))})(2^{-\mathrm{Rank}(r(B))}) = \sum_{r|\hat{f}} 2^{-m(\underline{d}_m(r)-d)}.\end{aligned}$$

Including the trivial solution, we get

$$\mathrm{Exp}(F(d)) = \mathrm{Exp}_{\mathbf{x}} \#\left\{ r_\mu \,\middle|\, \sum_\mu \mathbf{x}_\mu^{\mathrm{T}} r_\mu(B) = \mathbf{0}^{\mathrm{T}} \right\} \leq 1 + \sum_{r|\hat{f}} 2^{-m(\underline{d}_m(r)-d)}.$$

Finally,

$$\text{Exp}(2^{-\text{Rank}(X(d))}) \leq 2^{-md} + \sum_{r|\hat{f}} 2^{-m\underline{d}_m(r)} \leq \underline{c}_m 2^{-\min(md, m\underline{d}_m)}. \quad \square$$

Corollary 6.3. *The probability that* $\text{Rank}(X(d)) \leq \min(md, m\underline{d}_m) - \Delta$ *is bounded by* $\underline{c}_m 2^{-\Delta}$.

Lemma 6.4. *Let* $1 \leq d < \infty$. *Let* $s(\lambda)$ *be a factor of* $\hat{f}(\lambda)$. *Then*

$$\text{Exp}(2^{-\text{Rank}(X^{\text{T}}(d)s(B))}) \leq 2^{-md} + \sum_{r|(\hat{f}/s)} 2^{-m\deg(r)-\text{Rank}(rs(B))}.$$

Proof. Analogous to the previous lemma. $\square$

Lemma 6.5. *Select integers* d_1, d_2, m, n, *with* $m \geq n$. *(In our application we will have* $md_1 \geq nd_2$.*) Then*

$$\text{Exp}(2^{-\text{Rank}(W(d_1, d_2))}) \leq \underline{e} 2^{-\min(md_1, nd_2, n\underline{d}_n)}.$$

Proof. Again, we proceed by bounding the expected size of the right kernel of $W(d_1, d_2)$. Let $r(\lambda)$, $s(\lambda)$ be two univariate polynomials over $GF(2)$ whose product divides $\hat{f}(\lambda)$. A nonzero element of the right kernel of $W(d_1, d_2)$ corresponds to a collection of polynomials $s_\nu(\lambda)$ of degree at most $d_2 - 1$, not all zero, such that $\hat{\mathbf{z}} \equiv \sum_\nu s_\nu(B)\mathbf{z}_\nu$ satisfies $X^{\text{T}}(d_1)\hat{\mathbf{z}} = \mathbf{0}$. For such an element, set

$$s(\lambda) = \gcd(\hat{f}(\lambda), \{s_\nu(\lambda) | 1 \leq \nu \leq n\}).$$

If we first fix s_ν and then randomly select $\mathbf{z}$, the image $\hat{\mathbf{z}} = \sum_\nu s_\nu(B)\mathbf{z}_\nu$ has the same distribution as the image $s(B)\mathbf{z}_0$ with $\mathbf{z}_0$ chosen at random. So for fixed s_ν and random $\mathbf{x}$, $\mathbf{z}$, the probability that $\hat{\mathbf{z}}$ is orthogonal to $X^{\text{T}}(d_1)$ is the same as the probability that a random $\mathbf{z}_0$ is orthogonal to $X^{\text{T}}(d_1)s(B)$. By Lemma 6.4, this probability is bounded by

$$2^{-md_1} + \sum_{r|(\hat{f}/s)} 2^{-m\deg(r)-\text{Rank}(rs(B))}.$$

Corresponding to a given choice of s, the number of choices of s_ν is at most $2^{n(d_2-\deg(s))}$. So the expected number of choices of s_ν corresponding to a given s for randomly chosen $\mathbf{x}$, $\mathbf{z}$ is bounded by

$$2^{nd_2-n\deg(s)-md_1} + \sum_{r|(\hat{f}/s)} 2^{nd_2-n\deg(s)-m\deg(r)-\text{Rank}(rs(B))},$$

and the expected total number of choices, including the trivial choice $s_\nu = 0$, $1 \leq \nu \leq n$, is bounded by

$$1 + 2^{nd_2-md_1} \sum_{s|\hat{f}} 2^{-n\deg(s)} + \sum_{rs|\hat{f}} 2^{nd_2-m\deg(r)-n\deg(s)-\text{Rank}(rs(B))},$$

the latter sum being taken over pairs of polynomials (r, s) whose product di-

vides $\hat{f}$. Then

$$\begin{aligned}
&\mathrm{Exp}(2^{-\mathrm{Rank}(W(d_1,d_2))}) \\
&\quad \le 2^{-nd_2} + 2^{-md_1}\sum_{s|\hat{f}} 2^{-n\deg(s)} + \sum_{rs|\hat{f}} 2^{-m\deg(r)-n\deg(s)-\mathrm{Rank}(rs(B))} \\
&\quad \le 2^{-\min(md_1,nd_2,n\underline{d}_n)}\left(1 + \sum_{s|\hat{f}} 2^{-n\deg(s)} + \sum_{rs|\hat{f}} 2^{-(m-n)\deg(r)-n(\underline{d}_n(rs)-\underline{d}_n)}\right) \\
&\quad \le 2^{-\min(md_1,nd_2,n\underline{d}_n)}\underline{e}. \quad \square
\end{aligned}$$

Corollary 6.6. *If $\Delta \ge 0$ is an integer, then*

$$\mathrm{Prob}\{\mathrm{Rank}(W(d_1,d_2)) \le \min(md_1, nd_2, n\underline{d}_n) - \Delta\} \le \underline{e}2^{-\Delta}.$$

Now restrict to the case where $md_1 \ge \min(nd_2, n\underline{d}_n)$. Then with probability at least $1 - \underline{e}2^{-\Delta}$ we have

$$\mathrm{Rank}(W(d_1,d_2)) > \min(nd_2, n\underline{d}_n) - \Delta.$$

But we also have with certainty $\mathrm{Rank}(Z(d_2)) \le \min(nd_2, n\underline{d}_n)$. So with high probability (at least $15/16 = 1 - 2^{-4}$),

$$\dim(X(d_1)^{\perp} \cap \mathrm{Span}(Z(d_2))) \le 4 + \log_2 \underline{e};$$

that is, those vectors in the span of the columns of $Z(d_2)$ which annihilate all of $X^{\mathrm{T}}(d_1)$ form a low-dimensional space with high probability.

The vectors

$$\{B^{1+d-d'}\hat{\mathbf{w}}_l \mid 1 \le l \le n\}$$

lie in this small subspace, so (taking suitable linear combinations) most of them are zero.

Termination. With high probability, the algorithm terminates by the time the iteration number t reaches $\underline{d}_n(m+n)/m + O(1)$, and at that time it has produced n *solutions* $f_l^{(t)}$, meaning that the corresponding $\hat{\mathbf{w}}_l$ satisfies $B^{1+d-d'}\hat{\mathbf{w}}_l = \mathbf{0}$. This is based on four statements, which are related to the previous lemmas by specifying some $l \le n$ and substituting

$$\lambda^{(\deg_{\mathrm{nom}} f_l^{(t)})} f_{l,\nu}^{(t)}(1/\lambda) = r_\nu(\lambda), \qquad 1 \le \nu \le n.$$

1. From Lemma 6.2 and Corollary 6.3, with high probability, if $f_l^{(t)}$ is a solution, then $\deg_{\mathrm{nom}} f_l^{(t)} \ge \underline{d}_n - O(1)$.

2. From Lemma 6.5 and Corollary 6.6, with high probability, if $f_l^{(t)}$ is not a solution, then $\deg_{\mathrm{nom}} f_l^{(t)} \ge t - \underline{d}_n n/m - O(1)$. (From Condition (C1), if $\deg_{\mathrm{nom}} f_l^{(t)}$ is smaller than that, then the corresponding $B^{1+d-d'}\hat{\mathbf{w}}_l$ is orthogonal to at least $\underline{d}_n n/m$ successive blocks $\mathbf{x}^{\mathrm{T}}B^i$, and from Corollary 6.6, this implies that it is actually a solution with high probability.)

3. The average degree is $tm/(m+n) + t_0 n/(m+n) = tm/(m+n) + O(1)$.

4. Condition (C2) prevents the computation of more than n solutions.

Suppose $t > \underline{d}_n(m+n)/m$ but only $n - \delta$ solutions have been obtained, with

$\delta > 0$. From statements 1 and 2, with high probability the average nominal degree is at least

$$\begin{aligned}
&\frac{1}{n+m}[(n-\delta)(\underline{d}_n - O(1)) + (m+\delta)(t - \underline{d}_n n/m - O(1))] \\
&\quad = \frac{1}{n+m}[n\underline{d}_n - \delta\underline{d}_n + mt + \delta t - \underline{d}_n n - \delta\underline{d}_n n/m] - O(1) \\
&\quad = t\frac{m}{m+n} + \frac{\delta}{m+n}\left(t - \underline{d}_n\frac{m+n}{m}\right) - O(1),
\end{aligned}$$

which will violate statement 3, if t exceeds $\underline{d}_n(m+n)/m$ by a sufficient amount to account for the $O(1)$ terms.

Summary. We have attempted to argue that the algorithm will produce n solutions within $\underline{d}_n(m+n)/m+O(1)$ iterations. The arguments are probabilistic in nature, and perhaps not totally satisfying; rather than constituting a full proof, they should serve as a guide to understanding why the algorithm works. The encouraging signs are that the algorithm does work in practice and that the arguments point towards a robustness that should handle even the difficult case of a few eigenvalues with large multiplicity.

7. Choice of parameters

Evidently, the optimal choice of n is some multiple of the word size of the machine, because the cost of applying B is proportional to $\lceil n/(\text{word size})\rceil$ times the number of nonzeros of B.

Increasing m can decrease the cost of the applications of B, but also increases the overhead (the terms proportional to N^2).

It is tempting to set $m < n$, for example, $m = 1$, $n = 32$, and hope that $2N/n$ iterations would still suffices. But then the information gathered in the first stage,

$$\{a^{(i)} = (\mathbf{x}^{\mathrm{T}}B^i\mathbf{y})^{\mathrm{T}},\ 0 \le i \le 2N/n\},$$

would be insufficient to find a solution to $B\mathbf{w} = \mathbf{0}$. Indeed, the information $\{a^{(i)},\ 0 \le i \le 2N/n\}$ is less than the information $\{\mathbf{x}^{\mathrm{T}}B^i,\ 0 \le i \le 2N/n\}$, which does not give sufficient information about B to calculate a member of its kernel. In the beginning of §3 we gave reason to believe that about $N/n + N/m$ iterations are necessary (if B is of nearly full rank), not the $2N/n$ one might hope for, and this makes it disadvantageous to select $m = 1$.

Let L denote the number of rows in $f^{(t)}$. If $L > m + n$, then the matrix $h^{(t_0)}$ has rank only $m + n$ over $GF(2)[\lambda]$, and eventually we will get useless rows of all zeros for some $f_l^{(t)}$. On the other hand, if $L < m + n$, then the average degree increases by m/L, and it will take longer to find solutions. This is why we fix $L = m + n$.

Fallout. Our considerations of the choice of parameters can extend back to the original Wiedemann algorithm, which works in any finite field. For large finite fields, because we cannot pack several field elements into a single word, we no longer need n to be a multiple of the word size; n can be any integer, and it may be 1 as in the original Wiedemann algorithm. But m can be higher, say 3 or 4, and thereby we will cut the number of applications of B from $3N$ to $[2 + (n/m)]N + O(1)$, with a corresponding increase in the $O(N^2)$ overhead.

This represents a fractional decrease in the overall cost. (We have $2N/n + N/m + O(1)$ applications of B to blocks of n vectors each, and for large fields each such application of B is as expensive as n applications of B to a single vector, expect for the possibility of parallelization.)

8. Further research

Gustavson and Yun [4] show how to speed up the Berlekamp-Massey algorithm from time $O(N^2)$ to time $O(N \log^2 N)$. A similar savings can probably be effected on the block version of the Berlekamp-Massey algorithm which we use here. Unfortunately, the dominate factor in the computational cost is the $3N/32$ applications of the sparse matrix B to a vector block, and this would not be affected.

Parallel versions. If one allows n and m to grow to, say, $\sqrt{N}$, one can take better advantage of massively parallel machines. This is true for arbitrary finite fields, not just $GF(2)$. Two apparent drawbacks to parallelization are the space requirement, proportional to $N(m+n)$, and the requirement to solve dense $m \times (m+n)$ systems of linear equations on each of N/n iterations during Phase 2.

Inhomogeneous equations. We developed this algorithm for homogeneous equations, because that is the case of interest for integer factorization. For the inhomogeneous system of equations $B\mathbf{w} = \mathbf{b}$, where $\mathbf{b}$ is a block of at most n vectors, variants that can be tried include the following:

1. Set the first few columns of $\mathbf{y}$ equal to $\mathbf{b}$, and calculate the rest of $\mathbf{y}$ as $B\mathbf{z}$. Then hope that in the equation

$$\mathbf{x}_\mu^{\mathrm{T}} B^{j-d'} \sum_{\nu,k} f_{l,\nu}^{(t,k)} B^{d'-k} \mathbf{y}_\nu = 0$$

the coefficients of $B^0 \mathbf{y}_\nu$ form an invertible matrix, allowing one to solve for $\mathbf{y}$ in terms of vectors in the image of B.

2. Augment B to have columns equal to $\mathbf{b}$.

3. Apply a random perturbation to B as described in [6].

9. Summary

We have proposed an adaptation of the Wiedemann algorithm to the solution of sparse systems of linear equations over $GF(2)$. Our approach has been influenced by the application to integer factorization, where we have a large sparse asymmetric matrix B with more columns than rows and we wish to find nontrivial linear combinations of the columns that vanish. For large problems this approach apparently has smaller space requirements than structured Gaussian elimination, and smaller time requirements than the original Wiedemann algorithm as proposed in [9]. We are hopeful that it will enable one to solve a larger problem than was previous possible.

Acknowledgments

The author has benefited from several discussions with Andrew Odlyzko. In particular, Odlyzko suggested that the Wiedemann algorithm might be converted to block form, after we had discussed the present author's similar work on a

block version of the Lanczos algorithm [2]; he also gave me sample matrices from integer factorization for experimentation. Grateful acknowledgment is also made to Victor Miller, Larry Carter, and Jane Cullum for help in developing the block Lanczos algorithm, which led to the present paper. We thank the referees for seeing the strength of the results through the weakness of the presentation, and for encouraging us to improve the latter. They also pointed out several technical stumbling blocks, along with solutions to many of them.

BIBLIOGRAPHY

1. E. Cahen, *Théorie des nombres.* I, Hermann, Paris, 1914, pp. 336–338.
2. D. Coppersmith, *Solving linear equations over* $GF(2)$, IBM Research Rep. RC 16997, July 1, 1991.
3. J. L. Dornstetter, *On the equivalence between Berlekamp's and Euclid's algorithms*, IEEE Trans. Inform. Theory **33** (1987), 428–431.
4. F. G. Gustavson and D. Y. Y. Yun, *Fast algorithms for rational Hermite approximation and solution of Toeplitz systems*, IEEE Trans. Circuits and Systems **26** (1979), 750–755.
5. D. E. Knuth, *The art of computer programming.* Vol. 2: *Seminumerical algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1981, exercise 4.6.1.19, pp. 419, 621.
6. E. Kaltofen and B. D. Saunders, *On Wiedemann's method of solving sparse linear systems*, Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (H. F. Mattson, T. Mora, and T. R. N. Rao, eds.), Lecture Notes in Comput. Sci., vol. 539, Springer-Verlag, Berlin and New York, 1991, pp. 29–38.
7. B. A. LaMacchia and A. M. Odlyzko, *Solving large sparse linear systems over finite fields*, Advances in Cryptology—CRYPTO '90 (A. J. Menezes and S. A. Vanstone, eds.), vol. 537, Springer-Verlag, Berlin and New York, 1991, pp. 109–133.
8. J. A. Reeds and N. J. A. Sloane, *Shift-register synthesis (modulo* m), SIAM J. Comput. **14** (1985), 505–513.
9. D. H. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inform. Theory **32** (1986), 54–62.

IBM RESEARCH, T. J. WATSON RESEARCH CENTER, YORKTOWN HEIGHTS, NEW YORK 10598
E-mail address: copper@watson.ibm.com