A Block Lanczos Algorithm for Finding Dependencies over GF(2)

Peter L. Montgomery

780 Las Colindas Road, San Rafael, CA 94903–2346 USA.
Work performed at Centrum voor Wiskunde en Informatica, Amsterdam.

Abstract. Some integer factorization algorithms require several vectors in the null space of a sparse $m \times n$ matrix over the field GF(2). We modify the Lanczos algorithm to produce a sequence of orthogonal subspaces of $\mathrm{GF}(2)^n$, each having dimension almost N, where N is the computer word size, by applying the given matrix and its transpose to N binary vectors at once. The resulting algorithm takes about $n/(N-0.76)$ iterations. It was applied to matrices larger than $10^6 \times 10^6$ during the factorizations of 105–digit and 119–digit numbers via the general number field sieve.

1 Introduction

Some integer factorization algorithms require several nonzero vectors $\mathbf{x} \in \mathrm{GF}(2)^n$ such that $\mathbf{Bx} = \mathbf{0}$, where $\mathbf{B}$ is a given $m \times n$ matrix over the field GF(2), usually very sparse and with $m < n$. These include the (obsolete) continued fraction method [9, p. 381], quadratic sieve (QS) [13, 14], and number field sieve [3, 11]. For example, when factoring an integer M, the QS method finds congruences

$$a_j^2 \equiv \prod_{i=1}^{m} p_i^{b_{ij}} \pmod{M} \qquad (1 \le j \le n) \ .$$

Here the p_i are primes (or -1) and the b_{ij} are exponents, mostly zero. QS then tries to find $S \subseteq \{1, 2, \cdots, n\}$ such that both sides of $\prod_{j\in S} a_j^2 \equiv \prod_{j \in S} \prod_{i=1}^{m} p_i^{b_{ij}} \pmod{M}$ are perfect squares. The left product is automatically a square, but the right product is a square only if all exponents are even, i.e., if $\sum_{j\in S} b_{ij} \equiv 0 \pmod{2}$ for $1 \le i \le m$. This is equivalent to $\mathbf{Bx} \equiv \mathbf{0} \pmod{2}$, where $\mathbf{B} = (b_{ij})$, $\mathbf{x} = (x_j)$, and where $x_j = 1$ if $j \in S$ and $x_j = 0$ if $j \notin S$.

Traditionally one has solved $\mathbf{Bx} = \mathbf{0}$ over GF(2) by a variation of Gaussian elimination [9, p. 425], requiring about mn bits. When $\mathbf{B}$ is sparse, one can first apply structured Gaussian elimination [10, §5], replacing $\mathbf{B}$ by a dense matrix with about one third as many rows and columns. A gigabyte does not hold a dense $10^5 \times 10^5$ matrix, whereas we want to solve systems with n around 10^6.

LaMacchia and Odlyzko [10] implement variants to the Lanczos and conjugate gradient methods, as previously suggested by Odlyzko et al. [7, 12]. These methods repeatedly apply a symmetric $n \times n$ matrix to a vector. They store only a few temporary vectors and the original matrix, thus relieving the storage problem if the matrix is sparse. The methods were developed for use on real matrices, but work over other fields unless one encounters a vector orthogonal to itself. To avoid self–orthogonal vectors, they work in an extension field of GF(2) rather than GF(2) itself.

Wiedemann [15] proposes another iterative algorithm. His algorithm applies a (not necessarily symmetric) $n \times n$ matrix $\mathbf{B}$ to a vector approximately $2n$ times, and constructs the minimal polynomial of $\mathbf{B}$. Using this minimal polynomial, one can find vectors in the null space of $\mathbf{B}$, if $\mathbf{B}$ is singular. The method likewise requires storage only for the matrix $\mathbf{B}$ and for a few temporary vectors.

The Lanczos, conjugate gradient, and Wiedemann algorithms all apply the given matrix (or its transpose) to $\mathcal{O}(n)$ vectors. On a binary computer with N bits per word, one can apply a matrix to N independent vectors over GF(2) at once, using the machine's bitwise operators. We would like to reduce the iteration count from $\mathcal{O}(n)$ to $\mathcal{O}(n/N)$, by accomplishing N times as much work per iteration. Even if we do N times as many operations per iteration after applying the matrix, the total cost of applying the matrix will drop N–fold.

Our variation of Lanczos achieves this objective by decomposing $\mathrm{GF}(2)^n$ into several subspaces of dimension almost N which are pairwise orthogonal with respect to the symmetric $n \times n$ matrix $\mathbf{A} = \mathbf{B}^{\mathrm{T}}\mathbf{B}$. The resulting algorithm takes about $n/(N - 0.76)$ iterations. Each iteration applies the matrices $\mathbf{B}$ and $\mathbf{B}^{\mathrm{T}}$ to an $n \times N$ matrix and does a few supplementary operations (i.e., inner products of two $n \times N$ matrices, multiplication of an $n \times N$ matrix by an $N \times N$ matrix, multiplication and inversion of $N \times N$ matrices).

Don Coppersmith published a block Wiedemann algorithm [6] which needs about $3n/N$ applications of $\mathbf{B}$. The present work was inspired by a comment [6, p. 334] that Coppersmith had previously found a block Lanczos algorithm, but before this author had seen [5]. When $N \geq 16$, the present algorithm and [5] each need about 2/3 as many sparse matrix operations as [6], even if $\mathbf{B}$ is not symmetric. This algorithm constructs the orthogonal vectors differently than [5] and needs about 40% as many supplementary operations as [5].

Except for Gaussian elimination, these algorithms are probabilistic. They make random choices, and may fail for some of these choices. I have tried the proposed method on about 50 matrices, and have not experienced failure.

The methods herein work over other finite fields if one can do independent field operations in parallel, analogous to the bitwise operators for GF(2).

2 Notations

Throughout this paper, $\mathbf{A}$ denotes a symmetric $n \times n$ matrix over a field K. Two vectors $\mathbf{v}$, $\mathbf{w} \in K^n$ are said to be $\mathbf{A}$–*orthogonal* if $\mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{w} = 0$. If $\mathcal{V}$ and $\mathcal{W}$ are subspaces (or subsets) of K^n, then we define the block operations

$$\begin{aligned} \mathcal{V} + \mathcal{W} &= \{\mathbf{v} + \mathbf{w} : \mathbf{v} \in \mathcal{V} \text{ and } \mathbf{w} \in \mathcal{W}\} \ , \\ \mathbf{A}\mathcal{V} &= \{\mathbf{A}\mathbf{v} : \mathbf{v} \in \mathcal{V}\} \ , \\ \mathcal{V}^{\mathrm{T}}\mathcal{W} &= \{\mathbf{v}^{\mathrm{T}}\mathbf{w} : \mathbf{v} \in \mathcal{V} \text{ and } \mathbf{w} \in \mathcal{W}\} \ . \end{aligned}$$

Two subspaces $\mathcal{V}$ and $\mathcal{W}$ of K^n are said to be $\mathbf{A}$–orthogonal if $\mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{w} = 0$ for all $\mathbf{v} \in \mathcal{V}$ and $\mathbf{w} \in \mathcal{W}$; this is equivalent to $\mathcal{V}^{\mathrm{T}}\mathbf{A}\mathcal{W} = \{0\}$.

If $\mathbf{V}$ is an $n_1 \times n_2$ matrix, then $\langle\mathbf{V}\rangle$ denotes the subspace of K^{n_1} generated by the column vectors of $\mathbf{V}$.

If $\mathcal{W}$ is a subspace of K^n, then $\mathcal{O}(\mathcal{W})$ represents a vector in $\mathcal{W}$ or a matrix with column vectors in $\mathcal{W}$. It satisfies $\mathcal{O}(\mathcal{V}) + \mathcal{O}(\mathcal{W}) = \mathcal{O}(\mathcal{V} + \mathcal{W})$. If $\mathbf{M}$ is a matrix of suitable size, then we can replace $\mathcal{O}(\mathcal{W})\mathbf{M}$ by $\mathcal{O}(\mathcal{W})$, but cannot similarly simplify $\mathbf{M}\mathcal{O}(\mathcal{W})$.

We denote the number of bits per computer word by N.

The $k \times k$ identity matrix is denoted by $\mathbf{I}_k$.

3 Standard Lanczos

Suppose $\mathbf{A}$ is a symmetric positive definite $n \times n$ matrix over the field $K = \mathbb{R}$. If $\mathbf{b} \in \mathbb{R}^n$, then the standard Lanczos algorithm solves $\mathbf{A}\mathbf{x} = \mathbf{b}$ by iterating

$$\begin{aligned} \mathbf{w}_0 &= \mathbf{b} \ , \\ \mathbf{w}_i &= \mathbf{A}\mathbf{w}_{i-1} - \sum_{j=0}^{i-1} c_{ij}\mathbf{w}_j \qquad (i > 0), \quad \text{where} \quad c_{ij} = \frac{\mathbf{w}_j^{\mathrm{T}}\mathbf{A}^2\mathbf{w}_{i-1}}{\mathbf{w}_j^{\mathrm{T}}\mathbf{A}\mathbf{w}_j} \ , \end{aligned} \tag{1}$$

until $\mathbf{w}_i = \mathbf{0}$. Using induction on $\max(i, j)$ (and the symmetry of $\mathbf{A}$), we verify

$$\mathbf{w}_j^{\mathrm{T}}\mathbf{A}\mathbf{w}_i = 0 \qquad (i \neq j) \ . \tag{2}$$

If $i > n$, then the vectors $\mathbf{w}_0, \mathbf{w}_1, \cdots, \mathbf{w}_i$ are linearly dependent. Suppose $\sum_{j=0}^{i} a_j\mathbf{w}_j = \mathbf{0}$ where $a_i \neq 0$. Pre–multiply by $\mathbf{w}_i^{\mathrm{T}}\mathbf{A}$ to find $a_i\mathbf{w}_i^{\mathrm{T}}\mathbf{A}\mathbf{w}_i = 0$. By positive definiteness, $\mathbf{w}_i = \mathbf{0}$. Let m denote the first value of i such that $\mathbf{w}_i = \mathbf{0}$.

Define

$$\mathbf{x} = \sum_{j=0}^{m-1} \frac{\mathbf{w}_j^{\mathrm{T}}\mathbf{b}}{\mathbf{w}_j^{\mathrm{T}}\mathbf{A}\mathbf{w}_j}\mathbf{w}_j \ . \tag{3}$$

Then $\mathbf{A}\mathbf{x} - \mathbf{b} \in \langle\mathbf{A}\mathbf{w}_0, \ \mathbf{A}\mathbf{w}_1, \cdots, \mathbf{A}\mathbf{w}_{m-1}, \mathbf{b}\rangle \subseteq \langle\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{m-1}\rangle$ by (1). By construction, $\mathbf{w}_j^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{w}_j^{\mathrm{T}}\mathbf{b}$ for $0 \le j \le m-1$. Hence $(\mathbf{A}\mathbf{x} - \mathbf{b})^{\mathrm{T}}(\mathbf{A}\mathbf{x} - \mathbf{b}) = 0$ and $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Formula (1) appears to require adding suitable multiples of all earlier $\mathbf{w}_j$ when computing $\mathbf{w}_i$. However, the terms vanish when $j < i - 2$, since

$$\begin{aligned} \mathbf{w}_j^{\mathrm{T}}\mathbf{A}^2\mathbf{w}_{i-1} &= (\mathbf{A}\mathbf{w}_j)^{\mathrm{T}}\mathbf{A}\mathbf{w}_{i-1} \\ &= \left(\mathbf{w}_{j+1} + \sum_{k=0}^{j} c_{j+1,k}\mathbf{w}_k\right)^{\mathrm{T}} \mathbf{A}\mathbf{w}_{i-1} = 0 \qquad (j < i - 2) \end{aligned} \tag{4}$$

by (1) and (2). Hence (1) simplifies to

$$\mathbf{w}_i = \mathbf{A}\mathbf{w}_{i-1} - c_{i,i-1}\mathbf{w}_{i-1} - c_{i,i-2}\mathbf{w}_{i-2} \qquad (i \geq 2) \ . \tag{5}$$

The Lanczos algorithm requires at most n iterations. Each iteration of (5) applies $\mathbf{A}$ to one vector $\mathbf{w}_{i-1}$. The computations

$$c_{i,i-1} = \frac{(\mathbf{A}\mathbf{w}_{i-1})^{\mathrm{T}}(\mathbf{A}\mathbf{w}_{i-1})}{\mathbf{w}_{i-1}^{\mathrm{T}}(\mathbf{A}\mathbf{w}_{i-1})} \qquad \text{and} \qquad c_{i,i-2} = \frac{(\mathbf{A}\mathbf{w}_{i-2})^{\mathrm{T}}(\mathbf{A}\mathbf{w}_{i-1})}{\mathbf{w}_{i-2}^{\mathrm{T}}(\mathbf{A}\mathbf{w}_{i-2})}$$

can be done with three inner products if we assume (by induction) that $\mathbf{w}_{i-1}$, $\mathbf{w}_{i-2}$, $\mathbf{A}\mathbf{w}_{i-1}$, $\mathbf{A}\mathbf{w}_{i-2}$, and $\mathbf{w}_{i-2}^{\mathrm{T}}\mathbf{A}\mathbf{w}_{i-2}$ are known. Another inner product is used for $\mathbf{w}_i^{\mathrm{T}}\mathbf{b}$ while updating the partial sum of $\mathbf{x}$ in (3). If $\mathbf{A}$ averages d nonzero entries per row or column, then the cost per iteration is $\mathcal{O}(dn)$ to multiply by $\mathbf{A}$ and $\mathcal{O}(n)$ for the other vector arithmetic. The total time is $\mathcal{O}(dn^2) + \mathcal{O}(n^2)$.

The storage requirement is nominal: $\mathcal{O}(1)$ temporary vectors of length n, plus the matrix $\mathbf{A}$ itself.

4 Lanczos over Other Algebraic Domains

The Lanczos iterations (1), (3), and (5) use only rational arithmetic operations (no square roots or transcendental functions). If there is no round–off error, then the final vector $\mathbf{x}$ is an exact solution, not an approximation. As Odlyzko et al. [7] [10, §3] [12] observe, this makes Lanczos usable in other algebraic domains, although the method was discovered by numerical analysts.

Let $\mathbf{A}$ be a symmetric $n \times n$ matrix over a field K. Assume that $\mathbf{w}_i \neq \mathbf{0}$ for $0 \leq i < m$ and that $\mathbf{w}_m = \mathbf{0}$. When $K = \mathbb{R}$ and $\mathbf{A}$ is positive definite, the Lanczos vectors $\{\mathbf{w}_i\}_{i=0}^{m-1}$ satisfy

$$\begin{aligned} \mathbf{w}_i^{\mathrm{T}}\mathbf{A}\mathbf{w}_i &\neq 0 && (0 \leq i < m) \ , \\ \mathbf{w}_j^{\mathrm{T}}\mathbf{A}\mathbf{w}_i &= 0 && (i \neq j) \ , \\ \mathbf{A}\mathcal{W} &\subseteq \mathcal{W}, && \text{where} \quad \mathcal{W} = \langle \mathbf{w}_0, \mathbf{w}_1, \cdots, \mathbf{w}_{m-1} \rangle \ . \end{aligned} \tag{6}$$

If $\mathbf{b} \in \mathcal{W}$ and $\mathbf{x}$ is defined by (3), then we claim (6) implies $\mathbf{A}\mathbf{x} = \mathbf{b}$, without further assumptions about the field K (the proof in §3 assumes $K = \mathbb{R}$). As before, $\mathcal{W}^{\mathrm{T}}(\mathbf{A}\mathbf{x} - \mathbf{b}) = \{0\}$ by construction of $\mathbf{x}$. Also

$$\mathcal{W}^{\mathrm{T}}\mathbf{A}(\mathbf{A}\mathbf{x} - \mathbf{b}) = (\mathbf{A}\mathcal{W})^{\mathrm{T}}(\mathbf{A}\mathbf{x} - \mathbf{b}) \subseteq \mathcal{W}^{\mathrm{T}}(\mathbf{A}\mathbf{x} - \mathbf{b}) = \{0\} \ .$$

We know that $\mathbf{A}\mathbf{x} - \mathbf{b} \in \mathcal{W}$, say $\mathbf{A}\mathbf{x} - \mathbf{b} = \sum_{i=0}^{m-1} c_i \mathbf{w}_i$. Pre–multiply by $\mathbf{w}_i^{\mathrm{T}}\mathbf{A}$ to conclude $c_i = 0$ since $\mathbf{w}_i^{\mathrm{T}}\mathbf{A}\mathbf{w}_i$ is assumed nonzero. Since i is arbitrary, $\mathbf{A}\mathbf{x} = \mathbf{b}$.

When $K \neq \mathbb{R}$, if $\mathbf{w}_i$ is computed by (1) or (5), then the requirement $\mathbf{w}_i^{\mathrm{T}}\mathbf{A}\mathbf{w}_i \neq 0$ if $\mathbf{w}_i \neq 0$ in (6) may fail. If $|K| \gg n$, then we may be able to tolerate this risk [7, 10], esp. if we can rerun our problem with slightly different data whenever it fails.

We want to apply Lanczos to the field GF(2), over which about half of all vectors are $\mathbf{A}$–orthogonal to themselves, and need to vary our methods. The field GF(2) has its advantages, since one can apply the matrix $\mathbf{A}$ to N different vectors in $\mathrm{GF}(2)^n$ at once, using bitwise operators. We generalize (6) to allow a sequence of subspaces in place of the vectors $\{\mathbf{w}_i\}$, and adapt the Lanczos iteration (1) to the new framework.

5 Sequences of Orthogonal Subspaces

Let $\mathbf{A}$ be a symmetric matrix over a field K. Block Lanczos algorithms [8, Chapter 7] modify (6) to produce a sequence of subspaces $\{\mathcal{W}_i\}_{i=0}^{m-1}$ of K^n which are pairwise $\mathbf{A}$–orthogonal. The condition $\mathbf{w}_i^{\mathrm{T}}\mathbf{A}\mathbf{w}_i \neq 0$ in (6) is replaced by a requirement that no nonzero vector in $\mathcal{W}_i$ be $\mathbf{A}$–orthogonal to all of $\mathcal{W}_i$.

Definition 1. A subspace $\mathcal{W} \subseteq K^n$ is said to be $\mathbf{A}$–*invertible* if it has a basis $\mathbf{W}$ of column vectors such that $\mathbf{W}^{\mathrm{T}}\mathbf{A}\mathbf{W}$ is invertible.

The property of being $\mathbf{A}$–invertible is independent of the choice of basis, since any two bases for $\mathcal{W}$ are related by an invertible transformation.

If $\mathcal{W}$ is $\mathbf{A}$–invertible, then any $\mathbf{u} \in K^n$ can be uniquely written as $\mathbf{v} + \mathbf{w}$ where $\mathbf{w} \in \mathcal{W}$ and $\mathcal{W}^{\mathrm{T}}\mathbf{A}\mathbf{v} = \{0\}$. Indeed, if the columns of $\mathbf{W}$ are a basis for $\mathcal{W}$, then $\mathbf{w} = \mathbf{W}\left(\mathbf{W}^{\mathrm{T}}\mathbf{A}\mathbf{W}\right)^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{A}\mathbf{u}$.

The generalization of (6) to subspaces is

$$\begin{array}{rll} \mathcal{W}_i & \text{is } \mathbf{A}\text{–invertible} \ , & \\ \mathcal{W}_j^{\mathrm{T}}\mathbf{A}\mathcal{W}_i = \{0\} & (i \neq j) \ , & \\ \mathbf{A}\mathcal{W} \subseteq \mathcal{W}, & \text{where} \quad \mathcal{W} = \mathcal{W}_0 + \mathcal{W}_1 + \cdots + \mathcal{W}_{m-1} \ . & \end{array} \tag{7}$$

Assume (7). Given $\mathbf{b} \in \mathcal{W}$, we can construct an $\mathbf{x} \in \mathcal{W}$ such that $\mathbf{A}\mathbf{x} = \mathbf{b}$. Let $\mathbf{x} = \sum_{j=0}^{m-1} \mathbf{w}_j$, where $\mathbf{w}_j \in \mathcal{W}_j$ is chosen so that $\mathbf{A}\mathbf{w}_j - \mathbf{b}$ is orthogonal to all of $\mathcal{W}_j$. If the columns of $\mathbf{W}_j$ form a basis for $\mathcal{W}_j$, then

$$\mathbf{x} = \sum_{j=0}^{m-1} \mathbf{W}_j(\mathbf{W}_j^{\mathrm{T}}\mathbf{A}\mathbf{W}_j)^{-1}\mathbf{W}_j^{\mathrm{T}}\mathbf{b} \tag{8}$$

generalizes (3).

Fix $N > 0$. At step i, we will have an $n \times N$ matrix $\mathbf{V}_i$ which is $\mathbf{A}$–orthogonal to all earlier $\mathbf{W}_j$. The initial $\mathbf{V}_0$ is arbitrary. We select $\mathbf{W}_i$ using as many columns of $\mathbf{V}_i$ as we can, subject to the requirement that $\mathbf{W}_i$ be $\mathbf{A}$–invertible. More precisely, we try to replace the Lanczos iterations (1) by

$$\begin{aligned} \mathbf{W}_i &= \mathbf{V}_i\mathbf{S}_i \ , \\ \mathbf{V}_{i+1} &= \mathbf{A}\mathbf{W}_i\mathbf{S}_i^{\mathrm{T}} + \mathbf{V}_i - \sum_{j=0}^{i} \mathbf{W}_j\mathbf{C}_{i+1,j} \qquad (i \geq 0) \ , \\ \mathcal{W}_i &= \langle \mathbf{W}_i \rangle \ . \end{aligned} \tag{9}$$

Stop iterating if $\mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i = \mathbf{0}$, say for $i = m$.

Here $\mathbf{S}_i$ is an $N \times N_i$ projection matrix chosen so that $\mathbf{W}_i^{\mathrm{T}}\mathbf{A}\mathbf{W}_i$ is invertible while making $N_i \leq N$ as large as possible. The matrix $\mathbf{S}_i$ should be zero except for exactly one 1 per column and at most one 1 per row. These ensure that $\mathbf{S}_i^{\mathrm{T}}\mathbf{S}_i = \mathbf{I}_{N_i}$ and that $\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}}$ is a submatrix of $\mathbf{I}_N$ reflecting the vectors selected from $\mathbf{V}_i$. For example, if $N = 3$, then $\mathbf{S}_i = \begin{pmatrix} 0\,1\,0 \\ 0\,0\,1 \end{pmatrix}^{\mathrm{T}}$ selects the second and third columns of $\mathbf{V}_i$ for inclusion in $\mathbf{W}_i$.

Formula (9) for $\mathbf{V}_{i+1}$ tries to generalize (1) while ensuring $\mathbf{W}_j^{\mathrm{T}}\mathbf{A}\mathbf{V}_{i+1} = \{\mathbf{0}\}$ for $j \le i$ if the earlier $\mathbf{W}_j$ exhibit the desired $\mathbf{A}$–orthogonality. We use

$$\mathbf{C}_{i+1,j} = (\mathbf{W}_j^{\mathrm{T}}\mathbf{A}\mathbf{W}_j)^{-1}\mathbf{W}_j^{\mathrm{T}}\mathbf{A}(\mathbf{A}\mathbf{W}_i\mathbf{S}_i^{\mathrm{T}} + \mathbf{V}_i) \ . \tag{10}$$

The terms $\mathbf{V}_i - \mathbf{W}_i\mathbf{C}_{i+1,i}$ in (9) select any columns of $\mathbf{V}_i$ not used in $\mathbf{W}_i$; those columns are known to be $\mathbf{A}$–orthogonal to $\mathbf{W}_0$ through $\mathbf{W}_{i-1}$, and the choice of $\mathbf{C}_{i+1,i}$ adjusts them so they are $\mathbf{A}$–orthogonal to $\mathbf{W}_i$ as well. Without the $\mathbf{V}_i$ term, $\mathrm{rank}(\mathbf{V}_{i+1})$ would be bounded by $\mathrm{rank}(\mathbf{A}\mathbf{W}_i\mathbf{S}_i^{\mathrm{T}}) \le \mathrm{rank}(\mathbf{V}_i)$, and would soon drop to zero.

Theorem 2. *Equations (9) and (10) imply (7) if* $\mathbf{V}_m = \mathbf{0}$. *Furthermore,*

$$\mathbf{W}_j^{\mathrm{T}}\mathbf{A}\mathbf{V}_i = \mathbf{0} \qquad (0 \le j < i \le m) \ . \tag{11}$$

Proof. The selection of $\mathbf{S}_i$ ensures $\mathcal{W}_i = \langle\mathbf{W}_i\rangle$ is $\mathbf{A}$–invertible. The equation $\mathbf{W}_j^{\mathrm{T}}\mathbf{A}\mathbf{V}_i = \mathbf{0}$ implies $\mathbf{W}_j^{\mathrm{T}}\mathbf{A}\mathbf{W}_i = \mathbf{0}$ since $\mathbf{W}_i = \mathbf{V}_i\mathbf{S}_i$. It also implies $\mathbf{W}_i^{\mathrm{T}}\mathbf{A}\mathbf{W}_j = \mathbf{0}$ since $\mathbf{A}$ is symmetric.

We prove (11) by induction on i. Let $0 \le k < m$ and assume (11) holds for $0 \le j < i \le k$. This assumption is vacuously true when $k = 0$. If $0 \le j \le k$, then

$$\begin{aligned}\mathbf{W}_j^{\mathrm{T}}\mathbf{A}\mathbf{V}_{k+1} &= \mathbf{W}_j^{\mathrm{T}}\mathbf{A}(\mathbf{A}\mathbf{W}_k\mathbf{S}_k^{\mathrm{T}} + \mathbf{V}_k) - \sum_{i=0}^{k}\mathbf{W}_j^{\mathrm{T}}\mathbf{A}\mathbf{W}_i\mathbf{C}_{k+1,i} \\ &= \mathbf{W}_j^{\mathrm{T}}\mathbf{A}(\mathbf{A}\mathbf{W}_k\mathbf{S}_k^{\mathrm{T}} + \mathbf{V}_k) - \mathbf{W}_j^{\mathrm{T}}\mathbf{A}\mathbf{W}_j\mathbf{C}_{k+1,j} = \mathbf{0}\end{aligned}$$

by induction and the choice (10) of $\mathbf{C}_{k+1,j}$.

A corollary to (11) is $\mathcal{W}_j^{\mathrm{T}}\mathbf{A}\mathcal{W}_i = \{0\}$ if $i \ne j$.

Post-multiply the defining equation (9) for $\mathbf{V}_{i+1}$ by $\mathbf{S}_i$:

$$\begin{aligned}\mathbf{V}_{i+1}\mathbf{S}_i &= \mathbf{A}\mathbf{W}_i\mathbf{S}_i^{\mathrm{T}}\mathbf{S}_i + \mathbf{V}_i\mathbf{S}_i - \sum_{j=0}^{i}\mathbf{W}_j\mathbf{C}_{i+1,j}\mathbf{S}_i \\ &= \mathbf{A}\mathbf{W}_i + \mathbf{W}_i - \sum_{j=0}^{i}\mathbf{W}_j\mathbf{C}_{i+1,j}\mathbf{S}_i \ .\end{aligned}$$

This and (9) give

$$\begin{aligned}\mathbf{A}\mathbf{W}_i &= \mathbf{V}_{i+1}\mathbf{S}_i - \mathbf{W}_i + \sum_{j=0}^{i}\mathbf{W}_j\mathbf{C}_{i+1,j}\mathbf{S}_i = \mathbf{V}_{i+1}\mathbf{S}_i + \mathcal{O}(\mathcal{W}) \ , \\ \mathbf{V}_i &= \mathbf{V}_{i+1} - \mathbf{A}\mathbf{W}_i\mathbf{S}_i^{\mathrm{T}} + \sum_{j=0}^{i}\mathbf{W}_j\mathbf{C}_{i+1,j} = \mathbf{V}_{i+1} - \mathbf{A}\mathbf{W}_i\mathbf{S}_i^{\mathrm{T}} + \mathcal{O}(\mathcal{W}) \ .\end{aligned} \tag{12}$$

By hypothesis, $\mathbf{V}_m = \mathbf{0} = \mathcal{O}(\mathcal{W})$. By backwards induction and (12), $\mathbf{AW}_i = \mathcal{O}(\mathcal{W})$ and $\mathbf{V}_i = \mathcal{O}(\mathcal{W})$ for $0 \le i \le m-1$. Hence $\mathbf{A}\mathcal{W} \subseteq \mathcal{W}$. $\square$

The subspaces $\mathcal{W}_i$ generated by (9) have dimension at most N. This is immediate from (9) since $\mathcal{W}_i = \langle \mathbf{V}_i \mathbf{S}_i \rangle$ and $\mathbf{V}_i$ is an $n \times N$ matrix.

6 Simplifying the Block Lanczos Recurrence

In standard Lanczos, we simplified (1) to (5). The computation of $\mathbf{w}_i$ from $\mathbf{Aw}_{i-1}$ requires adjustments only by scalar multiples of $\mathbf{w}_{i-1}$ and $\mathbf{w}_{i-2}$, not by $\mathbf{w}_j$ for $j \le i-3$.

We would like to similarly optimize the computation of $\mathbf{V}_{i+1}$ in (9), using the invariant (11). We have some freedom in the choice of $\mathbf{S}_i$ since $\langle \mathbf{V}_i \rangle$ may have multiple bases.

If $j < i$, then the term $\mathbf{W}_j^{\mathrm{T}} \mathbf{AV}_i$ in (10) vanishes by (11). We attempt to simplify $\mathbf{W}_j^{\mathrm{T}} \mathbf{A}^2 \mathbf{W}_i$, using (9), (11), and the methods of (4):

$$\begin{aligned}
\mathbf{W}_j^{\mathrm{T}} \mathbf{A}^2 \mathbf{W}_i &= \left(\mathbf{S}_j^{\mathrm{T}} \mathbf{S}_j\right) \left(\mathbf{W}_j^{\mathrm{T}} \mathbf{A}^2 \mathbf{W}_i\right) = \mathbf{S}_j^{\mathrm{T}} (\mathbf{AW}_j \mathbf{S}_j^{\mathrm{T}})^{\mathrm{T}} \mathbf{AW}_i \\
&= \mathbf{S}_j^{\mathrm{T}} \left(\mathbf{V}_{j+1} - \mathbf{V}_j + \mathcal{O}(\mathcal{W}_0 + \cdots + \mathcal{W}_j)\right)^{\mathrm{T}} \mathbf{AW}_i \qquad (j < i) \\
&= \mathbf{S}_j^{\mathrm{T}} \mathbf{V}_{j+1}^{\mathrm{T}} \mathbf{AW}_i - \mathbf{W}_j^{\mathrm{T}} \mathbf{AW}_i = \mathbf{S}_j^{\mathrm{T}} \mathbf{V}_{j+1}^{\mathrm{T}} \mathbf{AW}_i \ .
\end{aligned} \tag{13}$$

If $\mathbf{S}_{j+1} = \mathbf{I}_N$ (so that $\mathbf{V}_{j+1} = \mathbf{W}_{j+1}$) and if $j < i-1$, then (13) vanishes since $\mathbf{W}_{j+1}^{\mathrm{T}} \mathbf{AW}_i = \mathbf{0}$.

In other cases equation (13) does not similarly simplify. We may be unable to force $\mathbf{C}_{i+1,j} = \mathbf{0}$ for $j = i-2$, but do want to force $\mathbf{C}_{i+1,j} = \mathbf{0}$ for $j \le i-3$. Then the recurrence (9) will simplify to

$$\mathbf{V}_{i+1} = \mathbf{AW}_i \mathbf{S}_i^{\mathrm{T}} + \mathbf{V}_i - \mathbf{W}_i \mathbf{C}_{i+1,i} - \mathbf{W}_{i-1} \mathbf{C}_{i+1,i-1} - \mathbf{W}_{i-2} \mathbf{C}_{i+1,i-2} \quad (i \ge 2). \tag{14}$$

Although (14) has more terms than (5), the time per iteration and the temporary storage requirements will remain acceptable using (14). Equation (14) remains valid for $i = 0$ and $i = 1$ if we define $\mathbf{V}_j = \mathbf{0}$ and $\mathbf{W}_j = \mathbf{0}$ for $j < 0$.

To achieve (14), we require that (13) vanish whenever $j \le i-3$. That is, we require $\mathbf{V}_{j+1}$ to be $\mathbf{A}$–orthogonal to $\mathbf{W}_{j+3}$ through $\mathbf{W}_m$. We achieve this by requiring that all vectors in $\mathbf{V}_{j+1}$ be used either in $\mathbf{W}_{j+1}$ or in $\mathbf{W}_{j+2}$. More precisely, we require

$$\langle \mathbf{V}_{j+1} \rangle \subseteq \mathcal{W}_0 + \mathcal{W}_1 + \cdots + \mathcal{W}_{j+2} \qquad (j \ge -1) \ . \tag{15}$$

Assuming (15), we try to simplify the matrix equation (14). Denote

$$\mathbf{W}_i^{\mathrm{inv}} = \mathbf{S}_i \left(\mathbf{W}_i^{\mathrm{T}} \mathbf{AW}_i\right)^{-1} \mathbf{S}_i^{\mathrm{T}} = \mathbf{S}_i \left(\mathbf{S}_i^{\mathrm{T}} \mathbf{V}_i^{\mathrm{T}} \mathbf{AV}_i \mathbf{S}_i\right)^{-1} \mathbf{S}_i^{\mathrm{T}} \ . \tag{16}$$

Each $\mathbf{W}_i^{\text{inv}}$ is a symmetric $N \times N$ matrix. Eliminate all references to $\mathbf{W}_i = \mathbf{V}_i \mathbf{S}_i$:

$$\begin{aligned}
\mathbf{V}_{i+1} &= \mathbf{A}\mathbf{V}_i\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}} + \mathbf{V}_i - \mathbf{V}_i\mathbf{S}_i\mathbf{C}_{i+1,i} - \mathbf{V}_{i-1}\mathbf{S}_{i-1}\mathbf{C}_{i+1,i-1} - \mathbf{V}_{i-2}\mathbf{S}_{i-2}\mathbf{C}_{i+1,i-2} \\
&= \mathbf{A}\mathbf{V}_i\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}} + \mathbf{V}_i - \mathbf{V}_i\mathbf{W}_i^{\text{inv}}\mathbf{V}_i^{\mathrm{T}}\mathbf{A}(\mathbf{A}\mathbf{V}_i\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}} + \mathbf{V}_i) \\
&\quad - \mathbf{V}_{i-1}\mathbf{W}_{i-1}^{\text{inv}}\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_i\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}} - \mathbf{V}_{i-2}\mathbf{W}_{i-2}^{\text{inv}}\mathbf{V}_{i-2}^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_i\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}} \quad .
\end{aligned} \tag{17}$$

Equation (17) appears to require four inner products: $\mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i$, $\mathbf{V}_i^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_i$, $\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_i$, and $\mathbf{V}_{i-2}^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_i$. We can express the latter two inner products in terms of the first two, using (10), (11), (12), and (14):

$$\begin{aligned}
\mathbf{S}_{i-1}^{\mathrm{T}}\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_i &= (\mathbf{A}\mathbf{W}_{i-1})^{\mathrm{T}}\mathbf{A}\mathbf{V}_i = \left(\mathbf{V}_i\mathbf{S}_{i-1} + \mathcal{O}(\mathcal{W}_0 + \cdots + \mathcal{W}_{i-1})\right)^{\mathrm{T}}\mathbf{A}\mathbf{V}_i \\
&= \mathbf{S}_{i-1}^{\mathrm{T}}\mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i \quad ,
\end{aligned}$$

$$\begin{aligned}
\mathbf{S}_{i-2}^{\mathrm{T}}\mathbf{V}_{i-2}^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_i &= (\mathbf{A}\mathbf{W}_{i-2})^{\mathrm{T}}\mathbf{A}\mathbf{V}_i = \left(\mathbf{V}_{i-1}\mathbf{S}_{i-2} + \mathcal{O}(\mathcal{W}_0 + \cdots + \mathcal{W}_{i-2})\right)^{\mathrm{T}}\mathbf{A}\mathbf{V}_i \\
&= \mathbf{S}_{i-2}^{\mathrm{T}}\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}\mathbf{V}_i \\
&= \mathbf{S}_{i-2}^{\mathrm{T}}\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}\Big(\mathbf{A}\mathbf{W}_{i-1}\mathbf{S}_{i-1}^{\mathrm{T}} + \mathbf{V}_{i-1} \\
&\qquad - \mathbf{W}_{i-1}\mathbf{C}_{i,i-1} + \mathcal{O}(\mathcal{W}_{i-2} + \mathcal{W}_{i-3})\Big) \\
&= \mathbf{S}_{i-2}^{\mathrm{T}}\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}\left(\mathbf{I}_n - \mathbf{V}_{i-1}\mathbf{W}_{i-1}^{\text{inv}}\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}\right)\left(\mathbf{A}\mathbf{W}_{i-1}\mathbf{S}_{i-1}^{\mathrm{T}} + \mathbf{V}_{i-1}\right) \\
&= \mathbf{S}_{i-2}^{\mathrm{T}}\left(\mathbf{I}_N - \mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}\mathbf{V}_{i-1}\mathbf{W}_{i-1}^{\text{inv}}\right) \\
&\qquad \left(\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_{i-1}\mathbf{S}_{i-1}\mathbf{S}_{i-1}^{\mathrm{T}} + \mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}\mathbf{V}_{i-1}\right) \quad .
\end{aligned}$$

Hence (17) simplifies to

$$\mathbf{V}_{i+1} = \mathbf{A}\mathbf{V}_i\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}} + \mathbf{V}_i\mathbf{D}_{i+1} + \mathbf{V}_{i-1}\mathbf{E}_{i+1} + \mathbf{V}_{i-2}\mathbf{F}_{i+1} \tag{18}$$

for $i \geq 0$, where

$$\begin{aligned}
\mathbf{D}_{i+1} &= \mathbf{I}_N - \mathbf{W}_i^{\text{inv}}\left(\mathbf{V}_i^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_i\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}} + \mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i\right) \quad , \\
\mathbf{E}_{i+1} &= -\mathbf{W}_{i-1}^{\text{inv}}\mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}} \quad , \\
\mathbf{F}_{i+1} &= -\mathbf{W}_{i-2}^{\text{inv}}\left(\mathbf{I}_N - \mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}\mathbf{V}_{i-1}\mathbf{W}_{i-1}^{\text{inv}}\right) \\
&\qquad \left(\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_{i-1}\mathbf{S}_{i-1}\mathbf{S}_{i-1}^{\mathrm{T}} + \mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}\mathbf{V}_{i-1}\right)\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}} \quad .
\end{aligned} \tag{19}$$

We define $\mathbf{W}_j^{\text{inv}}$ and $\mathbf{V}_j$ to be $\mathbf{0}$ and $\mathbf{S}_j$ to be $\mathbf{I}_N$ for $j < 0$.

7 Finding Dependencies over GF(2)

Let $\mathbf{B}$ be an $n_1 \times n_2$ matrix over the field $K = \mathrm{GF}(2)$, where $n_1 < n_2$. Then there exist at least $n_2 - n_1$ linearly independent vectors $\mathbf{x} \in K^{n_2}$ such that $\mathbf{B}\mathbf{x} = \mathbf{0}$. Some integer factorization algorithms [9, pp. 380ff.][11] require finding several (perhaps ten) such $\mathbf{x}$. In practice, the matrix $\mathbf{B}$ is large but very sparse.

The Lanczos algorithm requires that the matrix be symmetric. We let $n = n_2$ and $\mathbf{A} = \mathbf{B}^{\mathrm{T}}\mathbf{B}$. This $\mathbf{A}$ is symmetric, and any solution of $\mathbf{B}\mathbf{x} = \mathbf{0}$ will satisfy $\mathbf{A}\mathbf{x} = \mathbf{0}$ (although the converse need not be true if the rank of $\mathbf{A}$ is less than n_1).

Let N denote the computer word size, typically 32 or 64. Select a random $n \times N$ matrix $\mathbf{Y}$ over GF(2), compute $\mathbf{AY}$, and attempt to find an $n \times N$ matrix $\mathbf{X}$ such that $\mathbf{AX} = \mathbf{AY}$. If we succeed, then the column vectors of $\mathbf{X} - \mathbf{Y}$ will be random vectors in the null space of $\mathbf{A}$. If the rank of $\mathbf{A}$ is at least $\operatorname{rank}(\mathbf{B}) - N + 1$, then one can combine columns of $\mathbf{X} - \mathbf{Y}$ to find vectors in the null space of $\mathbf{B}$.

After selecting $\mathbf{Y}$, we initialize $\mathbf{V}_0 = \mathbf{AY}$ and proceed through the Lanczos iterations (18) until some $\mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i = \mathbf{0}$, say for $i = m$. Compute

$$\mathbf{X} = \sum_{i=0}^{m-1} \mathbf{W}_i \left(\mathbf{W}_i^{\mathrm{T}}\mathbf{A}\mathbf{W}_i\right)^{-1} \mathbf{W}_i^{\mathrm{T}}\mathbf{V}_0 = \sum_{i=0}^{m-1} \mathbf{V}_i \mathbf{W}_i^{\mathrm{inv}} \mathbf{V}_i^{\mathrm{T}}\mathbf{V}_0 \ . \tag{20}$$

Denote $\mathcal{W} = \mathcal{W}_0 + \cdots + \mathcal{W}_{m-1}$ and $\mathcal{W}_m = \langle \mathbf{V}_m \rangle$. Then $\mathcal{W}_m$ is $\mathbf{A}$–orthogonal to itself and to $\mathcal{W}$. By construction, $\mathbf{AX} - \mathbf{V}_0 \in \mathcal{W} + \mathcal{W}_m$. If $\mathbf{V}_m = \mathbf{0}$, then $\mathbf{AX} = \mathbf{V}_0 = \mathbf{AY}$.

Often the algorithm terminates with $\mathbf{V}_m^{\mathrm{T}}\mathbf{A}\mathbf{V}_m = \mathbf{0}$ but $\mathbf{V}_m \neq \mathbf{0}$. We argue heuristically that $\mathbf{A}\mathcal{W}_m \subseteq \mathcal{W}_m$. The matrix $\mathbf{V}_m$ is $\mathbf{A}$–orthogonal not only to itself but to $\mathbf{W}_j$ for $j < m$ by (11). In practice, this $\mathbf{V}_m$ has small rank (perhaps two). All $\mathbf{V}_j$ and $\mathbf{W}_j$ are contained in the Krylov subspace

$$\mathcal{V} = \langle \mathbf{V}_0 \rangle + \langle \mathbf{A}\mathbf{V}_0 \rangle + \langle \mathbf{A}^2\mathbf{V}_0 \rangle + \cdots \ . \tag{21}$$

If N is large (say $N \geq 16$), then the final $\mathbf{V}_m$ typically has precisely those vectors in $\mathcal{V}$ which are $\mathbf{A}$–orthogonal to all of $\mathcal{V}$, including themselves. See §8 for comments about the size of this subspace if $\mathbf{A}$ is random (which $\mathbf{B}^{\mathrm{T}}\mathbf{B}$ definitely is not). If this assumption about $\mathcal{V}$ is correct, then $\mathcal{V}$ is the direct sum of $\mathcal{W}$ and of $\langle \mathbf{V}_m \rangle = \mathcal{W}_m$. Hence $\mathbf{A}\mathcal{W}_m \subseteq \mathbf{A}\mathcal{V} \subseteq \mathcal{V} = \mathcal{W} + \mathcal{W}_m$. Suppose $\mathbf{w}_0 + \mathbf{w}_1 + \cdots + \mathbf{w}_m \in \mathbf{A}\mathcal{W}_m$, where each $\mathbf{w}_j \in \mathcal{W}_j$. We claim that $\mathbf{w}_j = \mathbf{0}$ for $0 \leq j < m$. We check that

$$\begin{aligned} \mathbf{w}_j^{\mathrm{T}}\mathbf{A}\mathcal{W}_j &= (\mathbf{w}_0 + \mathbf{w}_1 + \cdots + \mathbf{w}_m)^{\mathrm{T}}\mathbf{A}\mathcal{W}_j \subseteq (\mathbf{A}\mathcal{W}_m)^{\mathrm{T}}\mathbf{A}\mathcal{W}_j \\ &= \mathcal{W}_m^{\mathrm{T}}\mathbf{A}(\mathbf{A}\mathcal{W}_j) \subseteq \mathcal{W}_m^{\mathrm{T}}\mathbf{A}(\mathcal{W} + \mathcal{W}_m) = \{0\} \ . \end{aligned}$$

Since $\mathbf{w}_j \in \mathcal{W}_j$ and $\mathcal{W}_j$ is $\mathbf{A}$–invertible for $j < m$, this shows that $\mathbf{w}_j = \mathbf{0}$.

If this heuristic argument about $\mathcal{V}$ is correct, then the images $\mathbf{A}(\mathbf{X} - \mathbf{Y})$ and $\mathbf{A}\mathbf{V}_m$ are both in $\mathcal{W}_m$. The total rank of $\mathbf{X} - \mathbf{Y}$ and $\mathbf{V}_m$ is at most $2N$. Using Gaussian elimination, one can take linear combinations of $\mathbf{X} - \mathbf{Y}$ and $\mathbf{V}_m$ to find vectors in the null space of $\mathbf{A} = \mathbf{B}^{\mathrm{T}}\mathbf{B}$. The same construction can be used to find vectors in the null space of $\mathbf{B}$. More precisely, let $\mathbf{Z}$ denote the $n \times 2N$ matrix formed by concatenating the columns of $\mathbf{X} - \mathbf{Y}$ and $\mathbf{V}_m$. Compute $\mathbf{BZ}$, and find a matrix $\mathbf{U}$ (at most $2N \times 2N$) whose columns span the null space of $\mathbf{BZ}$. Then output a basis for $\mathbf{ZU}$. In practice, one does not compute $\mathbf{U}$ explicitly, but applies the same column operations to $\mathbf{Z}$ and $\mathbf{BZ}$.

8 Selecting $\mathbf{S}_i$ and $\mathbf{W}_i$

Recurrence (9) does not specify how to select $\mathbf{S}_i$ and $\mathbf{W}_i = \mathbf{V}_i\mathbf{S}_i$, except that

- $\mathbf{W}_i^{\mathrm{T}}\mathbf{A}\mathbf{W}_i$ must be invertible;
- $\mathrm{rank}(\mathbf{W}_i)$ should be as large as possible;
- Any column of $\mathbf{V}_{i-1}$ which was not used in $\mathbf{W}_{i-1}$ must be used now (15).

Let $\mathbf{Q}$ be a symmetric $N \times N$ matrix over a field K. Let $r = \mathrm{rank}(\mathbf{Q})$. We claim that if we select any r linearly independent columns of $\mathbf{Q}$, then the symmetric $r \times r$ submatrix of $\mathbf{Q}$ with the same row indices is invertible. After renumbering the rows and columns, we may write $\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \mathbf{Q}_{21} & \mathbf{Q}_{22} \end{bmatrix}$. Here $\mathbf{Q}_{11}$ is the symmetric $r \times r$ matrix which we claim is invertible, $\mathbf{Q}_{22}$ is symmetric, and $\mathbf{Q}_{21} = \mathbf{Q}_{12}^{\mathrm{T}}$. The assumption that the first r columns generate all of $\mathbf{Q}$ means that there exists an $r \times (N-r)$ matrix $\mathbf{T}$ such that $\begin{bmatrix} \mathbf{Q}_{12} \\ \mathbf{Q}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_{11} \\ \mathbf{Q}_{21} \end{bmatrix}\mathbf{T}$. This translates into $\mathbf{Q}_{12} = \mathbf{Q}_{11}\mathbf{T}$ and $\mathbf{Q}_{22} = \mathbf{Q}_{21}\mathbf{T} = \mathbf{Q}_{12}^{\mathrm{T}}\mathbf{T} = \mathbf{T}^{\mathrm{T}}\mathbf{Q}_{11}\mathbf{T}$. Hence

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{11}\mathbf{T} \\ \mathbf{T}^{\mathrm{T}}\mathbf{Q}_{11} & \mathbf{T}^{\mathrm{T}}\mathbf{Q}_{11}\mathbf{T} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_r & \mathbf{0} \\ \mathbf{T}^{\mathrm{T}} & \mathbf{I}_{N-r} \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{I}_r & \mathbf{T} \\ \mathbf{0} & \mathbf{I}_{N-r} \end{bmatrix} \;,$$

implying $\mathrm{rank}(\mathbf{Q}_{11}) = \mathrm{rank}(\mathbf{Q}) = r$.

Consequently all maximal $\mathbf{A}$–invertible subspaces of $\langle \mathbf{V}_i \rangle$ have dimension $\mathrm{rank}(\mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i)$. The selection of $\mathbf{W}_i$ can first include anything required by (15), namely $\mathbf{V}_i(\mathbf{I}_N - \mathbf{S}_{i-1}\mathbf{S}_{i-1}^{\mathrm{T}})$. If the corresponding columns of $\mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i$ are linearly dependent but the columns in $\mathbf{V}_i$ are independent, then the algorithm fails. Otherwise choose a spanning set of columns for $\mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i$ while trying to include the required columns, and choose $\mathbf{S}_i$ accordingly. Figure 1 summarizes my procedure. Afterwards I check whether all nonzero columns of $\mathbf{V}_{i+1}$ were chosen in $\mathbf{S}_i$ and/or $\mathbf{S}_{i-1}$.

The pseudocode asserts that some $\mathbf{M}[c_k, c_j]$ or $\mathbf{M}[c_k, c_j + N]$ is nonzero, where $k \geq j$. At all times, if $\mathbf{M}_1$ denotes the left half of $\mathbf{M}$ and $\mathbf{M}_2$ denotes its right half, then $\mathbf{M}_1 = \mathbf{M}_2\mathbf{T}$ since only row operations are performed. We attempt to get $\mathbf{I}_N$ on the left of $\mathbf{M}$, but occasionally zero an entire row. Let $\overline{S} = \{c_1, \cdots, c_{j-1}\} \setminus S$ denote the rows of $\mathbf{M}$ which have been zeroed. We consider three sets of vectors in $\mathrm{GF}(2)^n$:

1. Dependency vectors $\mathbf{v}_k$ for $k \in \overline{S}$. These satisfy $\mathbf{T}^{\mathrm{T}}\mathbf{v}_k = \mathbf{T}\mathbf{v}_k = \mathbf{0}$. The k–th element of $\mathbf{v}_k$ is 1; the index of any other nonzero element of $\mathbf{v}_k$ is in S. Row k of $\mathbf{M}$ and column k of $\mathbf{M}_2$ are zero.
2. Rows c_j to c_N of $\mathbf{M}_2$. These rows are linearly independent, since the initial $\mathbf{M}_2 = \mathbf{I}_N$ had full rank.
3. Columns k of $\mathbf{T}$, for $k \in S$ (same as row vectors, since $\mathbf{T}$ is symmetric). Column k of $\mathbf{M}_2\mathbf{T}$ matches that in $\mathbf{I}_N$.

Cmt. Inputs: $\mathbf{T} = \mathbf{V}_i^{\mathrm{T}} \mathbf{A} \mathbf{V}$ and $\mathbf{S}_{i-1}$, where $\mathbf{A}$ (and hence $\mathbf{T}$) is symmetric.
Cmt. Outputs: Set S for diagonal of $\mathbf{S}_i \mathbf{S}_i^{\mathrm{T}}$, and $\mathbf{W}_i^{\mathrm{inv}} = \mathbf{S}_i \left(\mathbf{S}_i^{\mathrm{T}} \mathbf{T} \mathbf{S}_i \right)^{-1} \mathbf{S}_i^{\mathrm{T}}$.
Construct an $N \times 2N$ block matrix $\mathbf{M}$, with $\mathbf{T}$ on the left and $\mathbf{I}_N$ on the right.
Cmt. Algorithm performs row operations on $\mathbf{M}$. It may zero an entire row.
Number columns of $\mathbf{T}$ as $c_1, c_2, \cdots, c_N$, with columns in $\mathbf{S}_{i-1}$ coming last.
Initialize $S = \emptyset$. **Cmt**. S has the columns selected from $\mathbf{V}_i$ for $\mathbf{W}_i$.
do $j = 1$ **to** N
  do $k = j$ **to** N **while** $(\mathbf{M}[c_j, c_j] = 0)$
    if $(\mathbf{M}[c_k, c_j] \neq 0)$ Exchange rows c_j and c_k of $\mathbf{M}$.
  end while
  if $(\mathbf{M}[c_j, c_j] \neq 0)$ **then** **Cmt**. Use column c_j of $\mathbf{V}_i$ in $\mathbf{W}_i$.
    $S = S \cup \{c_j\}$
    Divide row c_j of $\mathbf{M}$ by $\mathbf{M}[c_j, c_j]$. **Cmt**. A no–op over GF(2).
    Add multiples of row c_j to other rows of $\mathbf{M}$, to zero rest of column c_j.
  else **Cmt**. No pivot element found in column c_j.
    Cmt. Column c_j of $\mathbf{T}$ is linear combination of earlier ones. Skip it.
    do $k = j$ **to** N **while** $(\mathbf{M}[c_j, c_j + N] = 0)$
      if $(\mathbf{M}[c_k, c_j + N] \neq 0)$ Exchange rows c_j and c_k of $\mathbf{M}$.
    end do
    assert $(\mathbf{M}[c_j, c_j + N] \neq 0)$
    Add multiples of row c_j to other rows, to zero rest of column $c_j + N$.
    Zero row c_j of $\mathbf{M}$. **Cmt**. Will be zero for rest of algorithm.
  end if
  Cmt. Column c_j will remain unchanged for the duration of the algorithm.
end do
Copy right half of $\mathbf{M}$ into $\mathbf{W}_i^{\mathrm{inv}}$.

Fig. 1. Pseudocode for selecting $\mathbf{S}_i$ and $\mathbf{W}_i$

Sets 1 and 2 have a total of $|\overline{S}| + (N - j + 1) = N - |S|$ linearly independent vectors. The $|S|$ independent vectors in Set 3 are orthogonal to all vectors in Sets 1 and 2, and therefore span the orthogonal complement.

If $(\mathbf{M}_2 \mathbf{T})[c_k, c_j] = 0$ for all $k \geq j$, then column c_j of $\mathbf{T}$ is orthogonal to everything in Sets 1 and 2. This column must be a linear combination of vectors in Set 3. Let the dependency vector for $\mathbf{T}$ be $\mathbf{v} = \mathbf{v}_{c_j}$. By symmetry, this $\mathbf{v}$ is orthogonal to all of Set 3 and must be a linear combination of vectors in Sets 1 and 2. Since element c_j of $\mathbf{v}$ is nonzero, some vector in Set 1 or Set 2 must be nonzero there. This translates into $\mathbf{M}_2[c_k, c_j] \neq 0$ for some $k \geq j$, as required.

8.1 Rank of $\mathbf{W}_i$

Let $f_N(X)$ be the generating function for the corank of a random $N \times N$ matrix over the field GF(p). That is, the coefficient of X^m in $f_N(X)$ is the probability that the matrix has rank exactly $N - m$. If $q = 1/p$, then

$$\begin{aligned} f_0(X) &= 1 \ , \\ f_1(X) &= 1 - q + qX \ , \\ f_N(X) &= (1 - q + q^N X) f_{N-1}(X) + (q - q^N) f_{N-2}(X) \qquad (N \geq 2) \ . \end{aligned} \tag{22}$$

Equation (22) is derived by checking which elements in the first row of the random matrix are zero, using the methods of [2]. It implies the recurrence $f_N(pX) = f_N(X) + (1 - q^N) X f_{N-1}(X)$ for $N \geq 1$. As $N \to \infty$, $f_N \to f$ where $f(pX) = (1 + X) f(X)$. The solution is

$$f(X) = c_q \left(1 + \frac{X}{p-1} + \frac{X^2}{(p-1)(p^2-1)} + \frac{X^3}{(p-1)(p^2-1)(p^3-1)} + \cdots \right) \ .$$

Here $c_q = (1-q)(1-q^3)(1-q^5) \cdots$ to force $f(1) = 1$.

The expected rank $\mathrm{Er}_N = N - f'_N(1)$ satisfies

$$\begin{aligned} \mathrm{Er}_0 &= 0 \ , \\ \mathrm{Er}_1 &= 1 - q \ , \\ \mathrm{Er}_N &= (1-q)(1 + \mathrm{Er}_{N-1}) + q^N \mathrm{Er}_{N-1} + (q - q^N)(2 + \mathrm{Er}_{N-2}) \\ &= (1 + q - 2q^N) + (1 - q + q^N) \mathrm{Er}_{N-1} + (q - q^N) \mathrm{Er}_{N-2} \qquad (N \geq 2) \ . \end{aligned} \tag{23}$$

When $p = 2$ and $q = 1/2$, equation (23) implies $\mathrm{Er}_N \geq N - 1 + 2^{-N}$ for $N \geq 0$. For large N, approximate numerical values are $c_2 \approx 0.419422418$ and $\mathrm{Er}_N \approx N - 0.764499780$.

We conjecture that the average number of iterations needed is about n/Er_N for large N and n, subject to $N \ll n \ll 2^{N^2/2}$. The experimental data in Table 1 of §10 support this conjecture.

9 Cost of Block Lanczos

Each iteration computes $\mathbf{A}\mathbf{V}_i$, $\mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i$, and $\mathbf{V}_i^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_i = (\mathbf{A}\mathbf{V}_i)^{\mathrm{T}}\mathbf{A}\mathbf{V}_i$ from $\mathbf{V}_i$. After choosing $\mathbf{S}_i$, it computes $\mathbf{W}^{\mathrm{inv}}$ in (16). It updates the partial sums of $\mathbf{X} - \mathbf{Y}$ using (20), and computes $\mathbf{V}_{i+1}$ using (18) and (19).

Equation (20) appears to require one inner product per $\mathbf{V}_i^{\mathrm{T}}\mathbf{V}_0$ and one multiplication of the $n \times N$ matrix $\mathbf{V}_j$ by a $N \times N$ matrix. Henk Boender [1] observes that most of these inner product computations can be exchanged for some $N \times N$ matrix computations. Equation (15) implies $\langle \mathbf{V}_0 \rangle \subseteq \mathcal{W}_0 + \mathcal{W}_1$; if $i \geq 2$, then $\mathbf{V}_0^{\mathrm{T}}\mathbf{A}\mathbf{V}_i = \mathbf{0}$ by (11). By (18),

$$\begin{aligned} \mathbf{V}_{i+1}^{\mathrm{T}}\mathbf{V}_0 &= \left(\mathbf{A}\mathbf{V}_i\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}} + \mathbf{V}_i\mathbf{D}_{i+1} + \mathbf{V}_{i-1}\mathbf{E}_{i+1} + \mathbf{V}_{i-2}\mathbf{F}_{i+1} \right)^{\mathrm{T}} \mathbf{V}_0 \\ &= \mathbf{D}_{i+1}^{\mathrm{T}}\mathbf{V}_i^{\mathrm{T}}\mathbf{V}_0 + \mathbf{E}_{i+1}^{\mathrm{T}}\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{V}_0 + \mathbf{F}_{i+1}^{\mathrm{T}}\mathbf{V}_{i-2}^{\mathrm{T}}\mathbf{V}_0 \ . \end{aligned}$$

The inner products $\mathbf{V}_i^{\mathrm{T}}\mathbf{V}_0$, $\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{V}_0$, and $\mathbf{V}_{i-2}^{\mathrm{T}}\mathbf{V}_0$ are known by induction.

With this improvement, the cost of (16), (18), (19), and (20) is (for $i > 2$):

- One application of $\mathbf{A} = \mathbf{B}^{\mathrm{T}}\mathbf{B}$ to $\mathbf{V}_i$ to compute $\mathbf{A}\mathbf{V}_i$.
- Two inner products of pairs of $n \times N$ matrices to compute the $N \times N$ matrices $\mathbf{V}_i^{\mathrm{T}}\mathbf{A}\mathbf{V}_i$ and $\mathbf{V}_i^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_i$. (We assume that $\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}\mathbf{V}_{i-1}$ and $\mathbf{V}_{i-1}^{\mathrm{T}}\mathbf{A}^2\mathbf{V}_{i-1}$ are known by induction.)
- Selection of $\mathbf{S}_i$ subject to (15) and the computation of $\mathbf{W}_i^{\mathrm{inv}}$, as in §8.
- A few $N \times N$ matrix operations to compute $\mathbf{D}_{i+1}$, $\mathbf{E}_{i+1}$, and $\mathbf{F}_{i+1}$ in (19).
- Four multiplications of $n \times N$ matrices by $N \times N$ matrices and four additions of $n \times N$ matrices when computing $\mathbf{V}_{i+1}$ and the new partial sum of $\mathbf{X} - \mathbf{Y}$. (The post–multiplication by $\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}}$ in (18) is easy since $\mathbf{S}_i\mathbf{S}_i^{\mathrm{T}}$ is diagonal with zeros and ones. When $K = \mathrm{GF}(2)$, it can be done via bitwise ANDs.)

The conjectured iteration count is $n/\mathrm{Er}_N = \mathcal{O}(n/N)$ if the initial $\mathbf{V}_0$ is random.

If the matrix $\mathbf{B}$ averages d nonzero entries per column, then each multiplication by $\mathbf{A}$ takes time $\mathcal{O}(dn)$. An $\mathcal{O}(Nn)$ algorithm for inner products over GF(2) circularly shifts the first argument by k bits, and uses n ANDs and $n-1$ XORs to construct N bits of the inner product; this step is repeated for $0 \le k \le N-1$. The multiplications of an $n \times N$ matrix by an $N \times N$ matrix can similarly be done with $\mathcal{O}(Nn)$ operations on N–bit words.

The net time is $\mathcal{O}(dn) + \mathcal{O}(Nn)$ per iteration and $\mathcal{O}(dn^2/N) + \mathcal{O}(n^2)$ for the algorithm. Gaussian elimination takes time $\mathcal{O}(n^3)$. Block Lanczos is asymptotically superior (in time and space) to Gaussian elimination if $d \ll n$ and remains competitive with Gaussian elimination if $d = \mathcal{O}(n)$.

Coppersmith [6, pp. 342–343] shows how to compute the inner products and the products of $n \times N$ by $N \times N$ matrices more efficiently.

When $\mathbf{S}_{i-1} = \mathbf{I}_N$ (so that $\mathbf{W}_{i-1} = \mathbf{V}_{i-1}$), the formula for $\mathbf{F}_{i+1}$ simplifies to zero and the term $\mathbf{V}_{i-2}\mathbf{F}_{i+1}$ can be omitted from (18).

Storage requirements are low. Other than the matrix $\mathbf{A}$ itself, the algorithm needs only $\mathbf{V}_{i+1}$, $\mathbf{V}_i$, $\mathbf{V}_{i-1}$, $\mathbf{V}_{i-2}$, $\mathbf{A}\mathbf{V}_i$, the partial sums of $\mathbf{X} - \mathbf{Y}$, and some $N \times N$ matrices. The same storage can be used for $\mathbf{A}\mathbf{V}_i$ and $\mathbf{V}_{i+1}$. The implementation may need storage for the intermediate vector $\mathbf{B}\mathbf{V}_i$ during the computation of $\mathbf{A}\mathbf{V}_i = \mathbf{B}^{\mathrm{T}}(\mathbf{B}\mathbf{V}_i)$, but this can be avoided if $\mathbf{B}$ is stored by rows. At the end, the algorithm needs storage for $\mathbf{V}_m$, $\mathbf{B}\mathbf{V}_m$, $\mathbf{X} - \mathbf{Y}$, and $\mathbf{B}(\mathbf{X} - \mathbf{Y})$.

10 Experimental Results

In June, 1994, we applied the algorithm to an $828,077 \times 833,017$ matrix with $26,886,496$ nonzero entries as part of the record factorization of the 162–digit number $(12^{151} - 1)/11$ via the special number field sieve. The program was run on one processor of a Cray C90 at the Academic Computing Center Amsterdam (SARA), with $N = 64$. The algorithm terminated after 3.4 hours with $m = 13,098$ and $\dim(\mathcal{W}) = 828,075$.

Two days later we applied the algorithm to a $1,284,719 \times 1,294,861$ matrix with 38,928,220 nonzero entries as part of the factorization of a 105–digit cofactor of $3^{367}-1$ via the general number field sieve (GNFS). The program terminated after 7.3 hours with $m = 20,319$ and $\dim(\mathcal{W}) = 1,284,712$.

The program sizes were about 330 Mb and 480 Mb, respectively, compared to the machine's 2147 Mb. For Gaussian elimination, on a dense matrix with one third as many rows and columns, the sizes would be 9 Gb and 22 Gb.

Table 1 has the observed dimensions of $\mathcal{W}_0$ through $\mathcal{W}_{m-1}$. The "probability" column is based on the coefficients of $f_{64}(X)$. The low values (6 and 18) for $\dim(\mathcal{W}_i)$ occurred at the end, specifically when $i = m-1$ and the Krylov subspace (21) had been exhausted.

Table 1. Dimensions of $\mathcal{W}_i$ for two big systems

	Predicted	$828,077 \times 833,017$		$1,284,719 \times 1,294,861$	
$\dim(\mathcal{W}_i)$	probability	Frequency	Percent	Frequency	Percent
6	.00000	0	0.000	1	0.005
18	.00000	1	0.008	0	0.000
59	.00004	1	0.008	0	0.000
60	.00133	15	0.115	32	0.157
61	.01997	290	2.214	431	2.121
62	.13981	1854	14.155	2946	14.499
63	.41942	5508	42.052	8333	41.011
64	.41942	5429	41.449	8576	42.207
Total		13,098		20,319	
$\sum \dim(\mathcal{W}_i)$		828,075		1,284,712	

Scott Contini and Arjen Lenstra implemented the algorithm on Bellcore's MasPar 16K–1 with 16K processors [4]. The MasPar took 2.5 CPU days to find 151 column dependencies of a $1,472,607 \times 1,475,898$ matrix with 79,661,432 nonzero entries, after ignoring its 102 densest rows. These dependencies were later combined to get 49 true dependencies and to factor the 119–digit cofactor of the partition number $p(13171)$ via GNFS. Memory requirements were 24 Kb per node, or 390 Mb total. They estimate that Gaussian elimination would have required two CPU weeks and 15 Gb, after reducing the problem to a dense $362,397 \times 362,597$ system.

Don Coppersmith [5, §12] reports a need to preprocess the data to eliminate singletons, which are rows containing a single nonzero element. One normally does such preprocessing, since the corresponding column cannot be in a dependency. We have not experienced his difficulty when we omit this step, but lack a theoretical explanation for why the problem does not occur in our variation.

11 Acknowledgements

This work was supported by Stieltjes Institute for Mathematics, Leiden and by Centrum voor Wiskunde en Informatica, Amsterdam. The CPU time on the Cray C90 was provided by the Dutch National Computing Facilities Foundation NCF, with financial support from the Netherlands Organization for Scientific Research NWO. Thanks to Scott Contini of Bellcore for his constructive comments on an earlier version of this manuscript.

References

1. Henk Boender, Private communication, 1994.
2. Richard P. Brent and Brendan D. McKay, *On determinants of random symmetric matrices over* $\mathbb{Z}_m$, Ars Combinatoria **26A** (1988), 57–64.
3. J.P. Buhler, H.W. Lenstra, Jr., and Carl Pomerance, *Factoring integers with the number field sieve*, The Development of the Number Field Sieve (Berlin) (A.K. Lenstra and H.W. Lenstra, Jr., eds.), Lecture Notes in Mathematics, vol. 1554, Springer–Verlag, Berlin, 1993, pp. 50–94.
4. Scott Contini and Arjen K. Lenstra, *Implementation of blocked Lanczos and Wiedemann algorithms*, In preparation, 1995.
5. Don Coppersmith, *Solving linear equations over GF*(2)*: Block Lanczos algorithm*, Linear Algebra and its Applications **192** (1993), 33–60.
6. ———, *Solving homogeneous linear equations over GF*(2) *via block Wiedemann algorithm*, Mathematics of Computation **62** (1994), no. 205, 333–350.
7. Don Coppersmith, Andrew M. Odlyzko, and Richard Schroeppel, *Discrete logarithms in GF(p)*, Algorithmica **1** (1986), 1–15.
8. Jane K. Cullum and Ralph A. Willoughby, *Lanczos algorithms for large symmetric eigenvalue computations. Vol. I Theory*, Birkhäuser, Boston, 1985.
9. Donald E. Knuth, *Seminumerical algorithms*, The Art of Computer Programming, vol. 2, Addison–Wesley, Reading, MA, 2nd ed., 1981.
10. B.A. LaMacchia and A.M. Odlyzko, *Solving large sparse systems over finite fields*, Advances in Cryptology, CRYPTO '90 (A.J. Menezes and S.A. Vanstone, eds.), Lecture Notes in Computer Science, vol. 537, Springer–Verlag, pp. 109–133.
11. A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, and J.M. Pollard, *The factorization of the ninth Fermat number*, Mathematics of Computation **61** (1993), no. 203, 319–349.
12. A.M. Odlyzko, *Discrete logarithms in finite fields and their cryptographic significance*, Advances in Cryptology: Proceedings of EUROCRYPT 84 (New York) (T. Beth, N. Cot, and I. Ingemarsson, eds.), Lecture Notes in Computer Science, vol. 209, Springer–Verlag, pp. 224–314.
13. Carl Pomerance, *The quadratic sieve factoring algorithm*, Advances in Cryptology, Proceedings of EUROCRYPT 84 (New York) (T. Beth, N. Cot, and I. Ingemarsson, eds.), Lecture Notes in Computer Science, vol. 209, Springer–Verlag, pp. 169–182.
14. Robert D. Silverman, *The multiple polynomial quadratic sieve*, Mathematics of Computation **48** (1987), no. 177, 329–339.
15. Douglas H. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inform. Theory **32** (1986), no. 1, 54–62.