2003

TR-2003004: Superfast Algorithms for Singular Toeplitz/Hankel-like Matrices

Victor Y. Pan

Superfast Algorithms for Singular Toeplitz/Hankel-like Matrices

Victor Y. Pan
Department of Mathematics and Computer Science
Lehman College of CUNY, Bronx, NY 10468, USA
vpan@lehman.cuny.edu

May 6, 2003

Abstract

Applying the superfast divide-and-conquer MBA algorithm for generally singular $n \times n$ Toeplitz-like or Hankel-like integer input matrices, we perform computations in the ring of integers modulo a power of a fixed prime, especially power of 2. This is practically faster than computing modulo a random prime but requires additional care to avoid degeneration, particularly at the stages of compression of auxiliary matrices. We supply the necessary techniques. The resulting algorithm combined with Hensel's lifting and fast rational number reconstruction supports nearly optimal bit cost estimates for the solution of (possibly singular but) consistent Toeplitz/Hankel-like linear systems with integer coefficients (as well as for other fundamental problems of matrix computation). We arrive at nearly optimal bit cost estimates also for computing the univariate polynomial gcd and resultant, Padé approximation, rational interpolation, and Berlekamp–Massey's problem.

0 Introduction

0.1 Background and our progress

Matrices with the structure of Toeplitz/Hankel type are ubiquitous in computations in sciences, engineering, and signal processing (see, e.g., Kailath and Sayed (editors) 1999 [KS99], Pan 2000 [P00, Section 1.1], Pan 2001 [P01], and the bibliography therein). In computer algebra, some most fundamental problems amount to or can be reduced to solving (possibly singular but) consistent Toeplitz or Hankel systems of linear equations $M\mathbf{x} = \mathbf{b}$. Such systems with nonsingular integer input have been effectively solved in [P00], Pan 2002 [P02], [P02a] by relying on Hensel's lifting. We substantially accelerate the algorithms in [P02a] and extend them to the singular case by combining them with the superfast divide-and-conquer MBA algorithm by Morf 1974, 1980 [M74], [M80]

and Bitmead and Anderson 1980 [BA80]. This algorithm applies to structured matrices having small displacement rank (see Kailath et al. 1979 [KKM79], [KS99], and [P01] on this fundamental concept); Toeplitz (or Hankel) input is a special case treated in the same way. Practical efficiency of the computations grows if they are performed modulo a power p^u of a fixed prime p, $p = 2$ is the most desired choice. To arrive at practically promising algorithms, we meet quite a few technical challenges, e.g., we avoid degeneration of the compressed auxiliary matrices (see Sections 9–12).

0.2 The bit complexity estimates

Hereafter, log stands for $\log_2$, $m(n)$ is the arithmetic cost of multiplying two polynomials of degree n, and $\mu(d)$ is the bit operation cost of multiplying two integers in the range from -2^d to 2^d. We have

$$2n - 1 \le m(n) \le \min\{c_{class}n^2, c_k n^{\log 3}, (c_{ck} n \log n) \log\log n\}, \tag{0.1}$$

$$2d - 1 \le \mu(d) \le \min\{C_{class}d^2, C_k d^{\log 3}, (C_{ss} d \log d) \log\log d\}, \tag{0.2}$$

where $\log 3 = 1.5849625\ldots$, $c_{class} < c_k < c_{ck}$, $C_{class} < C_k < C_{ss}$, $c_{class}, C_{class}, c_k, C_k, c_{ck}$ and C_{ss} are constants, and the abbreviations "class", "k", "ck", and "ss" refer to the "classical", "Karatsuba's", "Cantor and Kaltofen's", and "Schönhage and Strassen's" algorithms, respectively (see, e.g., Bernstein, to appear [Ba]).

Our solution modulo p^u of a consistent linear system $M\mathbf{x} = \mathbf{b}$ for a Toeplitz-like $n \times n$ integer matrix M of rank ρ requires

$$B = O((m(n) + m(\rho)\log\rho)\mu(u\log p)) \tag{0.3}$$

bit operations. We allow failure with a probability of at most ϵ, do not count the cost of generating $O(n\log(n/\epsilon))$ random bits, and let $\|M\|_1 + \|\mathbf{b}\|_1 + p = O(n)$. Here and hereafter, $\|\cdot\|_1$ denotes the 1-norms of vectors and matrices. In [P02a] $\rho = n$ and the bit cost bound is inferior to (0.3),

$$B = O((m(n)n\mu(u\log p)\log u)/\log(u\log p)). \tag{0.4}$$

Bound (0.3) is proved for any choice of a fixed prime p and any pair of integers u and $g < u$ for which M/p^g can be inverted modulo p^u. The same bound is proved for $u = 1$ and a random prime p (see Theorems 8.1 and 11.2). For a fixed p and a fixed precision of computing, we have an upper bound u^+ on u. For a random input matrix M we rarely need to increase u above u^+ to support the inversion of M/p^g modulo p^u (see Wang 2002 [W02]). In this unlikely case, we may choose a distinct (fixed or random) prime p and repeat the computations.

Having the vector $\mathbf{x} \bmod p^u$ computed, we compute a rational solution $\mathbf{x}$ by first applying the lifting algorithm in [P02a] and then the rational number reconstruction algorithms in Pan and Wang 2002 [PW02], Wang and Pan 2003 [WP03]. Only the latter stage of reconstruction requires a higher precision

of computing. We obtain the best overall asymptotic bit cost estimates if we assume nearly linear $\mu(d)$ in (0.2) and choose u^+ of the order of $n/\log p$. Then we arrive at the overall bit cost in $O(m(\rho)\log\rho + m(n))\mu(n))$ or, for $\rho = n$, in $O(m(n)(\log n)\mu(n))$ (see our Corollary 12.1 and Theorem 6.2 in [P02a]). For $\rho = n$, the latter estimate is within the factor of $m(n)\mu(n)/n^2$ from the information lower bound $n^2 \log n$, showing the number of bits in the output.

0.3 Extensions

Besides solving a linear system $M\mathbf{x} = \mathbf{b}$, the extended MBA algorithm computes modulo p^{g+1} the determinant of M, the rank of M, and a vector from the null space of M or a shortest generator for a matrix whose columns form a basis for the null space of M. The bit cost bound of (0.3) applies. Furthermore, the known reduction techniques (see Brent et al. 1980 [BGY80], [P01]) immediately enable extension of the algorithm and its cost estimates to computing the univariate polynomial gcd, lcm, and resultant, Padé approximation, rational interpolation and Berlekamp–Massey's linear recurrence computation.

The algorithm can be easily extended to the case of rational input values and to structured matrices of other types (e.g., Vandermonde and Cauchy–Pick type) [P01]. In this case the choice of the exponent u in the base modulus p^u for a fixed p grows with the magnitudes of the determinant of the input matrix and the least common denominator of the input values. For the highly important matrices of the Cauchy–Pick type, the magnitudes can be quite large, and so one may shift to the Toeplitz/Hankel-like input by using the displacement transformation (Pan 1990 [P90], [P01]) with the subsequent truncation and scaling of the resulting values to ensure integrality.

Of course, the algorithm can be also applied to solving a general linear system $M\mathbf{x} = \mathbf{b}$ modulo p or p^g. This case is technically much simpler, because all the compression problems disappear, but is less rewarding because the resulting bit cost estimates grow to the known level [MS99], [MSa].

0.4 Organization of the paper

In the next three sections, we recall some background on general structured matrices, Toeplitz/Hankel-like matrices, and Hensel's lifting. In Sections 4–7, we elaborate upon the MBA algorithm for recursive factorization of a matrix and its extensions. In Sections 9 and 10, we compress Toeplitz-like matrices (avoiding degeneration) in the ring of integers modulo p^{g+1}. We estimate the arithmetic cost in Section 8 and the Boolean (bit operation) cost in Sections 11 and 12.

1 Displacement representation of structured matrices

This paper specializes to the structure of Toeplitz/Hankel type, but it is more convenient to state some basic definitions for a more general class of structured matrices (cf. [P01]).

Definition 1.1. Displacement operators L of Stein and Sylvester types *map a matrix M into its displacements $L(M) = \nabla_{A,B}(M) = AM - MB$ and $L(M) = \Delta_{A,B}(M) = M - AMB$, respectively. A and B are called* operator matrices, *$r = \operatorname{rank} L(M)$ is called the L-*rank *or the* displacement rank *of M. If*

$$L(M) = GH^T, \tag{1.1}$$

where

$$G = (\mathbf{g}_1, \dots, \mathbf{g}_l) \quad \textit{and} \quad H = (\mathbf{h}_1, \dots, \mathbf{h}_l) \tag{1.2}$$

are $l \times n$ matrices and M and $L(M)$ are $n \times n$ matrices, then the matrix pair (G, H) is called a generator *of length l for $L(M)$ and an L-*generator *(or a* displacement generator*) of length l for M, $l \geq r$. (A pair G, H in (1.1) is nonunique for a fixed $L(M)$.) If M is an $m \times n$ matrix and l is small relatively to m and n (say, if $l = O(1)$ as $\min\{m, n\} \to \infty$), then M is said to have L-*structure *or to be an L-*structured matrix. *(Both Sylvester and Stein type operatprs ∇ and Δ would fit in this paper (cf. our Theorem A.2 and [P01]); somewhat randomly, we selected Sylvester's ∇.)*

The next theorem is crucial for the efficiency of the MBA algorithm.

Theorem 1.2. *[P01, Theorem 4.6.4]. Given an L-generator (G_1, H_1) of length l for a matrix M having L-rank $r \leq l$, it is sufficient to use $O(l^2 n)$ arithmetic operations to compute an L-generator (G, H) of length r for the matrix M.*

The following results are immediately verified.

Theorem 1.3. *Let M be a nonsingular matrix, then $\nabla_{A,B}(M^{-1}) = -M^{-1}\nabla_{B,A}(M)M^{-1}$.*

Theorem 1.4. *For two pairs of scalars a and b and matrices M and N of the same size and any linear operator L (in particular, for $L = \nabla_{A,B}$ for any pair of matrices A and B), we have $L(aM + bN) = aL(M) + bL(N)$.*

Theorem 1.5. *[P01, Theorem 1.5.4]. For a 5-tuple $\{A, B, C, M, N\}$ of matrices of compatible sizes, we have*

$$\nabla_{A,C}(MN) = \nabla_{A,B}(M)N + M\nabla_{B,C}(N).$$

Based on the results in this subsection, we may represent L-structured $m \times n$ matrices M with mn entries in compressed form via $(m + n)l$ entries of their short L-generators and operate with them according to the following flowchart:

Theorems 1.2–1.5 support the OPERATE stage, in which in addition to saving the memory space, we may usually dramatically decrease computational time (see the next two sections). On the DECOMPRESS stage, see our Theorem 2.6, [P01, Chapter 6] and Pan and Wang 2003 [PW03].

2 Toeplitz-like and Hankel-like matrices

Definition 2.1. *$T = (t_{i,j})$ is a Toeplitz matrix if $t_{i,j} = t_{i+1,j+1}$ for every pair of its entries $t_{i,j}$ and $t_{i+1,j+1}$. Especially, for any scalar f and vector $\mathbf{v} = (v_i)_{i=0}^{n-1}$, define the $n \times n$ unit f-circulant matrix $Z_f = (z_{i,j})$, $z_{i,i-1} = 1$, $i = 2, \dots, n$; $z_{1,n} = f$, $z_{i,j} = 0$ for other pairs (i, j), and the f-circulant matrix with the first column $\mathbf{v}$, $Z_f(\mathbf{v}) = \sum_{i=0}^{n-1} v_i Z_f^i$. (Note that $Z_f^n = fI$.)*

$H = (h_{i,j})$ is a Hankel matrix if $h_{i,j} = h_{i-1,j+1}$ for every pair of its entries $h_{i,j}$ and $h_{i-1,j+1}$. Especially define the reflection matrix $J = (j_{g,h})$, $j_{g,n+1-g} = 0$ for $g = 0, \dots, n-1$, $j_{g,h} = 0$ for $h+g \neq n$. ($J\mathbf{v} = (v_{n-i-1})_{i=0}^{n-1}$ for any vector $\mathbf{v} = (v_i)_{i=0}^{n-1}$, $J^2 = I$.)

Theorem 2.2. *TJ and JT are Hankel matrices if T is a Toeplitz matrix, and HJ and JH are Toeplitz matrices if H is a Hankel matrix.*

Theorem 2.3. *For any pair of distinct scalars e and f and any Toeplitz matrix T, there exist nonunique pairs $(Z_e(\mathbf{u}), Z_f(\mathbf{v}))$ and $(Z_0(\mathbf{w}), Z_0(\mathbf{x}))$ such that $T = Z_e(\mathbf{u}) + Z_f(\mathbf{v}) = Z_0(\mathbf{w}) + Z_0^T(\mathbf{x})$. (Every matrix $Z_0(\mathbf{v})$ is lower triangular.)*

Theorem 2.4. *Given an $m \times n$ Toeplitz or Hankel matrix, its multiplication by a vector is a subproblem of multiplication of two polynomials of degrees $m+n-2$ and $n-1$, whose coefficients are given by the entries of the input matrix and vector, respectively.*

Definition 2.5. *An $m \times n$ matrix M is Toeplitz-like if $r = \operatorname{rank} L(M)$ is small relatively to $m+n$ (say, $r = O(1)$ as $\min\{m, n\} \to \infty$), and if M is given with its L-generator of length $l = O(r)$, where $L = \nabla_{Z_e, Z_f}$, $L = \nabla_{Z_e^T, Z_f^T}$, $L = \Delta_{Z_e, Z_f^T}$, $L = \Delta_{Z_e^T, Z_f}$ for a fixed pair of scalars e and f. A matrix M is Hankel-like if MJ or JM is Toeplitz-like (and so the problems of solving Toeplitz-like and Hankel-like linear systems are reduced to each other).*

Here is the displacement of T with $l \leq 2$ [P01, page 120]:

$$\nabla_{Z_f, Z_e}(T) = Z_f T - T Z_e = (Z_0 J\mathbf{t}_- - e\mathbf{t})\mathbf{e}_{n-1}^T + \mathbf{e}_0(fJ\mathbf{t} - Z_0^T\mathbf{t}_-)^T \qquad (2.1)$$

for $T = (t_{i-j})_{i,j=0}^{n-1}$, $\mathbf{t} = (t_i)_{i=0}^{n-1}$, $\mathbf{t}_- = (t_{-i})_{i=0}^{n-1}$, and any pair of scalars e and f.

In this paper it is sufficient to use e and f in the set $\{-1, 0, 1\}$ (see Appendix).

Theorem 2.6. *[P01, Examples 4.4.2]. Let* (1.1) *hold. Then*

$$(e-f)M=\sum_{j=1}^{l}Z_e(\mathbf{g}_j)Z_f(J\mathbf{h}_j)\ \textit{if}\ L=\nabla_{Z_e,Z_f},\ e\neq f,$$

$$(e-f)M=\sum_{j=1}^{l}Z_e^T(J\mathbf{g}_j)Z_f^T(\mathbf{h}_j)\ \textit{if}\ L=\nabla_{Z_e^T,Z_f^T},\ e\neq f.$$

Theorems 1.2, 2.2 and 2.6 together imply the following corollary.

Corollary 2.7. *An $n\times n$ Toeplitz-like or Hankel-like matrix M of displacement rank r given with its displacement generator of length l can be multiplied by a vector by using $O(m(n)l)$ ring operations or alternatively $O(m(n)r+l^2n)$ field operations for $m(n)$ in (0.1).*

The next results extend the displacement representation of a Toeplitz/Hankel-like matrix to its blocks.

Theorem 2.8. *Let* $P_0=\begin{pmatrix}I_l & 0\\ 0 & 0\end{pmatrix}$, $P_1=\begin{pmatrix}0 & 0\\ 0 & I_l\end{pmatrix}$ *for any fixed l. Then* $\mathrm{rank}(Z_fP_i-P_iZ_f)\le 2$ *for $i=0,1$ and any f.*

Corollary 2.9. $\mathrm{rank}(L(P_iMP_j)-P_iL(M)P_j)\le 4$ *for $i,j\in\{0,1\}$; $L=\nabla_{Z_e,Z_f}$, $L=\nabla_{Z_e^T,Z_f^T}$.*

Theorem 2.8 and Corollary 2.9 have simple constructive proofs, that is, given $i,j\in\{0,1\}$ and an L-generator for 2×2 block matrix M of length l, we immediately compute an L-generator of length of at most $l+4$ for the block P_iMP_j.

3 Hensel's lifting for a linear system of equations

Let us briefly recall lifting computation in [P02a]. Its input consists of a prime p, three integers $g\ge 0$, $h>0$, and $k>0$, a vector $\mathbf{b}\in\mathbb{Z}^n$, and two matrices $M\in\mathbb{Z}^{n\times n}$ and $Q=(M/p^g)^{-1} \bmod p^{g+k}$ satisfying $QM-p^gI=0 \bmod p^{g+k}$. In h lifting steps, the vector $x^{(h)}=(M/p^g)^{-1}\mathbf{b} \bmod p^{g+kh}$ is output. Each of the h steps essentially amounts to multiplying each of M and Q by a vector with the precision of $O(g\log\gamma)$ bits for $\gamma=n+p+\|M\|_1+\|\mathbf{b}\|_1$. For a sufficiently large $h=O(\log_{p^k}\|M\|_1^{2n-1}\|\mathbf{b}\|_1)$, the unique rational vector $\mathbf{y}=p^g\mathbf{x}$ is recovered from $\mathbf{x}^{(h)}$ such that $M\mathbf{x}=\mathbf{b}$. In [P02a, Corollary 5.1 and Theorem 6.1], the overall bit cost of lifting is bounded by

$$B=O(m(n)n\mu(g+k)\log n) \tag{3.1}$$

for $m(n)$ in (0.1), $\mu(d)$ in (0.2). This includes the cost of precomputation of Q [P02a, Algorithms 6.1 or 6.2] and randomized recovery of $\mathbf{x}$ from $\mathbf{x}^{(h)}$ [PW02],

[WP03] but not the generation of $O(n \log^2 n)$ random bits, and we assume that $\log \gamma = O(\log n)$, $\log(1/\epsilon) = O(\log n)$ and that the computation either fails with a probability of at most ϵ or outputs the correct solution $\mathbf{x}$.

4 Recursive block triangular factorization of a strongly nonsingular matrix

To extend the algorithms of [P02a] to computations with singular matrices M, we first apply the known divide-and-conquer MBA algorithm that recursively factorizes a preconditioned input matrix to compute and to invert its nonsingular submatrix of the maximal rank. Theorem A.2 and the algorithms in Sections 9 and 10 enable us to keep the intermediate matrices in this computation in compressed form. We assume that the computation is performed modulo a prime p or a prime power p^{g+k}. The same algorithm may also substitute for Algorithms 6.1 and 6.2 in [P02a] to compute the inverse of a nonsingular matrix modulo p^{g+k} (to initialize Hensel's lifting). For moderately large p^{g+k}, the bit cost of this computation is small because the arithmetic cost is low (see Section 8), and this enables us to improve the overall bit cost bounds of [P02a] in the nonsingular case. Furthermore, the overall bit cost is dominated at the lifting stage, and so we extend the estimates of [P02a] to singular Toeplitz-like inputs. In this and the next sections we prepare the ground for these results, that is, we recall and elaborate upon the MBA algorithm; then in Sections 9–12 we describe our compression techniques for structured matrices and estimate the bit cost of the resulting algorithms, which support the acceleration and extension.

Definition 4.1. *Let $M^{(k)}$ denote the $k \times k$ leading principal (northwestern) submatrix of a matrix M. M has generic rank profile if $M^{(k)}$ are nonsingular matrices for $k = 1, \dots, \rho$ where $\rho = \operatorname{rank} M$. A nonsingular matrix having generic rank profile is called strongly nonsingular. Define the* Schur complement *of a general 2×2 block matrix*

$$M = \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix}, \tag{4.1}$$

with nonsingular $k \times k$ block $M_{00} = M^{(k)}$ as follows:

$$S = S(M, M_{00}) = S^{(k)}(M) = M_{11} - M_{10} M_{00}^{-1} M_{01}. \tag{4.2}$$

Applying the block Gauss–Jordan elimination to M, obtain the well-known block triangular factorization

$$M = \begin{pmatrix} I & 0 \\ M_{10} M_{00}^{-1} & I \end{pmatrix} \begin{pmatrix} M_{00} & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & M_{00}^{-1} M_{01} \\ 0 & I \end{pmatrix}. \tag{4.3}$$

$$\det M = (\det M_{00}) \det S. \tag{4.4}$$

If M is nonsingular, then (4.1) implies that S is nonsingular, and we obtain

$$M^{-1} = \begin{pmatrix} I & -M_{00}^{-1}M_{01} \\ 0 & I \end{pmatrix} \begin{pmatrix} M_{00}^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -M_{10}M_{00}^{-1} & I \end{pmatrix}. \tag{4.5}$$

(4.3) implies the following theorem.

Theorem 4.2. *S^{-1} is the southwestern block of M^{-1}.*

Factorizations (4.3) and (4.5) can be recursively applied to the matrices M_{00} and S provided that the respective leading principal blocks of M_{00} and S are nonsingular. This is always the case where M is strongly nonsingular. Indeed, we first observe that Schur complementation and projection are transitive.

Theorem 4.3. *Suppose that $h > 0$, $k > l > 0$, and $M^{(k)}$ and $M^{(l)}$ are nonsingular matrices. Then a) $(S^{(l)}(M))^{(h)} = S^{(l)}(M^{(h+l)})$, b) $S^{(k-l)}(S^{(l)}(M)) = S^{(k)}(M)$.*

Now, Theorems 4.2 and 4.3 a) together imply the following corollary.

Corollary 4.4. *If an $n \times n$ matrix M is strongly nonsingular, then so are all its Schur complements $S^{(l)}(M^{(k)})$, $l = 1, \dots, k-1$; $k = 2, \dots, n$.*

Choose $M_{00} = M^{(k)}$ with $k = \lceil n/2 \rceil$, apply factorizations (4.3) and (4.5) recursively to M_{00} and S, and stop when you arrive at 1-by-1 matrices. This defines the *balanced complete recursive (block triangular) factorization* (we use the abbreviation BCRF) of a strongly nonsingular matrix M and its inverse. Let us associate the BCRF of M with a binary tree $T_{n,n}$: M is the root with two children M_{00} and S, recursive extension of (4.3) and (4.5) to every internal node N of the tree defines two diagonal blocks (which are M_{00} and S for $N = M$) represented as the two children of N in the tree. The leaves of the tree are 1-by-1 matrices. Figure 1 and 2 show two sample trees for 8×8 and 5×5 matrices M where we write $S^{(k,l)} = S^{(l)}(M^{(k)})$, $k \geq l$ (see Theorem 4.3 a) and $S^{(k,0)} = M^{(k)}$.

Algorithm 4.5. *The BCRF of a strongly nonsingular matrix.*

INPUT: *a strongly nonsingular $n \times n$ matrix M.*

OUTPUT: *M^{-1} and the BCRF of M and M^{-1}.*

COMPUTATIONS: *Define the BCRF tree $T(M) = T_{n,n}$, then recursively compute and invert all matrices associated with its nodes according to the following rules.*

1. *The left child of any node is a leading principal block of its parent.*

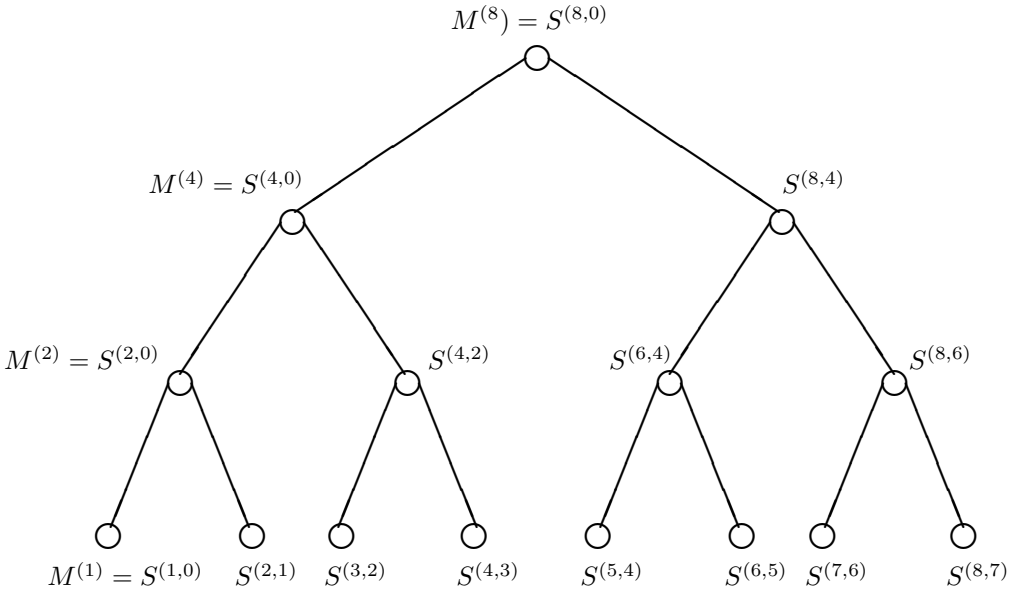

Figure 1: Generic BCRF tree $T_{8,8} = T(M)$ for an 8×8 matrix M.

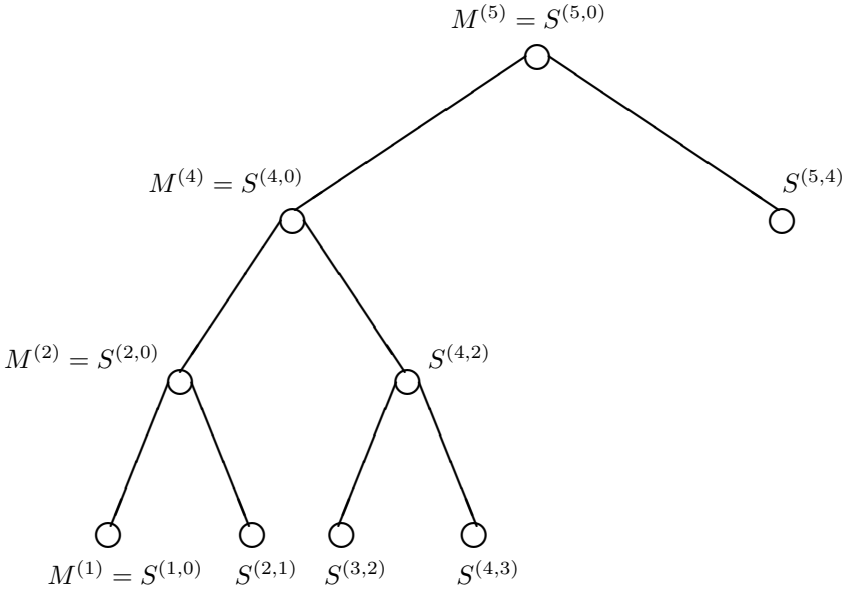

Figure 2: Generic BCRF tree $T_{5,5} = T(M)$ for a 5×5 matrix M.

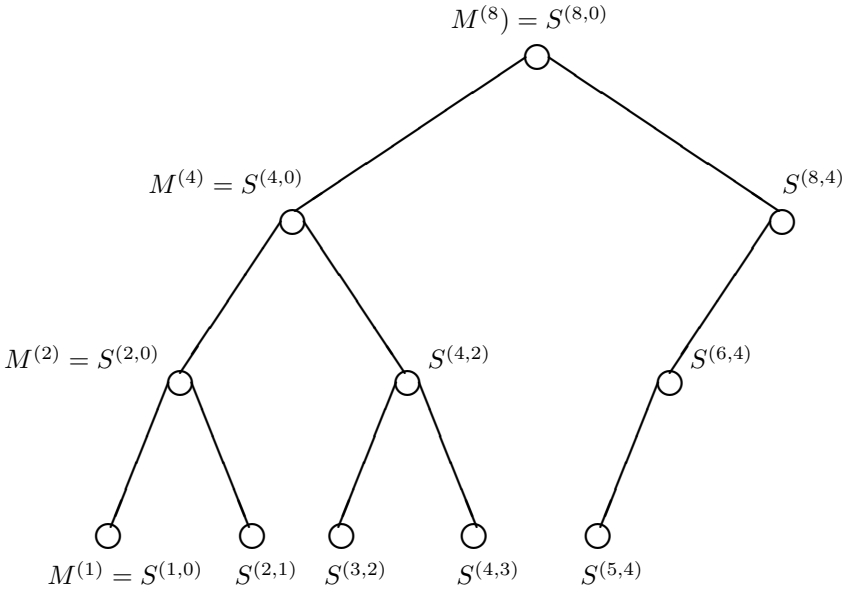

Figure 3: The BCRF tree $T(M) = T_{8,5}$ for Algorithm 4.5 applied to an 8×8 matrix M of rank 5 having generic rank profile.

2. *Invert the leaves (1-by-1 matrices) directly, then recursively invert all other nodes based on factorization* (4.5) *and its recursive extension. That is, in each recursive step first invert both children; immediately thereafter invert their parent. Inverting two siblings, proceed in the following order: invert the left sibling first, then immediately compute and invert its right sibling based on* (4.2) *or its extension.*

3. *Stop when the root M is inverted.*

It is immediately verified that the latter rules completely and correctly define the computation of the BCRF. For example, given an 8×8 matrix M in Figure 1, they define the following order (where $c(N)$ means "compute N", $i(N)$ means "invert N"):

$i(S^{(1,0)})$, $c(S^{(2,1)})$, $i(S^{(2,1)})$, $i(S^{(2,0)})$, $c(S^{(4,2)})$, $i(S^{(3,2)})$, $c(S^{(4,3)})$, $i(S^{(4,3)})$, $i(S^{(4,2)})$, $i(S^{(4,0)})$, $c(S^{(8,4)})$, $i(S^{(5,4)})$, $c(S^{(6,5)})$, $i(S^{(6,5)})$, $i(S^{(6,4)})$, $c(S^{(8,6)})$, $i(S^{(7,6)})$, $c(S^{(8,7)})$, $i(S^{(8,7)})$, $i(S^{(8,6)})$, $i(S^{(8,4)})$, $i(S^{(8,0)})$.

Due to (4.4), we immediately extend Algorithm 4.5 to computing $\det M$.

5 Extension to general input matrices

We first assume input matrices having generic rank profile. Then we relax this assumption.

Algorithm 5.1. *The BCRF of a submatrix of a matrix having generic rank profile.*

INPUT: *a matrix M having generic rank profile.*

OUTPUT: *$\rho = \operatorname{rank} M$ and the BCRF of $M^{(\rho)}$ and its inverse.*

COMPUTATIONS: *Proceed as in Algorithm 4.5 but with an aditional provision in the case where a computed leaf $S^{(k+1,k)}$ turns out to be the (1-by-1) null matrix; in this case output $\rho = k$ and the BCRF of the matrices $M^{(\rho)}$ and its inverse and stop. If $S^{(k+1,k)} \neq 0$ for all $k \leq n-1$, then write $\rho = n$ and stop when M is inverted (as in Algorithm 4.5).*

The BCRF tree $T(M) = T_{n,\rho}$ computed with Algorithm 5.1 is a subtree of the BCRF tree $T_{n,n}$ associated with Algorithm 4.5. For $\rho < n$, the computation stops where it reaches a leaf equal to 0. Then we define the output subtree as follows: first delete from $T_{n,n}$ the leaf $S^{(\rho+1,\rho)}$ together with all unprocessed leaves, that is, all leaves $S^{(i+1,i)}$ for $i \geq \rho$; then recursively delete every parent having no child. The BCRF tree $T_{8,5}$ is shown in Figure 3.

6 Extension to computing the null space and solving a singular linear system

Let us show some immediate applications of the BCRF. Assume that $\rho = \operatorname{rank} M$, $M^{(\rho)}$ is nonsingular, and ρ and $(M^{(\rho)})^{-1} = M_{00}^{-1}$ have been computed by means of Algorithms 4.5 and 5.1. Let us compute a basis for (or a vector from) the null space of M and solve a linear system $M\mathbf{x} = \mathbf{b}$ or determine its inconsistency. Write $\mathbf{v}^{(h)}$ for the subvector made up of the first h components of a vector $\mathbf{v}$ and write $\mathbf{0}_h$ for the null vector of dimension h.

General solution to a consistent linear system $M\mathbf{x} = \mathbf{b}$ is given by the vectors $\mathbf{x} = \mathbf{x}^{(0)} + \mathbf{z}$, where $\mathbf{x}^{(0)}$ is a fixed specific solution and $\mathbf{z}$ is any vector in the null space of M. Combining Algorithms 6.1 and 6.3 below supports computing such a general solution.

Algorithm 6.1. *The null space.*

INPUT: *a field $\mathbb{F}$, a matrix $M \in \mathbb{F}^{n\times n}$, $\rho = \operatorname{rank} M$ such that $\rho < n$ and $M^{(\rho)}$ is nonsingular, and the inverse of $M^{(\rho)}$.*

OUTPUT: *a basis for the null space of M.*

COMPUTATIONS: *Substitute* $M_{00} = M^{(\rho)}$ *into* (4.1). *Compute the matrix* $K = -M_{00}^{-1}M_{01}$ *and output the columns of the matrix* $N = \binom{K}{I_{n-p}}$.

Algorithm 6.2. *A nontrivial vector in the null space.*

INPUT: *as in Algorithm 6.1.*

OUTPUT: *a vector* $\mathbf{v} \neq \mathbf{0}$ *in the null space of* M.

COMPUTATIONS: *the same as in Algorithm 6.1 but restricted to a single selected column (e.g., the first column) of* M_{01}, K *and* N.

Algorithm 6.3. *A linear system.*

INPUT: *as in Algorithm 6.1, but complemented by a vector* $\mathbf{b} \in \mathbb{F}^n$.

OUTPUT: *either INCONSISTENT or a solution* $\mathbf{x}$ *to the linear system of equations* $M\mathbf{x} = \mathbf{b}$.

COMPUTATIONS: *Compute the vector* $\mathbf{x}^{(\rho)} = (M^{(\rho)})^{-1}\mathbf{b}^{(\rho)}$. *Substitute the vector* $\mathbf{x} = (\mathbf{x}^{(\rho)T}, \mathbf{0}^T)^T$ *into the linear system* $M\mathbf{x} = \mathbf{b}$. *If* $M\mathbf{x} = \mathbf{b}$, *output* $\mathbf{x}$. *Otherwise output INCONSISTENT.*

7 Yielding the generic rank profile property

If M is an $n \times n$ integer (or more generally, real) nonsingular matrix, then we may compute the BCRF of any of the two strongly nonsingular matrices M^TM or MM^T and then compute and output $M^{-1} = (M^TM)^{-1}M^T = M^T(MM^T)^{-1}$ and $(\det M)^2 = \det(M^TM) = \det(MM^T)$. In fact we also have $\|W\|_2 \geq \|W^{(k)}\|_2$, $\|W\|_2 \geq \|S(W, W^{(k)})\|_2$, $|\det W| \geq |\det W^{(k)}|$, and $|\det W| \geq |\det S(W, W^{(k)})|$ for all k if $W = M^TM$ or $W = MM^T$. The recipe does not work for singular matrices M such as $M = \left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, but various effective randomized preconditioners (see Kaltofen and Saunders 1991 [KS91], Kaltofen and Pan 1991 [KP91], Bini and Pan 1994 [BP94], [P01, Sections 5.6 and 5.7], and the bibliography therein) probabilistically ensure the generic rank profile property for any M. Let us specify a preconditioner well suited for Toeplitz/Hankel-like computation.

Theorem 7.1. *[KS91]. Let* $M \in \mathbb{R}^{n\times n}$ *for any ring* $\mathbb{R}$. *Let* S *be a finite set of cardinality* $|S|$ *in* R *or its extension. Let* L *and* U^T *be a pair of* $n \times n$ *unit lower triangular Toeplitz matrices, defined by the* $2n-2$ *subdiagonal entries of the pair of the first columns of* L *and* U^T. *Let these* $2n-2$ *entries be randomly and independently of each other selected from the set* S *under the uniform probability distribution on* S. *Then the preconditioned matrix* UML *has generic rank profile with a probability of at least* $1 - \bar{\epsilon}$, *where* $\bar{\epsilon} = \rho^2/|S|$, $\rho = \operatorname{rank} M \leq n$.

For $\mathbb{R} = \mathbb{Z}$ and a fixed positive $\bar{\epsilon}$, it is sufficient to choose

$$S = \{z \in \mathbb{Z} : |z| \le \lceil n^2/(2\bar{\epsilon})\rceil\}. \tag{7.1}$$

Under (7.1) we generate at most $(2n-2)\lceil\log(2\lceil n^2/(2\bar{\epsilon})\rceil)\rceil$ random bits and have

$$|UML| \le n^2|U| \cdot |M| \cdot |L| \le n^6|M|/(4\bar{\epsilon}^2). \tag{7.2}$$

Finally, statistics in [W02] shows that a random Toeplitz matrix in $\mathbb{Z}_{p^{g+k}}$ is likely to be strongly nonsingular already for moderately large $g+k$.

8 Arithmetic computational cost estimates

Theorem 8.1. *Let M be a Toeplitz/Hankel-like matrix given with its (shortest) displacement generator of length r. Then the arithmetic cost of performing Algorithms 4.5, 5.1, 6.1–6.3 and preconditioning is bounded as follows: $O(m(n)r^2\log n)$ with $m(n)$ in (0.1) for Algorithm 4.5 (which also covers computing $\det M$); $O(m(\rho)r^2\log\rho)$, with $\rho = \operatorname{rank} M$, for Algorithm 5.1; $O((n-\rho)m(\rho)r)$ for Algorithm 6.1; $O(m(\rho)r)$ for Algorithm 6.2; $O(m(n)r)$ for Algorithm 6.3; $O(m(n)r^2)$ for the preconditioning of M with M^T, and $O(m(n)r)$ for the Toeplitz preconditioning with U and L as in [KS91]; the latter bound does not cover the cost of generating the $(2n-2)\lceil\log(2\lceil n^2/(2\bar{\epsilon})\rceil)\rceil$ random bits involved. For Algorithm 6.1, the arithmetic cost bound can be turned into $O(m(n)r)$ if we represent the output matrix K with its short displacement generator.*

Proof. W.l.o.g., suppose that n is a power of two, let $F(n), I(n), M(n)$ and $A(n)$ be equal to the arithmetic cost for computing BCRF, matrix inversion, multiplication and addition/subtraction, respectively, assuming $n \times n$ input and nonsingularity whenever it comes to inversion. Hereafter, $M(n,q,r)$ denotes the arithmetic cost of the $n \times q$ by $q \times r$ matrix multiplication. Then

$$\begin{aligned} F(n) &\le 2F(n/2) + 3M(n/2) + I(n/2) + A(n/2),\\ I(n) &\le 2I(n/2) + 6M(n/2) + 2A(n/2). \end{aligned}$$

Substitute the bound on $I(k)$ recursively for $k = n/2, n/4, \dots, 1$; obtain that

$$\begin{aligned} F(n) = O(M(n)), \quad & I(n) = O(M(n)) \quad && \text{if } M(n) \ge Cn^{\omega},\\ F(n) = O(M(n)\log n), \quad & I(n) = O(M(n)\log n) && \text{if } M(n) \le n\log^c n, \end{aligned}$$

where c, C, and w are three constants, $C > 0$, $3 \ge \omega > 1$. If M is a Toeplitz/Hankel-like matrix given with its shortest displacement generator of length r, then in computations in Algorithms 4.5, 5.1, and 6.1–6.3 we may represent all matrices with their shortest displacement generators of length $O(r)$ based on Theorems 1.2–1.5, 2.6, 2.8, and 4.2. Likewise, preconditioning by M^T or by U and L still keeps the length of the displacement generators in $O(r)$. By

Corollary 2.7 for Toeplitz/Hankel-like matrices M given with their displacement generators of length of at most r, we have $M(n) = O(r^2 m(n))$; consequently

$$M(\rho, \rho, n-\rho) = O(m(n)r^2),\ M(\rho, \rho, 1) = O(m(\rho)r),\ M(n, \rho, 1) = O(rm(n)).$$

Thus, in the Toeplitz/Hankel case, we have for Algorithm 4.5 that

$$F(n) = O(m(n)r^2 \log n),\quad I(n) = O(m(n)r^2 \log n).$$

The same bound holds for computing $\det M$ due to (4.4) and also for Algorithm 5.1 with n replaced by ρ. The arithmetic cost of preconditioning by the matrix M^T is $M(n)$ and by the matrices U and L is in $O(nm(n))$ (see Section 7). Summarizing, we arrive at the theorem. □

Remark 8.2. *For smaller ρ and/or larger r, it may be faster to operate with the entries rather than displacement generators, e.g., we have the bounds of $O(\min\{m(n)r^2 \log n, n^\omega\})$ for Algorithm 4.5, $O(\min\{m(\rho)r^2 \log \rho, \rho^\omega\})$ for Algorithm 5.1, and so on. Here $O(n^\omega)$ bounds the arithmetic cost of $n \times n$ matrix multiplication.*

9 Compression of a Toeplitz-like matrix

Definition 9.1. *Let $m \in \mathbb{Z}$, $U \in \mathbb{Z}^{k\times l}$, and $V \in \mathbb{Q}^{k\times l}$. Let every entry of V be either 0 or the ratio of two coprimes. Then $\delta(V)$ is the least common denominator of the entries of $V \neq 0$, $\delta(0) = 1$. U has the order $s = \mathrm{ord}_m(U)$ in m if s is the maximal integer such that $m^{-s}U \in \mathbb{Z}^{k\times l}$ for $U \neq 0$, $s = \infty$ for $U = 0$. A prime p is coprime with V if $\mathrm{ord}_p(V) = 0$. Write $\tilde{V} = \delta(V)V \in \mathbb{Z}^{k\times l}$, $g_m(V) = \mathrm{ord}_m(\delta(V))$, then $\mathrm{ord}_m(V) = \mathrm{ord}_m(\tilde{V}) - \mathrm{ord}_m\, g_m(V)$. Write*

$$\begin{aligned} g &= g_p(M^{-1}), \\ g_p(L) &= g_p(1 - ef) \text{ for } L = \Delta_{A,B}, \\ g_p(L) &= g_p(e - f) \text{ for } L = \nabla_{A,B}, \end{aligned} \tag{9.1}$$

for L in Theorem 2.6 and choose the integers e and f such that $g_p(L) \leq 1$. (Then we call the operator L permissible.*)*

Problem 9.2. *Given a generator G_V, H_V of length l for a matrix $V = L(M) \in \mathbb{Z}^{n\times k}$ of rank r, compute a generator G, H of length r for V such that*

$$g_p(G) \geq 0,\quad g_p(H) \geq 0. \tag{9.2}$$

The next algorithm solves Problem 9.2, where $G_V = V$, $H_V = I$, $l = n$.

Algorithm 9.3. *Basic compression via Gaussian elimination.*

INPUT: *the entries of a matrix $V \in \mathbb{Z}^{n\times k}$ of rank r.*

OUTPUT: *a generator G, H of length r for V.*

COMPUTATIONS: *Apply Gaussian elimination with column pivoting to compute lower triangular (generally rectangular) matrices U^T and L (not to be confused with an operator L above and in Sections 1 and 2 and the matrix L in Section 7) and a permutation matrix P such that $V = LUP$ and at every elimination step the order in p of the pivot element is minimum in its row. The output matrices G and H are defined as the $n \times r$ submatrices formed by the first r columns of L and $P^T U^T$, respectively.*

Correctness follows because all diagonal entries of U are coprime with p (by construction of U), $\operatorname{rank} U \geq \operatorname{rank} L = \operatorname{rank} V = r$, and all but the first r columns of L vanish, so $LUP = V$.

The algorithm performs $O(nkr)$ arithmetic operations with ratios η/δ of pairs of coprimes η and $\delta > 0$ such that $\operatorname{ord}_p(\delta) = 0$, and if $r = O(1)$, then $\log(|\eta|\delta) = O(\log |V|)$. Here and hereafter, $|V| = |(v_{i,j})_{i,j}| = 1 + \max_{i,j} |v_{i,j}|$.

Theorem 9.4. *For $r = O(1)$, Algorithm 9.3 requires $O(nk\mu(\log |V|))$ bit operations for $\mu(d)$ in (0.2).*

Corollary 9.5. *Let L in Theorem 2.6 be a permissible operator. Let $V = L(M)$ be given with its entries and let it have rank $r = O(1)$. Then bit cost estimate (3.1) proved in [P02a] for Toeplitz matrices also holds for the above Toeplitz-like matrix $M = L^{-1}(V)$.*

10 Compression of short generators

Theorem 4.1 enables extension of the estimates in [P02a] even where a matrix V is given with its entries. Let us next introduce techniques that similarly compress a matrix V given with its short although not shortest generator. These techniques extend our Theorem 1.1 to bound also the precision of computing, which is again critically important in the MBA algorithm.

Algorithm 10.1. *Mulitiplication with basic compression.*

INPUT: *a prime p, a matrix $V \in \mathbb{Z}^{n \times k}$ of an unknown rank r for $k = O(n)$, and its rational generator G_V, H_V of length $l \geq r$.*

OUTPUT: *a shortest rational generator G, H for V satisfying* (9.2).

COMPUTATIONS:

1. *Compute $V = G_V H_V^T$.*
2. *Apply Algorithm 9.3.*

The algorithm uses $O(kln)$ arithmetic operations. To control the precision of computing at stage 1, first compute the integers

$$t_G = g_p(G_V), \quad t_H = g_p(H_V), \quad t = t_G + t_H,$$

and

$$u = 1 + \max\{t, \lceil \log_p(2|V|) \rceil\}, \tag{10.1}$$

then compute in $\mathbb{Z}_{p^u}$ the matrices $p^{t_G} G_V$, $p^{t_H} H_V$ (both are coprime with p), $p^t V = (p^{t_G} G_V)(p^{t_H} H_V^T)$, and V; finally recover V in $\mathbb{Z}^{n \times k}$ from $V \bmod p^u$. This yields the precision of $\lceil u \log p \rceil$ bits, so the bit cost in Stage 1 is in

$$B_+ = O(klnu(u \log p)). \tag{10.2}$$

The latter bound dominates the bound at stage 2 (based on Theorem 4.1).

The next algorithm applies to the same input as Algorithm 10.1 and also computes a shortest generator for V. For $l = O(1)$, it is faster by the factor of k but instead of (9.2) supports only the weaker bounds of

$$g_p(G) + g_p(H) \geq g_p(G_V) + g_p(H_V). \tag{10.3}$$

Algorithm 10.2. *Compression via repeated multiplication and basic compression.*

INPUT/OUTPUT: *as in Algorithm 10.1 but with* (10.3) *replacing* (9.2).

COMPUTATIONS:

1. *Apply Algorithm 9.3 to the input matrix* $\tilde{G_V} = \delta(G_V) G_V$ *to compute its generator* $\bar{G}, \bar{H}$ *of length* $\bar{r} = \operatorname{rank} G_V \leq l$.
2. *Compute the* $\bar{r} \times l$ *matrix* $W = \bar{H}^T H_V^T$ *and apply Algorithm 9.3 to the matrix* $\tilde{W} = \delta(W) W$ *to compute its shortest generator* $\hat{G}, H$.
3. *Compute the matrix* $G = \bar{G}\hat{G}/(\delta(G_V)\delta(\bar{H}^T H_V^T))$. *Output* G, H.

Correctness follows from correctness of Algorithm 9.3 and the observation that the matrices G and H have full rank. The arithmetic cost is $O(l^2 n)$. For $l = O(1)$, the precision of computing is $O(u \log p)$. This yields the bit cost bound

$$B = O(n\mu(u \log p)). \tag{10.4}$$

Remark 10.3. *Algorithm 10.2 supports our Theorem 1.2 for* $G_1 = G_V, H_1 = H_V$ *complemented with bounds* (10.1) *and* (10.3).

Remark 10.4. *Both Algorithms 10.1 and 10.2 can be applied to scaled integer generator* $\tilde{G_V}, \tilde{H_V}$ *and performed in* $\mathbb{Z}_{p^u}$. *From the (scaled) output generator* G, H *in* $\mathbb{Z}_{p^u}$, *the (scaled) output in* $\mathbb{Z}$ *can be recovered because* u *is large enough.*

11 The BCRF modulo a random prime

Problem 11.1. *Let UML be a randomly preconditioned matrix of Section 7, let p be a random prime in a fixed range $(x, y]$, and estimate the error/failure probability of computing the BCRF of $(UML)^{-1}$ in $\mathbb{Z}_p$.*

Algorithms in [PW02], [P02a], [WP03] and in our Sections 6–8 enable an extension of the BCRF in $\mathbb{Z}_p$ to computing rational solution of various related problems. The base moduli p or p^{g+k} can be chosen sufficiently small to have the overall bit cost dominated at the lifting stage, and so all the bit cost estimates in [P02a] are immediately extended. Furthermore, at the expense of a slowdown by the factor of $\log n$, one may lift the BCRF to compute $\det M$ and derandomize the rational number reconstruction stage.

Our next goal is to relate the likelihood of degeneration to the ranges for p, U, and L and to the overall bit cost. Due to Sections 9 and 10, we may discount the impact of the matrix compression stage. It remains to estimate the impact of the choice of p and to combine this estimate with the results of Section 7. By choosing p unsuccessfully, we may turn a divisor into zero in some division (at the leaf level) in the BCRF tree.

(4.4) and Theorems 4.2 and 4.3 imply that this occurs only if $\det((UML)^{(i)}) \bmod p$ vanishes for some i. The following simple extension of Theorem 7.1 in [P02a] bounds the probability.

Theorem 11.2. *Suppose that $g+k$ is a positive integer, a matrix $M \in \mathbb{Z}^{n\times n}$ has a rank ρ and has generic rank profile, and a prime p is randomly sampled under the uniform probability distribution in the range $(y/20, y]$, where $y = n^b \ln |M| \geq 114$, $b \geq 2$. Then M has rank ρ and has generic rank profile in $\mathbb{Z}^{n\times n}_{p^{g+k}}$ with a probability of $1 - P_{g+k}$, where $P_{g+k} < c\rho^2 n^{-b}/(g+k) = (c\rho^2 \ln |M|)/((g+k)y)$, for $c = 16\alpha/(\alpha-1) = 3.278885445\ldots$ and $\alpha = \ln 114/(16 \ln 5.7)$.*

Preconditioning in Section 7 does not affect the asymptotic bit cost bound in Theorem 11.2 except that $2n-2$ random parameters must be generated and $|M|$ should increase to $|M|n^6/(4\bar{\epsilon}^2)$. One may verify that a matrix W is the inverse of V by computing a shortest displacement generator for $WV - I$ and checking if it vanishes or not. Here we assume that the matrices W and V are given by their short displacement generators. To estimate the computational cost, one may apply Theorems 1.4, 1.5, and 8.1 and Algorithms 9.3, 10.1, and/or 10.2. The next result combines Theorems 7.1 and 11.2 and exploits Algorithms 6.1–6.3.

Theorem 11.3. *Suppose that we are given two positive numbers ϵ and $\bar{\epsilon}$ and a sufficiently large positive y and let a prime p be randomly sampled under the uniform probability distribution in the range $(y/20, y]$. Let a displacement generator G, H of length $r = O(1)$ be given in $\mathbb{Z}_p$ for an $n \times n$ matrix M. Let $\rho > 1 + n^2/(2\epsilon)$ (cf. (7.1)) and $y \geq (c/\epsilon)\rho^2 \ln(|M|n^6)/(4\bar{\epsilon}^2)) \geq 114$ for the constant c in Theorem 11.2 and $\rho = \operatorname{rank} M$. Then it is sufficient to generate $2n-2$ random integers in the range $(-\lceil n^2/(2\bar{\epsilon})\rceil, \lceil n^2/(2\bar{\epsilon})\rceil]$ of (7.1), defining two*

unit lower triangular Toeplitz matrices L and U^T, and in addition to perform $O((m(n) + m(\rho)\log\rho)\mu(\log p))$ bit operations for $m(n)$ in (0.1) and $\mu(d)$ in (0.2) in order to output either FAILURE with a probability of at most $\epsilon + \bar{\epsilon}$ or a shortest displacement generator in $\mathbb{Z}_p$ for $((UML)^{(\rho)})^{-1}$. The latter bit cost bounds also cover solving in $\mathbb{Z}_p$ a linear system $M\mathbf{x} = \mathbf{b}$ (or determining that the system is inconsistent in $\mathbb{Z}_p$), as well as computing a vector from the null space of M, and a displacement generator for a matrix whose columns form a basis for the null space of M.

12 The BCRF modulo a power of a fixed prime

Suppose we compute the BCRF of a matrix $(M/p^g)^{-1}$ in $\mathbb{Z}_{p^{3g+k}}$ for a fixed prime p and integers $g \geq 0$, $k > 0$, such that

$$g \geq g_+ = \max_{1 \leq i \leq \rho} g_p((M^{(i)})^{-1}) \tag{12.1}$$

and M has rank ρ and generic rank profile in $\mathbb{Z}$. (The latter requirement should be stated for preconditioned (sub)matrices in Section 7; we assume it for M to simplify the notation.) Under (12.1), we have $g_p((M^{(i)})^{-1}) \leq g$ for all i, and so, by Theorems 4.2 and 4.3, $g_p(V) \leq g$ for every block diagonal entry V in the BCRF of $(M/p^g)^{-1}$. To keep all block entries in (4.5) in $\mathbb{Z}^{i\times i}$, we multiply each of the three factors by p^g; then we multiply the factors together in $\mathbb{Z}_{p^{3g+k}}$. The product $p^{3g}M^{-1}$ has the order of at least $2g$ in p, so we rescale it down to p^gM^{-1}, and similarly proceed in each step of the BCRF. This enables us to keep the precision within $\lceil (3g+k)\log p \rceil$ bits during the entire BCRF process of computing the matrix $(M/p^g)^{-1} = p^gM^{-1}$.

Due to our algorithms in Sections 9 and 10, the same conclusion applies to the computations where M has displacement rank $r = O(1)$, and we operate with matrices represented in the compressed form via their displacement generators. Summarizing (for $k = 1$) we obtain the next estimates.

Theorem 12.1. *For a Toeplitz-like matrix $M \in \mathbb{Z}^{n\times n}$ of rank ρ having generic rank profile, and for an integer g in (12.1), a shortest displacement generator of the matrix $p^g(M^{(\rho)})^{-1}$ can be computed in $\mathbb{Z}_{p^{3g+1}}$ by using $O((m(n)+m(\rho)\log\rho)\mu((g+1)\log p))$ bit operations for $m(n)$ in (0.1) and $\mu(d)$ in (0.2).*

Remark 12.2. *With adding a random (Toeplitz-like) matrix of a small rank k to the input matrix M (as in Section 8 in [P02a]), one may probabilistically decrease the integer parameter g_+. With (subsequent) randomized multiplicative Toeplitz preconditioning in Section 7, Theorem 12.1 can be probabilistically extended to any matrix $M \in \mathbb{Z}^{n\times n}$. The affect of the latter preconditioning on g_+ of (12.1) (possibly nil) should be tested experimentally.*

To decrease the bit cost bound in Theorem 12.1, one should choose g closer to the unknown value g_+. To achieve this, apply the above computations for

$g = 1, 2, 4, \ldots$, doubling g every time the computation fails and stopping where either it first goes through (then, clearly, $g \le 2g_+$) or g exceeds a selected upper bound. In the latter case, one may replace p with another fixed or random prime or apply a small rank perturbation as in the above remark and repeat the computations.

Under the model of computing where the precision is not restricted and the bit cost of integer multiplication is nearly linear (see (0.2)), one may apply Theorem 12.1 for

$$p^{3g+1} > 2\|M\|^{2\rho-1}\|\mathbf{b}\|; \tag{12.2}$$

then the rational vector $(M^{(\rho)})^{-1}\mathbf{b}$ can be recovered from its value modulo p^{3g+1} via the rational number reconstruction algorithms in [PW02], [WP03], by using $O(\rho\mu(d)\log d)$ bit operation for $d = \log(2\|M\|^{2\rho-1}\|\mathbf{b}\|)$ or alternatively $O(\rho\mu(d))$ by using randomization. (With the transition to modular computations and rational numbers reconstruction at the end, we avoid the delicate and still open technical problem of bounding the magnitudes of the rational entries of the displacement generators of the auxiliary matrices.)

Corollary 12.3. *For a matrix M in Theorem 12.1 and a vector $\mathbf{b} \in \mathbb{Z}^n$, a rational solution to a consistent linear system of equations $M\mathbf{x} = \mathbf{b}$ can be computed by using $O((\rho\log d + m(n) + m(\rho)\log\rho)\mu(d))$ bit operations or alternatively $O(\rho d^2 + (m(n) + m(\rho)\log\rho)\mu(d))$ where $d = O(\log(2\|M\|^{2\rho-1}\|\mathbf{b}\|)) = O(\log\rho)$, $\rho\log d = O(m(\rho)\log\rho)$ if $\log(\|M\| + \|\mathbf{b}\|) = \rho^{O(1)}$.*

References

[ABM99] J. Abbott, M. Bronstein, T. Mulders. Fast Deterministic Computations of the Determinants of Dense Matrices, *Proc. of International Symposium on Symbolic and Algebraic Computation (ISSAC'99)*, 197–204, ACM Press, New York, 1999.

[Ba] D. J. Bernstein, Multidigit Multiplication for Mathematicians, *Advances in Applied Mathematics*, to appear. Available from http://cr.yp.to/papers.html

[BA80] R. R. Bitmead, B. D. O. Anderson, Asymptotically Fast Solution of Toeplitz and Related Systems of Linear Equations, *Linear Algebra and Its Applications*, **34**, 103–116, 1980.

[BGY80] R. P. Brent, F. G. Gustavson, D. Y. Y. Yun, Fast Solution of Toeplitz Systems of Equations and Computation of Padé Approximations, *J. Algorithms*, **1**, 259–295, 1980.

[BP94] D. Bini and V. Y. Pan, *Polynomial and Matrix Computations, v. 1: Fundamental Algorithms*, Birkhäuser, Boston, 1994.

[CFG99] G. Cooperman, S. Feisel, J. von zur Gathen, G. Havas, GCD of Many Integers, *Computing and Combinatorics, Lecture Notes in Computer Science*, **1627**, 310–317, Springer, Berlin, 1999.

[CK91] D.G. Cantor, E. Kaltofen, On Fast Multiplication of Polynomials over Arbitrary Rings, *Acta Informatica*, **28(7)**, 697–701, 1991.

[D82] J. D. Dixon, Exact Solution of Linear Equations Using p-adic Expansions, *Numerische. Math.*, **40**, 137–141, 1982.

[EGV00] W. Eberly, M. Giesbrecht, G. Villard, On Computing the Determinant and Smith Form of an Integer Matrix, *Proc. 41st Annual Symposium on Foundations of Computer Science (FOCS'2000)*, 675–685, IEEE Computer Society Press, Los Alamitos, California, 2000.

[GL96] G. H. Golub, C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1996.

[K95] E. Kaltofen, Analysis of Coppersmith's Block Wiedemann Algorithm for the Parallel Solution of Sparse Linear Systems, *Math. Comput.*, **62 (210)**, 777–806, 1995.

[KKM79] T. Kailath, S. Y. Kung, M. Morf, Displacement Ranks of Matrices and Linear Equations, *Journal of Mathematical Analysis and Applications*, **68, 2**, 395–407, 1979.

[KP91] E. Kaltofen, V. Y. Pan, Processor Efficient Parallel Solution of Linear Systems over an Abstract Field, *Proceedings of 3rd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'91)*, 180–191, ACM Press, New York, 1991.

[KS91] E. Kaltofen, B. D. Saunders, On Wiedemann's Method for Solving Sparse Linear Systems, *Proceedings of AAECC-5, Lecture Notes in Computer Science*, **536**, 29–38, Springer, Berlin, 1991.

[KS99] T. Kailath, A. H. Sayed (editors), *Fast Reliable Algorithms for Matrices with Structure*, SIAM Publications, Philadelphia, 1999.

[M74] M. Morf, Fast Algorithms for Multivariable Systems, Ph.D. Thesis, *Department of Electrical Engineering, Stanford University*, Stanford, CA, 1974.

[M80] M. Morf, Doubling Algorithms for Toeplitz and Related Equations, *Proceedings of IEEE International Conference on ASSP*, 954–959, IEEE Press, Piscataway, New Jersey, 1980.

[MC79] R. T. Moenck, J. H. Carter, Approximate Algorithms to Derive Exact Solutions to Systems of Linear Equations, *Proceedings of EUROSAM, Lecture Notes in Computer Science*, **72**, 63–73, Springer, Berlin, 1979.

[MSa] T. Mulders, A. Storjohann, Certified Dense Linear System Solving, Preprint, 2001.

[MS99] T. Mulders, A. Storjohann, Rational Solution of Singular Linear Systems, *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC'99)*, 242–249, ACM Press, New York, 1999, Preprint, 2001.

[P87] V. Y. Pan, Complexity of Parallel Matrix Computations, *Theoretical Computer Science*, **54**, 65–85, 1987.

[P88] V. Y. Pan, Computing the Determinant and the Characteristic Polynomials of a Matrix via Solving Linear Systems of Equations, *Information Processing Letters*, **28**, 71–75, 1988.

[P90] V. Y. Pan, On Computations with Dense Structured Matrices, *Mathematics of Computation*, **55 (191)**, 179–190, 1990.

[P92] V. Y. Pan, Parametrization of Newton's Iteration for Computations with Structured Matrices and Applications, *Computers and Mathematics (with Applications)*, **24(3)**, 61–75, 1992.

[P00] V. Y. Pan, Parallel Complexity of Computations with General and Toeplitz-like Matrices Filled with Integers and Extensions, *SIAM J. Comput.*, **30 (4)**, 1080–1125, 2000.

[P01] V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Birkhäuser/Springer, Boston/NewYork, 2001.

[P02] V. Y. Pan, Can We Optimize Toeplitz/Hankel Computations? *Proc. of the Fifth International Workshop on Computer Algebra in Scientific Computing (CASC'02)*, Yalta, Crimea, Sept. 2002 (E. W. Mayr, V. G. Ganzha, E. V. Vorozhtzov, Editors), 253–264, Technische Universitat München, Germany, 2002.

[P02a] V. Y. Pan, Nearly Optimal Toeplitz/Hankel Computations, *TR 2002 017*, PhD Program in Computer Science, Grad. Center of City University of New York, 2002.

[PW02] V. Y. Pan, X. Wang, Acceleration of Euclidean Algorithm and Extensions, *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC'02)*, 207–213, ACM Press, New York, 2002.

[PW03] V. Y. Pan, X. Wang, Inversion of Displacement Operators, *SIAM Journal on Matrix Analysis and Applications*, **24, 3**, 660–677, 2003.

[PZ00] V. Y. Pan, A. Zheng, Superfast Algorithms for Cauchy-like Matrix Computations and Extensions, *Linear Algebra and Its Applications*, **310**, 83–108, 2000.

[W02] X. Wang, How frequently is a matrix nonsingualr? *TR 2002 018*, PhD Program in Computer Science, Grad. Center of City University of New York, 2002.

[WP03] X. Wang, V. Y. Pan, Acceleration of Euclidean Algorithm and Rational Number Reconstruction, *SIAM Journal on Computing*, **32,2**, 548556, 2003.

A Appendix

Theorem A.1. *For any 4-tuple of scalars (a, b, e, f) and any matrix M, the matrices $\Delta_{A(a),B(b)}(M) - \Delta_{A(e),B(f)}(M)$ and $\nabla_{A(a),B(b)}(M) - \nabla_{A(e),B(f)}(M)$ have rank of at most 2, provided that $A(u) = Z_u$, $B(v) = Z_v$; $A(u) = Z_u$, $B(v) = Z_v^T$; $A(u) = Z_u^T$, $B(v) = Z_v$, or $A(u) = Z_u^T$, $B(v) = Z_v^T$.*

Proof. Combine the relations

$$\Delta_{A,B}(M) - \Delta_{E,F}(M) = EM(F - B) + (E - A)MB;$$
$$\nabla_{A,B}(M) - \nabla_{E,F}(M) = (A - E)M + M(F - B);$$
$$\text{rank}(A - E) \le 1 \text{ if } A = Z_a, E = Z_e \text{ or if } A = Z_a^T, E = Z_e^T;$$
$$\text{rank}(F - B) \le 1 \text{ if } B = Z_b, F = Z_f \text{ or if } B = Z_b^T, F = Z_f^T.$$

□

Theorem A.2. *[P01, Theorem 1.3.1]. If an operator matrix A is nonsingular, then $\nabla_{A,B}(M) = A\Delta_{A^{-1},B}(M)$. If an operator matrix B is nonsingular, then $\nabla_{A,B}(M) = -\Delta_{A,B^{-1}}(M)B$.*