Smith Normal Form of Dense Integer Matrices, Fast Algorithms into Practice *

David Saunders
University of Delaware
Newark, DE, USA
saunders@udel.edu

Zhendong Wan
University of Delaware
Newark, DE, USA
wan@udel.edu

ABSTRACT

We present a variation of the fast Monte Carlo algorithm of Eberly, Giesbrecht and Villard for computing the Smith form of an integer matrix. It is faster in practice, but with the same asymptotic complexity, and it also handles the singular case. Then we will apply the key principle to improve Storjohann's algorithm and Iliopoulos' algorithm. We have a soft linear time algorithm for the special case of a diagonal matrix. A local Smith form Algorithm is also considered.

We offer analysis and experimental results regarding these algorithms, with a view to the construction of an adaptive algorithm exploiting each algorithm at it's best range of performance. Finally, based on this information, we sketch the proposed structure of an adaptive Smith form algorithm for matrices over the integers. Our experiments use implementations in *LinBox*, a library for exact computational linear algebra available at `linalg.org`.

Categories and Subject Descriptors

J [**2**]: *mathematics*

General Terms

Algorithms, Performance

Keywords

Integer matrix, Smith normal form

1. INTRODUCTION

The Smith form plays an important role in the study of homology groups, systems theory and other areas. In 1861, Smith[17] proved that any integer matrix can be diagonalized using elementary row and column operations. That is, for every integer matrix A, there exist unimodular matrices U and V, and a unique diagonal matrix S with non-negative diagonal entries s_i, for $i = 1, \cdots, n$, such that $A = USV$, and $s_i | s_{i+1}$, for $i = 1, \cdots, n-1$. The matrix S is called the Smith normal form or simply Smith form of A, these s_i are called the invariant factors of A. Here an integer matrix is called unimodular if and only if it's determinant is a unit, ie 1 or -1.

Over the past thirty years, steady progress has been made in algorithms for the Smith forms. The asymptotically fastest algorithm for Smith forms given to date is in [14]. It requires a remarkably small $\mathrm{O}^{\sim}(n^{2.697263} \log ||A||)$ bit operations, but it may not be practical.

In this paper, our starting point is a variation of the Eberly, Giesbrecht and Villard Smith form algorithm[11], which we believe is important in the practical view. We make some improvements and extend it to the singular case. In the original algorithm, it only handles nonsingular matrices. After that, we will review some previous algorithms and point out improvements which are beneficial in practice. In particular, we will review algorithms of Storjohann[18], Iliopoulos/Hafner&McCurley[13, 12], and Dumas, et al[8]. We will provide a mixture of improvements, analysis, and some interesting experiments, all working toward the design of a synthesis of these algorithms into an adaptive algorithm which is fastest yet in practice, approaching optimal asymptotic complexity and with very good constant factors.

If $\exists c$, such that $f(n) = \mathrm{O}(g(n) \log^c(g(n)))$, we say $f(n) = \mathrm{O}^{\sim}(g(n))$. Also $\|A\|$ denotes $\max\{|A_{i,j}|\}$, and it's log bounds the bit length of the entries. We will use ω to denote the exponent for matrix multiplication. Using standard matrix multiplication, $\omega = 3$, while the best known algorithm of Coppersmith and Winograd[6] allows $\omega = 2.376$.

2. MODIFIED EGV ALGORITHM

The Smith form algorithm of Eberly, Giesbrecht, and Villard[11] (EGV for short) for nonsingular integer matrices computes first the last invariant factor, and then the i^{th} invariant factors in a binary search.

The EGV algorithm, and our variant, requires much more effort to get the right values with respect to small prime factors than with respect to larger prime factors. By contrast modular methods are fastest when the modulus is small. Our adaptive algorithm will exploit this tradeoff. Thus we express the next two invariant factor finding algorithms in terms of a threshold on the size of primes being correctly handled. Borrowing the *smooth/rough* adjectives from number theory, a number can be written uniquely as a product

*Research was supported by National Science Foundation Grants CCR-0112807 and CCR-0098284.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISSAC'04, July 4–7, 2004, Santander, Spain.
Copyright 2004 ACM 1-58113-827-X/04/0007 ...$5.00.

of a smooth part (product of small primes) and a rough part (product of large primes). More precisely, we call a number *k-smooth* if it is a product of primes smaller than k and *k-rough* if it is a product of primes no less than k. These invariant factor algorithms compute the k-rough part of the invariant factor. Note that 1-rough part of an integer is itself. However, as we discuss later, it is generally better in practice to choose k a bit larger and compute the smooth part by a different method such as Iliopoulos' or local elimination.

We will start with a new algorithm to find the k-rough part of the last invariant factor. Then we will describe a new algorithm to find the k-rough part of an arbitrary i^{th} invariant factor of an $n \times m$ integer matrix, where $1 \leq i \leq$ rank (A). It may be used to compute the largest invariant factor of a singular matrix, and is also faster than EGV's when i is not too large.

Then we will present an algorithm similar to and extending the EGV algorithm to compute the Smith forms of integer matrices, which may be singular or nonsingular.

2.1 Computing the last invariant factor of an integer matrix

Let $s_i(A)$ denote the i^{th} invariant factor of matrix A.

We first extend the [11, Algorithm: LargestInvariantFactor] to singular matrices

ALGORITHM 1. *LastInvariantFactor*
Input:

- *A, an $n \times n$ integer matrix.*
- *$k \geq 1$, a smooth/rough threshold.*
- *ϵ, an error probability requirement.*

Output:

- *The k-rough part of the n^{th} invariant factor of A. Monte Carlo with error probability at most ϵ.*

Procedure:

1. *Compute r = rank of A. Monte Carlo with error probability at most $\epsilon/2$.*
2. *If $r < n$, then return* 0.
3. *Compute result with error probability at most $\epsilon/2$, by [11, Algorithm: LargestInvariantFactor] or [16, Algorithm Leading Smith Factor].* In EGV the error probability is 2/3, but we can achieve any error probability by repeating the loop [11, steps 3 - 6 in Algorithm: LargestInvariantFactor] enough times.
4. Return the k-rough part of s.

We will see the bad impact of small primes in the proof, and again later when computing the i^{th} invariant factor.

THEOREM 1. *Given a positive constant k, error bound ϵ, and $n \times n$ integer matrix A, there exists a Monte Carlo algorithm which computes the k-rough part of the last invariant factor of A with error probability at most ϵ and requires*

$$O(n^3(\log n + \log ||A||) \log(\log n + \log ||A||) \log(1/\epsilon))$$

bit operations.

PROOF. Algorithm LastInvariantFactor meets the requirement.

Rank can be computed Monte Carlo in $O\left(n^3 M(b) \log(1/\epsilon)\right)$ time with error probability at most $\epsilon/2$, where $M(b)$ is the cost of basic arithmetic operations mod $p < b$, and b is described below. To see this, first note that from Hadamard's bound, the number of prime factors in any one $r \times r$ minor of A is at most $r(\log r + \log ||A||) \leq n(\log n + \log ||A||)$. If A has rank r then it has a $r \times r$ minor with nonzero determinant. Fix such a minor. From number theory, we know $p_n < n(\ln n + \ln \ln n - \frac{1}{2})$ if $n \geq 20$, where p_n is the n^{th} prime number - eg. see [19, Theorem:18.7]. Let $s = 2n(\log n + \log ||A||)$. If we choose $b = 2s\ln(s) + 72 > p_s$, then there are at least s primes in $[0, \cdots, b-1]$. We choose a prime at random less than b. The probability that our random prime is not a factor of this determinant is at least 1/2. And for those primes p, A has rank r in $\boldsymbol{Z}_p$. We need to repeat $(\log(2/\epsilon))$ times in order to meet the $\epsilon/2$ error probability requirement. By using FFT -see [19, Section:8.3], $M(b) = \mathrm{O}(\log b \log\log b)$, and we will assume it in the rest.

[11, Algorithm:LargestInvariantFactor] with 2 iterations computes the last invariant factor of a nonsingular matrix with error probability at most 2/3. For small k, $k < 10$, we can repeat $\log(2/\epsilon)/\log(1.5)$ to achieve $\epsilon/2$ error probability. For big k, $k >= 10$, If we repeat loop step 3 - 6, r times, the error probability for the k-rough part will be at most

$$\begin{array}{ll} & \sum_{p \geq k} (\frac{1}{M} \lceil \frac{M}{p} \rceil)^r \\ \leq & \sum_{k \leq p \leq M} (\frac{1}{M} \lceil \frac{M}{p} \rceil)^r + \sum_{M < p} (\frac{1}{p})^r \\ \leq & \sum_{k \leq p \leq M} ((\frac{1}{M})^r + (\lceil \frac{M}{p} \rceil)^r) 2^{r-1} + \sum_{M<p} (\frac{1}{p})^r \\ \leq & 2^{r-1} (\sum_{k \leq p \leq M} (\frac{1}{M})^r + \sum_{k \leq p} (\frac{1}{p})^r) \\ \leq & 2^{r-1} ((\frac{1}{M})^{r-1} + \frac{1}{r-1} (\frac{1}{k-1})^{r-1}). \end{array}$$

If we choose $M > k$, the the error probability for r iterations, will be at most $\frac{r}{r-1}(\frac{2}{k-1})^{r-1}$. If we choose $r = \log(4/\epsilon)/\log((k-1)/2)$, then the error probability will be at most $\epsilon/2$. Each iteration requires $\mathrm{O}(n^3 d \log d)$ bit operations[1], where $d = \log n + \log ||A||$. So Algorithm LastInvariantFactor requires $\mathrm{O}(n^3 d \log d \log(1/\epsilon))$ bit operations. Note that if if k is reasonably large, the number of iterations in the algorithm can be much fewer than if the contribution of small primes is also required. This is important from a practical point of view. In our experiments we see dramatically the bad impact of those small primes. □

2.2 Computing the i^{th} Invariant Factor of an integer matrix

In [11, Algorithm:OneInvariantFactor] the idea is to obtain the i-th invariant factor by solving a rational linear system $A'x = b$, in which the denominator of the solution vector x is likely to reveal s_i. This is done using a preconditioning: $A' = A + B$, where random B has rank $n - i$. It also requires a precomputed largest invariant factor. Here we propose an alternative preconditioning: $A' = LAR$, where random L is $i \times n$ and random R is $m \times i$. With this preconditioning, we do not require $m = n$ or a precomputed last invariant factor. Here is outline of our algorithm.

ALGORITHM 2. *i^{th}InvariantFactor (Outline)*
Input:

- *A, an $n \times m$ integer matrix, w.l.g. $m \leq n$.*

[1]Thanks to E. Kaltofen, who pointed out a log factor could be removed by using fast Chinese remainder algorithm.

- *i, an integer, $1 \le i \le$ rank(A).*
- *$k \ge 1$, a smooth/rough threshold.*
- *ϵ, an error probability requirement.*

Output:

- *s_i, the k-rough part of the i^{th} invariant factor of A, with error probability at most ϵ.*

Procedure:

1. *set $s_i := 0$;*
2. *repeat the following steps r times. The required r is a function of the input parameters, discussed in the proof of theorem 2.*
   a. *Randomly choose $i \times n$ matrix L.*
   b. *Randomly choose $m \times i$ matrix R.*
   c. *Compute s_i', k-rough part of the last invariant factor of LAR.*
   d. *s_i := GCD(s_i, s_i').*
3. *return s_i.*

Comment: This algorithm will be faster than the similar algorithm in [11], when i is small, since we solve an $i \times i$, rather than $n \times n$, linear system in order to find the i^{th} invariant factor. On the other hand, when i is large, it is slower due to the higher cost of computing the preconditioned matrix LAR. See section 4.2, especially figure 4, for experimental measurements.

THEOREM 2. *Given a positive constant k, there exists a Monte Carlo algorithm which, for $i \le$ rank(A), computes the k-rough part of the i^{th} invariant factor of an integer matrix A with error probability at most ϵ and requires*

$$O(mni(\log n + \log ||A||) \log(\log n + \log ||A||) \log^2(1/\epsilon))$$

bit operations.

The proof appears after these four lemmas, which will contribute to it. In these lemmas, we assume L, A, R are $i \times n$,$n \times m$, $m \times i$ integer matrices, respectively.

LEMMA 1. *$s_i(LAR)$ is always a multiple of $s_i(A)$.*

PROOF. If $s_i(A) = 0$, then rank$(A) < i$, so rank$(LAR) \le$ rank$(A) < i$, thus $s_i(LAR)$ is 0. If $s_i(A) = 1$, obviously the lemma is true. Otherwise, let us have a look at the Smith form of LA and A mod $s_i(A)$. A over $\boldsymbol{Z}_{s_i(A)}$ has the Smith form S = diagonal$(s_1, \ldots, s_r, 0, \ldots, 0)$, for some $r < i$. So there exists corresponding integer matrices U and V, such that and det(U) and det(V) are coprime to $s_i(A)$, $A = USV$. So $LA \equiv LUSV \equiv \begin{bmatrix} L'_{11}S_{11} & 0 \\ 0 & 0 \end{bmatrix} V$ mod $s_i(A)$, where L'_{11}, S_{11} are the first $r \times r$ principal submatrices of LU and S, respectively. Since det(V) is coprime to $s_i(A)$, $\begin{bmatrix} L'_{11}S_{11} & 0 \\ 0 & 0 \end{bmatrix} V$ has the same Smith form as $\begin{bmatrix} L'_{11}S_{11} & 0 \\ 0 & 0 \end{bmatrix}$ mod $s_i(A)$. Since $r < i$, $s_i(\begin{bmatrix} L'_{11}S_{11} & 0 \\ 0 & 0 \end{bmatrix}) = 0$ mod $s_i(A)$. So $s_i(LA)$ is a multiple of $s_i(A)$. For a similar reason, $s_i(LAR)$ is a multiple of $s_i(LA)$. So $s_i(LAR)$ is a multiple of $s_i(A)$. $\square$

The following lemma explores the sufficient condition for $p \nmid \frac{s_i(LAR)}{s_i(A)}$.

LEMMA 2. *For any given prime p, there exists an $n \times i$ full column rank matrix M such that for all L, $p \nmid \frac{s_i(LA)}{s_i(A)}$ only if det$(LM) \neq 0$ mod p.*

PROOF. First let us look at the local Smith forms of A and LA at prime p. There exist integer matrices U, V and D, such that, $A = UDV$, det$(U) \neq 0$ mod p, det $(V) \neq 0$ mod p, and D is the local Smith form of A at p. Represent U, D as block matrices, $U = \begin{bmatrix} M & M' \end{bmatrix}$, $D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}$, where M is an $n \times i$ matrix, D_1 is an $i \times i$ matrix. $LA = \begin{bmatrix} LMD_1 & LM'D_2 \end{bmatrix} V$. If det$(LM) \neq 0$ mod p and since the entries of D_2 are multiples of those in D_1, it follows that D_1 is the local Smith form of LA at p. Note that M has full column rank in $\boldsymbol{Z}_p$. $\square$

By the lemma above, we know $p \nmid \frac{s_i(LAR)}{s_i(A)}$ only if det(LM_1) $\neq 0$ mod p and det$(M_2R) \neq 0$ mod p, where M_1, M_2 are the matrices when lemma 2 is applied to A and LA, respectively. Note this sufficient condition for $p \nmid \frac{s_i(LAR)}{s_i(A)}$ is similar to the sufficient condition in [11, Section:3.1]. We will use this sufficient condition in the following two lemmas exploring the probabilty $p \nmid \frac{s_i(LAR)}{s_i(A)}$ for random matrices L and R.

LEMMA 3. *For an even positive integer β, if the entries of integer matrix L and R are selected uniformly and independently from the set of integers between 0 and $\beta - 1$. Then for any prime p, if $p \le \frac{\beta}{21i^2}$, then $p \nmid \frac{s_i(LAR)}{s_i(A)}$ with probability at least $\frac{2}{25}$ if $p = 2$; $\frac{1}{5}$ if $p = 3$; $\frac{1}{2}$ if $p = 5$; and $1 - \frac{11}{5(p-1)}$ if $p \ge 7$.*

This is the result [11, Lemma:3.4]. And the next lemma follows from the proof of [11, Lemma:3.7].

LEMMA 4. *For a positive integer β, if the entries of integer matrix L and R are selected uniformly and independently from the set of integers between 0 and $\beta - 1$. Then for any prime p, $p > \beta$, then $p \nmid \frac{s_i(LAR)}{s_i(A)}$ with probability at least $1 - \frac{2i}{\beta}$.*

PROOF OF THEOREM2. The i^{th}InvariantFactor algorithm meets the requirement. The proof is similar to proof in [11, Section 3].

By lemma 1, we know at any stage the k-rough part of $s_i(A)$ divides the algorithm's variable s_i. Given an even integer β, for those primes $p \le \frac{\beta}{21i^2}$, if the entries of the integer matrices L and R are selected uniformly and independently from the set of integers between 0 and $\beta - 1$ we obtain an upper bound for Prob$(p \mid \frac{s_i}{s_i(A)})$ using Lemma 3. Let k be given and let E be the event that some prime p in the range $k \le p \le \frac{\beta}{21i^2}$ divides $\frac{s_i}{s_i(A)}$. If we repeat step 2 a total of r times, $r \ge 3$, and with L and R entries chosen in $0 \ldots \beta - 1$, we have Prob$(E) \le (\frac{23}{25})^r + (\frac{4}{5})^r + (\frac{1}{2})^r + \sum_{7 \le p} (\frac{11}{5(p-1)})^r$ $\le 3(\frac{23}{25})^r + \frac{5}{r-1}(\frac{11}{25})^r \le 4(\frac{23}{25})^r$. So we can choose $r = \log(12/\epsilon)/\log(25/23)$ to ensure that Prob$(E) < \epsilon/3$. For $k > 5$, if we r times repeat step 2 with L and R as above, we have Prob$(E) \le \sum_{k \le p} (\frac{11}{5(p-1)})^r \le \frac{k-2}{r-1}(\frac{11}{5(k-2)})^r$. We need at most $r = \log(3/\epsilon)/(\log(\frac{5(k-2)}{11}))$ iterations to achieve $\epsilon/3$ error probability for the k-rough part. The notes at the end of the proof of Theorem 1 also apply here.

For those primes $p > \frac{\beta}{21i^2}$, let the entries of integer matrix L and R be selected uniformly and independently from the set of integers between 0 and $\frac{\beta}{21i^2} - 1$. Let E' be the event that some prime $p > \frac{\beta}{21i^2}$ divides $\frac{s_i(LAR)}{s_i(A)}$. We have $\text{Prob}(E') \leq \frac{42i^3}{\beta}$, by lemma 4. Note that, after a first trial with entries in $[0, \cdots, \beta - 1]$, there are at most $\log(\det(LAR)) \leq i(\log i + \log ||(LAR)||)$ distinct primes that divides $s_i(A)$, since $s_i(LAR)$ is a multiple of $s_i(A)$. Also note that $\log ||(LAR)|| \leq \log(n^2\beta^2) + \log ||A||$. Then $\text{Prob}(E') \leq \frac{42i^3}{\beta} i(\log i + \log(n^2\beta^2) + \log ||A||) \leq \frac{42i^4}{\beta}(3\log n + 2\log\beta + \log ||A||)$. We can choose $\beta = 164 \max\{16, 11\log n + \log ||A||\} i^4$, such that $\frac{42i^4}{\beta}(3\log n + 2\log\beta + \log ||A||) < 1/2$. If we repeat step 2 with the choices above for L and R for r' times, then $\text{Prob}(E') \leq (\frac{1}{2})^{r'}$. If we choose $r' = \log(3/\epsilon)/\log 2$, then $\text{Prob}(E') < \epsilon/3$. From our experiments, these additional trials to ensure no extra big primes are not necessary in practice.

Also if we compute the the k-rough part of the last invariant factor of LAR with our algorithm LastInvariantFactor with input error probability $\frac{\epsilon}{3r+3r'}$, obviously, the algorithm i^{th}InvariantFactor will compute the k-rough part of the i^{th} invariant factor of A with error probability at most ϵ. Each iteration requires $O(mni(\log n + \log ||A||) \log(\log n + \log ||A||) \log(1/\epsilon))$ bit operations, so the total cost is $O(mni(\log n + \log ||A||) \log(\log n + \log ||A||) \log^2(1/\epsilon))$ bit operations. □

2.3 Putting them together: the Smith form

Without loss of generality, we assume that A is an $n \times m$ integer matrix, where $m \leq n$.

ALGORITHM 3. *SmithForm* [variation of EGV's]
Input:

- *A, a $n \times m$ integer matrix, w.l.g. $m \leq n$.*
- *k, a smooth/rough threshold.*
- *ϵ, an error probability requirement.*

Output:

- *$S = (s_1, \ldots, s_m)$, array of the k-rough parts of invariant factors of A with error probability at most ϵ.*

Procedure:

1. *Compute r := rank(A).* [Monte Carlo with error probability at most ϵ/m].
2. *Compute s_1 = the k-rough part of $GCD_{1\leq i\leq n, 1\leq j\leq m}(A_{ij})$.*
3. *Compute $s_r = i^{th}$InvariantFactor($A, r, k, \epsilon/r$).* [Monte Carlo with error probability at most ϵ/r].
4. For i from 2 to $r-1$ do (s_i) := UNKNOWN.
   For i from $r+1$ to m do $s_i := 0$.
5. Call *SmithFormBinarySearch*$(A, 1, r, k, \epsilon/r, S)$ [defined below].
6. Return $(s_1, \ldots, s_m)$.

Note if $k = 1$, then the algorithm above will compute the Smith form of A with error probability at most ϵ. And this algorithm can handle the case of a singular matrix, while the algorithm (EGV) in [11] from which we have developed it treats only the non-singular case. Here we have the same asymptotic running time. In practice, we can expect some speedup since, for smaller i, we find the s_i faster.

FUNCTION 1. *SmithFormBinarySearch*
Input:

- *A, an $n \times m$ integer matrix, $m \leq n$.*
- *i, j, integers, and $1 \leq i, j \leq$ rank(A).*
- *k, a smooth/rough threshold.*
- *ϵ, error probability requirement.*
- *Array of invariant factors S, with s_i and s_j already defined.*

Output:

- *$(s_i, \ldots, s_j)$ are now defined, with s_l = the k-rough part of the i^{th} invariant factor of A, for $i \leq l \leq j$.*

Procedure:

1. *If $i + 1 \geq j$ return.*
2. *If $(s_i = s_j)$*
   For l from $i+1$ to $j-1$ do $s_l := s_i$.
   return.
3. *$mid := \lfloor (i+j)/2 \rfloor$.*
4. *if ($mid < .6m$)* [The magic number .6 is explained in section 4.4.] *Compute k-rough part of s_{mid} by our method,*
   else
   Compute the k-rough part of s_{mid} by EGV's method.
   [Either way it is Monte Carlo with error probability at most ϵ].
5. *Call* SmithFormBinarySearch$(A, i, mid, k, \epsilon, S)$.
   Call SmithFormBinarySearch$(A, mid, j, k, \epsilon, S)$.
6. *Return S.*

THEOREM 3. *Given a positive integer k, there exists a Monte Carlo algorithm which will compute the k-rough part of the Smith form of A with error probability at most ϵ, and requires $O(n^{3.5}(\log n + \log ||A||)^{1.5} \log(\log n + \log ||A||) \log n$ $(\log(1/\epsilon) + \log n)^2)$ bit operations.*

PROOF. The algorithm SmithForm above meets the requirements. The algorithm is essentially doing a binary search for the distinct k-rough part invariant factors of A. Integer matrix A has at most $O(n^{1/2}(\log n + \log ||A||)^{1/2})$ distinct invariant factors, see [11, Theroem 4.1]. Computing the k-rough part of an i^{th} invariant factor of A with error probability at most ϵ/r requires $O(mni(\log n + \log ||A||) \log(\log n + \log ||A||) (\log(1/\epsilon) + \log n)^2)$ bit operations. Thus $O(n^{3.5}(\log n + \log ||A||)^{1.5} \log(\log n + \log ||A||) \log n(\log(1/\epsilon) + \log n)^2)$ bit operations are required in total.

If the rank is less than or equal to 1, the error probability is at most $\epsilon/m \leq \epsilon$. Otherwise, the first invariant factor will be computed exactly without any error and the i^{th}InvariantFactor algorithm computes each other distinct invariant correctly with error probability at most ϵ/r, on the condition that it computes the rank correctly. So the algorithm computes all distinct invariant factors correctly with error probability at most $1-(1-\epsilon/r)^{r-1}$, again on the condition that it computes the rank correctly. Thus algorithm will return the Smith form of A with error probability at most $1-(1-\epsilon/r)^r \leq \epsilon$. □

If fast matrix multiplication is available, we can use it for a fast algorithm to compute the Smith form of a singular matrix. This is done by modifying algorithm [11, FastInvariantFactors] As we have done above, replacing their algorithm to find the i^{th} invariant factor with our i^{th}InvariantFactor algorithm.

3. OTHER SMITH FORM ALGORITHMS

The idea of computing the last invariant factor first may also be used to improve algorithms of Storjohann[18] and Iliopoulos/Hafner&McCurley[13, 12]. These are good candidates for computing the smooth part of the invariant factors.

3.1 Improving Iliopoulos' and Storjohann's

ALGORITHM 4. *SmithForm* [variation of Storjohann's / Iliopoulos' algorithm]
Input:

- *A, a $n \times m$ integer matrix, $m <= n$.*

Output:

- *$S = (s_1, \ldots, s_m)$, the invariant factors of A.*

Procedure:

1. *Compute the rank of A, $r := Rank(A)$*
2. *Compute s'_r, the last invariant factor of a non-singular $r \times r$ minor of A* [s_r divides s'_r].
3. *Use Storjohann's / Iliopoulos' elimination method to compute the Smith form of A mod s'_r.*

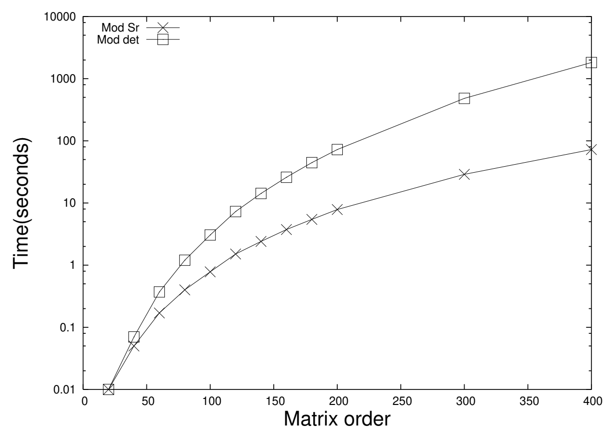
Storjohann's algorithm requires $O^\sim(n^\omega \log(d))$ bit operations, and Iliopoulos' algorithm requires $O(n^4(\log d)^2)$ bit operations, where d is the modulus used (determinant or the largest invariant factor). Always $d = O^\sim(n \log ||A||)$.

Correctness of the two above algorithms is a corollary of the following statement. Suppose A is an $n \times m$ integer matrix, and d is an integer, if let $S = \mathrm{diag}(s_1, \ldots, s_r, 0, \ldots, 0)$ be the Smith form of A over the integers, and let $S' = \mathrm{diag}(s'_1, \ldots, s'_r, 0, \ldots, 0)$ be the Smith form of A modulo d, where r is the integer rank of A, then s'_i is an associate of $\gcd(s_i, d)$. This follows since, over a principal ideal ring, any matrix has a unique Smith form, c.f.[5, Theorem:15.24].

Note that s'_r may be much less than the determinant and the cost of it's computation is comparable to computation of the determinant, in the worst case. Note that J. Abbott and M. Brostein and T. Mulders in [1] showed a fast way to the determinant by computing the last invariant factor first.

Iliopoulos' algorithm done mod the last invariant factor can be very efficient in practice. For example, figure 1 compares the run times of the determinant and last invariant factor versions on matrices randomly equivalent to a diagonal matrix with entries random in $1 \cdots 100$. Such a matrix has a determinant about $2n$ digits long and a last invariant factor, the lcm of the n 2-digit entries, that is generally much smaller. We have also noted that the last invariant factor phase only takes around 10% of the total running time.

Figure 1: Comparison between two elimination phases modulo s_r and modulo det by Iliopoulos' elimination method. Smaller is better.



3.2 Smith form of diagonal matrices

Iliopoulos' elimination produces an equivalent diagonal matrix. Computation of the Smith form of this diagonal matrix is rather straightforward using n^2 (extended) gcd computations[13], a cost that is much below the elimination to diagonal form. Nonetheless, in this section we present a faster method to compute the Smith form of a diagonal matrix. It is soft linear in the size of the input. This speedup may be significant in some settings. In any case, it is an interesting and easily implemented method.

Given a matrix $A = \mathrm{diag}(a_1, \ldots, a_n)$ we show how to compute its Smith form. Define the input size to be $N = \sum_{i=0}^{n}(\mathrm{len}(a_i))$, where $\mathrm{len}(a) = 1 + \log(|a|)$. The run time is soft linear in N, $O^\sim(N)$. A basic observation is that diag(a, b) is equivalent to $\mathrm{diag}(g, l)$, where $g = \gcd(a, b)$ and $l = \mathrm{lcm}(a, b)$. Furthermore, the cost of this operation is $O(G(\mathrm{len}(a) + \mathrm{len}(b)))$, where G(m), the cost of gcd/lcm computation size m numbers, is $O(m \log^2(m) \log\log(m))$ using the fast gcd [19]. Also note that the memory occupied by g and l, is the same as that for a and b since $ab = gl$, and thus $\log(a) + \log(b) = \log(g) + \log(l)$. We will compute our Smith form by a series of such transformations, each replacing two elements by their gcd and lcm.

For $\mathrm{diag}(a_1, \ldots, a_n)$, the n diagonal entries can be partitioned into $n/2$ pairs and the gcd/lcm replacement done on each pair independently. Using the fast methods, the cost of these $n/2$ operations is $\sum_{i=1}^{n/2} O(G(\mathrm{len}(a_{i_1}) + \mathrm{len}(a_{i_2})))$. Because $m \log^2(m) \log\log(m)$ is of the form m times a nondecreasing function, this cost is $O(G(\sum_{i=1}^{n} \mathrm{len}(a_i)))$, i.e. softly linear in the input size.

The main idea is that sorting networks can be simply modified to compute the Smith form. Sorting networks are circuits consisting of comparator units wired together. Each comparator has two input wires and two output wires producing the min of the two inputs at one output wire and the max at the other. If the min/max comparators are replaced by gcd/lcm butterflies (producing gcd at one output, lcm at the other), a sorting network will compute the Smith form. To see this, observe that locally at each prime p, the gcd/lcm butterfly *is* a min/max comparator ($\gcd(p^e, p^f) = \min(p^e, p^f)$). As is well known, A matrix is in Smith form if and only if it is in Smith form locally at every prime.

Without loss of generality, let n be a power of two and let us consider staged sorting networks in which each stage is a circuit of depth one having n inputs feeding $n/2$ comparators, and n outputs feeding the next stage. Sorting networks such as Batcher's odd-even merge [3], and the variant described in [7] require $O(\log^2(n))$ stages. Also there is a staged sorting network requiring only $O(n \log(n))$ time[2], but with large constant factor. Any such staged network may be used to compute diagonal Smith forms in soft linear time, since each stage costs soft linear time producing output of the same total size as that of the input, and there are logarithmically many stages. We have proven

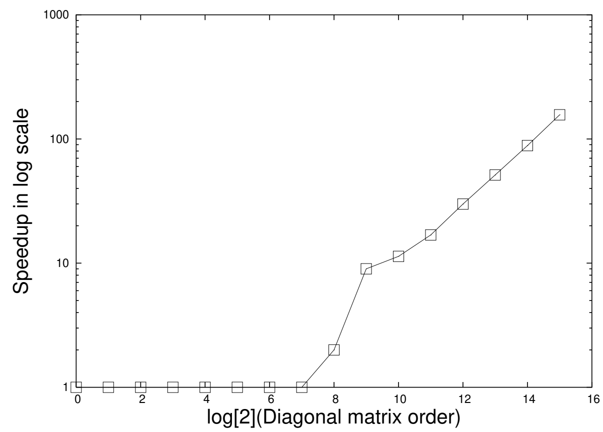
Theorem 4. *Let A be a diagonal matrix over a Principal Ideal Ring R. A staged sorting network with the min/max comparators replaced by gcd/lcm butterflies will compute the Smith form of A. If $R = Z$, the cost is $O^{\sim}(N)$, where N is the sum of the lengths of the diagonal entries of A.*

If d is an upper bound for $\log(|a_i|)$, we may also observe that the run time is $O^{\sim}(nd)$.

We thank a referee for pointing out that a softly linear method already exists as a consequence of Bernstein's "coprime base" algorithm[4]. Given $(a_1, \ldots, a_n)$ he computes the unique $(p_1, \ldots, p_m)$, such that the $p_i's$ are relatively prime to each other and each a_i is a product of powers of the p_i. He also computes $e_{i,j}$, the power of p_i in a_j, and by sorting the $e_{i,j}$ for each i the invariant factors may be obtained. The cost estimate given in [4, Theorem 18.2] for the coprime basis computation is $O(N \log^4 N \log n\mu(N))$, where $\mu(k)$ is the cost of arithmetic on numbers of size k. The factoring of the a_i against the basis costs less[4, Theorem 21.3]. Clearly, so does the sorting, so that basis construction is the dominant cost. The algorithm presented here, when using the asymptotically fastest sorting network, has a runtime bound which is a factor of $\log^3(N)$ smaller.

In figure 2 we compare experimentally the gcd/lcm sorting network approach to the standard n^2 gcd's approach. The latter method is, for diagonal matrix $(d_1, \ldots, d_n)$, to do: for i from 1 to $n-1$, for j from $i+1$ to n or until $d_i = 1$, replace d_i and d_j by $\gcd(d_i, d_j)$ and $\mathrm{lcm}(d_i, d_j)$ respectively. Our measurements, show that the speedup is substantial and eventually tends to be linear in n, as expected.

Figure 2: Diagonal matrix Smith form speedup (classical method time / sorting network time).



3.3 Local Smith Form

The local approach can be useful for the smooth primes, particularly 2, and maybe for some other small primes. To that purpose, we discuss a way to compute S_{p^e}, for a prime power p^e. The local Smith form algorithm we describe is valid over any local principal ideal ring in which, beside the usual arithmetic functions, a gcd function and exact division function are provided. The important fact about a local PIR we use is that the gcd of a set of elements is an associate of some element of that set. For example, in $\boldsymbol{Z}_{81}$, $\gcd(18, 27, 54) = 9$, and 9 is an associate of 18. The algorithm below is similar to the local Smith form algorithm in Dumas, et al[8]. More recently, Lübeck discusses local Smith form computation[15]. In the case of an elimination step in which the remaining block has a nontrivial gcd, that algorithm removes a factor of p whereas this version removes in one step the gcd of the elements, possibly a higher power of p. We also remark that it does not explicitly use the prime of the local ring.

Algorithm 5. *LocalSmithForm*
Input: A, an $n \times m$ matrix over a Local PIR.
Output: S, the Smith form of A. Procedure:

1. *Find s, t such that $g := a_{s,t}$ is gcd($\{a_{i,j}\}$) [gcd of all entries], up to a unit factor.*
2. (a) *if g is a unit,*
   Swap row 1 with row s and col 1 with col t.
   For each $i \in [2 \ldots n]$, eliminate $a_{i,k}$ (subtract the appropriate multiple of row i from row l).
   Set A' to A with the first row and column removed.
   Return $S := (1, S')$, where S' = LocalSmithForm(A').
   (b) *if $g = 0$, return $S := (0)$, the $n \times m$ zero matrix.*
   (c) *else [g is a non trivial factor.]*
   Set $A' = A/g$, i.e. exactly divide each entry by g.
   Compute S' = LocalSmithForm(A').
   Return $S := g \times S'$.

For our purposes in this paper, the Local PIR of interest is $\boldsymbol{Z}_q$ for a prime power q dominating the power of that prime in the largest invariant factor of an integer matrix. Let $M(q)$ represent the cost of arithmetic operations mod q, The runtime of the LocalSmithForm algorithm is then evidently $O(n^3 M(q)) = O(n^3 \log q \log\log q)$.

In practice we expect this local approach to be most effective for the small primes. When q is word-size, basic arithmetic operations take constant time and gcd's take $O(\log(q))$ time, so we expect runtime to be $O(n^3 \log(q))$ in that case. More particularly, For small powers of 2 there is a very fast way to handle the basic arithmetic. Most 32 bit word machines have machine integer arithmetic which is actually arithmetic mod $q = 2^{32}$. This means we can avoid explicit normalization of the results of the basic arithmetic operations mod q. Also we can exploit the machine shift operations to further speedup gcd computations and exact divisions by a power of 2. Measurements of a preliminary implementation in this style are showing a speedup by a factor of 15 relative to using the standard word-size modular arithmetic. Another way to minimize normalizations is to exploit dot products[10] or block operations using floating point BLAS[9]. The local smith form algorithm is very similar to a LU decomposition, which can be done in a "left looking" way to use dot products or be done in a blocked way. However, these approaches break down each time the invariant factors step up, $s_i \neq s_{i+1}$. We intend to explore such variants in future, including how to finesse the breakdowns.

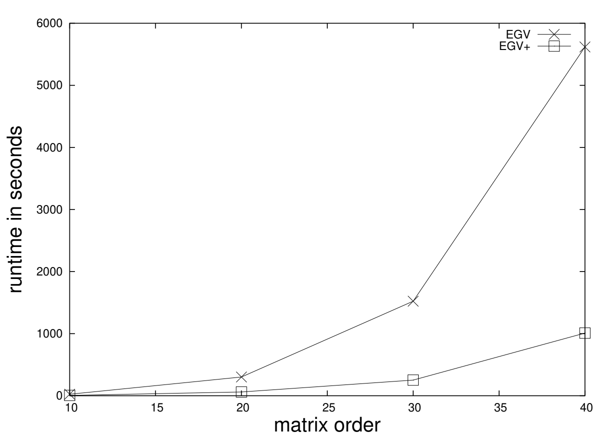
4. ADAPTIVES AND EXPERIMENTS

In this section, we will show some interesting experimental results. We denote by EGV+ our variant in section 2 of the EGV algorithm. Both EGV and EGV+ suffer from the fact that many repetitions are necessary to have good probability for the smooth part of the i^{th} invariants, i.e., no small primes excluded from consideration, see notes in the proof of Theorem 2. The number of repetitions used in EGV/EGV+ can be significantly reduced if we only compute the rough part of the invariant factors. So we propose a strategy of using different approaches for the rough part and the smooth part.

4.1 Two algorithms for rough parts

EGV and EGV+ are good candidates for the rough part of Smith forms. Experiments confirm an advantage for EGV+. All measurements were taken on an Intel PM laptop running Redhat 9, with 0.5GB RAM, and operating at 1.3GHz. For the first experiment, we constructed random matrices by multiplication of L, D and U, where L is random unit lower triangular, D is a diagonal matrix with half it's diagonal entries zero and the other half random in 1..5, and U is a random upper triangular matrix. The random entries in L and U were random 32 bit integers. From figure 3, we can clearly see some benefit from our improvements.

Figure 3: Time to compute the Smith form by EGV algorithm and EGV+ algorithm. Smaller is better.



EGV and EGV+ are quite similar, the only difference between them lies in the choice of LAR or $A+UV$ for computing an arbitrarily indexed invariant factor. EGV uses $A + UV$ for all i-th invariant factors. EGV+ uses one or the other depending on the size of the index. In following subsection, we will discuss this idea in detail.

4.2 Two i^{th} invariant factor methods

In [11], in order to compute the i^{th} invariant factor of A, random $n \times (n-i)$ matrix U and random $(n-i) \times n$ matrix V are constructed. Then the last invariant of $A + UV$ is computed.

In the LAR approach, we construct random $i \times n$ matrix L and $n \times i$ matrix R, then we compute the last invariant factor of $i \times i$ matrix LAR. So generally, when i increases, the time to compute the i^{th} invariant factor will increase in our method, while in the $A + UV$ method the time will decrease. Which way should we compute the i^{th} invariant factor?

The two methods choose elements in random sets of size $O^{\sim}(n^4)$ and have the same success probability for the same number of iterations. In this estimate, we will use $\beta = n^4$ for the size. Let K be the number of iterations in the last invariant factor algorithm.

Our LAR method roughly requires $(in^2 + i^2 n)\log(\beta) + Ki^3 \log(\beta^2 n^2)$ word-size arithmetic operations, in order to compute the i^{th} invariant factor. The $A + UV$ method roughly requires $(n-i)n^2 \log(\beta) + Kn^3 \log(\beta^2(n-i))$ word-size integer arithmetic operation.

At the crossover point, $(in^2 + i^2 n)\log(\beta) + Ki^3 \log(\beta^2 n^2) = (n-i)n^2 \log(\beta) + Kn^3 \log(\beta(n-i))$.

After division by $n^3 \log(n)$ and simplifying, we have

$$10K(i/n)^3 + 4(i/n)^2 + 8(i/n) - (4 + 9K) = 0.$$

The crossover ratio i/n will increase as K increases. For instance, when $K = 1$, the crossover is at $i/n = 0.7670$ and when $K = 10$, $i/n = 0.9396$.

In the limit, $\lim_{K\to\infty}(i/n) = \sqrt[3]{9/10} \approx 0.9655$.

In practice, we may be some distance away from this limit. Figure 4 shows times for the two approaches for selected sizes i in a random 400×400 matrix. We note that the crossover is near $i = 0.6n$. For smaller i, LAR will be faster, but when $i \geq 0.6n$ the $A + UV$ will be faster.

Thus we can easily have an adaptive algorithm to compute the i^{th} invariant factors. When i/n is less than $.6n$ use LAR, otherwise use $A+UV$. We have not yet formulated our best opinion concerning where the crossover threshold will be. A crossover threshold which increases with increasing n may be appropriate.

4.3 Two algorithm for smooth parts

For the smooth parts of invariant factors, Iliopoulos' algorithm and the local algorithm are good candidates. So may be Storjohann's, but we do not yet have an implementation of that algorithm. Note that Iliopoulos' algorithm can handle several prime powers q_i at once, i.e. compute mod $m = \prod(q_i)$, whereas the local algorithm requires independent computations for each prime power.

In figure 5 we plot runs of these two algorithms on 1000×1000 matrices which are random equivalents to random diagonal matrices in $\boldsymbol{Z}_q$, for various powers q of small primes. The Local algorithm is 2 or 3 times faster than Iliopoulos'.

This suggests that it may be preferable to use Iliopoulos' algorithm when s_r contains three or more q_i whose product

Figure 4: Time to compute the i^{th} invariant factor. Smaller is better.

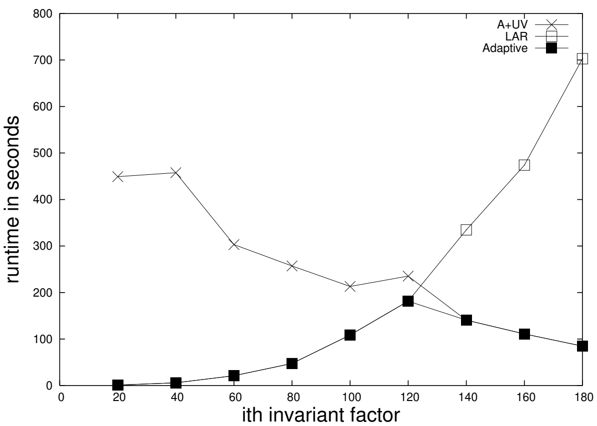
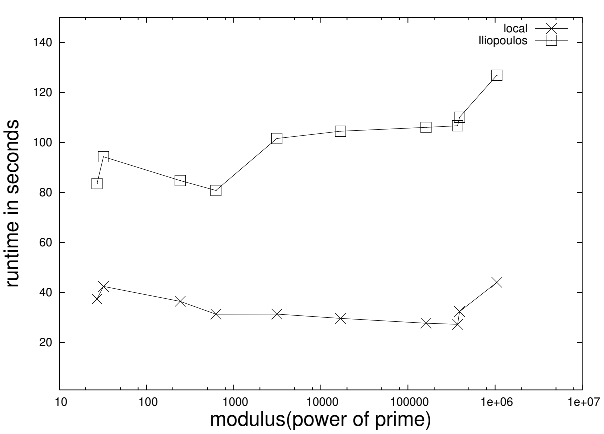


Figure 5: Time for Smith form mod a prime power, 1000 by 1000 matrix, local vs. Iliopoulos'. Smaller is better.



is still word-size, but to use the local algorithm when there are only one or two small primes present or for each single q_i which is itself a rather large prime power. We can combine the two methods into an adaptive one. Further tuning and measurements are necessary to determine the best approach here.

4.4 Adaptive Smith form algorithm

The algorithms we have discussed can be combined into an adaptive algorithm to obtain considerable speedup in the run time. To do this requires setting various thresholds and we have not yet found the values we would like to propose for a "production" code. However, we can describe the basic outline of the combination algorithm. The strategy will be to get the rough parts of Smith forms using EGV+, and to get the smooth parts of Smith forms using combination of LocalSmithForm and Iliopoulos' method. Storjohann's algorithm also may belong in the mix.

5. CONCLUSION

For Smith form computation of integer matrices, best performance in practice seems to be a mixture of the algorithms whose asymptotic complexity is best or near best. An adaptive framework for such a mixture has been developed here.

6. REFERENCES

[1] J. Abbott, M. Brostein, and T. Mulders. Fast deterministic computation of determinants of dense matrices. In *Proc. ISSAC'99*, pages 197 – 204. ACM Press, 1999.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c\lg(n)$ parallel steps. *Combinatorica*, 3:1–19, 1983.

[3] K.E. Batcher. Sorting networks and their applications. volume 32, pages 307–314, 1968.

[4] Daniel J. Bernstein. Factoring into coprimes in essentially linear time. *Journal of Algorithms*, to appear.

[5] William C. Brown. *Matrices over commutative rings*. Marcel Dekker Inc., New York, NY, 1993.

[6] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proc. 19th ACM STOC*, pages 1 – 6, 1987.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Second edition*. MIT Press, 2001.

[8] J. Dumas, B.Saunders, and G. Villard. Smith form via the valence : Experience with matrices from homology. In *Proc. ISSAC'00*, pages 95 – 105. ACM Press, 2000.

[9] J. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. In *Proc. ISSAC'02*, pages 63 – 74. ACM Press, 2002.

[10] A. Duran, D. Saunders, and Z. Wan. Rank of sparse integer matrices. In *Siamla 2003*, 2003.

[11] W. Eberly, M. Giesbrecht, and G. Villard. On computing the determinant and smith form of an integer matrix. In *Proc. 41st FOCS*, pages 675 – 687, 2000.

[12] J. L. Hafner and K. S. McCurley. Asymptotically fast triangularization of matrices over rings. *Siam J. Comput.*, 20:1068–1083, 1991.

[13] C. Iliopoulos. Worst-case complexity bounds on algorithms for computing the canonical structure of finite abelian groups of the hermite and smith normal forms of an integer matrix. *SIAM J. Comput.*, Vol. 18, No.4:658 – 669, 1989.

[14] E. Kaltofen and G. Villard. On the complexity of computing determinants. Technical Report 36, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, October 2003.

[15] Frank Lübeck. Computation of elementary divisors of integer matrices. *Journal of Symbolic Computation*, 33, NO 1:57 – 65, 2002.

[16] V. Pan and X. Wang. Acceleration of euclidean algorithm and extensions. In *Proc. ISSAC'02*, pages 207 – 213. ACM Press, 2002.

[17] H. M. S. Smith. On systems of indeterminate equations and congruences. *Philos. Trans.*, pages 293–326, 1861.

[18] A. Storjohann. Near optimal algorithms for computing smith normal forms of integer matrices. In *Proc. ISSAC'96*, pages 267 – 274. ACM Press, 1996.

[19] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.