Elementary Algebra Revisited: Randomized Algorithms

Gene Cooperman and George Havas

Abstract. We look at some simple algorithms for elementary problems in algebra that yield dramatic efficiency improvements over standard methods through randomization. The randomized algorithms are, in a sense, "obvious". Their formal statement was delayed partly by the need for rigorous analysis, but more so by the need to re-think traditional approaches to elementary algorithms. We illustrate this philosophy with some basic problems in computational number theory (GCD of many integers), linear algebra (low-rank Gaussian elimination) and group theory (random subproducts for subgroup construction), along with a brief survey of other areas.

1. Introduction

Because the textbook algorithms of elementary algebra seem to be so efficient, researchers often neglect to examine still more efficient randomized algorithms. Nevertheless, there is scope for some interesting and potentially very important applications of randomization.

A paper of this length cannot hope to provide a survey of the subject. So, it is our intention, rather, to illustrate the utility of randomization in several recent innovations that are related to our own work. The topics chosen reflect elementary algorithms that are either known or can easily be taught to high school students. Often, it is not difficult to see a candidate for a randomized algorithm, but a formal analysis may be more difficult. It is hoped that these examples will inspire more people to look at randomization as a source of greater efficiency. We would also like to point out the excellent survey paper [**13**] in this volume, concerning generation of randomized non-singular matrices, k-subspaces and other structures of interest.

In keeping with our philosophy, the ideas for randomization are simple. The "interesting bits" are in the analysis. In this paper, we discuss the GCD problem for many integers, one of the oldest algorithms of elementary algebra, in detail, while surveying several other algorithms. In addition to GCD's, we give examples of some applications of randomizations to low-rank Gaussian elimination, and to computing canonical forms of matrices.

For GCD's, one wishes to find the GCD of k integers, $n_1, \ldots, n_k$, with few GCD computations. The key idea of the algorithm is to form two random linear sums of the form $\sum_{i=1}^{k} a_i n_i$, and take the GCD of those two random linear sums.

1991 *Mathematics Subject Classification.* Primary 68Q25; Secondary 11-04, 15-04, 20-04.
Supported in part by NSF Grant CCR-9509783.
Supported in part by the Australian Research Council.

It is clear that this computation yields a multiple of the true GCD. Hence, we describe an "error" as occurring when the two random linear sums each include an additional prime factor that is not present in the k integers. The issue is to determine a maximum magnitude for the randomly chosen coefficients, a_i, sufficient to guarantee low probability of error. A refined version of such an algorithm is given in [**3**] together with a rigorous analysis and some experimental results.

For low-rank Gaussian elimination, one uses a similar trick. If one can rapidly determine the rank, r, of a matrix and form r independent linear combinations of matrix rows, then the Gaussian elimination algorithm admits an obvious optimization. The algorithm consists of considering the rows, $v_1, \ldots, v_k$, of a matrix, and forming r random linear sums, $\sum_{i=1}^{k} a_i v_i$, where the random coefficients, a_i, are now chosen independently from the set $\{0, 1\}$.

The ideas above are specified in the context of integers and vectors, respectively. The ideas can be generalized in the direction of mathematical groups by considering random subproducts for groups. This yields some surprisingly strong results.

Finally, the use of randomization in computing canonical forms of matrices shows performance gains over deterministic algorithms. The randomized algorithms achieve close to theoretically optimal performance in some practical circumstances.

2. GCD's of many integers

Consider a model of computation where calculating GCDs is relatively expensive in comparison to other arithmetic. This is a reasonably realistic model. Thus, we wish to do few GCD computations in comparison with the naive algorithm (iterative computation of pairwise GCDs), which in the worst case requires $k-1$ GCD calculations.

THEOREM 1. *For positive integers $n_1, \ldots, n_k$, one can probabilistically find $\gcd(n_1, \ldots, n_k)$ at the cost of $4k$ arithmetic operations, one GCD calculation, and the computation of $2k$ random integers chosen in the range from 1 to M, where $M = \max(1024\lceil \frac{\log_2 k}{150} + 0.1\rceil, (n_1)^2, \ldots, (n_k)^2)$. All operations involve quantities of size at most $M \sum_{i=1}^{k} n_i \leq kM \max(n_1, \ldots, n_k)$. The probability of error is bounded above by 0.622.*

PROOF. Without loss of generality, we assume that $n_1 \leq n_2 \leq \cdots \leq n_k$.

The reduction algorithm consists of computing $z = \gcd(\sum_{i=1}^{k} a_i n_i, \sum_{i=1}^{k} b_i n_i)$, where $a_i, b_i \in [1, M]$ are randomly chosen integers in the specified range. The value of z will be a multiple of $\gcd(n_1, \ldots, n_k)$.

We now consider $x = z/\gcd(n_1, \ldots, n_k)$. We bound the probability that $x \neq 1$. The analysis consists of finding an upper bound on the probability that x has a prime factor, p, for each of three cases: (A) $M < p \leq kM^{3/2}$, (B) $M/10 < p \leq M$, and (C) $p \leq M/10$. (Note $x \leq kM^{3/2}$, and so no other cases are possible.) The sum of these three probability bounds will then be shown to be less that the probability stated in the theorem.

CASE A: $(M < p \leq kM^{3/2})$

In this case, p is co-prime to n_k since $p > n_k$ and p is prime. Since $M < p$ and $\sum_{i=1}^{k} b_i n_i \leq kM^{3/2}$, there are at most $\lceil \log(kM^{3/2})/\log M \rceil$ distinct primes, p, that divide $\sum_{i=1}^{k} b_i n_i$. Consider $\sum_{i=1}^{k} a_i n_i$ as a function of a_k. Since $a_k \leq M < p$, for each fixed p there is at most one value of a_k for which p divides $sum_{i=1}^{k} a_i n_i$. Since a_k is chosen uniformly from the range $[1, M]$, the probability that a prime $p > M$

divides x is bounded above by $\lceil \log(kM^{3/2})/\log M \rceil/M$. So, the probability that p divides x is bounded above by
$\lceil \log(kM^{3/2})/\log M \rceil/M \leq (\frac{\log_2 k}{10} + \frac{3}{2})/M \leq 15\lceil \frac{\log_2 k}{150} + 0.1 \rceil/M$.

CASE B: $(M/10 < p \leq M)$
Choose s to be the smallest positive integer such that p^s does not divide $\gcd(n_1, \ldots, n_k)$. (Hence, $s \geq 1$ always.) Choose j such that p^s does not divide n_j. Fix the a_i's arbitrarily except for a_j, and consider $\sum_{i=1}^k a_i n_i$ as a function of a_j. If p^s divides $\sum_{i=1}^k a_i n_i$ for a given value of a_j, then the next value of a_j for which p^s divides $\sum_{i=1}^k a_i n_i$ will be $a_j + p$. Since a_j is chosen uniformly from the range $[1, M]$, the probability that p^s divides $\sum_{i=1}^k a_i n_i$ is at most $\lceil M/p \rceil/M \leq \frac{1}{p} + \frac{1}{M} \leq 2/p$. Similarly, the probability that p^s divides $\sum_{i=1}^k b_i n_i$ is at most $2/p$. Since the a_i and b_i were chosen independently, the probability that p divides x is at most $4/p^2$. So, the overall probability that p divides x for some prime in the given range is bounded above by $\sum_{p=(M/10)+1}^{M} (4/p^2) \leq \int_{p=M/10}^{M} (4/p^2)\, dp = 36/M$.

CASE C: $(p \leq M/10)$
Choose s and j as in the previous case. One follows the argument of the previous case. Since a_j is chosen uniformly from the range $[1, M]$ while now $p \leq M/10$, one can conclude for this case that the probability that p^s divides $\sum_{i=1}^k a_i n_i$ is at most $\lceil M/p \rceil/M \leq \frac{1}{p} + \frac{1}{M} \leq \frac{1}{p} + \frac{1}{10p} = \frac{11}{10}/p$. Similarly, the probability that p^s divides $\sum_{i=1}^k b_i n_i$ is at most $\frac{11}{10}/p$. Since the a_i and b_i were chosen independently, the probability that p divides x is at most $(\frac{11}{10}/p)^2$. So, the overall probability that p divides x for some prime in the given range is bounded above by $(\frac{11}{10}/2)^2 + (\frac{11}{10}/3)^2 + (\frac{11}{10}/5)^2 + (\frac{11}{10}/7)^2 + \sum_{p=11}^{M/10} (\frac{11}{10}/p)^2 < 0.51004 + \int_{p=10}^{M/10} (\frac{11}{10}/p)^2\, dp = 0.51004 + (\frac{11}{10})^2(\frac{1}{10} - 10/M) < 0.571$.

Hence, the sum of the probabilities of error for the three cases is

$$0.571 + 36/M + 15\lceil \frac{\log_2 k}{150} + 0.1 \rceil/M \leq 0.622$$

based on the choice of M in the hypothesis. This completes the proof. □

REMARK 2. *A different approach to solving this problem is implicit in* [**8**, Proof of Lemma 1.2].

As is always the case with Monte Carlo algorithms, the probability can be improved by repeating the algorithm. Specifically, the probability that r invocations all return an erroneous answer is at most 0.622^r.

2.1. Las Vegas conversion. Any Monte Carlo algorithm which has an easy check as to whether the solution is correct may be simply converted to a Las Vegas algorithm: run the algorithm, check the solution, and repeat till the solution is confirmed.

In this case checking the solution is very easy: check that the hypothetical GCD exactly divides each of the k input numbers. Thus, adding k more arithmetic operations to the algorithm, we get a Las Vegas algorithm for which the expected number of invocations needed to achieve a correct answer is at most $1 + 0.622 + 0.622^2 + 0.622^3 + \cdots < 2.65$. Multiplying 2.65 by the appropriate complexity estimate yields the following corollary.

COROLLARY 3. *One can find* $\gcd(n_1, \ldots, n_k)$ *for arbitrary positive integers* $n_1, \ldots, n_k$ *with an expected cost of at most* $13.25k$ *arithmetic operations,* $5.3k$ *random integers in the range* $[1, M]$*, and* 2.65 *GCDs. All operations involve quantities of size at most* $kM^{3/2}$*, with* M *defined as in Theorem 1.*

Notice that $\lceil \frac{\log_2 k}{150} + 0.1 \rceil = 1$ for $k \leq 10^{40}$. It is worth remarking that the only uses of division in this algorithm are in the GCD algorithm and in the Las Vegas verification of the answer. All other operations consist of multiplication and addition, which are usually faster.

2.2. Improved Reliability. The above analysis can be strengthened to yield a probability of error of at most 0.2. Define a *spurious prime factor* as a prime factor of x. (Recall that x was defined as $\gcd(\sum_{i=1}^{k} a_i n_i, \sum_{i=1}^{k} b_i n_i) / \gcd(n_1, \ldots, n_k)$.)

REMARK 4. *The largest contribution to the above probability comes from small primes. For example, the possibility of additional prime factors* 2 *or* 3 *occurring spuriously in the answer contributed a probability of error of more than* 0.4369*. If one separately checks for factors of* 2 *and* 3 *in the answer and replaces* 0.622 *by the finer estimate of* 0.6219*, the probability of error is reduced to at most* $0.6219 - 0.4369 = 0.185$.

In implementations, this cost saving may be worthwhile. Since the answer returned, y, will be a multiple of the true answer, it requires one to directly find the largest power of 2 that divides $\gcd(y, n_1, \ldots, n_k)$ and similarly for 3. Since $y \leq kM$, there are at most $\log_2 kM$ additional spurious prime factors in y. So, the additional computation costs $O(k \log(kM))$ using an obvious algorithm.

However, the worst case computational cost occurs only if y has many prime factors corresponding to 2 and 3. In typical cases, there will be at worst a small constant number, r, of such factors, and so the additional computation costs will typically involve only $k + 5r$ additional divisions. This is seen by first finding how many times 2 and 3 occur as factors of y and then carrying out the k divisions into $n_1, \ldots, n_k$ and up to r adjustments to y as spurious prime factors corresponding to 2 or 3 are found. This operation should be carried out before the Las Vegas test at the end to see if y divides n_i for all i. If one can be assured that r is small (for example, if the inputs, $n_1, \ldots, n_k$ are drawn from a "random" distribution for appropriate definition of random), then this yields a still faster GCD algorithm. Taking $r < k/5$ yields a Monte Carlo algorithm with probability of error at most 0.2. The expected number of invocations is then reduced from 2.65 to 1.25 at the cost of raising the $5k$ arithmetic operations per invocation to at most $7k$.

Still further heuristics are possible. As just one example, consider including the result of a previous invocation as an additional input, n_{k+1} in the next invocation.

3. Random Subproducts for Group Theory

Random subproducts provide a useful unifying framework for both the randomized GCD algorithm, discussed above, and for the low-rank Gaussian elimination algorithm, discussed in the next section. Random subproducts were first used in a more limited setting (for orbits in permutation groups [**2**]). A survey of many of the implications of random subproducts is found in [**4**]. The more general description here has surprising consequences.

Definition. A *random subproduct* of a sequence of group elements $(g_1, g_2, \ldots, g_k)$ is an instance of a product $g_1^{e_1} g_2^{e_2} \cdots g_k^{e_k}$ where the e_i are independent random variables uniformly distributed over $\{0, 1\}$.

We also refer to a *random subproduct on a set of group elements*, instead of a random subproduct of a sequence, when an arbitrary ordering of the set suffices.

THEOREM 5. [**1**, Lemma 2.4] *Let H be a non-trivial, proper subgroup of the finite group G, and let G have a generating set S of size s. If $g \in G$ is a random subproduct on S, then one can compute g at the cost of at most s group multiplications, and* $\Pr(g \notin H) \geq 1/2$.

PROOF. For some i, $g_i \notin H$, (since $H \neq G$). Let $w = u g_i^{\epsilon_i} v$, with $g_i \notin H$, $g_{i+1}, \ldots, g_r \in H$. So, $v \in H$.
Case I: $u \in H$. With probability $1/2$, $\epsilon_i = 1$, and so $w = u g_i v \notin H$.
Case II: $u \notin H$. With probability $1/2$, $\epsilon_i = 0$, and so $w = uv \notin H$. □

Note that this algorithm makes no mention of a particular group representation. This "representation-free" character of the result also provides a strong result for the case of matrix groups, which are notoriously difficult to analyze. The ideas of this section yield an almost trivial algorithm to find generators for the normal closure of a group, first described in [**4**]. The normal closure algorithm also illustrates the general ideas of this philosophy.

Definition. Let $H \subseteq G$, with $H = \langle U \rangle$ and $G = \langle S \rangle$. A *random normal subproduct with respect to U and S* is a conjugate of the form h^g where h is a random subproduct on U and g is a random subproduct on S.

THEOREM 6. [**1**, Lemma 3.6] *Let $H = \langle U \rangle$ and $G = \langle S \rangle$ be subgroups of a (possibly infinite) group J, with U and S finite. Suppose that H is not normalized by G. Let h^g be a random normal subproduct with respect to U and S. Then*

$$\Pr(h^g \in \langle H^G \rangle \setminus H) \geq 1/4.$$

The proof of this theorem is clear by analogy with Theorem 5. Its usage is more interesting. In looking for a normal closure, $K = \langle H^G \rangle$, we initially set $K = \langle U \rangle$. Our goal is to add generators to K, until $K = \langle H^G \rangle$. Theorem 6 asserts that a random normal subproduct will provide a "new" generator in $\langle K^G \rangle \setminus K$ with probability at least $1/4$. The random normal subproduct is added to the generators of K, and the previous random normal subproduct construction is repeated.

One can show that the algorithm operates on $O(l^2)$ generators, where l is an upper bound on the length of a subgroup chain from its representation and assuming $\max(|S|, |U|) = O(l)$. One can estimate l based either on the representation as a permutation or matrix group, or one can take $l = \log_2 |G|$ for $H < G$. So $4l$ random normal subproducts suffice in some averaged sense for full generation of $\langle H^G \rangle$. Formally, one uses Chernoff's theorem to show cl random normal subproducts suffice with high probability, for c a constant that is larger than 4 by some small factor.

Other similar ideas are described in [**1**], where random subproducts are used for fast algorithms that test solvability and nilpotency of groups in $O(l^2 \log^4 l)$ group operations, that find a normal closure in $O(l \log^4 l)$ (by a different algorithm), and that find a generating set of size $O(\log |G|)$ from an initial generating set, S, in $O(|S| \log l)$ group operations.

The last result is important for the next section, and so we highlight it.

THEOREM 7. [**1**, Theorem 2.3] *Let $G = \langle S \rangle$ be a finite group. Let l be a known upper bound on the length of all subgroup chains in G. Then for any fixed parameter p such that $0 < p < 1$, with probability at least p one can find a generating set S' with $|S'| = O(l)$, using $O(|S| \log l)$ group operations.*

4. Low-rank Gaussian elimination

It is well known that Gaussian elimination operates in $O(n^3)$ time or operations in a given field, for n the matrix dimension. For low-rank matrices of rank $r < n$, this time can be reduced to $O(nr^2 + n^2 \log^2 r)$ field operations. This should be compared with the standard upper bound (Gaussian elimination) of $O(n^2 r)$ and the lower bound of $O(n^2)$ required simply to look at all matrix entries.

THEOREM 8. *A system of n linear equations of rank r (where r is not known a priori) can be solved (or found inconsistent) (with a small probability of error) using $O(nr^2 + n^2 \log^2 r)$ field operations.*

A useful way to understand this construction is by using the ideas of random subproducts, discussed in the previous section. The idea is to consider an additive group on vectors in dimension n. Consider the group generated by the row vectors of the matrix. A maximum subgroup chain in this group is of length $l = r$. If one could efficiently construct $O(r)$ generators for this group, it is clear that $O(nr^2)$ field operations would then suffice for the Gaussian elimination. Theorem 7 allows us to assume $l = r$, and find $O(r)$ generators in $O(n^2 \log r)$ field operations.

In this specialized setting, $l = r$ and a randomized subproduct of the row vectors v_i becomes $\sum_{i=1}^{n} e_i v_i$ for random $e_i \in \{0, 1\}$. We initially set $K = \text{span}(0)$ for 0 the zero vector. We add cr random subproducts to the generators of K for suitable $c > 4$, where each random subproduct has probability at least $1/4$ of being "new", by Theorem 5. This requires $crn = O(nr)$ operations. So if the rank, r, is known a priori, we form a rectangular $cr \times n$ matrix and carry out Gaussian elimination on that matrix in $O(nr^2)$ field operations.

We now describe the full algorithm. The only additional idea is to guess the rank and invoke "repeated doubling". One initially guesses a rank of $\tilde{r} = 1$ and carries out Gaussian elimination on the resulting $c\tilde{r} \times n$ matrix, and determines an estimated rank, $\tilde{r}'$, of the matrix with $c\tilde{r}$ row vectors. One then doubles the guessed rank, $\tilde{r}$, repeats the Gaussian elimination, and compares the newly estimated rank, $\tilde{r}'$, with the old value. If the values are the same, then we have discovered the rank with high probability, and we have successfully carried out a Gaussian elimination. If the ranks are different, we double the guessed rank, $\tilde{r}$, again, and repeat the process. It is clear that $\lceil \log_2 r \rceil$ iterations will be required (where r is the true rank), with each iteration requiring $O(n\tilde{r}^2)$ field operations. The time is dominated by the last iteration with $\tilde{r} \leq 2r$. So, the overall time is $O(nr^2)$, if one is given the $c\tilde{r}$ row vectors. As we have seen, one can form the $c\tilde{r}$ row vectors using $O(n^2 \log \tilde{r})$ field operations for each of the $O(\log r)$ iterations, yielding the overall estimate of $O(nr^2 + n^2 \log^2 r)$.

If partial information is known, a priori, about r, then one can often achieve a better complexity. If the rank, r, is known exactly, then the time can be reduced to $O(nr^2 + n^2 \log r)$ in an obvious manner.

If $r < \epsilon m$ for some small $\epsilon > 0$ but r itself is not bounded, we can achieve considerable savings by first replacing the m rows by a set of $O(r)$ rows using our Monte Carlo algorithms. With large probability, the rows obtained will generate

the same row space. The new rows are obtained as sums of subsets of the original rows and will require $O(m \log r)$ row operations to compute. Subsequently, we can complete the work by Gaussian elimination, using $O(r^2)$ row operations.

5. Computing canonical forms of matrices

There has been much recent work on finding theoretically efficient and practical algorithms for computing canonical forms of matrices over various computational domains. In a well-defined sense finding optimal ways to obtain the canonical forms is NP-hard, and worst-case complexity for standard algorithms is exponential in terms of both space and time [**6, 12**]. Randomization has proved to be a powerful tool in addressing such difficult problems. Recent progress includes [**7, 8, 9, 10, 11, 15, 17, 18**], which cover a comprehensive range of algorithms including: heuristically driven with excellent practical performance; deterministic with guaranteed performance; and randomized. In the randomized context both Monte Carlo and Las Vegas algorithms have been developed.

Interesting applications of randomization to Smith normal form computation problems for integer matrices are given by Giesbrecht [**8, 9**] as Monte Carlo algorithms using black-box methods. The GCD of many integers problem of Section 2 arises in this context. The upshot is algorithms with improved space and time complexity for both sparse and dense input. However no comparably efficient Las Vegas versions are known.

In a polynomial matrix context, Storjohann and Labahn [**17**] present a Las Vegas probabilistic algorithm which finds the Smith normal form in $\mathbf{Q}[x]^{n\times n}$ of a nonsingular input matrix in $\mathbf{Z}[x]^{n\times n}$. Their complexity results improve significantly on previous algorithms in both theory and practice.

References

[1] L. Babai, G. Cooperman, L. Finkelstein, E.M. Luks, and Á. Seress, "Fast Monte Carlo Algorithms for Permutation Groups", *J. Computer and System Sciences* **50** (1995) 296–308.

[2] L. Babai, E. Luks, and Á. Seress, "Fast Management of Permutation Groups", *Proc. 29th IEEE FOCS* (1988) 272–282.

[3] G. Cooperman, S. Feisel, J. von zur Gathen, G. Havas, "GCD of many integers", *Northeastern University Tech. Rept. NU-CCS-97-12* (1997).

[4] G. Cooperman and L. Finkelstein, "Combinatorial Tools for Computational Group Theory", *Groups and Computation*, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. **11** (1993) 53–86.

[5] G. Cooperman, L. Finkelstein and B. York, "Parallel Implementations of Group Membership and the Method of Random Subproducts", *Proc. of 1992 Dartmouth Institute for Advanced Graduate Studies in Parallel Computation Symposium (DAGS'92)*, D. Johnson, F. Makedon, and P. Metaxas (eds.), Department of Mathematics and Computer Science, Dartmouth College, Hanover, N.H., pp. 94–100.

[6] X.G. Fang and G. Havas, "On the worst-case complexity of integer gaussian elimination", *ISSAC'97* (Proc. 1997 Internat. Sympos. Symbolic Algebraic Comput.) ACM Press (1997) 28–31.

[7] M. Giesbrecht, "Nearly optimal algorithms for canonical matrix forms", *SIAM Journal on Computing* **24** (1995) 948–969.

[8] M. Giesbrecht, "Fast computation of the Smith normal form of an integer matrix", *ISSAC'95* (Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation), ACM Press (1995) 110–118.

[9] M. Giesbrecht, "Probabilistic computation of the Smith normal form of a sparse integer matrix", *Algorithmic Number Theory*, Springer Lecture Notes in Computer Science **1122** (1996) 173–186.

[10] G. Havas and B.S. Majewski, "Integer matrix diagonalization", *J. Symbolic Computation* **24** (1997) 399–408.
[11] G. Havas, B.S. Majewski and K.R. Matthews, "Extended gcd and Hermite normal form algorithms via lattice basis reduction", *Experimental Mathematics* **7** (1998) 125–135.
[12] G. Havas and C. Wagner, "Matrix reduction algorithms for Euclidean rings", *Proc. 3rd Asian Symposium Computer Mathematics* (to appear).
[13] I. Pak, "When and How n Choose k", DIMACS Ser. Discrete Math. Theoret. Comput. Sci. (this volume).
[14] A. Storjohann, "Near optimal algorithms for computing Smith normal forms of integer matrices", *ISSAC'96* (Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation), ACM Press (1996) 267–274.
[15] A. Storjohann, "Computing Hermite and Smith Normal Forms of Triangular Integer Matrices", *Linear Algebra and its Applications* (to appear).
[16] A. Storjohann and G. Labahn, "Asymptotically fast computation of Hermite normal forms of integer matrices", *ISSAC'96* (Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation), ACM Press (1996) 259–266.
[17] A. Storjohann and G. Labahn, "A Fast Las Vegas Algorithm for Computing Smith Normal Form of a Polynomial Matrix", *Linear Algebra and its Applications* **253** (1997) 155–173.
[18] C. Wagner, "Normalformberechnung von Matrizen über euklidischen Ringen", Dissertation, Fachbereich Mathematik und Informatik der Universität/GH Essen (1997).

College of Computer Science, Northeastern University, Boston, MA 02115, USA
E-mail address: gene@ccs.neu.edu

Centre for Discrete Mathematics and Computing, Department of Computer Science and Electrical Engineering, The University of Queensland, Queensland 4072, Australia
E-mail address: havas@csee.uq.edu.au