Bounds on Sample Space Size for Matrix Product Verification

Donald D. Chinn*
Rakesh K. Sinha†
Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, Washington, U.S.A. 98195

November 28, 1992

Abstract

We show that the size of any sample space that could be used in Freivalds' probabilistic matrix product verification algorithm for $n \times n$ matrices is at least $\frac{n-1}{\epsilon}$ if the error probability is at most ϵ. We also provide a characterization of any sample space for which Freivalds' algorithm has error probability at most ϵ. We then provide a generalization of Freivalds' algorithm and provide matching lower and upper bounds on the error probability in terms of the sample space size and running time.

1 Introduction

Given two $n \times n$ matrices A and B, computing their product is a classic problem in algebraic complexity theory. We consider a related decision problem.

The matrix product verification problem is to decide, given three $n \times n$ matrices A, B, and C, whether $AB \neq C$. We are assuming that all our matrices are over some integral domain D. A probabilistic algorithm with one-sided error solves the matrix product verification problem if it rejects with probability 1 if $AB = C$, and accepts with probability at least $\frac{1}{2}$ if $AB \neq C$.

Freivalds' [4] original matrix product verification algorithm is to choose a $v \in \{0,1\}^n$ from the 2^n possible n-vectors and test whether $(AB)v = Cv$. This algorithm works when the matrices have entries from some ring (which may or may not be an integral domain).

*This material is based upon work supported in part by the National Science Foundation under Grant CCR-9002891.

†This material is based upon work supported in part by the National Science Foundation under Grants CCR-8858799 and CCR-8907960.

Algorithm 1 (Freivalds' original algorithm)

1. Choose $v \in \{0,1\}^n$ randomly and uniformly.
2. **if** $ABv \neq Cv$

   2a. **then return** "$AB \neq C$" (accept)

   2b. **else return** "$AB = C$" (reject).

Using the associativity of matrix multiplication, we can perform the operations as follows: $(A(Bv)) = Cv$, which only requires $O(n^2)$ operations. Contrast this with the best known deterministic algorithm for matrix product verification, which performs matrix multiplication on A and B. Multiplication of two $n \times n$ matrices takes $O(n^{2.376})$ steps [2].

Instead of choosing v from the set $\{0,1\}^n$, let us allow step 1 of Algorithm 1 to choose v from an arbitrary set of n-vectors S. (Throughout this paper, we use *vector* to mean a tuple of elements from D.)

Algorithm 2

1. Choose $v \in S$ randomly and uniformly, where S is a finite set of n-vectors.
2. **if** $ABv \neq Cv$

   2a. **then return** "$AB \neq C$" (accept)

   2b. **else return** "$AB = C$" (reject).

The motivation behind this generalization of Algorithm 1 is that the size of the sample space in Algorithm 1 is 2^n, which implies that n random bits are required to run the algorithm. If we can find a smaller sample space S such that Algorithm 2 still has constant error probability then we can reduce the number of random bits needed to solve the matrix product verification problem.

Naor and Naor [8] were the first to construct a polynomial size sample space for Algorithm 2. They gave two constructions, both of which achieve constant error probability. Their first construction is of size $O(n)$ but works only for matrices over $GF(2)$. The second construction works for matrices over an arbitrary ring but is of $\Theta(n^2)$ size. Alon *et al.* [1] gave three constructions for the case of matrices over $GF(2)$. The sample spaces they constructed are each of size $(\frac{n}{2\epsilon-1})^2$, where ϵ is the error probability of Algorithm 2.

Kimbrel and Sinha [6] constructed a sample space of size $\left\lceil \frac{n-1}{\epsilon} \right\rceil$ such that the error probability for Algorithm 2 is no more than ϵ. Their algorithm works for the case when the matrices have entries from an integral domain D with at least $\left\lceil \frac{n-1}{\epsilon} \right\rceil$ elements. (For small prime fields, they construct a sample space of size $\left\lceil \frac{n-1}{\epsilon} \right\rceil$ whose elements are vectors over some *extension field*. In this case, computations are performed in the extension field and therefore the algorithm is slightly inefficient. The exact running time depends on the order of the base field and the value of ϵ.) Their algorithm is very simple to describe for the case when $|D| \geq \left\lceil \frac{n-1}{\epsilon} \right\rceil$:

Algorithm 3 (Kimbrel and Sinha)

1. Let X be a set of $\left\lceil \frac{n-1}{\epsilon} \right\rceil$ distinct elements of D.
2. Choose $x \in X$ randomly and uniformly.
3. $v \leftarrow [x^0\ x^1\ \ldots\ x^{n-1}]^T$.
4. **if** $ABv \neq Cv$

   4a. **then return** "$AB \neq C$" (accept)

   4b. **else return** "$AB = C$" (reject).

Algorithm 3 still runs in $O(n^2)$ time with error probability at most ϵ, yet uses only $\lceil \log_2(n-1) \rceil + \left\lceil \log_2(\frac{1}{\epsilon}) \right\rceil$ random bits [6, Theorem 5]. In particular, if $\epsilon = \frac{1}{2}$, then Algorithm 3 uses only $\lceil \log_2(n-1) \rceil + 1$ random bits. Can we find a smaller S such that Algorithm 2 works? For example, can we use a pseudorandom number generator that uses $o(\log_2 n)$ random bits, such as the almost k-wise generators of Naor and Naor [8], Alon *et al.* [1], and Even *et al.* [3], to generate the n-vectors of S for Algorithm 2 and still achieve an $O(n^2)$ running time?

Section 2 answers this last question in the negative by providing a lower bound on the size of the sample space in terms of the error probability. Section 3 provides a characterization of those sample spaces S such that Algorithm 2 has an error probability that matches the bound of Section 2. Sections 4, 5, and 6 provide a generalization of Algorithm 2 and give matching lower and upper bounds on the error probability in terms of the sample space size and running time.

We will be repeatedly using the following simple fact from linear algebra.

Fact 1: Given any $n \times m$ matrix A with entries from an integral domain D, $rank(A) \leq n-1$ if and only if there is a non-zero n-vector v such that $vA = 0$.

Fields are assumed to be the underlying structure in standard treatments of rank. Since we are dealing with integral domains, we have to be careful in defining *rank*. Given any set of vectors, we define their *rank* to be the cardinality of the largest subset of vectors such that any non-zero combination of these vectors is non-zero. The *rank* of a matrix A is defined as the rank of the set of columns of A. This definition of rank turns out to be equivalent to the standard definition of rank over the field of quotients, $Q(D)$, of D. Note that Fact 1 would be immediate if the row rank of A and the column rank of A were equal, a familiar fact if D were a field. We include a proof for the general case of integral domains in the Appendix.

Notation: $< a, b >$ denotes the dot product of two vectors a and b.

2 A Lower Bound on the Size of the Sample Space

Theorem 1: Let S be the sample space used in Algorithm 2. If S contains a set of k distinct vectors $\{v_1, v_2, \ldots, v_k\}$ with rank at most $n-1$, then there are matrices A, B, and C such that $AB \neq C$ and $ABv_i = Cv_i$ for all i. Thus the probability of error is at least $\frac{k}{|S|}$.

Proof: Suppose the sample space S of n-vectors used in Algorithm 2 is such that there are k vectors $v_1, v_2, \ldots, v_k$ with rank at most $n-1$.

Now consider the $n \times k$ matrix

$$V = [\ v_1 \mid v_2 \mid \ \ldots \ \mid v_k\ ]$$

That is, V is obtained by adjoining these n-vectors of S.

By assumption, $rank(V) \leq n-1$. From Fact 1, there is a non-zero n-vector x such that $< x, v_i > = 0$ for all i.

Now consider the (non-zero) $n \times n$ matrix M whose first row is x and all other entries are 0. The matrix product Mv_i is 0 for any v_i chosen in Algorithm 2, since $< x, v_i >$ and $< 0, v_i >$ are both 0 for all i.

Now construct matrices A, B, and C such that $M = (AB - C)$. Then $AB \neq C$ and $ABv_i = Cv_i$ for all i. □

Corollary 2: Let S be the sample space used in Algorithm 2. Then the probability of error is at least $\min\{\frac{n-1}{|S|}, 1\}$.

Proof: Follows from Theorem 1 and the fact that any set of $n-1$ or fewer vectors has rank at most $n-1$. □

Corollary 3: If the probability of error in Algorithm 2 is at most $\epsilon < 1$, then the sample space for the algorithm must be of size at least $(n-1)/\epsilon$, and therefore at least $\log_2(n-1) + \log_2 \frac{1}{\epsilon}$ random bits must be used.

Proof: From Corollary 2, $\epsilon \geq \frac{n-1}{|S|}$, which gives the desired bound on $|S|$. □

This lower bound is tight for sufficiently large integral domains: it is matched by Algorithm 3.

3 A Characterization of the Sample Space

Under the conditions of Theorem 1, the probability of error is at least $\frac{k}{|S|}$. This section provides a converse to that result.

Theorem 4: Let S be the sample space used in Algorithm 2. If every set of k distinct vectors in S has rank n, then the probability of error of Algorithm 2 is no more than $\frac{k-1}{|S|}$.

Proof: We will prove that for every non-zero matrix $M = AB - C$, there are at most $k-1$ vectors in S whose product with M is 0.

Suppose this is not true. Then for at least k vectors of S (call them $v_1, \ldots, v_k$ and view them as column vectors) and some non-zero matrix M, $Mv_i = 0$.

Consider the matrix $V = [v_1 \mid v_2 \mid \ \ldots \ \mid v_k]$. Since M is non-zero, there exists a non-zero row m such that $mV = 0$. Therefore, by applying Fact 1, we conclude that $rank(V) \leq n-1$, which contradicts our assumption that every set of k vectors has rank n. □

Note that in Freivalds' original algorithm (Algorithm 1), every set of $2^{n-1}+1$ n-vectors over $\{0,1\}$ has rank n (since there are too many vectors to be contained in a subspace of dimension $n-1$), so that the error probability is at most $\frac{2^{n-1}}{2^n} = \frac{1}{2}$. In Kimbrel and Sinha's algorithm (Algorithm 3), every set of n vectors has rank n (since it forms a Vandermonde matrix), so the error probability is at most $\frac{n-1}{\lceil (n-1)/\epsilon \rceil} < \epsilon$.

4 A Generalization of Freivalds' Algorithm

Algorithm 2 uses a single test vector v to decide whether $AB = C$. We can modify the algorithm so that a carefully selected *set* of vectors v can be used to test whether $AB = C$. The idea is to trade time for the amount of randomness used. Any randomized algorithm running in time T and using R random bits can be easily transformed into a deterministic algorithm that runs in $T \cdot 2^R$ time. The hope is that some clever choice of parameters for Algorithm 4 will yield a deterministic algorithm that is more efficient than the best known (deterministic) algorithm for multiplying matrices.

Algorithm 4

1. Let $X = \{1, \ldots, r\}$.
2. Let W be a nonempty finite set of functions that map elements of X to n-vectors.
3. Choose an $l \in X$ uniformly at random.
4. For each $s \in W$
   4a. Let $v = s(l)$.
   4b. **if** $ABv \neq Cv$, then **return** "$AB \neq C$" (accept).
5. **return** "$AB = C$" (reject).

Analysis: Steps 1 and 2 require no time, since we view X and W as fixed sets. Step 3 takes $O(\log r)$ steps. Step 5 takes one step.

Step 4a takes time $t_s(l)$, where $t_s(l)$ is the number of steps it takes to compute $s(l)$ from l. Steps 4b takes $O(n^2)$ steps. So step 4 takes a total of $O(n^2\ |W| + t)$ steps, where $t = \sum_{s \in W} t_s(l)$.

The entire algorithm takes $O(n^2\ |W| + t + \log_2 r)$ steps and uses $\log_2 r$ random bits. We can adapt this algorithm to run deterministically in $O((n^2\ |W| + t + \log_2 r)\ r)$ steps by examining all choices of l in X.

Notice that both Algorithm 1 and Algorithm 3 are special cases of Algorithm 4. For Algorithm 1, $X = \{1, 2, \ldots, 2^n\}$ and $W = \{bin(x)\}$, where $bin(x)$ is a vector corresponding to the n-bit binary representation of the integer $x-1$; for Algorithm 3, $X = \{1, \ldots, \left\lceil \frac{n-1}{\epsilon} \right\rceil\}$ and $W = \{J(x)\}$, where $J(x) = [x^0\ x^1\ \ldots\ x^{n-1}]^T$. In Section 6 we will see another example of Algorithm 4, this one having $|W| > 1$ and $|X| < \lceil (n-1)/\epsilon \rceil$.

5 A Lower Bound for the Generalized Freivalds Algorithm

Theorem 5: For any choice of W and r in Algorithm 4, if there is a set of $k \leq r$ distinct indices $\{i_1, i_2, \ldots, i_k\}$ such that $rank\{s(i_j) \mid s \in W, 1 \leq j \leq k\}$ is at most $n-1$, then there are matrices A, B, and C such that $AB \neq C$ and Algorithm 4 rejects for each choice of $l \in \{i_1, i_2, \ldots, i_k\}$ in step 3. Thus the probability of error is at least $\frac{k}{r}$.

Proof: The proof mirrors that of Theorem 1. Consider the $n \times |W|k$ matrix V obtained by adjoining the $|W|k$ vectors $\{s(i_j) \mid s \in W, 1 \leq j \leq k\}$.

By assumption, $rank(V) \leq n-1$. From Fact 1, there is a non-zero n-vector x such that the inner product of x with each column in V is zero.

Now consider the (non-zero) $n \times n$ matrix M whose first row is x and all other entries are 0. The product of M with each column of V is zero. Construct matrices A, B, and C such that $M = (AB - C)$. Notice that Algorithm 4 rejects for $l = i_j$ if and only if the product of M with each vector in $\{s(i_j) \mid s \in W\}$ is zero. Then $AB \neq C$ and yet Algorithm 4 rejects for each choice of $l \in \{i_1, i_2, \ldots, i_k\}$ in step 3. □

Corollary 6: For any choice of W and r in Algorithm 4, if $|W| = w$ then there exists an input $< A, B, C >$ such that $AB \neq C$ and Algorithm 4 rejects with probability at least $\frac{1}{r}(\left\lceil \frac{n}{w} \right\rceil - 1)$.

Proof: Consider any $k = \left\lceil \frac{n}{w} \right\rceil - 1$ indices in the statement of Theorem 5. Then the set $\{s(i_j) \mid s \in W, 1 \leq j \leq k\}$ has at most $w \cdot k < w \cdot \frac{n}{w} = n$ vectors and therefore has rank at most $n-1$. □

Corollary 7: For any choice of W and r in Algorithm 4, if $|W| = w$, then $w(r+1) > n$.

Proof: Assume $w(r+1) \leq n$. Then by Corollary 6, the probability that Algorithm 4 rejects is at least $\frac{1}{r}(\left\lceil \frac{n}{w} \right\rceil - 1) \geq \frac{1}{r}(\lceil r+1 \rceil - 1) = 1$. This contradicts the assumption that the probability of error is less than 1. □

Corollary 8: Let T be the running time of Algorithm 4. Then $(r+1)T \geq n^3$.

Proof: This follows directly from Corollary 7 and the running time analysis of Algorithm 4, which shows that $T \geq |W|n^2$. □

Corollary 9: For Algorithm 4, if $T = O(n^2)$ then $r = \Omega(n)$.

This again shows that Kimbrel and Sinha's algorithm (Algorithm 3) is asymptotically optimal, at least with respect to this generalization of Freivalds' algorithm.

The next theorem shows a matching upper bound for the generalized Freivalds algorithm.

6 An Upper Bound for the Generalized Freivalds Algorithm

Theorem 10: For any choice of W and r in Algorithm 4, if for every set of $k \leq r$ distinct indices $\{i_1, i_2, \ldots, i_k\}$ the set of vectors $\{s(i_j) \mid s \in W, 1 \leq j \leq k\}$ has rank n, then the probability of error of is no more than $\frac{k-1}{r}$.

Proof: The proof mirrors that of Theorem 2. We will prove that for every non-zero matrix $M = AB - C$, there are at most $k-1$ choices of l in step 3 such that Algorithm 4 rejects.

Suppose this is not true. Then for at least k choices of l (call them $i_1, \ldots, i_k$) the algorithm rejects. Again, notice that Algorithm 4 rejects for $l = i_j$ if and only if the product of M with each vector in $\{s(i_j) \mid s \in W\}$ is zero. Consider the matrix V obtained by adjoining the vectors in $\{s(i_j) \mid s \in W, 1 \leq j \leq k\}$. Then the product of M with each column in V is zero. Since M is non-zero, there exists a non-zero row m such that $mV = 0$. Therefore, by applying Fact 1, we conclude that $rank(V) \leq n-1$, which contradicts our assumption. □

From Corollary 6, the probability of error is at least $\frac{1}{r}(\left\lceil \frac{n}{w} \right\rceil - 1)$. Here we give an *explicit* algorithm that achieves this error probability. We assume that the elements of our matrices are from an integral domain that has at least r elements. For the special case of small prime fields $GF(p)$, we can use techniques similar to Kimbrel and Sinha [6, Section 4] to achieve the same error probability by increasing the running time by a factor of $\log_p r$.

Theorem 11: For any integers w and r, and any integral domain D with at least r elements, we can construct a set W of cardinality w such that the error probability of Algorithm 4 for matrices over D is at most $\frac{1}{r}(\left\lceil \frac{n}{w} \right\rceil - 1)$. Also, its running time is $O(n^2\ w)$.

Proof: Let $X = \{1, 2, \ldots, r\}$, and let $x_1, \ldots, x_r$ be r distinct elements of D.

Let $q = \left\lceil \frac{n}{w} \right\rceil$.

Define $s_1, \ldots, s_w$ to be mappings from integers to n-vectors such that

$$s_{i,j}(l) = \begin{cases} x_l^{(j \bmod q)} & \text{if } (i-1)q \leq j \leq \min(iq-1, n-1) \\ 0 & \text{otherwise} \end{cases}$$

for $0 \leq j \leq n-1$, where $s_{i,j}(l)$ is the j-th element of $s_i(l)$.

The function s_i can be viewed as a function that generates the powers (from 0 to $q-1$) of x_l, places them in the i-th block of q elements, and sets all other components to 0.

Now let $W = \{s_1, \ldots, s_w\}$.

We now prove the correctness of Algorithm 4 with this X and W.

Case 1: If $AB = C$, then $ABv = Cv$ for all v, so $\Pr(ABv = Cv) = 1$.

Case 2: If $AB \neq C$, then we need to show that for at most $(\left\lceil \frac{n}{w} \right\rceil - 1) = q - 1$ choices of $l \in X$, $(AB - C)s(l) = 0$ for all s in W.

Let $M = AB - C$. Since M is non-zero, there is a non-zero row of M. Suppose $m = [m_1\ m_2\ \ldots\ m_n]$ is a non-zero row of M. Consider the set of integers l such that $(AB - C)s(l) = 0$ for all choices of s in W. For each such l,

$$\begin{aligned} m_1 \cdot (x_l)^0 + m_2 \cdot (x_l)^1 + \cdots + m_q \cdot (x_l)^{q-1} &= 0 \\ m_{q+1} \cdot (x_l)^0 + m_{q+2} \cdot (x_l)^1 + \cdots + m_{2q} \cdot (x_l)^{q-1} &= 0 \\ \vdots & \vdots \\ m_{(w-1)q+1} \cdot (x_l)^0 + \cdots + m_n \cdot (x_l)^{n-(w-1)q-1} &= 0. \end{aligned}$$

These w equations correspond to each of the w functions of W.

Since m is non-zero, at least one of these polynomials is non-zero. Since each of these polynomials has degree at most $q-1$, there are at most $q-1$ values of l for which all these equations are satisfied [7, Theorem 2, Section IV.3.3].

The running time of the algorithm is $O(n^2\ w)$, since $t = \sum_{s \in W} t_s(l) = nw$. □

Corollary 12: If $wr \geq \frac{n}{\epsilon}$, then there is a choice of W of cardinality w such that the error probability of Algorithm 4 is at most ϵ. Thus, if $\epsilon = \frac{1}{2}$, then $wr \geq 2n$ is sufficient.

Proof: From Theorem 11, the error probability of Algorithm 4 is at most $\frac{1}{r}(\left\lceil \frac{n}{w} \right\rceil - 1) \leq \frac{1}{r}(\lceil r\epsilon \rceil - 1) \leq \frac{1}{r} \cdot r\epsilon = \epsilon$. □

7 Conclusions

We have proved that Naor and Naor, Alon *et al.*, and Kimbrel and Sinha's algorithms use an asymptotically optimal number of random bits with respect to the Freivalds approach to matrix product verification. We also have provided a generalization of the Freivalds approach to matrix product verification and have shown that Algorithm 3 is optimal with respect to that generalization, in that the number of random bits used in Algorithm 3 matches the lower bound for all instances of Algorithm 4 that run in $O(n^2)$ time.

We have shown that given w and r such that $wr \geq 2n$, there exist sets W and X with cardinality w and r (respectively) such that Algorithm 4 runs in $O(n^2\ w)$ time and uses $\log_2 r$ random bits with error probability at most $\frac{1}{2}$. Also, the bounds on the product wr are tight (to within a factor of 2). Another way to view this result is that for each bit of randomness we save by halving the size of X, we double the running time of Algorithm 4 (by doubling the size of W).

More importantly, we have shown that no pseudorandom number generator that uses $\log_2 n - \omega(1)$ random bits can be used to produce an instance of Algorithm 2 or 4 that runs in $O(n^2)$ time.

Appendix

Throughout this section, *vector* is understood to mean an n-tuple of elements from an integral domain, whether or not the integral domain is a field.

Let D be an integral domain. A *D-module* is like a vector space except that the scalars are from D, which is not necessarily a field. See [5] for a more careful and general definition. (Modules are in general defined for rings.)

Definition 13: Let M be a D-module. Let $v_1, \ldots, v_r \in M$. The vectors in the set $\{v_1, \ldots, v_r\}$ are *linearly independent over D* if $\sum_{j=1}^{r} a_j v_j = 0$ implies that $a_j = 0$ for $j = 1, \ldots, r$.

Definition 14: Let M be a D-module. Given any set of vectors $\{v_1, v_2, \ldots, v_r\} \subseteq M$, $rank_D(\{v_1, v_2, \ldots, v_r\})$ is defined to be the cardinality of the largest subset of $\{v_1, v_2, \ldots, v_r\}$ that is linearly independent over D.

Definition 15: Let V be a vector space over a field F. Given any set of vectors $\{v_1, v_2, \ldots, v_r\} \subseteq V$, $rank_F(\{v_1, v_2, \ldots, v_r\})$ is defined to be the cardinality of the largest subset of $\{v_1, v_2, \ldots, v_r\}$ that is linearly independent over F.

Let D be an integral domain. The *field of quotients of D*, denoted $Q(D)$, bears the same relation to D as the field of rational numbers to the integral domain of integers. See [7, Section IV.2.2] for a precise definition. Elements of $Q(D)$ can be represented by a pair of elements (a, b) from D, where b is non-zero, and a is zero if and only if the element represented in $Q(D)$ is zero. For the rational numbers, this pair corresponds to the number $\frac{a}{b}$.

Theorem 16: Let M be the D-module consisting of the n-tuples over D (i.e. the direct product D^n) as its elements (and the usual "vector" operations defined for it). Let V be the vector space consisting of n-tuples over $Q(D)$.

Then if $v_1, \ldots, v_r \in M$ ($v_1, \ldots, v_r$ can be viewed as elements of V),

$$\sum_{i=1}^{r} a_i v_i = 0 \text{ and } a_i \in D \Rightarrow a_i = 0 \quad (i = 1, \ldots, r)$$

if and only if

$$\sum_{i=1}^{r} b_i v_i = 0 \text{ and } b_i \in Q(D) \Rightarrow b_i = 0 \quad (i = 1, \ldots, r).$$

That is, $\{v_1, \ldots, v_r\}$ are linearly independent over D if and only if $\{v_1, \ldots, v_r\}$ are linearly independent over $Q(D)$, when $v_1, \ldots, v_r$ are viewed as elements of M and V, respectively.

Proof: ($\Leftarrow$) If there are no $b_1, \ldots, b_r \in Q(D)$ (not all zero) such that $\sum_{i=1}^{r} b_i v_i = 0$, there are no $a_1, \ldots, a_r \in D$ (not all zero) such that $\sum_{i=1}^{r} a_i v_i = 0$, since D is isomorphic to a subdomain of $Q(D)$.

($\neg \Leftarrow \neg$) Suppose there were $b_1, \ldots, b_r \in Q(D)$ (not all zero) such that $\sum_{i=1}^{r} b_i v_i = 0$. Then since $b_i \in Q(D)$, $b_i = (c_i, d_i)$ for $c_i, d_i \in D$, where not all the c_i are zero, and none of the d_i is zero. Let $d = \prod_{i=1}^{r}(d_i, 1)$. Then $d \sum_{i=1}^{r} b_i v_i = 0$ implies $\sum_{i=1}^{r} d b_i v_i = 0$, which implies that $\sum_{i=1}^{r} [d\,((c_i, d_i))]\, v_i = 0$.

But $e_i = d \cdot (c_i, d_i)$ is the image of an element of D. Therefore there are $a_1, \ldots, a_r \in D$ (not all zero) such that $\sum_{i=1}^{r} a_i v_i = 0$, by letting $a_i = e_i$ from above. $\square$

Corollary 17: Given any integral domain D, the D-module M consisting of the n-tuples over D, and any set of vectors $\{v_1, v_2, \ldots, v_r\} \subseteq M$,

$$rank_D(\{v_1, v_2, \ldots, v_r\}) = rank_{Q(D)}(\{v_1, v_2, \ldots, v_r\}).$$

Proof of Fact 1: We first claim that $rank_D$(columns of A) = $rank_{Q(D)}$(columns of A) = $rank_{Q(D)}$(rows of A) = $rank_D$(rows of A).

The first and third equalities follow from Corollary 17. The second equality is a well-known fact from linear algebra.

So $rank_D$(columns of A) $\leq n-1 \iff rank_D$(rows of A) $\leq n-1 \iff$ there is a non-zero x such that $xA = 0$, where the second equivalence follows from the definition of $rank_D$. $\square$

Acknowledgements

We are grateful to Martin Tompa for his encouragement and insights in this work.

References

[1] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost k-wise independent random variables. In *31st Annual Symposium on Foundations of Computer Science*, pages 544–553, St. Louis, MO, October 1990. IEEE.

[2] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[3] G. Even, O. Goldreich, M. Luby, N. Nisan, and B. Veličković. Approximations of general independent distributions. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 10–16, Victoria, BC, Canada, May 1992.

[4] R. Freivalds. Fast probabilistic algorithms. In *Mathematical Foundations of Computer Science: Proceedings, 8th Symposium*, volume 74 of *Lecture Notes in Computer Science*, pages 57–69. Springer-Verlag, 1979.

[5] I. N. Herstein. *Topics in Algebra.* John Wiley & Sons, second edition, 1975.

[6] T. Kimbrel and R. K. Sinha. A probabilistic algorithm for verfiying matrix products using $O(n^2)$ time and $\log_2 n + O(1)$ random bits. Technical Report TR 91-08-06, University of Washington Department of Computer Science and Engineering, August 1991. To appear in *Information Processing Letters.*

[7] John D. Lipson. *Elements of Algebra and Algebraic Computing.* Addison-Wesley, Reading, MA, 1981.

[8] J. Naor and M. Naor. Small-bias probability spaces: efficient constructions and applications. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 213–223, Baltimore, MD, May 1990.