A Probabilistic Algorithm for Verifying Matrix Products Using $O(n^2)$ Time and $\log_2 n + O(1)$ Random Bits.

Tracy Kimbrel*
Rakesh Kumar Sinha†
Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, Washington, U.S.A. 98195

August 8, 1991

Abstract

A one-sided error probabilistic algorithm is given that determines, for $n \times n$ input matrices A, B, and C, whether $AB \neq C$, using $O(n^2)$ multiplications and additions and $\lceil \log_2 n \rceil + 1$ random bits. We further show how to reduce the error probability to ϵ with only an additional $\lceil \log_2(\frac{1}{\epsilon}) \rceil$ random bits.

*This material is based upon work supported under a National Science Foundation Graduate Fellowship.

†This material is based upon work supported by the National Science Foundation under grants CCR–8858799 and CCR–8907960.

1. Introduction

Given two $n \times n$ matrices A and B, computing their product is a classic problem. We consider a related decision problem: given three $n \times n$ matrices A, B, and C, how difficult is it to verify whether $AB = C$? Freivalds [2] gave a probabilistic algorithm to verify matrix products using $O(n^2)$ multiplications and additions, and n bits of randomness. The algorithm accepts the set $\{\langle A, B, C\rangle \mid AB \neq C\}$ with one-sided error. That is, the algorithm always rejects if $AB = C$, and accepts with probability at least $\frac{1}{2}$ if $AB \neq C$.

Freivalds' algorithm selects a vector $\vec{v}$ at random from $\{-1, 1\}^n$. It then computes the products $A(B\vec{v})$ and $C\vec{v}$, and accepts (reports that $AB \neq C$) if and only if $AB\vec{v} \neq C\vec{v}$. A similar method is used here, but the vector $\vec{v}$ is chosen from a set of less than $4n$ vectors, requiring only $\lceil \log_2 n \rceil + 1$ random bits.

Naor and Naor [6], and Alon, Goldreich, Håstad, and Peralta [1] give constructions of polynomial size (polynomial in n) sample spaces of n-bit vectors, yielding $O(\log n)$ randomness solutions to this problem. Our method is similar to the constructions of Alon *et al.*, particularly their powering construction. The constructions of Alon *et al.* can be used to implement Freivalds' algorithm using $2 \log_2 n + O(1)$ random bits. In the case of matrices over GF(2), the construction of Naor and Naor based on linear codes requires only $\log_2 n + O(1)$ random bits; in the general case of matrices over an arbitrary ring, their method uses $2 \log_2 n + O(\log \log n)$ bits. While Naor and Naor and Alon *et al.* have solved a more general problem, our algorithm yields a slightly better constant (in terms of the random bit requirement) in the general case. More importantly, our algorithm is much simpler and more direct, and it can easily be extended to make the probability of error less than any ϵ with only $\lceil \log_2(\frac{1}{\epsilon}) \rceil$ additional random bits.

In section 2, we present the simplest version of the algorithm for integer matrices. Section 3 shows how to improve the bit complexity, section 4 extends the algorithm to matrices over finite fields, and section 5 shows how to reduce the error probability.

2. The Algorithm

In order to carry out Step 1 below using exactly $\log_2(2n)$ random bits, assume that n is a power of 2 (otherwise, modify Step 1 below to choose x between 1 and m, where m is the smallest power of 2 which is larger than $2n$). Also assume that the input matrices are over the integers; we will later modify the algorithm for matrices over other algebraic structures. The one-sided error algorithm is as follows:

Algorithm 1

```
    **Step 1** choose a random integer $x$ uniformly between 1 and $2n$.
    **Step 2** $\vec{v} \longleftarrow [x^0, \ldots, x^{n-1}]^T$.
    **Step 3** if $AB\vec{v} \neq C\vec{v}$ then *accept* else *reject*.
```

We will need the following definition to prove the correctness of Algorithm 1.

Definition: The *Vandermonde* matrix $V = V(x_1, \ldots, x_n)$ of n distinct elements $x_1, \ldots, x_n$ is formed by letting the j^{th} column (for $1 \leq j \leq n$) be composed of the first n powers of x_j; that is, $V_{ij} = x_j^{i-1}$.

The following lemma is easily proved using the interpolation theorem.

Lemma 1: Over any integral domain, every Vandermonde matrix of n distinct non-zero elements is nonsingular.

(See Proposition 1, page 300 of Lipson [5] for a proof.)

We are now ready to state our first result.

Theorem 2: Algorithm 1 accepts the triple of matrices $\langle A, B, C\rangle$ with probability greater than $\frac{1}{2}$ if $AB \neq C$, and rejects if $AB = C$. The algorithm uses $3n^2+O(n)$ multiplications, $3n^2+O(n)$ additions, and $\lceil \log_2 n \rceil + 1$ random bits.

Proof:

Time complexity: By associativity, $AB\vec{v}$ can be computed as $A(B\vec{v})$. $C\vec{v}$ and $B\vec{v}$ can each be computed using n^2 multiplications and $n^2 - n$ additions.

$AB\vec{v}$ can then be computed using another n^2 multiplications and $n^2 - n$ additions.

Randomness: $\lceil \log_2 n \rceil + 1$ random bits are sufficient to choose $x \in \{1, \ldots, 2 \cdot 2^{\lceil \log_2 n \rceil}\}$.

Correctness: If $AB = C$, then $AB\vec{v} = C\vec{v}$ for all $\vec{v}$, so $\Pr(AB\vec{v} = C\vec{v}) = 1$, and the algorithm (correctly) rejects.

If $AB \neq C$, then $AB\vec{v} = C\vec{v}$ for at most $n-1$ of the $2n$ or more choices of x in Step 1. To see this, notice that any such choice of x produces a column vector $\vec{v}$ such that $AB\vec{v} = C\vec{v}$. If there were n distinct such choices $x_1, \ldots, x_n$ of x, this would produce a matrix $V = V(x_1, \ldots, x_n)$ which is nonsingular (by Lemma 1) and which satisfies $ABV = CV$, contradicting the hypothesis that $AB \neq C$. Thus, if $AB \neq C$, the algorithm accepts with probability greater than $\frac{1}{2}$. $\square$

3. Improving Bit Complexity

The algorithm as outlined above suffers from several shortcomings. The test vector $\vec{v}$ may have very large entries (as large as $(2n)^{n-1}$). If the input matrices A, B, and C all have small entries, this will lead to a running time (counting bit operations rather than integer multiplications and additions) no better than even the simplest deterministic algorithm for matrix multiplication. We fix this problem by modifying step 2 of Algorithm 1.

Algorithm 2
```
    Step 1 choose a random integer $x$ uniformly between 1 and $2n$.
    Step 2 let $p$ be any prime between $2n$ and $4n$.
               $\vec{v} \longleftarrow [x^0 \bmod p, \ldots, x^{n-1} \bmod p]^T$.
    Step 3 if $AB\vec{v} \neq C\vec{v}$ then *accept* else *reject*.
```

The correctness of the modified algorithm follows from the fact that Lemma 1 applies to $GF(p)$. If we consider the set of vectors $(x_j^0, \ldots, x_j^{n-1})$ for $1 \leq j \leq 2n$ with the entries reduced modulo p, any n choices form a Vandermonde matrix which is nonsingular over $GF(p)$, and therefore nonsingular over the integers. The rest of the proof of correctness remains unchanged.

For every $k \geq 1$, there is a prime between k and $2k$ (Theorem 418 of Hardy and Wright [3]); thus, there is some prime p, $2n \leq p \leq 4n$. Let m be the maximum of $4n$ and the largest absolute value in A, B, and C, and let $b = \lceil \log_2 m \rceil$. We can use the sieve method to find p. It is straightforward to see that this requires $O(n^{3/2})$ additions of $O(\log n)$-bit integers. (One can use Merten's lemma (Theorem 427 of Hardy and Wright [3]) to argue that the number of additions is only $O(n \log \log n)$.) The overall running time of the algorithm is $O(n^2 b^2)$ (counting bit operations) using simple algorithms for integer multiplication and addition.

Theorem 3: Algorithm 2 accepts the triple of matrices $\langle A, B, C \rangle$ with probability greater than $\frac{1}{2}$ if $AB \neq C$, and rejects if $AB = C$. Let m be the maximum of $4n$ and the largest absolute value in A, B, and C, and let $b = \lceil \log_2 m \rceil$. The algorithm uses $O(n^2 b^2)$ bit operations and $\lceil \log_2 n \rceil + 1$ random bits.

4. Matrices over Finite Fields

Notice that Algorithm 1 will work as long as the entries of the input matrices come from an integral domain. However, a far more severe restriction is the implicit assumption that the domain contains at least $2n$ elements so that Step 1 can be carried out. In particular, Algorithm 1 will fail for the important case of testing matrix products over $GF(p)$ where $p < 2n$. Here we suggest a modification of Algorithm 1 to deal with this case.

Let p^α be the smallest integral power of p which is larger than $2n$. In fact $p^\alpha = O(pn)$ suffices.

Fact: If A, B, and C are matrices over $GF(p)$, then $AB = C$ in $GF(p) \Leftrightarrow AB = C$ in $GF(p^\alpha)$.

Since $GF(p^\alpha)$ contains more than $2n$ elements, we can use the method of Algorithm 1 to test matrix products in $GF(p^\alpha)$. Computing the first n powers of x requires $O(n\alpha^2)$ $GF(p)$–operations. Since the entries of A, B, and C come from $GF(p)$, computing $A(B\vec{v})$ and $C\vec{v}$ requires only $O(n^2\alpha)$ $GF(p)$–operations. The only complication is that an irreducible monic polynomial of degree α over $GF(p)$ is needed to do $GF(p^\alpha)$ arithmetic. Such a polynomial can be found within the overall time bound of $O(n^2\alpha)$; for instance, the method of Shoup [8] can be used. However, a much more naive and simple algorithm will suffice for our purposes. Consider the two cases $p \leq \sqrt{2n}$ and $p > \sqrt{2n}$ separately.

Suppose $p > \sqrt{2n}$. In this case, $\alpha = 2$. If we can find a quadratic nonresidue y in $GF(p)$, then $x^2 - y$ is an irreducible monic polynomial over $GF(p)$. Finding a quadratic nonresidue by squaring each element in $GF(p)$ requires only $O(n)$ $GF(p)$–operations.

Now suppose $p \leq \sqrt{2n}$. In this case, we can enumerate all the monic polynomials of degree at most α and find an irreducible polynomial using trial division. This requires $O(p^{\alpha+\lfloor\frac{\alpha}{2}\rfloor}\alpha^2)$ $GF(p)$–operations. The claim is that this number is at most $O(n^2)$. To see this consider the three cases $p \leq (2n)^{\frac{1}{4}}$, $(2n)^{\frac{1}{4}} < p \leq (2n)^{\frac{1}{3}}$, and $(2n)^{\frac{1}{3}} < p \leq \sqrt{2n}$. In the first case, the number of $GF(p)$–operations is $O(n^{\frac{15}{8}}\alpha^2) = O(n^2)$. In the next two cases, α is respectively 4 and 3. Again, it is easy to verify the claim in both these cases.

Algorithm 3

```
    Step 1 choose a random integer $i$ uniformly between 1 and $2n$.
           let $x$ be the $i^{th}$ element (lexicographically) of $GF(p^\alpha)$.
    Step 2 $\vec{v} \longleftarrow [x^0, \ldots, x^{n-1}]^T$.
    Step 3 if $AB\vec{v} \neq C\vec{v}$ over $GF(p^\alpha)$ then accept else reject.
```

Theorem 4: Algorithm 3 accepts the triple of matrices $\langle A, B, C\rangle$ over $GF(p)$ with probability greater than $\frac{1}{2}$ if $AB \neq C$, and rejects if $AB = C$. The algorithm uses $O(n^2 \log n)$ $GF(p)$–operations and $\lceil \log_2 n \rceil$ +1 random bits.

5. Reducing the Probability of Error

Suppose we wish to reduce the probability of error to ϵ. The straightforward approach involves running Algorithm 1 for k independent trials, where $(\frac{1}{2})^k < \epsilon$, or $k > \log_2(\frac{1}{\epsilon})$. This strategy requires $\lceil \log_2(\frac{1}{\epsilon})\rceil \lceil \log_2(n)\rceil$ random bits and $\Omega(n^2 \log(\frac{1}{\epsilon}))$ operations. Using the techniques of Impagliazzo and

Zuckerman [4] we can achieve the same error bound with only $O(\log(\frac{1}{\epsilon}))$ additional random bits. Here we show how our method can easily be extended to achieve the same error bound with only $\lceil \log_2(\frac{1}{\epsilon}) \rceil$ additional random bits and no increase in the number of integer multiplications and additions.

Algorithm 4

```
    **Step** 1 Choose a random integer $x$ uniformly between 1 and $\lceil \frac{n}{\epsilon} \rceil$.
    **Step 2** $\vec{v} \longleftarrow [x^0, \ldots, x^{n-1}]^T$.
    **Step 3** if $AB\vec{v} \neq C\vec{v}$ then *accept* else *reject*.
```

With this modification, the probability of making an error is at most $\frac{n-1}{n/\epsilon} < \epsilon$.

Theorem 5: Algorithm 4 accepts the triple of integer matrices $\langle A, B, C \rangle$ with probability greater than $(1 - \epsilon)$ if $AB \neq C$, and rejects if $AB = C$. The algorithm uses $O(n^2)$ integer multiplications and additions, and $\lceil \log_2 n \rceil + \lceil \log_2(\frac{1}{\epsilon}) \rceil$ random bits.

This technique can be used in conjunction with that of Section 3 used to reduce the size of the entries of the test vector modulo a prime in the case of integer matrices, for $\epsilon = \Omega(1/n)$. Finding the first prime greater than $\frac{n}{\epsilon}$ seems to be prohibitive in running time for smaller values of ϵ. Pritchard [7] shows how to find the smallest prime larger than N using $O(\frac{N \log N}{\log \log N})$ bit operations. Even if we use this result, our algorithm would still run in time $\Omega(n^3)$ for all $\epsilon = O(\frac{1}{n^2})$.

This method extends to the problem of testing matrix products over finite fields provided ϵ is not too small. However, the sophisticated algorithm of Shoup must be used to find an irreducible monic polynomial rather than the simple method described earlier. It would be nice to find a simpler algorithm to find a monic irreducible polynomial without a prohibitive running time.

Acknowledgements

We are grateful to Martin Tompa for many ideas and constant encouragement, to Richard Ladner for the idea of using extension fields, to Avi Wigderson for pointing out the results of Naor and Naor and Alon *et al.* , and to Paul Beame and Simon Kahan for many helpful discussions.

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, Oct. 1990. IEEE.

[2] 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.

[3] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford University Press, fifth edition, 1979.

[4] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *30th Annual Symposium on Foundations of Computer Science*, pages 248–253, Research Triangle Park, NC, Oct. 1989. IEEE.

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

[6] 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.

[7] P. Pritchard. A sublinear additive sieve for finding prime numbers. *Communications of the ACM*, 24:18–23, 1981.

[8] V. Shoup. New algorithms for finding irreducible polynomials over finite fields. In *29th Annual Symposium on Foundations of Computer Science*, pages 283–290, White Plains, NY, Oct. 1988. IEEE.