Fast Nondeterministic Matrix Multiplication via Derandomization of Freivalds' Algorithm*

Jiří Wiedermann

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz

Abstract. We design two nondeterministic algorithms for matrix multiplication. Both algorithms are based on derandomization of Freivalds' algorithm for verification of matrix products. The first algorithm works with real numbers and its time complexity on Real RAMs is $O(n^2 \log n)$. The second one is of the same complexity, works with integer matrices on a unit cost RAM with numbers whose size is proportional to the size of the largest entry in the underlying matrices. Our algorithms bring new ideas into the design of matrix multiplication algorithms and open new avenues for their further development. The results pose exciting questions concerning the relation of the complexity of deterministic versus nondeterministic algorithms for matrix multiplication, and complexity of integer versus real matrices multiplication.

1 Introduction

Matrix multiplication probably is one of the most closely watched areas of algorithm design. A great attention is paid even to minor advancements in this field. Undoubtedly, this is because of the immense practical importance of the problem backed by its deep algorithmic beauty. Matrix multiplication has attracted considerable attention for more than four decades and the challenge is whether or not matrix multiplication can be done in quadratic time. The origin of this conjecture lies back in the late nineteen sixties when Volker Strassen discovered an algorithm for matrix multiplication of complexity $O(n^{2.807})$ [23] (in terms of the number of arithmetical operations). This has been an unexpected and significant improvement over the classical algorithm of complexity $O(n^3)$. Since then the search for matrix multiplication algorithms of complexity $O(n^{2+\omega})$, for $0 \leq \omega < 1$, has started and over the years it has proceeded through several incremental improvements of the ω exponent. The best current upper bound achieves the algorithm by Vasilevska Williams from 2011 with $\omega = 0.3729$.

The increased interest in matrix multiplication algorithms came about in the same time as the increased interest in randomized algorithms. Randomization has become a subject of systematic investigation as a "cheap" tool for solution

* This research was partially supported by RVO 67985807 and the GA ČR grant No. P202/10/1333.

J. Diaz et al.(Eds.): TCS 2014, LNCS 8705, pp. 123–135, 2014.

of problems for which no efficient deterministic algorithms have been known. Could randomization also help in matrix multiplication?

Perhaps this was the motivation of Rusins Freivalds who in 1977 designed a randomized algorithm with a bounded error probability for verifying matrix multiplication in quadratic randomized time [10], [11]. This algorithm has become one of the first algorithms showing the advantage of randomized algorithm over the deterministic ones also in a domain related to matrix multiplication and no wonder that Freivalds' algorithm has become a standard textbook example illustrating the power of randomized computations over the deterministic ones (cf. [7],[17]). At the same time this result has indicated that for a task related to matrix multiplication, viz. its verification, a quadratic time bound can be achieved.

The idea of Freivalds' algorithm is as follows. In order to verify whether $\mathbf{AB} = \mathbf{C}$ for any three matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ of real numbers of size $n \times n$, Freivalds' algorithm chooses a specific (column) vector $\mathbf{x}$ of length n and compares the product $\mathbf{ABx}$ with the product $\mathbf{Cx}$. Both products can be computed using $O(n^2)$ arithmetical operations (the former product thanks to the associativity of the matrix products: $(\mathbf{AB})\mathbf{x} = \mathbf{A}(\mathbf{Bx})$). The entries of vector $\mathbf{x}$ are uniformly chosen from the set $\{0, 1\}$. It can be shown that if $\mathbf{AB} \neq \mathbf{C}$, then the Freivalds algorithm returns a wrong answer with the probability at most $1/2$ (cf. [7]). The probability of error can be reduced to $1/2^k$ by performing k independent iterations of the algorithm.

Similarly as in the case of matrix multiplication algorithms research effort aiming at the performance improvement of Freivalds' algorithm has followed. The goal was to diminish the amount of randomness in this algorithm (cf. [18], [14]). This has culminated by the recent result of Korec and Wiedermann [15] showing that integer matrix product verification can entirely be derandomized. For real matrices only one random real number is needed in order to verify the correctness of matrix product with probability one. Both verification algorithms run in quadratic time w.r.t. the number of arithmetic operations. The deterministic algorithm for integer matrix verification has straightforwardly lead to the design of a nondeterministic algorithm for computing the product of two integer matrices in quadratic time, answering thus positively the conjecture concerning the complexity of matrix product, albeit only for integer matrices and for non-deterministic algorithms.

In order to completely understand the development sketched above one also has to pay attention to the underlying models of computation. The standard computational model for matrix product is by arithmetic circuits (aka straight line programs) over some field $\mathbb{F}$. The inputs to the circuit are the entries of the two matrices and the output from the circuit are the entries of the resulting matrix product. All entries are elements of $\mathbb{F}$. In the circuit's gates arithmetic operations with the field elements are allowed.

For verification of matrix products a stronger computational model is used. It is a probabilistic model allowing random moves and in addition to the standard arithmetic operations also comparisons between the elements of the field. For the

case of integer matrices an appropriate underlying model is probabilistic unit-cost RAM (cf [1]). When one wants to take into account the size of integers manipulated during a computation the log-cost RAM model should be used. For the case of real matrices, the so-called Real RAM (aka BSS machine [4]) model is used. Essentially, Real RAM is a Random Access Machine with registers that can store arbitrary real numbers and that can compute rational functions over reals at unit cost.

Quantum algorithms for matrix product verification and matrix multiplication have been considered, too (cf. [8], [16]).

Finally, in [15] for computing the product of integer matrices in quadratic time a nondeterministic model has been used. Bringing nondeterminism into the matrix multiplication means a further shift along the scale of the computational power of the underlying computational devices. As we shall see later the departure from the standard models of arithmetical circuits towards more powerful models of computations is not a purposeless move serving for circumventing the existing obstacles by using more powerful tools. Rather, such a proceeding allows approaching the problem from an other end. It might help in the development of a theoretical background for the development of more practical methods.

The present paper fits into the line of research efforts sketched above. Namely, it investigates the benefits that can be gained from considering nondeterministic algorithms in the context of matrix multiplication. It is based on the previous paper [15] in which the framework for the present approach has been established. It solves some problems that have remained open in the previous paper. Progress along these lines has been enabled by using a new fast deterministic criterion suitable for verification of matrix products over infinite and finite fields. Specifically, the paper brings an efficient algorithm for the nondeterministic multiplications (or deterministic verification of products) of matrices with real entries. On a Real RAM this task can be done deterministically using $O(n^2 \log n)$ arithmetical operations. When compared to the probabilistic algorithm for the latter task from [15], the factor of $\sim \log n$ seems to be a penalization for derandomization of the Freivalds' algorithm. Next, an algorithm of similar complexity is designed for multiplication of integer matrices using modular arithmetic. Its advantage over the quadratic-time algorithm for the same task designed in [15] is that the new algorithm does not compute with large numbers; the size of all numbers in this algorithm is bounded by the size of the largest entry in the underlying matrices.

The contribution of the present paper does not merely lie in the design of the previous algorithms which establish new upper bounds on the complexity of matrix multiplication on the non-deterministic models of computations. What is perhaps more important is that these algorithms make use of algebraic techniques that so far have not been often considered in the matrix multiplication research. Among other things these techniques include elements of the theory of Vandermonde matrices, properties of polynomial roots, multipoint polynomial evaluation, primality certificates, fast Fourier transform and number theoretic transform. It is hoped that the insight gained by the use of such techniques will

help to understand better the nature of matrix multiplication algorithms and eventually bring a further progress to the underlying field.

Our results raise a number of interesting questions to which we presently do not know the answers. For instance, is multiplication of integer matrices easier than multiplication of real matrices, under the unit cost model? So far we do not know an algorithm for nondeterministic multiplication of real matrices of quadratic complexity, albeit such an algorithm for integer matrices in known. By the way, the existence of the latter algorithm implies that any proof for superquadratic lower bound for integer matrix multiplication must be constructed so as not to hold for nondeterministic computations or must take into account the size of integers stored in the RAM registers. Nevertheless, perhaps the most important question arising from our research is the question, whether nondeterminism helps in matrix multiplication. A positive answer would be a surprise since it would put matrix multiplication among the problems that separate determinism from nondeterminism. In any case, our results seem to strengthen the hope that there exist substantially better deterministic algorithms for matrix multiplication than those known today.

The structure of the paper is as follows. In Section 2 those results from the predecessor paper [15] are briefly reviewed that will be used in the elaboration of the new results presented here. Especially, Lemma 1 is presented here that forms the starting point of all subsequent considerations, inclusively of a nondeterministic Algorithm 1 for multiplication of integer matrices in quadratic time on a unit cost RAM (with no bounds on the size of RAM registers). In Section 3 it is shown that ideas from Algorithm 1 cannot be straightforwardly used for multiplication of real matrices. To that end a corollary of Lemma 1 is proved serving as a new criterion enabling efficient verification of the product of real matrices. This results into the design of a nondeterministic Algorithm 2 for multiplication of real matrices on Real RAM in time $O(n^2 \log n)$. In Section 4 we return to the problem of multiplication of integer matrices using only registers of bounded-size. This is achieved by using modular arithmetic over a properly selected finite field. The resulting Algorithm 3 is of complexity $O(n^2 \log n)$ on a unit-cost RAM and computes with registers whose size is proportional to the size of the largest entry in the underlying integer matrices. In order to achieve their complexity as stated here both Algorithm 2 and Algorithm 3 make use of an appropriate version of the fast Fourier transform tuned to the underlying number field. Section 5 contains an overview of the results presented in this paper, open problems and conclusions.

2 Preliminaries

The key to all results in this paper is the following lemma from [15] which is related to the theory of Vandermonde matrices. Here, we recall this lemma inclusively of its proof because we will refer to it in our further considerations:

Lemma 1. *If* $\mathbf{D} \neq \mathbf{0}$ *is a real matrix of size* $n \times n$ *then there are at most* $n-1$ *real numbers* r *such that* $\mathbf{D}(1, r, r^2, \ldots, r^{n-1})^T = \mathbf{0}$

Proof: Since $\mathbf{D} = \{d_{i,j}\} \neq \mathbf{0}$ at least one row of $\mathbf{D}$ is non-zero. In the resulting matrix-vector product there is one algebraic equation in indeterminate r of degree less than n corresponding to this non-zero row. If j is the index of this row then the respective equation is of form $\sum_{i=1}^{n-1} d_{i,j}x^i = 0$ and for some i, $d_{i,j} \neq 0$. This equation has at most $n-1$ real roots. □

Based on the previous lemma a simple probabilistic algorithm of quadratic time complexity for verifying the product of two matrices has been designed in [15]. We sketch the respective algorithm since it will be the basis of our further improvements.

To verify $\mathbf{AB} = \mathbf{C}$, we pick a "random" real number r, create vector $\mathbf{x} = (1, r, r^2, \ldots, r^{n-1})^T$ and compute $\mathbf{Y} = \mathbf{A}(\mathbf{Bx}) - \mathbf{Cx}$.

If $\mathbf{Y} = \mathbf{0}$ then $\mathbf{AB} = \mathbf{C}$ with probability 1 (because among all reals there are at most $n-1$ "bad" numbers r causing $(\mathbf{AB} - \mathbf{C})\mathbf{x} = \mathbf{0}$ even if $\mathbf{AB} \neq \mathbf{C}$).

If $\mathbf{Y} \neq \mathbf{0}$ then $\mathbf{AB} \neq \mathbf{C}$ "for sure".

Obviously, $\mathbf{Y}$ can be computed in $O(n^2)$ operations if matrix-vector products are computed first.

The probabilistic step in this algorithm could be eliminated if we could deterministically choose r in such a way that $(\mathbf{AB} - \mathbf{C})\mathbf{x} = \mathbf{0}$ if and only if $\mathbf{AB} = \mathbf{C}$. That is, if $\mathbf{D} = \mathbf{AB} - \mathbf{C} = \{d_{i,j,}\}$, such an r cannot be a root of any polynomial of form $P_i(r) = \sum_{j=1}^{n} d_{i,j}x^{j-1} = 0$ for $i = 1, 2, \ldots, n$. Such an r can be found using the so-called Cauchy's bound ([9], p. 122, or the textbook [13], p. 82):

Theorem 1. *Let $P(x) = a_k x^k + a_{k-1}x^{k-1} + \ldots + a_1 x + a_0$ be a polynomial with real coefficients. If x is a real root of $P(x) = 0$ then $|x| < 1 + A/|a_k|$, with $A = \max_{i=0}^{k-1}\{|a_i|\}$.*

One can see that in order to upper-bound the magnitude of the roots we have to know coefficient a_k and the maximum of the absolute value of all coefficients in a polynomial. In our case, $k = n$ in the previous theorem, $d_{i,j} = \sum_{j=1}^{n} a_{i,j}b_{j,i} - c_{i,j}$ and $P_i(r) = \sum_{j=1}^{n} d_{i,j}r^{j-1} = 0$ for $i = 1, 2, \ldots, n$.

Let $c_{max} = \max\{|a_{i,j}|, |b_{i,j}|, |c_{i,j}|\}$. Then the maximal coefficient in any polynomial — the value of A — can be upper-bounded by $nc_{max}^2 + c_{max}$. Further, for integer matrices, and only for integer matrices, for any i, the absolute value of the leading coefficient in front of the highest power of r in $P_i(r)$ can be lower-bounded by 1 since the leading coefficient must be an integer.

Then, from Cauchy's bound it follows that for any i the absolute values of the roots of polynomial $P_i(r)$ with integer coefficients are upper-bounded by $\alpha = nc_{max}^2 + c_{max} + 1$.

Thus, choosing $r \geq \alpha$ in the previous proposition will guarantee that $P_i(r) \neq 0$ and hence $\mathbf{Dx} = \mathbf{0}$ can only hold for $\mathbf{D} = \mathbf{0}$. From these considerations a deterministic algorithm of quadratic complexity for integer matrix multiplication verification and nondeterministic algorithm for integer matrix multiplication follow easily (cf. [15]). Here we will only give the algorithm for nondeterministic matrix multiplication that will be the subject of our further improvements (cf. Algorithm 1).

Algorithm 1:
A non-deterministic integer matrix multiplication algorithm:

1. Input non-zero integer matrices $\mathbf{A}$ and $\mathbf{B}$;
2. Guess matrix $\mathbf{C} \neq \mathbf{0}$ with the absolute values of entries bounded by $r = \alpha$;
3. Compute vector $\mathbf{x} = (1, r, r^2, \ldots, r^{n-1})^T$;
4. If $\mathbf{A}(\mathbf{Bx}) = \mathbf{Cx}$ then output $\mathbf{C}$.

In [15] it is shown that Algorithm 1 can be generalized to the case of matrices of rational numbers given as numerator and denominator pairs.

Note that the seemingly similar idea of guessing $\mathbf{C}$ and its subsequent probabilistic verification *à la* Freivalds cannot work since through the error margin of the respective algorithm, however small, matrices $\mathbf{C}$ can come through for which $\mathbf{AB} \neq \mathbf{C}$. Thus, getting rid of probabilistic features of a verification algorithm for matrix multiplication turns out to be a crucial ingredient for the success of our nondeterministic algorithm.

3 Non-deterministic Multiplication of Real Matrices

Can we adjust Algorithm 1 to also work with real matrices? Unfortunately, no. What will not go through for reals is the upper bound on the magnitude of the maximal root for a class of polynomials with real coefficients depending on matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. Namely, although we can estimate the size of A in Cauchy's bound, we cannot get a lower bound on a_k whose value can be arbitrary close to 0. This will make the value of A/a_k arbitrarily large and subsequently we cannot get a bound on $|x|$. Thus, it appears that we cannot easily find number r such that $\mathbf{AB} = \mathbf{C}$ if and only if $\mathbf{ABx} = \mathbf{Cx}$ for $\mathbf{x} = (1, r, r^2, \ldots, r^{n-1})^T$.

Nevertheless, from Lemma 1 it follows that no matter how we pick n distinct real numbers $r_1, r_2, \ldots, r_n$, among them there will be at least one number r_j that is not a root of the respective polynomial. Such an r_j will certify $\mathbf{AB} \neq \mathbf{C}$ if and only if $\mathbf{ABx}_j \neq \mathbf{Cx}_j$ for $\mathbf{x}_j = (1, r_j, r_j^2, \ldots, r_j^{n-1})^T$. This suggests a possible strategy for a verification algorithm. For $j = 1, 2, \ldots, n$ we test whether $\mathbf{ABx}_j = \mathbf{Cx}_j$. If the equality will be confirmed for all j then $\mathbf{AB} = \mathbf{C}$. Otherwise $\mathbf{AB} \neq \mathbf{C}$.

Thanks to a special form of vectors $\mathbf{x}_j$ it is promising to realize that the product $\mathbf{Bx}_j$ for all $j = 1, 2 \ldots, n$ can be computed faster than in cubic time. Namely, we can reorganize the computation of $\mathbf{Bx}_j$ for $j = 1, 2, \ldots, n$ in such a way that we compute the value of each polynomial defined by each row of $\mathbf{B}$, at points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$.

Let $P(i, x) = \sum_{j=1}^{n} b_{i,j} x^{j-1}$ be the polynomial corresponding to the product of the i-th row of matrix $\mathbf{B}$ with vector $\mathbf{x} = (1, x, x^2, \ldots, x^{n-1})^T$. Then for each i we have to compute $P(i, x)$ in n different points $x = r_1, r_2, \ldots, r_n$. This is a typical task of multipoint polynomial evaluation that for any set of points can be solved by divide-and-conquer method. This leads to complexity of $O(n \log^2 n)$

operations per one row of $\mathbf{B}$ (i.e., per one i) — cf. [1], [5], [6]. When the points are selected to be the roots of unity fast Fourier transform can be used (cf. [1]). This method saves one factor of $\log n$ in the resulting complexity estimate. In a similar way we can compute $\mathbf{Cx}_i$.

Unfortunately, for computing the next product $\mathbf{A}(\mathbf{Bx}_i)$ the advantage of a specific form of vector $\mathbf{Bx}_i$ is lost and this seems to lead again to an algorithm of cubic complexity. Fortunately, a slightly more complicated verification criterion based on the following corollary of Lemma 1 solves the problem:

Corollary 1. *Let* $\mathbf{D}$ *be a real matrix of size* $n \times n$, *let* r_i *for* $i = 1, 2, \ldots, 2n-1$ *be* $2n-1$ *different reals numbers, let* $\mathbf{x}_i = (1, r_i, r_i^2, \ldots, r_i^{n-1})^T$.

Then $\mathbf{D} = \mathbf{0}$ *if and only if for all* $i = 1, 2, \ldots, 2n-1$, $\mathbf{x}_i^T \mathbf{D} \mathbf{x}_i = 0$.

Proof: The left-to-right implication is obvious. For the proof of the reverse direction suppose that under the assumptions of the corollary $\mathbf{D} \neq \mathbf{0}$. Then at least one element of $\mathbf{D}$ would be non-zero. In the vector-matrix-vector product $\mathbf{x}_i^T \mathbf{D} \mathbf{x}_i$ this element will give rise to a polynomial in indeterminate r_i of degree at most $2n-2$. This polynomial has at most $2n-2$ real roots and therefore it cannot turn to zero in $2n-1$ distinct real numbers. This is a contradiction and therefore $\mathbf{D} = \mathbf{0}$. □

Translating back to our context of matrix multiplication the previous corollary implies a strategy for a verification algorithm. For $i = 1, 2, \ldots, 2n-1$, we test whether $(\mathbf{x}_i^T \mathbf{A})(\mathbf{Bx}_i) = \mathbf{x}_i^T(\mathbf{Cx}_i)$. If this test fails for some i then $\mathbf{AB} \neq \mathbf{C}$. Otherwise $\mathbf{AB} = \mathbf{C}$ (cf. Algorithm 2). The advantage of the above mentioned criterion is that for computing $\mathbf{x}_i^T \mathbf{A}$, $\mathbf{Bx}_i$, and $\mathbf{Cx}_i$, respectively, again fast multipoint polynomial evaluation algorithms can be used. This time, however, we must evaluate the respective polynomials in $2n-1$ different points, but this does not change the asymptotic complexity of the evaluation task.

Algorithm 2:
A non-deterministic real matrix multiplication algorithm:

1. Input real matrices $\mathbf{A} \neq \mathbf{0}$ and $\mathbf{B} \neq \mathbf{0}$;
2. Guess matrix $\mathbf{C} \neq \mathbf{0}$;
3. if (for $i = 1, 2, \ldots, 2n-1$ and $\mathbf{x}_i = (1, i, i^2, \ldots, i^{n-1})^T$, $(\mathbf{x}_i^T \mathbf{A})(\mathbf{Bx}_i) = \mathbf{x}_i^T(\mathbf{Cx}_i)$), then output($\mathbf{C}$)

Therefore, all the necessary vector-matrix or matrix-vector products can be computed in time $O(n^2 \log^2 n)$ arithmetic operations, or even in $O(n^2 \log n)$ using fast Fourier transform. Hence the entire verification can be done within the same arithmetic complexity.

Theorem 2. *For a Real RAM there exists a non-deterministic algorithm for computing matrix product of real matrices using* $O(n^2 \log n)$ *arithmetical operations.*

For bounded coefficient arithmetic circuits over the real or complex numbers Raz [22] proved a lower bound of order $\Omega(n^2 \log n)$. This bound does not apply to our case since we are using a different computational model than Raz — namely the model where nondeterministic computations with arbitrary large reals and equality tests are allowed.

4 Non-deterministic Multiplication of Integer Matrices Using Modular Arithmetic

Algorithm 1 from Section 2 has one serious drawback: it computes with very large integers since the largest entry in vector $\mathbf{x}$ is as high as $(2n-1)^{n-1}$. In matrix $\mathbf{D} = \mathbf{AB} - \mathbf{C}$ this entry can subsequently be multiplied by α, where α, defined in Section 2, denotes the bound on the maximal value of coefficient of matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. That is, the bit representation of such an entry requires $O(n \log n + \log \alpha) = O(n \log n)$ bits for a sufficiently large n. Hence, for such n the size of this entry is dominated by the term $O(n \log n)$ rather than by the size of the maximal possible coefficient in matrix $\mathbf{D}$. Fortunately, this situation can be remedied by using modular arithmetic.

However, one must be careful when trying to adjust the basic Algorithm 1 from Section 2 to modular arithmetic. The first problem arises because, as it appears, in finite fields no analogue of Cauchy's bound is known. This is probably due to the fact that finite fields cannot be ordered. A number that is not a root of a polynomial in the ring $\mathbb{Z}$ of integers may become a root of the same polynomial in a finite field $\mathbb{Z}_p$, for some p prime. For instance, in $\mathbb{Z}$ polynomial $x^2 + 1$ has no real roots. The same polynomial in $\mathbb{Z}_5$ has two roots: 2 and 3. Thus, we must use a different criterion than that from Lemma 1 for identifying a zero matrix.

It appears that we can design a criterion similar to that used for the case of real numbers (cf. Corollary 1). This is possible since a polynomial of degree n has at most n roots in any finite field. The criterion from Corollary 1 can be adjusted to the case of finite fields thanks to the following lemma:

Lemma 2. *Let* $\mathbf{D}$ *be an integer matrix of size* $n \times n$ *with* $\max\{|d_{i,j}|\} \leq \delta$. *Let* $\mathbb{Z}$ *be the ring of integers and let* $\mathbb{Z}_p$ *be a finite field of integers* $\mod p$, *for some prime* $p > \max\{\delta, 2n-1\}$. *Let further* r_i *for* $i = 1, 2, \ldots, 2n-1$, $r_i < p$ *be* $2n-1$ *different integers and let* $\mathbf{x}_i = (1, r_i, r_i^2, \ldots, r_i^{n-1})^T$.

Then $\mathbf{D} = \mathbf{0}$ *(in* $\mathbb{Z}$*) if and only if for all* $i = 1, 2, \ldots, 2n-1$, $\mathbf{x}_i^T \mathbf{D} \mathbf{x}_i \equiv 0 \pmod{p}$ *(in* $\mathbb{Z}_p$*).*

Proof: The left-to-right implication is obvious. For the proof of the reverse direction suppose that under the assumptions of the lemma $\mathbf{D} \neq \mathbf{0}$. Thanks to this and the choice of p at least one element $d_{i,j}$ of $\mathbf{D}$ must be non-zero $\mod p$: $d_{i,j} \not\equiv 0 \pmod{p}$. This means that $\mathbf{D} \not\equiv \mathbf{0} \pmod{p}$. In the vector-matrix-vector product $(\mathbf{x}_i^T \mathbf{D} \mathbf{x}_i) \mod p$ this element will give rise to a non-zero polynomial in indeterminate r_i of degree at most $2n-2$. This polynomial has at most $2n-2$ roots in $\mathbb{Z}_p$ and therefore it cannot turn to zero in $2n-1$ distinct numbers r_i

mod $p = r_i$. This, however, implies that it cannot hold $\mathbf{x}_i^T \mathbf{D} \mathbf{x}_i \equiv 0 \pmod{p}$ for all $i = 1, 2, \ldots, 2n-1$. This is a contradiction. □

This Lemma opens the way for designing an algorithm using modular arithmetic for integer matrix multiplication similar to Algorithm 2 from Section 3.

In order to do so we have to choose a field of a suitable characteristic p. We will want to keep p relatively small, with $p \geq \alpha$. While avoiding computations with numbers above p, thanks to the definition of α such a choice of p will preserve the value of entries of matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ when considered in $\mathbb{Z}$. However, one must be careful with this idea since in arithmetic $\pmod{p}$ each element (except zero) has two representations: one positive and one negative: for any a with $-p < a < p$, $a \equiv a - p \pmod{p}$. Therefore, when guessing the values of matrix $\mathbf{C}$ the correct values (with a correct sign) — i.e., those from $\mathbb{Z}$ — must be guessed because these are the values that would eventually be output. This is because, as we shall see, the correctness check is performed in arithmetic $\pmod{p}$ (as suggested by Lemma 2) through which both the positive and the negative representation of the same non-zero number could pass. This could be prevented once we knew what would be the sign of each entry in $\mathbf{C} = \{c_{i,j}\}$. In general, it appears that for arbitrary matrices $\mathbf{A}$ and $\mathbf{B}$ the sign of each $c_{i,j}$ cannot be determined without actually computing it. Nevertheless, for special matrices this can be done. E.g., when all entries of $\mathbf{A}$ and $\mathbf{B}$ are positive, so will be the entries of $\mathbf{C}$. This motivates the following definition.

A matrix $\mathbf{A}$ with all entries non-negative (non-positive) will be called a *non-negative (non-positive) matrix.* A matrix $\mathbf{A}$ that is either non-negative or non-positive will be called an *equisigned* matrix. Observe that thanks to the definition of matrix product in which rows are multiplied by columns the product of two equisigned matrices is again an equisigned matrix.

For any real matrix $\mathbf{A} = \{a_{i,j}\}$, $\mathbf{A}^+$ is a matrix defined as $\mathbf{A}^+ = \{a^+_{i,j}\}$, with $a^+_{i,j} = a_{i,j}$ if $a_{i,j} > 0$ and $a^+_{i,j} = 0$ if $a_{i,j} \leq 0$. Matrix $\mathbf{A}^-$ consisting of non-positive entries of matrix $\mathbf{A}$ is defined similarly. Obviously, $\mathbf{A}^+$ and $\mathbf{A}^-$ are equisigned matrices and any non-equisigned matrix $\mathbf{A}$ can be uniquely written as $\mathbf{A} = \mathbf{A}^+ + \mathbf{A}^-$. The following lemma is nearly obvious:

Lemma 3. *Let* $\mathbf{A}$, $\mathbf{B}$ *and* $\mathbf{C}$ *be three matrices of real numbers of size* $n \times n$. *Then* $\mathbf{AB} = \mathbf{C}$ *if and only if* $\mathbf{A}^+\mathbf{B}^+ + \mathbf{A}^-\mathbf{B}^- = \mathbf{C}^+$ *and* $\mathbf{A}^+\mathbf{B}^- + \mathbf{A}^-\mathbf{B}^+ = \mathbf{C}^-$.

It follows that instead of multiplying two general matrices $\mathbf{A}$ and $\mathbf{B}$ it is enough to add four products of equisigned matrices. Note that by this we have solved the problem of a possible dual representation of entries in matrix $\mathbf{C}$ in modular arithmetic. In an equisigned matrix all elements are either non-negative or non-positive. Therefore, in the sequel, w.l.o.g. we will only consider multiplication of equisigned matrices.

Now, let us return to the problem of computing the product of integer matrices using arithmetic $\pmod{p}$.

The question remains how to find a suitable p efficiently, i.e., in a way that would not deteriorate the (nearly) quadratic algebraic complexity of the verification algorithm. To that end we use the Bertrand's postulate stating that for any

integer $k > 1$, there is a prime number p such that $k < p < 2k$ (cf. [21]). Since we are aiming at a nondeterministic algorithm a suitable prime p will be guessed and subsequently tested for primality. An appropriate test for our purpose is the check devised by Pratt in 1975 [19] that can be nondeterministically realized with the help of $O(\log^2 p)$ modular multiplication on a unit-cost RAM.

The resulting Algorithm 3 mirroring Algorithm 2 is given below. This algorithm is based on Lemma 2 in which we put $\mathbf{D} = \mathbf{AB} - \mathbf{C}$. In order to determine p we need to estimate δ from Lemma 2, i.e., to bound the absolute values of $\mathbf{D}$.

Clearly, the maximal values in $\mathbf{C}$ are bounded by $\beta = n(\max\{|a_{i,j}|, |b_{i,j}|\})^2$ and hence the maximal absolute value in matrix $\mathbf{D}$ is at most $\delta = 2\beta$. Thus, choosing $p > \max\{\delta, 2n-1\}$ will do.

Algorithms 3:
A log-cost efficient non-deterministic multiplication algorithm for equisigned integer matrices:

1. Input two non-zero equisigned matrices $\mathbf{A}$ and $\mathbf{B}$;
2. Guess the equisigned matrix $\mathbf{C} \neq \mathbf{0}$ with the absolute values of its entries bounded by β;
3. Check, whether the signs of elements of $\mathbf{C}$ correspond to the signs of the input equisigned matrices;
4. Compute $\alpha := \max\{2\beta, 2n-1\}$;
5. Guess $p : \alpha < p < 2\alpha$ and using Pratt's algorithm verify that p is a prime;
6. For $i = 1, 2, \ldots, 2n-1$ compute vectors $\mathbf{x}_i = (1, i, i^2 \bmod p, \ldots, i^{n-1} \bmod p)^T$;
7. Using multipoint polynomial evaluation in arithmetic $\bmod p$ deterministically verify whether (for $i = 1, 2, \ldots, 2n-1$, $(\mathbf{x}_i^T \mathbf{A})(\mathbf{B}\mathbf{x}_i) = \mathbf{x}_i^T(\mathbf{C}\mathbf{x}_i)$);
   if YES then Output $\mathbf{C}$.

As in the case of the previous algorithm the number of operation $(\bmod p)$ of this algorithm is $O(n^2 log^2 n)$. Its correctness follows from several facts. After Step 2, there exist many computational paths (one for each guess of $\mathbf{C}$) in the computation, but the verification in Step 7 selects exactly one of them satisfying $(\mathbf{x}_i^T \mathbf{A})(\mathbf{B}\mathbf{x}_i) \equiv \mathbf{x}_i^T(\mathbf{C}\mathbf{x}_i) \pmod p$ for all i. Note that by choosing $r_i = i$ for $i = 1, 2, \ldots, 2n-1$ the values of r_is are all different and therefore by Lemma 2 in which we put $\mathbf{D} = \mathbf{AB} - \mathbf{C}$ the latter verification is equivalent to verifying $\mathbf{AB} = \mathbf{C}$. Because the equisigned matrices were considered in Steps 1 and 2 matrix $\mathbf{C}$ verified and produced in Step 7 has the correct values (and signs). Obviously, since all computation is done $(\bmod p)$ the size of all RAM registers is bounded by $O(\log p) = O(\log \alpha)$.

Theorem 3. *For a unit-cost RAM there exists a non-deterministic integer matrix multiplication algorithm using $O(n^2 \log^2 n)$ modular arithmetical operations with operands proportional to the size of the largest entry in all involved matrices.*

Consider now the complexity of the previous computation under the logarithmic cost. Let $M(q)$ be the complexity of multiplying q-bit numbers on a log-cost

RAM. The best known upper bound $M(q) = \log q \; 2^{\Theta(\log^* q)}$ is due to Fürer [12] and De, Saha, Kurur and Saptharishi [2]. When measured in bit operations, $O(\log^3 p \; 2^{\Theta(\log^* q)})$ steps suffice to check a proof of p's primality in Step 4.

Since $p : \alpha < p < 2\alpha$ is the largest number occurring in the computation, the bit complexity of Step 6 and 7 is $O(n^2 \log^2 nM(\log p))$. For sufficiently large n the latter value dominates the complexity of the entire algorithm.

Similarly as in the case of computing with real numbers we can select the interpolation points used in multipoint polynomial evaluation (in Step 7) carefully in order to exploit a variant of fast Fourier transform also in the case of a finite field. Such transform is called number theoretic transform (NTT). Here an additional problem arises — namely the fact that not every finite field possesses an n-th root of unity. There exists numerous literature devoted to the optimal selection of parameters n (number of points), p and m (the size of the base field which must be of form p^m for p prime and m a natural number) of NTT (cf. [20] or [1]). The basic results here are that a logarithmic factor in the number of arithmetic operations can be saved, indeed, when compared with the general case of multipoint polynomial evaluation over arbitrary set of points. We will not follow this line any further because it is not directly related to the complexity of matrix products.

Theorem 4. *For a log-cost RAM there exists a non-deterministic integer matrix multiplication algorithm using $O(n^2 \log nM(\log p))$ bit operations.*

Obviously, this algorithm can be implemented on the standard nondeterministic model of Turing machines with a similar efficiency, too. This is to be compared to the bit complexity of the best known deterministic algorithms which is of order $O(n^\omega M(\log p)$, for some $\omega : 2 < \omega < 3$. For any $\omega > 2$ the latter expression is asymptotically worse than our estimation from the previous theorem.

5 Conclusion

We have presented two nondeterministic algorithms for matrix multiplication. Both algorithms make use of a derandomized version of Freivalds' algorithm for verification of matrix products. The first algorithm deals with real numbers and on a Real RAM is of time complexity $O(n^2 \log n)$. The second one computes with integers and on a unit-cost RAM is also of complexity $O(n^2 \log n)$. Moreover, the latter algorithm computes with numbers whose size is bounded by the maximal number occurring within the matrices. The respective complexity bounds hold when variants of fast Fourier transform are used for the multipoint polynomial evaluation used by both algorithms. With respect to their complexity both algorithms are the fastest known sequential algorithms for computing the product of integer and real matrices, respectively.

Of course, due to their nondeterministic nature our algorithms are more of a theoretical than of a practical value. Nevertheless, both of them can be used for deterministic matrix multiplication verification.

But there are more lessons to be taken from our results. First, for the case of integer matrices, they show that any lower bound for matrix multiplication which would be greater than our quadratic upper bound must avoid proof techniques that work both for deterministic and nondeterministic computations. For integer matrices such a lower bound must probably be dependent on the size of matrix entries since Algorithm 1 is of quadratic complexity (i.e., it is optimal). Second, our results pose interesting questions that we cannot answer at present: is there an intrinsic difference between the complexity of integer and real matrices multiplication (cf. Algorithm 1 vs. Algorithm 2)? Does matrix multiplication belong among problems that separate determinism from nondeterminism, i.e., does the use of nondeterminism in matrix multiplication help? Last but not least, it is interesting to observe that in order to prove our results we have used, in addition to nondeterminism, more powerful tools from computational algebra than those usually used in the design of fast "classical" matrix multiplication algorithms: comparisons, theory of Vandermonde matrices, properties of polynomial roots, primality certificate, modular arithmetic, and fast Fourier transform. Could it be the case that the use of such techniques is the key to the design of fast deterministic matrix multiplication algorithms?

We believe that our results have brought a further insight into the nature of matrix multiplication. They strengthen the hope that perhaps deterministic algorithms for matrix product of a similar complexity will be found.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms, p. 470. Addison-Wesley (1974)
2. De, A., Kurur, P.P., Saha, C., Saptharishi, R.: Fast Integer Multiplication Using Modular Arithmetic. SIAM J. Comput. 42(2), 685–699 (2013)
3. Ar, S., Blum, M., Codenotti, B., Gemmel, P.: Checking approximate computations over the reals. In: Proc. 25th ACM STOC, pp. 786–795 (1993)
4. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation, p. 452. Spinger (1998)
5. Borodin, A., Moenck, R.: Fast modular transform. Journal of Computer and System Sciences 8, 366–386 (1974)
6. Bostan, A., Schost, E.: Polynomial evaluation and interpolation on special sets of points. Journal of Complexity 21(4), 420–446 (2005)
7. Brassard, G., Bratley, P.: Fundamentals of Algorithmics, 524 pages. Prentice Hall, Englewood Cliffs (1996)
8. Buhrman, H., Špalek, R.: Quantum verification of matrix products. In: Proceedings of SODA, pp. 880–889 (2006)
9. Cauchy, A.-L.: Mémoire sur la résolution numérique des équations et sur la théorie de l'élimination. Exercices de mathématiques, 40e et 41e livraisons, dans OC (2), IX, 87–161 (1829)
10. Freivalds, R.: Probabilistic machines can use less running time. In: Information Processing 1977, Proceedings of the IFIP Congress 1977, pp. 839–842 (1977)
11. Freivalds, R.: Fast Probabilistic Algorithms. In: Bečvář, J. (ed.) Mathematical Foundations of Computer Science 1979. LNCS, vol. 79, pp. 57–69. Springer, Heidelberg (1979)

12. Fürer, M.: Faster Integer Multiplication. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13 (2007)
13. Householder, A.: The Numerical Treatment of a Single Nonlinear Equation. McGraw-Hill, New York (1970)
14. Kimbrel, T., Sinha, R.K.: A probabilistic algorithm for verifying matrix products using $O(n^2)$ time and $\log_2 n + O(1)$ random bits. Inf. Process. Lett. 45(2), 107–110 (1993)
15. Korec, I., Wiedermann, J.: Deterministic verification of integer matrix multiplication in quadratic time. In: Geffert, V., et al. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 375–382. Springer, Heidelberg (2014)
16. Le Gall, F., Nishimura, H.: Quantum Algorithms for Matrix Products over Semirings. In: Ravi, R., Gørtz, I.L. (eds.) SWAT 2014. LNCS, vol. 8503, pp. 331–343. Springer, Heidelberg (2014)
17. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, New York (1995)
18. Naor, J., Naor, M.: Small-bias probability spaces: efficient constructions and applications. SIAM J. Comput. 22(4), 838–856 (1993)
19. Pratt, V.: Every prime has a succinct certificate. SIAM Journal on Computing 4, 214–220 (1975)
20. Preparata, F.P., Sarwate, D.V.: Computational complexity of Fourier transform over finite fields. Math. of Comp. 31(139), 740–751 (1977)
21. Ramanujan, S.: A Proof of Bertrand's Postulate. J. Indian Math. Soc. 11, 181–182 (1919)
22. Raz, R.: On the complexity of matrix product. In: Proc. of the 34th Annual ACM Symposium on Theory of Computing. ACM Press (2002)
23. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. 13, 354–356 (1969)
24. Vassilevska Williams, V.: Multiplying matrices faster than Coppersmith-Winograd. In: ACM STOC, pp. 887–898 (2012)