University of Padova - DEI
Course “Introduction to Natural Language Processing”, Academic Year 2002-2003

Term paper
Fast Matrix Multiplication

PhD Student: Carlo Fantozzi, XVI ciclo

1 Introduction and Contents

This term paper illustrates some results concerning the fast multiplication of $n \times n$ matrices: we use the adjective “fast” as a synonym of “in time asymptotically lower than n^3”. This subject is relevant to natural language processing since, in 1975, Valiant showed [27] that Boolean matrix multiplication can be used to parse context-free grammars (or CFGs, for short): as a consequence, a fast boolean matrix multiplication algorithm yields a fast CFG parsing algorithm. Indeed, Valiant’s algorithm parses a string of length n in time proportional to $T_{\mathrm{BOOL}}(n)$, i.e. the time required to multiply two $n \times n$ boolean matrices. Although impractical because of its high constants, Valiant’s algorithm is the asymptotically fastest CFG parsing solution known to date. A simpler (hence nearer to practicality) version of Valiant’s algorithm has been devised by Rytter [20].

One might hope to find a fast, practical parsing scheme which do not rely on matrix multiplication: however, some results seem to suggest that this is a hard quest. Satta has demonstrated [21] that *tree-adjoining grammar* (TAG) parsing can be reduced to boolean matrix multiplication. Subsequently, Lee has proved [18] that any CFG parser running in time $O\left(gn^{3-\epsilon}\right)$, with g the size of the context-free grammar, can be converted into an $O\left(m^{3-\epsilon/3}\right)$ boolean matrix multiplication algorithm*; the constants involved in the translation process are small. Since canonical parsing schemes exhibit a linear dependence on g, it can be reasonably stated that fast, practical CFG parsing algorithms can be translated into fast matrix multiplication algorithms. These reductions of matrix multiplication to parsing, together with the fact that extensive research has failed to discover a practical fast algorithm for matrix multiplication, provide evidence of the difficulty of finding a practical, sub-cubic parsing scheme.

The contents of this term paper are as follows.

- In Section 2 we first survey available solutions for multiplying arbitrary (i.e., non-boolean) matrices. In particular, we focus our attention on the well-known $O\left(n^{\log 7}\right)$ algorithm of Strassen, which is regarded as the only sub-cubic solution of practical relevance for the problem under consideration. We give an upper bound on the constant hidden by the big-oh notation, and we also hint at some design issues that must be faced in real-world implementations of the algorithm.

- In Section 3 we introduce the problem of boolean matrix multiplication, and we point out the differences with the problem of multiplying arbitrary matrices. We also sketch a way of computing the product of boolean matrices by making use of an algorithm for arbitrary matrices.

- In Section 4 we show how, given an algorithm $\mathcal{A}$ for the multiplication of $n \times n$ matrices defined over a *ring*, a randomized algorithm can be devised which multiplies boolean matrices by making use of $\mathcal{A}$ and with a logarithmic slowdown with respect to $\mathcal{A}$. This result is basically a solution of Exercise 31-1 in [10]; we remark that the section on boolean matrix multiplication has been completely removed in the second edition of the book.

*Context-free languages are a proper subset of the languages generated by TAGs; however, Satta’s reduction explicitly relies on TAG properties that allow non-context-free languages to be generated, hence it cannot be applied to CFG parsing.

- In Section 5 we mainly focus on a problem which is related to matrix multiplication, namely *matrix multiplication checking*. A matrix multiplication checker takes as input three square matrices A, B, and C and reveals whether or not C is the product of A and B. The relevance of this problem to natural language processing is due to the fact that Harrison and Havel provide [14, 15] a reduction of boolean matrix checking to context-free recognition. In the same section, we also give some bibliographical notes on two further problems: *matrix multiplication witnesses* and *transitive closure*.

2 Fast Matrix Multiplication

The standard algorithm for multiplying two $n \times n$ matrices requires exactly n^3 multiplications and $n^2(n-1)$ additions, for a total of $2n^3 - n^2$ scalar arithmetic operations. The first and most famous fast algorithm for matrix multiplication (or MM, for short) is certainly the one due to Strassen [23], which requires only $O\left(n^{\log 7}\right) = O\left(n^{2.808}\right)$ operations. Strassen's algorithm is based on a nontrivial way of multiplying 2×2 matrices using 7 (instead of 8) multiplications at the price of increasing the number of additions from 4 to 18; some of the additions are actually subtractions, so the algorithm works for matrices defined over a *ring*[†]; see Section 3 for a formal definition of what a ring is.

If $n \geq 2$ is even, the algorithm of Strassen can be easily presented in the following, recursive fashion. The product $C = AB$ can be decomposed as

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}.$$

The four submatrices contained in C can be computed by first calculating the 7 auxiliary submatrices

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\ P_2 &= (A_{21} + A_{22})B_{11}, \\ P_3 &= A_{11}(B_{12} - B_{22}), \\ P_4 &= A_{22}(B_{21} - B_{11}), \\ P_5 &= (A_{11} + A_{12})B_{22}, \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}), \end{aligned}$$

and then by appropriately combining the auxiliary submatrices to obtain the final result:

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} P_1 + P_4 - P_5 + P_7 & P_3 + P_5 \\ P_2 + P_4 & P_1 - P_2 + P_3 + P_6 \end{bmatrix}.$$

Due to the recursive nature of the algorithm, its running time $T_{\mathrm{STRAS}}(n)$, measured in terms of arithmetic operations, is given by a recurrence equation. Under the hypothesis that n is a power of 2, such equation is simply

$$T_{\mathrm{STRAS}}(n) = \begin{cases} 7T_{\mathrm{STRAS}}(n/2) + 18(n/2)^2 & \text{for } n > 2, \\ 25 & \text{for } n = 2, \end{cases}$$

Our calculations show that $T_{\mathrm{STRAS}}(n) \leq 7n^{\log 7}$. Quoting Strassen himself, [16] gives the following, tighter bound:

$$T_{\mathrm{STRAS}}(n) \leq 4.7n^{\log 7}.$$

These bounds show that the constant hidden by the big-oh notation is quite low.

[†] For the sake of completeness, we add that Strassen's algorithm does not require multiplication to be commutative.

The results of Strassen's work started a competition among researchers to find faster and faster solutions for matrix multiplication. An historical account of the progresses made on the subject can be found in [24]; [18] has a graph plotting the exponent ω of the best available matrix multiplication algorithm as it decreases from 3 (before 1969) to 2.376 (from 1987 [8] to now). The fastest MM algorithm currently known, which exhibits time complexity $O\left(n^{2.376}\right)$, is due to Coppersmith and Winograd [9]. It must be remarked that the constants involved in algorithms improving on Strassen's result are very large: as is often the case in Computer Science, the asymptotically faster an algorithm is, the larger the constants. As a practical consequence, these fast MM algorithms are only of theoretical interest.

As for Strassen's algorithm itself, its practicality needs some consideration. The algorithm can be applied in a straightforward fashion if n is a power of 2, but a real-world implementation must efficiently handle matrices of arbitrary size: in this scenario, nontrivial design issues arise. In dealing with such issues, the number of arithmetic operations performed does not give a good predictor of actual performance [16]: as a consequence, optimal design choices depend on the platform for which the algorithm is being tuned. Among design choices, the most significant is probably the one of the so-called "cutoff point", i.e. the matrix size n at which the overhead due to Strassen's algorithm makes it convenient to truncate the recursion and switch to a trivial, $O\left(n^3\right)$ solution. Empirical studies [5, 16, 25] show that the cutoff point is between 64 and 128 depending on the platform, hence we can say that Strassen's algorithm offers a clear performance advantage if $n > 100$.

For the sake of completeness, we note that a bibliographical search for randomized MM algorithms yielded no results.

3 Boolean Matrix Multiplication

In this section we analyze the problem of multiplying two $n \times n$ boolean matrices A and B, so as to determine $C = AB$. We call this the *boolean matrix multiplication* problem, or BMM for short. We recall that the elements of boolean matrices belong to the finite set $\{0, 1\}$; the product AB is performed as for matrices of real numbers, provided the familiar sum and product operators are replaced by the OR operator and the AND operator, respectively. As a consequence, the generic element c_{ij} of C, $1 \leq i, j \leq n$, is given by

$$c_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj}).$$

The AND and OR operators are denoted with the usual symbols $\wedge$ and $\vee$. The following facts can be proved.

- The 5-tuple $Q \triangleq \{\{0, 1\}, \vee, \wedge, 0, 1\}$ is a *quasiring*.
- Matrices over a *ring* form a ring.
- Matrices over a quasiring form a quasiring, hence boolean matrices are defined over a quasiring.

Since there is a bit of confusion in the literature about the terms "quasiring" and "semiring", we hereby include a full definition of a quasiring. A 5-tuple $(S, \oplus, \odot, \bar{0}, \bar{1})$ is a quasiring if it satisfies the following properties.

1. If $a \in S$ and $b \in S$, then $a \oplus b \in S$. $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ for all $a, b, c \in S$. $a \oplus \bar{0} = \bar{0} \oplus a = a$ for all $a \in S$. (We say that $(S, \oplus, \bar{0})$ is a *monoid*.)
2. $a \odot \bar{0} = \bar{0} \odot a = \bar{0}$ for all $a \in S$. (We say that $\bar{0}$ is an *annihilator*.)
3. $a \oplus b = b \oplus a$ for all $a, b \in S$. (We say that the $\oplus$ operator is *commutative*.)

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Table 1: Truth tables of the AND, OR and XOR operators.

4. $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$ for all $a, b, c \in S$. (We say that the $\odot$ operator *distributes* over $\oplus$.)

A quasiring $(S, \oplus, \odot, \bar{0}, \bar{1})$ is a ring if every element $a \in S$ admits an *additive inverse*, i.e. an element $\bar{a} \in S$ such that $a \oplus \bar{a} = \bar{a} \oplus a = \bar{0}$. We explicitly note that 1 does not have an additive inverse in Q, therefore Q is a quasiring but not a ring.

The asymptotically fastest way known to perform BMM is to rely on algorithms for multiplying arbitrary, dense matrices. According to [18], the fastest algorithms that do not follow this idea are the so-called "four Russians" algorithm [4], with a worst-case running time of $O\left(n^3/\log n\right)$, and its variant due to Rytter [19], which uses compression to reduce the running time to $O\left(n^3/\log^2 n\right)$. As it can be seen, the asymptotic performance of these algorithms is really poor if compared to that of the fast MM strategies we cited in Section 2: indeed, these algorithms are asymptotically slower than Strassen's, which is also considered practical.

It must be noticed that many fast matrix multiplication algorithms work for matrices defined over the set of natural or real numbers, which silently implies that the matrices are defined over a ring or a *field*. However, these algorithms can be employed to solve the BMM problem by making use of the following, straightforward procedure.

Let $\mathcal{A}$ be an algorithm for multiplying matrices over a ring R
By making use of $\mathcal{A}$, calculate the product C' of A and B over R
for $i \leftarrow 1$ to n
  for $j \leftarrow 1$ to n
    if $c'_{ij} = 0$ **then** $c_{ij} \leftarrow 0$ **else** $c_{ij} \leftarrow 1$

An often neglected point in this procedure is the analysis of the space which is required to store C' and the auxiliary data structures possibly needed by $\mathcal{A}$. It is easy to see that $0 \leq c'_{ij} \leq n$, hence $\log n$ bits suffice to store a single element of C': since we are dealing with $n \times n$ matrices, we can reasonably suppose that a memory word has at least $\log n$ bits. However, nothing can be said *a priori* concerning the intermediate results generated by $\mathcal{A}$: large intermediate values may require many machine words to be stored or raise numerical stability issues, thus hampering the practicality of the whole procedure. Modular arithmetic can be used to keep intermediate results within given bounds: for example, in [12] it is shown how Strassen's algorithm can be adapted to BMM by performing all computations modulo $n + 1$.

4 Shamir's Boolean Matrix Multiplication Algorithm

Let us denote with $\oplus$ the eXclusive OR (XOR) operator. The 5-tuple

$$R \triangleq \{\{0, 1\}, \oplus, \wedge, 0, 1\}$$

is a ring: in fact,

a	b	c	$a \oplus (b \oplus c)$	$(a \oplus b) \oplus c$
0	0	0	$0 \oplus 0 = 0$	$0 \oplus 0 = 0$
0	0	1	$0 \oplus 1 = 1$	$0 \oplus 1 = 1$
0	1	0	$0 \oplus 1 = 1$	$1 \oplus 0 = 1$
0	1	1	$0 \oplus 0 = 0$	$1 \oplus 1 = 0$
1	0	0	$1 \oplus 0 = 1$	$1 \oplus 0 = 1$
1	0	1	$1 \oplus 1 = 0$	$1 \oplus 1 = 0$
1	1	0	$1 \oplus 1 = 0$	$0 \oplus 0 = 0$
1	1	1	$1 \oplus 0 = 1$	$0 \oplus 1 = 1$

Table 2: The XOR operator is associative.

1. Table 1 trivially demonstrates that R is closed under $\oplus$ and that 0 is an identity for $\oplus$. Furthermore, $\oplus$ is associative, as proved by Table 2. Consequently, $(R, \oplus, 0)$ is a monoid.
2. 0 is an annihilator with respect to the AND operation (see Table 1).
3. The $\oplus$ operator is commutative: the proof is, once again, obtained by inspecting Table 1.
4. The $\wedge$ operator distributes over $\oplus$, that is, $a \wedge (b \oplus c) = a \wedge b \oplus a \wedge c$. Since the set of numbers we are considering contains only two elements, the easiest way to prove this fact is by direct inspection of the 8 possible combinations of a, b and c: the results of such inspection are contained in Table 3.

a	b	c	$a \wedge (b \oplus c)$	$a \wedge b \oplus a \wedge c$
0	0	0	$0 \wedge 0 = 0$	$0 \oplus 0 = 0$
0	0	1	$0 \wedge 1 = 0$	$0 \oplus 0 = 0$
0	1	0	$0 \wedge 1 = 0$	$0 \oplus 0 = 0$
0	1	1	$0 \wedge 0 = 0$	$0 \oplus 0 = 0$
1	0	0	$1 \wedge 0 = 0$	$0 \oplus 0 = 0$
1	0	1	$1 \wedge 1 = 1$	$0 \oplus 1 = 1$
1	1	0	$1 \wedge 1 = 1$	$1 \oplus 0 = 1$
1	1	1	$1 \wedge 0 = 0$	$1 \oplus 1 = 0$

Table 3: The AND operator distributes over the XOR operator.

5. Every element $a \in R$ has an *additive inverse*, that is, an element $\bar{a} \in R$ such that $a \oplus \bar{a} = \bar{a} \oplus a = 0$: to be precise, the inverse of 0 is 0 and the inverse of 1 is 1. The proof of this statement is given, once again, by Table 1.

By what we said in Section 3, we can immediately conclude that matrices defined over R form a ring, therefore they can be multiplied by making use of any algorithm $\mathcal{A}$ for matrices defined over a ring. We will now show how to solve the problem of multiplying two $n \times n$ boolean matrices A and B by repeatedly applying $\mathcal{A}$.

Let $a_{ij} \in \{0, 1\}$ be the element in row i and column j of matrix A; we will indicate the corresponding elements of B and $C = AB$ as b_{ij} and c_{ij}, respectively. The first step of our strategy prescribes to generate A' from A using the randomized procedure that follows.

- If $a_{ij} = 0$, then $a'_{ij} = 0$.

- If $a_{ij} = 1$, then let $a'_{ij} = 0$ with probability $1/2$ and let $a'_{ij} = 1$ with probability $1/2$. The random choices for distinct matrix entries are independent.

Once A' has been built, it is possible to calculate $C' \triangleq A'B$ in the ring R by making use of algorithm $\mathcal{A}$. There is a strict correlation between C' and the matrix product C we are trying to calculate: as a matter of fact, the following lemma can be proved.

Lemma 4.1 *If $c_{ij} = 0$, then $c'_{ij} = 0$. If $c_{ij} = 1$, then $c'_{ij} = 1$ with probability no smaller than $1/2$.*

Proof Element c_{ij} of C can be expressed as

$$c_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj}) = (a_{i1} \wedge b_{1j}) \vee (a_{i2} \wedge b_{2j}) \vee \ldots \vee (a_{in} \wedge b_{nj}), \tag{1}$$

while element c'_{ij} of C' can be calculated as follows:

$$c'_{ij} = \bigoplus_{k=1}^{n} (a'_{ik} \wedge b_{kj}) = (a'_{i1} \wedge b_{1j}) \oplus (a'_{i2} \wedge b_{2j}) \oplus \ldots \oplus (a'_{in} \wedge b_{nj}). \tag{2}$$

If $c_{ij} = 0$, then all the terms in parentheses that appear in Equation (1) are equal to zero: we will now show that, in this case, all the terms in Equation (2) are zero as well, thus proving that $c'_{ij} = 0$. Consider the generic term $(a_{ik} \wedge b_{kj})$, with $1 \leq k \leq n$: by the truth table of the AND operator, this term evaluates to 0 if at least one of a_{ik} and b_{kj} is zero. If $b_{kj} = 0$, we can immediately conclude that $a'_{ik} \wedge b_{kj} = 0$ in Equation (2). Instead, if $b_{kj} \neq 0$ then it must necessarily be $a_{ik} = 0$: in this case, the procedure that was used to generate A' ensures that $a'_{ik} = 0$ with probability 1, hence $a'_{ik} \wedge b_{kj} = 0$.

If $c_{ij} = 1$, then one or more of the terms which appear in Equation (1) are equal to one, and all the others are obviously equal to zero. By the proof for the case $c_{ij} = 0$, we can straightforwardly conclude that if a term evaluates to zero in Equation (1), then the corresponding term in Equation (2) is also zero: as a consequence, we will concentrate on the values of k such that $a_{ik} \wedge b_{kj} = 1$. For any such value, it must necessarily be $a_{ik} = b_{kj} = 1$, thus $a'_{ik} = 1$ with probability $1/2$: by the truth table of the XOR operator, the probability of having $a'_{ik} \wedge b_{kj} = 0$ is therefore $1/2$. To sum things up, we can say that the probability of having $a'_{ik} \wedge b_{kj} = 0$ is 1 if $a_{ik} \wedge b_{kj} = 0$, and it is $1/2$ otherwise. Now, the probability of c'_{ij} being 1 is at least

$$1 - P[a'_{ik} \wedge b_{kj} = 0 \ \forall k] - P[a'_{ik} \wedge b_{kj} = 1 \ \forall k],$$

which can be immediately rewritten as

$$1 - \prod_{k=1}^{n} P[a'_{ik} \wedge b_{kj} = 0] - \prod_{k=1}^{n} (1 - P[a'_{ik} \wedge b_{kj} = 0])$$

because the values assumed by the n terms are not correlated. Unfortunately, to give an upper bound on $P[c'_{ij} = 1]$ it is necessary to distinguish between two cases. If all the terms in Equation (1) evaluate to 1, then

$$P[a'_{ik} \wedge b_{kj} = 0 \ \forall k] = P[a'_{ik} \wedge b_{kj} = 1 \ \forall k] = 1/2^n$$

and consequently $P[c'_{ij} = 1] \geq 1 - 1/2^{n-1} \geq 1/2$. If at least one term in Equation (1) evaluates to 0, then $P[a'_{ik} \wedge b_{kj} = 1 \ \forall k] = 0$ and

$$P[c'_{ij} = 1] \geq 1 - \prod_{k=1}^{n} P[a'_{ik} \wedge b_{kj} = 0] = 1 - 1/2^m,$$

where $m \geq 1$ is the number of terms that evaluate to 1 in Equation (1). Once again, we can therefore conclude that $P[c'_{ij} = 1] \geq 1/2$. □

The lemma shows that matrix C' gives useful information on C: in particular, if $c'_{ij} = 1$ then we can immediately conclude $c_{ij} = 1$, since, by Lemma 4.1, a value of 0 in c_{ij} always yields $c'_{ij} = 0$. The interpretation of C' is more complicated if $c'_{ij} = 0$. This event, in fact, can be explained in two different ways:

1. $c_{ij} = 0$, which necessarily gives $c'_{ij} = 0$;
2. $c_{ij} = 1$, which, by Lemma 4.1, gives $c'_{ij} = 0$ with probability smaller than $1/2$.

In other words, if $c'_{ij} = 0$ there is a certain probability that $c'_{ij} \neq c_{ij}$; however, a technique can be devised to reduce this probability to arbitrarily small values through the repeated evaluation of $A'B$ with ℓ different instances of A', named $A'_1, \ldots, A'_\ell$. This technique gives birth to Shamir's algorithm, whose pseudo-code is given in Figure 1. The SHAMIR-MULT algorithm calculates the desired product

```
SHAMIR-MULT(A, B)
    C ← 0
    for k ← 1 to ℓ
        Generate A'_k
        Compute C'_k = A'_k B by making use of algorithm 𝒜
        Let c'^(k)_ij be the element which occupies row i and column j of C'_k
        for i ← 1 to n
            for j ← 1 to n
                if c'^(k)_ij = 1 then c_ij ← 1
    return C
```

Figure 1: Shamir's boolean matrix multiplication algorithm.

C of boolean matrices A and B; by what we said above, C may contain errors because no C'_k gives completely reliable information on C. We will now analyze the probability of making an error, and we will show that such probability decreases with ℓ. Consider the generic element c_{ij} of C: it is easy to see that the algorithm makes a mistake only if the true value of c_{ij} is 1, while $c'^{(k)}_{ij}$ never takes such value for every $1 \leq k \leq \ell$. The probability that this happens is

$$P[c_{ij} \text{ is assigned a wrong value}] = P[c'^{(k)}_{ij} = 0 \ \forall k] = \prod_{k=1}^{\ell} P[c'^{(k)}_{ij} = 0] < 1/2^{\ell}. \tag{3}$$

The above result has been obtained by remembering that the matrices C'_k are generated in independent events, and by plugging in the probabilities stated in Lemma 4.1. Equation (3) gives the probability of having an error at a specific position inside matrix C: the probability of having at least one error in C is upper bounded by

$$\sum_{i,j=1}^{n} P[c_{ij} \text{ is assigned a wrong value}] < n^2/2^{\ell},$$

therefore the probability of C being correct is at least

$$1 - n^2/2^{\ell}. \tag{4}$$

We are finally ready to state the main result of this section.

Theorem 4.2 *Let $\mathcal{A}$ be an algorithm that computes the product of $n \times n$ dense matrices over a ring in time $T(n)$. For any constant $k > 0$, there exists an $O(T(n) \log n)$ randomized algorithm that computes the product of two $n \times n$ boolean matrices with probability at least $1 - 1/n^k$.*

Proof Consider the boolean matrix multiplication algorithm of Figure 1, and let $\ell \stackrel{\Delta}{=} \lceil (2+k)\log n \rceil$. The generation of A'_k for a fixed k, and the subsequent inspection of C'_k, takes time $\Theta\left(n^2\right)$, while the calculation of C'_k clearly requires time $T(n)$. Since $T(n) = \Omega\left(n^2\right)$ and there are ℓ matrices to generate and check, the running time of the algorithm in Figure 1 is

$$O\left(\left(n^2 + T(n)\right)\ell\right) = O\left(T(n)(2+k)\log n\right) = O\left(T(n)\log n\right).$$

The probability of the solution being correct is given by Equation (4) and is

$$1 - \frac{n^2}{2^\ell} \geq 1 - \frac{n^2}{n^{2+k}} = 1 - \frac{1}{n^k}.$$

□

For example, if $\mathcal{A}$ is the well-known algorithm of Strassen, then $T(n) = O\left(n^{\log_7}\right)$ and SHAMIR-MULT gives an $O\left(n^{\log_7}\log n\right)$ BMM algorithm.

5 Matrix Multiplication Checking and More

Up to now we have examined solutions to the problem of determining the product $C = AB$ of square matrices A and B. In this section, we will shift our attention to the problem of *matrix multiplication checking*: A, B and C are given, and our aim is to verify whether $C = AB$. A naive solution to this problem calculates $C' = AB$, then compares C' with C; the speed and practicality of this approach entirely depend on the matrix multiplication algorithm which is adopted. To the best of our knowledge, this is the best deterministic solution available to date.

Randomized solutions exist that can do asymptotically better. In 1979, Freivalds proposed [13] a randomized algorithm that performs $n \times n$ matrix multiplication checking in time $O\left(n^2\right)$. In this section, we describe and analyze the following, generalized version of Freivalds' algorithm [7].

MULT-CHECK(A, B, C)
1. Randomly choose a vector v from a finite set S of test vectors.
2. Compute ABv.
3. Compute Cv.
4. **if** $ABv = Cv$ **then return** "C is the product of A and B"
5. **else return** "C is not the product of A and B".

Since the matrix-vector product is associative, Step 2 can be performed by calculating $A(Bv)$. As a consequence, the above algorithm requires at most $3n^2$ multiplications and $3n(n-1)$ additions.

As it can be easily seen, the solution returned by Freivalds' algorithm may be incorrect. To be precise, if $AB = C$ then the algorithm always returns the right answer; however, if $AB \neq C$ then the algorithm returns a wrong answer with a certain probability. We will now give an upper bound on the probability of error based on the *rank* of the set S of test vectors; recall that the rank of S is the cardinality of the largest subset $S' \subseteq S$ such that any linear combination of vectors in S' gives a nonzero result. Moreover, recall that the rank of a matrix M is the rank of the set generated by the columns of M. Finally, in our analysis we will make use of the following

Fact 5.1 *Given an $n \times m$ matrix M with entries from an integral domain, $\mathrm{rank}(M) < n$ if and only if there is a nonzero vector v such that $v^\top M = 0$.*

The following theorem proves[‡] that the error rate of Freivalds' algorithm depends on $\mathrm{rank}(S)$.

[‡]Note that the proof, found in [7], needs modifications to work with boolean matrices.

Theorem 5.2 *Let S be the set of test vectors used in Freivalds' algorithm. If every set of k distinct vectors in S has rank n, then the probability of error of the algorithm is at most $(k-1)/|S|$.*

Proof Consider the matrix $M \triangleq AB - C$. It is straightforward to see that Freivalds' algorithm returns an incorrect answer only if $M \neq 0$ and $Mv = 0$, therefore the probability of error p of the algorithm is

$$p = \frac{|\{v \in S \mid Mv = 0\}|}{|S|}.$$

To prove that the theorem holds, we will now show that there are no more than $k-1$ vectors in S whose product with M is 0. Suppose this assertion is not true: it is then possible to find k vectors $v_1, \ldots, v_k$ such that $Mv_i = 0$ for $1 \leq i \leq k$. Now, consider the matrix

$$V \triangleq \begin{bmatrix} v_1 & v_2 & \cdots & v_k \end{bmatrix}.$$

Since $MV = 0$ and $M \neq 0$, there exists a nonzero row m of M such that $mV = 0$. Therefore, by Fact 5.1 we can conclude that $\mathrm{rank}(V) < n$, which contradicts the hypothesis that every set of k vectors taken from S has rank n. □

In the original version of Freivalds' algorithm, S is the set of vectors defined over $\{0,1\}$: in other words, $v \in S$ if and only if $v \in \{0,1\}^n$. For this choice of S, Theorem 5.2 immediately leads to the following

Corollary 5.3 *For any constant $k > 0$, there exists an $O\left(n^2\right)$ randomized algorithm that verifies the product of two $n \times n$ matrices with probability at least $1 - 1/2^k$.*

Proof (sketch) Let $S = \{v \mid v \in \{0,1\}^n\}$ be the set of test vectors as defined in the original formulation of Freivalds' algorithm: any subset $S' \subset S$ with $|S'| = 2^{n-1} + 1$ has rank n (there are simply too many vectors in S' to be contained in a space of dimension $n-1$), therefore we can apply Theorem 5.2 with $k = 2^{n-1} + 1$ and $|S| = 2^n$, and conclude that the probability of error is at most $1/2$. The error rate stated in the corollary is obtained by iterating Freivalds' algorithm k times; the running time is trivially obtained by observing that each iteration requires $O\left(n^2\right)$ arithmetic operations and comparisons. □

Since 1979, the researchers have focused on reducing the cardinality of set S, therefore reducing the number of random bits required by the algorithm. As we have already said, the original formulation due to Freivalds uses a set of test vectors whose cardinality is 2^n, thus requiring n random bits to choose an element of S. Alon et al. [1] give a test vector space of size $O\left(n^2\right)$, which can be used to implement Freivalds' algorithm with only $2\log n + O(1)$ random bits. The number of random bits is further reduced to $\log n + O(1)$ in [26]; however, this construction is limited to matrices defined over $\mathrm{GF}(2)$, that is, over the finite field with arithmetic modulo 2. Finally, in [17] the algorithm of Freivalds is further modified so that exactly $\lceil \log n \rceil + 1$ random bits are enough to choose an element from the set of test vectors.

As a final remark, we just cite two more problems which are related to boolean matrix multiplication and, consequently, to natural language processing. The first problem is the computation of *witnesses* for BMM: given two $n \times n$ boolean matrices, a witness for an element $c_{ij} \in C = AB$ is an integer k such that $a_{ik} = b_{kj} = 1$. Solving the problem of witnesses for given matrices A and B means finding a witness (if one exists) for all pairs (i,j), $1 \leq i,j \leq n$. Both randomized and deterministic solutions exist for this problem, which is at least as difficult as BMM (it is straightforward to build a reduction): see e.g. [2, 3, 22] for details.

The last problem we are willing to cite goes under the name of *transitive closure*[§]. Given a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, the transitive closure of G is a graph $G' = (V, E')$ such that, for all $u, v \in V$, $(u, v) \in E'$ if and only if there is a path in G that connects u to v. The relation between transitive closure and BMM is clear if the sets E and E' are represented through adjacency matrices; as a matter of fact, the asymptotically fastest algorithms for transitive closure rely on BMM as a primitive [6]. "Traditional" algorithms for transitive closure [10] take $O(mn)$ time in the worst case. For transitive closure, see [4, 12]. Transitive closure has also been studied for undirected graphs [22]; in recent years, research has been focused on dynamically changing graphs [6, 11].

References

[1] Noga Alon, Oded Goldreich, Joan Hastad and René Peralta. Simple constructions of almost k-wise independent random variables. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 544–553, St. Louis, Missouri, October 1990.

[2] Noga Alon, Zvi Galil, Oded Margalit and Moni Naor. Witnesses for boolean matrix multiplications and for shortest paths. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 417–426, Pittsburgh, Pennsylvania, October 1992.

[3] Noga Alon and Moni Naor. Derandomization, witnesses for boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16(4–5): 434–449, 1996.

[4] V. L. Arlazarov, E. A. Dinic, M. M. Konrod and I. A. Faradzev. On economical constructions of the transitive closure of an oriented graph. *Soviet Mathematics Doklady*, 11:1209-1210, 1970.

[5] David H. Bailey. Extra-high speed matrix multiplication on the Cray-2. *SIAM Journal on Scientific and Statistical Computing*, 9(3):603–607, 1988.

[6] Surender Baswana, Sandeep Sen and Ramesh Hariharan. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. In *Proceedings of the 34th Annual ACM Conference on Theory of Computing*, pages 117–123, Montreal, Canada, May 2002.

[7] Donald D. Chinn and Rakesh K. Sinha. Bounds on sample space size for matrix product verification. Technical Report TR-92-12-01, University of Washington, Department of Computer Science and Engineering, 1992.

[8] Don Coppersmith and Samuel Winograd. Matrix multiplication via arithmetic progression. In *Proceedings of the 19th Annual ACM Conference on Theory of Computing*, pages 1–6, New York, 1987.

[9] Don Coppersmith and Samuel Winograd. Matrix multiplication via arithmetic progression. *Journal of Symbolic Computation*, 9:251–280, 1990.

[10] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest. *Introduction to Algorithms*, 1st edition. MIT Press, Cambridge, Massachusetts, 1990.

[11] Camil Demetrescu and Giuseppe F. Italiano. Fully dynamic transitive closure: breaking through the $O(n^2)$ barrier. In *Proceedings of the the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 381–389, Redondo Beach, California, November 2000.

[§] In some papers, it is also called *all pairs reachability*.

[12] Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *Proceedings of the 12th IEEE Symposium on Switching and Automata Theory*, pages 129–131. IEEE Computer Society, October 1971.

[13] Rusins Freivalds. Fast probabilistic algorithms. In *Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science*, LNCS 74, pages 57–69. Olomouc, Czechoslovakia, September 1979.

[14] Michael A. Harrison and Ivan M. Havel. On the parsing of deterministic languages. *Journal of the ACM*, 21(4):525–548, October 1974.

[15] Michael A. Harrison *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Massachusetts, 1978.

[16] Steven Huss-Lederman, Elaine M. Jacobson, J. R. Johnson, Anna Tsao and Thomas Turnbull. Strassen's algorithm for matrix multiplication: modeling, analysis, and implementation. Technical Report CCS-TR-96-147, Argonne National Laboratory, Center for Computing Sciences, November 1996.

[17] Tracy Kimbrel and Rakesh Kumar Sinha. A probabilistic algorithm for verifying matrix products using $O\left(n^2\right)$ time and $\log_2 n + O\left(1\right)$ random bits. *Information Processing Letters*, 45(2):107–110, 1993.

[18] Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *Journal of the ACM*, 49(1):1–15, January 2002.

[19] Wojciech Rytter. Fast recognition of pushdown automaton and context-free languages. *Information and COntrol*, 67(1–3):12–22, 1985.

[20] Wojciech Rytter. Context-free recognition via shortest paths computation: a version of Valiant's algorithm. *Theoretical Computer Science*, 143(2):343–352, 1995.

[21] Giorgio Satta. Tree adjoining grammar parsing and boolean matrix multiplication. *Computational Linguistics*, 20(2):173–192, 1994.

[22] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400-403, December 1995.

[23] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[24] Volker Strassen. *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, chapter "Algebraic complexity theory", pages 633–672. Elsevier and MIT Press, 1990.

[25] Mithuna S. Thottethodi, Siddhartha Chatterjee and Alvin L. Lebeck. Tuning Strassen's matrix multiplication for memory efficiency. In *Proceedings of SC98*, Orlando, Florida, November 1998. (Distributed on CD-ROM.)

[26] Joseph Naor and Moni Naor. Small-bias probability spaces: efficient constructions and applications. In *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, pages 213–223, Baltimore, Maryland, May 1990.

[27] Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10:308-315, 1975.