Fast multiplication of matrices over a finitely generated semiring

Daniel Andrén, Lars Hellström, and Klas Markström

Abstract. In this paper we show that $n \times n$ matrices with entries from a semiring $\mathcal{R}$ which is generated additively by q generators can be multiplied in time $\mathcal{O}(q^2 n^\omega)$, where n^ω is the complexity for matrix multiplication over a ring (Strassen: $\omega < 2.807$, Coppersmith and Winograd: $\omega < 2.376$).

We first present a combinatorial matrix multiplication algorithm for the case of semirings with q elements, with complexity $\mathcal{O}(n^3 / \log_q^2 n)$, matching the best known methods in this class.

Next we show how the ideas used can be combined with those of the fastest known boolean matrix multiplication algorithms to give an $\mathcal{O}(q^2 n^\omega)$ algorithm for matrices of, not necessarily finite, semirings with q additive generators.

For finite semirings our combinatorial algorithm is simple enough to be a practical algorithm and is expected to be faster than the $\mathcal{O}(q^2 n^\omega)$ algorithm for matrices of practically relevant sizes.

1. Introduction

Ever since the advent of Strassen's fast matrix multiplication method [Str69] there has been an active search for new fast matrix multiplication methods. Most of this work have focused on bilinear methods of the same general type as Strassen's method, see [BCS97] for a thorough survey of these methods. Methods of this type usually require that the elements of the matrices have additive inverses and are therefore naturally restricted to matrices with elements from a ring.

Another line of investigation has focused on so-called Boolean matrix multiplication, where the matrices have Boolean values as elements and multiplication and addition are replaced by $\wedge$ (logical AND) and $\vee$ (logical OR) respectively. Here we are no longer dealing with a ring but only a *semiring*, which is the more general algebraic structure obtained by

Key words and phrases. matrix multiplication, semiring.

no longer requiring the existence of additive inverses in the definition of a ring. This and other semirings appears naturally in some important applications such as the study of formal languages, see e.g. [Goo99]. For this problem, fast matrix multiplication methods fall into two categories: on one hand those which do a reduction to integer matrices and then employ a bilinear method, such as Strassen's $\mathcal{O}(n^{\log_2 7})$ method, for the new matrix, and on the other hand combinatorial methods which work within the Boolean semiring itself.

The first sub-cubic method of the latter class of algorithms was given in [ADKF70], requiring $\mathcal{O}(n^3/\log n)$ operations for an $n \times n$-matrix and has over the years been improved in various ways. Here we can mention [AS88] which improves the complexity to $\mathcal{O}(n^3/\log^{1.5} n)$ with a quite simple algorithm and [Ryt85] which gives the asymptotically fastest known method with a complexity of $\mathcal{O}(n^3/\log^2 n)$. The last method is quite complicated and has not been considered to be practical. Rytter's method is also somewhat roundabout in that it is really a method for recognition of context free languages; as shown by [Val75, Lee02] boolean matrix multiplication and parsing of context free languages have mutually dependent complexities.

In [RH88] the reduction to integer matrices was extended from boolean matrices to matrices with entries from a semiring with q elements. In this algorithm the problem is reduced to multiplying q^2 pairs of integer 0/1-matrices.

In this paper we will present two algorithms for multiplication of matrices with elements from any finite semiring $\mathcal{R}$. The first algorithm is a combinatorial method which is simpler than Rytter's method, but achieves the same complexity, $\mathcal{O}(n^3/\log_q^2 n)$, where q is the size of the semiring.

Next we give a multilinear algorithm which combines some of the ideas from the combinatorial algorithm with fast multiplication of matrices with elements from a ring. This algorithm works for semirings with q additive generators, i.e. every element can be written as a linear combination of some set of q elements from the semiring. The running time of the algorithm is $\mathcal{O}(q^2 n^\omega)$, where ω is the exponent for matrix multiplication over a ring. Standard matrix multiplication gives $\omega \leqslant 3$ and Strassen [Str69] showed that it can be lowered to $\omega < 2.807$, with a practical method. Coppersmith and Winograd [CW90] hold the current record with the upper bound $\omega < 2.376$. For finite semirings in which addition is idempotent our multilinear algorithm is formally equivalent to the algorithm from [RH88].

2. The combinatorial algorithm

2.1. The problem. We want to multiply two $n \times n$ matrices A and B with entries from a finite semiring $\mathcal{R}$ with q elements. We assume that we can do the semiring operations of addition and multiplication in $\mathcal{O}(1)$ time (i.e. independent of n, but may be dependent on q). In addition we also assume that we can compute an integer multiple $s \leqslant n$ of any semiring element (i.e., the sum of s identical terms) in time $\mathcal{O}(1)$. This can, using the identity

$$\underbrace{a + \cdots + a}_{s \text{ terms}} = \underbrace{(1 \cdot a) + \cdots + (1 \cdot a)}_{s \text{ terms}} = (\underbrace{1 + \cdots + 1}_{s \text{ terms}}) \cdot a,$$

be done by precomputing a table of the integer multiples $\leqslant n$ of the semiring unit 1 and then use a table lookup together with a semiring multiplication to calculate the multiple in constant time.

We also assume that table lookups can be made in time $\mathcal{O}(1)$ and that matrices can be indexed without problem. The last assumption is realistic as long as the word length of the computer used is of order $\Theta(\log n)$.

2.2. The algorithm. To multiply the $n \times n$ matrix A by the $n \times n$ matrix B we begin by blocking the rows of A k at a time and likewise with the columns of B so we have n/k block-rows A_i of type $[k \times n]$ of A and n/k block-columns B_j of type $[n \times k]$ of B. We then proceed by doing n^2/k^2 block-multiplications $A_i B_j$ of type $[k \times n][n \times k]$. By doing these multiplications in $\mathcal{O}(n)$ time we get an $\mathcal{O}\big(n^3/k^2\big)$ matrix multiplication algorithm.

One way to compute the product $A_i B_j$ is to sum up the n products $\mathbf{a}_{i\ell}\mathbf{b}_{\ell j}$ of type $[k \times 1][1 \times k]$, i.e., each $\mathbf{a}_{i\ell}$ is the ℓth *column* (of length k) from A_i and each $\mathbf{b}_{\ell j}$ is the ℓth *row* (of length k) from B_j;

$$A_i B_j = \sum_{\ell=1}^{n} \mathbf{a}_{i\ell}\mathbf{b}_{\ell j}.$$

We now observe that if we choose k well we have fewer than n different $\mathbf{a}_{i\ell}\mathbf{b}_{\ell j}$-products, henceforth $(\mathbf{a}, \mathbf{b})$-products, so if we instead count the number of times each distinct $(\mathbf{a}, \mathbf{b})$-product occurs in the sum, we can compute the product $\mathbf{ab}$ once and then take a *weighted* sum (according to the number of occurrences of each $(\mathbf{a}, \mathbf{b})$-product) as the answer.

$$A_i B_j = \sum_{\mathbf{a},\mathbf{b} \in \mathcal{R}^k} s(\mathbf{a}, \mathbf{b})\, \mathbf{ab} \qquad \text{where} \quad s(\mathbf{a}, \mathbf{b}) = \Big|\big\{\ell \in [n] \;\big|\; \mathbf{a}_{i\ell} = \mathbf{a}, \mathbf{b}_{\ell j} = \mathbf{b}\big\}\Big|$$

Thus we proceed by first counting the number of occurrences of each pair $(\mathbf{a},\mathbf{b})$, where $\mathbf{a}$ and $\mathbf{b}$ are k-vectors of semiring elements, among the $(\mathbf{a},\mathbf{b})$-products. The counting can be done by first creating a 2-dimensional array of integers, indexed by $\mathbf{a}$ and $\mathbf{a}$, with each entry initialized to 0. Next we scan through our two block column and increase the entry corresponding to each pair $(\mathbf{a},\mathbf{b})$ as they are encountered. This will take time $\mathcal{O}(n)$ since we have n $(\mathbf{a},\mathbf{b})$-products in a block-product. Next we form all possible products of pairs of k-vectors and finally we add the correct multiple of each product to the final sum.

Since the total length of an $(\mathbf{a},\mathbf{b})$-pair is $2k$ there are at most q^{2k} different pairs. To multiply one pair and add the weighted product to the result we need $\mathcal{O}(k^2)$ semiring operations. This gives us a total of $\mathcal{O}(k^2q^{2k})$ operations, so if we can choose k such that this is $\mathcal{O}(n)$ we have our algorithm.

If we choose $k \sim \frac{1}{2}\log_q n - \log_q \log_q n$ we get

$$
\begin{aligned}
k^2q^{2k} \sim \left(\tfrac{1}{2}\log_q n - \log_q\log_q n\right)^2 q^{\log_q n - 2\log_q\log_q n} = \\
\left(\tfrac{1}{4}\log_q^2 n - \log_q n \log_q\log_q n + (\log_q\log_q n)^2\right)\frac{n}{\log_q^2 n} = \\
\frac{n}{4}\left(1 - 4\frac{\log_q\log_q n}{\log_q n} + 4\left(\frac{\log_q\log_q n}{\log_q n}\right)^2\right) = \mathcal{O}(n)
\end{aligned}
$$

and this gives us our $\mathcal{O}\left(\frac{n^3}{\log_q^2 n}\right)$ algorithm.

2.3. Preprocessing optimisations. The part of this algorithm which complexity-wise is most critical is that of *counting* $(\mathbf{a},\mathbf{b})$-products, so the way this may be done in $\mathcal{O}(n)$ time warrants an explanation. If each k-vector $\mathbf{a}_{i\ell}$ or $\mathbf{b}_{\ell j}$ was to be read from memory as k separate elements then these memory accesses alone would constitute $\mathcal{O}(nk)$ operations and render the total complexity $\mathcal{O}(n^3/\log_q n)$ rather than $\mathcal{O}(n^3/\log_q^2 n)$. The vectors $\mathbf{a}_{i\ell}$ and $\mathbf{b}_{\ell j}$ must instead be encoded so that they fit into individual machine words, so that each can be read in $\mathcal{O}(1)$ time. This is not as difficult as it may sound at first, because the above choice of k makes $\left|\mathcal{R}^k\right| < \sqrt{n}$; any word large enough to hold a row or column index can comfortably encode even a pair of k-vectors. As the reencoding of A and the reencoding of B can be carried out independently of each other, the total time it takes is no more than $\mathcal{O}(n^2)$, and this preprocessing is thus dominated by the main step in the algorithm.

The preprocessing required to determine the integer multiples of the semigroup unit merely consists of n semiring additions, so this $\mathcal{O}(n)$ step is similarly dominated by the main step in the algorithm.

If any of the matrices are known to be sparse an additional preprocessing step can be added, where for each block row and block column we record the indices of non-zero subrow and subcolumns. Using this information we make sure that we only consider $(\mathbf{a}, \mathbf{b})$-products which are non-zero, thereby reducing the complexity according to the degree of sparsity.

3. The multilinear algorithm

3.1. The problem. We want to multiply two $n \times n$ matrices A and B with entries from an additively finitely generated semiring $\mathcal{R}$ with q additive generators.

Definition 3.1. A semiring $\mathcal{R}$ is additively finitely generated if there exists a set $S = \{s_1, \ldots, s_q\} \subseteq \mathcal{R}$ such that every element $z \in \mathcal{R}$ can be written as $z = \sum_i a_i s_i$, where the a_i belongs to some ring $\mathcal{R}_c$.

Note that this does not mean that every linear combination of elements from S gives an element from $\mathcal{R}$. We assume that multiplication of the additive generators has been specified as $s_i s_j = \sum_k \epsilon_{kij} s_k$.

The simplest example of a semiring with a finite number of additive generators is of course the natural numbers $\mathbb{N}$. Any infinite, additively idempotent semiring will require an infinite number of additive generators. Here a natural example is the tropical semiring over $\mathbb{R}$, using max as addition and $+$ as multiplication.

We assume that we can do the semiring operations of addition and multiplication in $\mathcal{O}(1)$ time (i.e. independent of n, but may be dependent on q).

3.2. The algorithm. The fastest known method for boolean matrix multiplication is based on a reduction to integer matrices, see e.g. [CLRS01] for a textbook treatment. The basic idea is that given two boolean matrices A and B we interpret the boolean values 0 and 1 as integers, use a fast integer matrix multiplication method to compute $C' = AB$, and finally replace all non-zero entries of C' by 1 to get a matrix C which is the boolean matrix product of A and B. In [RH88] this approach is also extended to show that for a finite semiring with q elements matrix multiplication be reduced to the multiplication of q^2 pairs of integer matrices.

With our combinatorial method in mind we can interpret C'_{ij} as simply counting the number of products $A_{i\ell}B_{\ell j}$ which give a non-zero contribution to C_{ij}, and the final step in going from C' to C as simply performing the semiring sum of those products. This point of view lends itself to immediate generalisation for more general semirings.

A rough outline of our algorithm will be

(1) Given matrices A and B we create two auxiliary matrices A' and B'. If position (i,j) in A is $r = \sum_i a_i s_i$ we will set the same position in A' to $\sum_k a_k x_{s_k}$, where x_{s_k} is a formal variable, and B' is constructed in the same way from B.

(2) Compute $A'B' = C'$ using a fast multilinear algorithm. This will be possible since the elements in A' and B' belong to a ring. In fact they will be low degree polynomials.

(3) Construct the matrix $C = AB$ by transforming the polynomials at the entries of C' into elements of the semiring.

Let us now fill in the details of this outline.

To evaluate the multiplication $AB = C$ of two $n \times n$ matrices A and B over a finitely generated semiring $\mathcal{R}$ with q generators we start by mapping the entries of the matrices to a semigroup algebra $\mathcal{R}_c[\mathcal{R}]$; in other words we map a semiring element $r = \sum_i a_i s_i \in \mathcal{R}$ to the element $\sum_k a_k x_{s_k} \in \mathcal{R}_c[\mathcal{R}]$. The basis elements x_{s_k} of the algebra are multiplied according to the rule $x_{s_i} x_{s_j} = \sum \epsilon_{kij} x_{s_k}$. Addition in $\mathcal{R}_c[\mathcal{R}]$ is however strictly the addition of an $\mathcal{R}_c$-module; the addition of $\mathcal{R}$ is not used in the definition of $\mathcal{R}_c[\mathcal{R}]$.

Let A' and B' be the matrices where we have sent the elements a_{ij} and b_{ij} from A and B respectively to $x_{a_{ij}}$ and $x_{b_{ij}}$. The product $C' = A'B'$ will contain formal polynomials of the form

$$c'_{ij} = \sum_{r \in S} d_{ijr} x_r.$$

These polynomials will count the number of times $r \in \mathcal{R}$ occurs in the sum that make up the position c_{ij} in C. We can evaluate these polynomials in the semiring by mapping the x_r back to $r \in \mathcal{R}$. This will take q multiplications and $q-1$ additions for each c_{ij}, in total $\mathcal{O}(qn^2)$ algebraic operations.

The product $A'B' = C'$ is computed with matrices over the semigroup algebra $\mathcal{R}_c[\mathcal{R}]$, which in particular is also a ring, so we can use a fast matrix multiplication algorithm, such as Strassen's method, and only do $\mathcal{O}(n^\omega)$ ring operations. These operations will be addition and multiplication in

$\mathcal{R}_c[\mathcal{R}]$, each of which can trivially be carried out in $\mathcal{O}(q^2)$ operations in $\mathcal{R}$ and $\mathcal{R}_c$ (although it may be possible to lower this exponent for particular cases of $\mathcal{R}$). This gives us an algorithm that will perform the product $A'B' = C'$ in $\mathcal{O}(q^2 n^\omega)$ algebraic operations. Since forming C' will be the dominant contribution to the complexity we have an $\mathcal{O}(q^2 n^\omega)$ algorithm for matrix multiplication over a finitely generated semiring.

4. Comparing the two methods for finite semirings

A disadvantage of the multilinear matrix multiplication method, as compared to the combinatorial method, is that it needs more memory. While the combinatorial method can be carried out in an amount of memory that is bounded by a universal constant multiple of the input data size, the matrices over the ring $\mathcal{R}_c[\mathcal{R}]$ in the integer method contain qn^2 integers and can thus be expected to require q times as much memory as the input data did. This q is still a constant as far as the asymptotics are concerned, but it varies with $\mathcal{R}$ and should be taken into account when choosing between the methods.

Further, if we ignore the constants in the $\mathcal{O}$-notation we see that the multilinear method will be faster than the combinatorial methods when $n^\omega < \frac{n^3}{\log_q^2 n}$. If we take $q = 2$ and compare the two methods when Strassen's method is used in the multilinear algorithm we find that the combinatorial method has the advantage for $n < 2^{59}$. With the bound for ω given by Coppersmith and Winograd this is reduced to $n < 2^{11}$. In both cases we have ignored multiplicative constants but given the size of the constants involved it is safe to say that in a practical implementation, for small q, the combinatorial method will be faster unless n is very large.

References

[ADKF70] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradžev. The economical construction of the transitive closure of an oriented graph. *Dokl. Akad. Nauk SSSR*, 194:487–488, 1970.

[AS88] Michael D. Atkinson and N. Santoro. A practical algorithm for Boolean matrix multiplication. *Inform. Process. Lett.*, 29(1):37–38, 1988.

[BCS97] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1997. With the collaboration of Thomas Lickteig.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms.* MIT Press, Cambridge, MA, second edition, 2001.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9(3):251–280, 1990.

[Goo99] Joshua Goodman. Semiring parsing. *Comput. Linguist.*, 25(4):573–605, 1999.

[Lee02] Lillian Lee. Fast context-free grammar parsing requires fast Boolean matrix multiplication. *Journal of the ACM*, 49(1):1–15, 2002.

[RH88] Daniel J. Rosenkrantz and Harry B. Hunt, III. Matrix multiplication for finite algebraic systems. *Inform. Process. Lett.*, 28(4):189–192, 1988.

[Ryt85] Wojciech Rytter. Fast recognition of pushdown automaton and context-free languages. *Inform. and Control*, 67(1-3):12–22, 1985.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[Val75] Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. System Sci.*, 10:308–315, 1975.

Department of Mathematics, Umeå University, SE-901 87 Umeå, Sweden

E-mail address: Daniel.Andren@math.umu.se

E-mail address: Lars.Hellstrom@residenset.net

E-mail address: Klas.Markstrom@math.umu.se