Information Processing Letters 29 (1988) 37–38
North-Holland

15 September 1988

A PRACTICAL ALGORITHM FOR BOOLEAN MATRIX MULTIPLICATION *

Michael D. ATKINSON and N. SANTORO

School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6

Communicated by W.L. Van der Poel
Received 20 May 1987

An algorithm is given for multiplying two $n \times n$ Boolean matrices. It has time complexity $O(n^3/(\log n)^{1.5})$ and requires $n \log_2 n$ bits of auxiliary storage.

Keywords: Boolean matrix multiplication, time vs. storage tradeoffs

Research on fast methods for multiplying two $n \times n$ Boolean matrices has followed two directions. On the one hand, there are the asymptotically fast methods derived from algorithms to multiply matrices with integer entries. On the other hand, it has been of interest to devise algorithms which, while inferior in the asymptotic sense, are simple enough to be advantageously implemented. This paper addresses the latter class of algorithms. The first such algorithm (apart from the classical straightforward $O(n^3)$ algorithm) was given in [3] and is known, at least in Western literature, as the Four Russians' algorithm. It requires $O(n^3/\log n)$ time and auxiliary storage of $O(n^3/\log n)$ bits; simple modifications can reduce the amount of auxiliary storage to $O(n^2)$. More recent work on the problem has focussed on reducing the amount of auxiliary storage [5] and on exploiting properties of the arrays to be multiplied [4,6].

The purpose of this paper is to point out that a rather simple and known technique can be used to derive a Boolean matrix multiplication algorithm which has a lower time complexity and requires less auxiliary storage than the Four Russians' method. This technique was originally proposed to compute matrix products over the field of integers modulo two [2, Problem 6.16] and has been employed to reduce the complexity of Boolean implementations of asymptotically fast matrix multiplication algorithms [1].

The essential idea is to partition each of the $n \times n$ matrix factors into $(n/k)^2$ $k \times k$ square submatrices, calculate the product by block matrix multiplication using $(n/k)^3$ multiplications and $(n/k)^3$ additions of $k \times k$ matrices, and do these matrix multiplications and additions by table look-up. Specifically the algorithm is as follows.

Algorithm Multiply

```
($A$, $B$: $n \times n$ input Boolean matrices; $C$: the matrix product $A \times B$);

Choose a suitable submatrix block size $k$;

for every $k \times k$ matrix $P$ and every $k \times k$ matrix $Q$
  $Product[P, Q] := P \times Q$;
  $Sum[P, Q]     := P + Q$
endfor;

Partition each of $A$ and $B$ into $(n/k)^2$ $k \times k$ submatrices $A_{ij}$ and $B_{ij}$ and build the corre-
```

* This research was supported by the National Science and Engineering Research Council of Canada under Grants Nos. A2419 and A2415.

sponding partition of the product by:

```
for every pair $(i, j)$ with $1 \leqslant i, j \leqslant n/k$
  $C_{ij} := 0$;
  for every $r$ with $1 \leqslant r \leqslant n/k$
    $C_{ij} := Sum[C_{ij}, Product[A_{ir}, B_{rj}]]$
  endfor;
endfor;
```

The two arrays *Product* and *Sum* are slightly unusual in having subscripts which are themselves square Boolean arrays. However, since arrays are generally stored as vectors (row major or column major storage) it is not difficult to interpret a Boolean array as an integer row subscript or column subscript into the arrays *Product* and *Sum*. An implementation in a high-level language might adopt the policy of encoding every $k \times k$ submatrix of A and B as an integer and representing *Product* and *Sum* as integer arrays with integer subscripts.

Proposition. *The algorithm requires* $2k^2 4^{k^2}$ *bits of auxiliary storage and has time complexity* $O(k^3 4^k + (n/k)^3)$.

Proof. Each of the tables *Product* and *Sum* has 2^{k^2} rows and 2^{k^2} columns, and so has 4^{k^2} entries each of which is a $k \times k$ Boolean matrix. Thus, *Product* and *Sum* occupy a total of $2k^2 4^{k^2}$ bits of storage. To build the table *Product* requires 4^{k^2} $k \times k$ Boolean matrix products to be calculated. The time required for this is $O(k^3 4^{k^2})$. Similarly, the construction of *Sum* requires $O(k^2 4^{k^2})$ time. The final part of the algorithm accesses the tables *Product* and *Sum* $(n/k)^3$ times and each access requires constant time. The other execution time costs (such as the matrix initializations $C_{ij} := 0$, and the placing of each C_{ij} in the correct position in the output array) are all $O(n^2)$. □

The choice of the parameter k offers a trade-off between time and space complexity. Taking $k = \sqrt{\log_4 n}$ results in an auxiliary space requirement of $2n \log_4 n = n \log_2 n$ bits. For this choice of k, the time complexity of the algorithm is easily seen to be

$$c_k \cdot c_A \left(n^3/(\log_2 n)^{1.5}\right) + \text{lower order terms},$$

where $c_k = \sqrt{8} = 2.828\ldots$, and c_A is the (implementation-dependent) constant representing the time required to perform an access of the form $Sum[X, Product[Y, Z]]$. The value c_k can be reduced by taking k to be larger. For example, $k = \sqrt{\log_2 n}$ yields $c_k = 1$ and still results in the modest space requirement of $2n^2 \log_2 n$ bits of auxiliary storage.

This technique can obviously be applied to other small algebras apart from the $\{0, 1\}$ Boolean algebra, as in the case described in [2, Problem 6.16]. For example, if the entries of the matrices to be multiplied are integers in $\mathbb{Z}_m$ for some small modulus m, we could construct the multiplication and addition tables for $k \times k$ matrices over $\mathbb{Z}_m$; taking $k = \sqrt{(\log_m n)/2}$ these would occupy just $2m^{2k^2}k^2 = n \log_m n$ integer locations of storage (or $n \log_2 n$ bits of storage). Once these tables are available, the $n \times n$ matrix product can be found in $n^3/k^3 = 2.828n^3(\log_m n)^{1.5}$ table accesses of the type $Sum[X, Product[Y, Z]]$.

Also in this case, a trade-off between the space and the (multiplicative constant in the) time complexity is implied from the choice of parameter k.

References

[1] L. Adleman, K.S. Booth, F.P. Preparata and W.L. Ruzzo, Improved time and space bounds for Boolean matrix multiplication, *Acta Informatica* **11** (1978) 61–75.

[2] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[3] V.L. Arlazarov, E.A. Dinic, M.A. Kronrod and I.A. Faradzev, On economical construction of the transitive closure of a directed graph, *Dokl. Akad. Nauk SSSR* **194** (1970) 487–488 (in Russian).

[4] N. Santoro, Four $O(N^2)$ multiplication methods for sparse and dense Boolean matrices, *Proc. 10th Conf. on Numerical Mathematics and Computing*, In: *Congressus Numerantium Vol. 31* (Utilitas Mathematica, Winnipeg, 1981) 241–252.

[5] N. Santoro and J. Urrutia, An improved algorithm for Boolean matrix multiplication, *Computing* **36** (1986) 375–382.

[6] J. Vyskoc, A note on Boolean matrix multiplication, *Inform Process Lett.* **19** (1984) 249–251.