The University of Warwick

THEORY OF COMPUTATION

REPORT

No. 2

COMPLEXITY OF MONOTONE NETWORKS FOR

BOOLEAN MATRIX PRODUCT

MICHAEL S. PATERSON

Department of Computer Science
University of Warwick
COVENTRY CV4 7AL
ENGLAND.

July 1974

COMPLEXITY OF MONOTONE NETWORKS FOR BOOLEAN MATRIX PRODUCT

by

Michael S.Paterson

Mailing address:

Department of Computer Science
University of Warwick
COVENTRY CV4 7AL
Warwickshire
ENGLAND.

Complexity of monotone networks for Boolean matrix product.

Abstract.

Any computation of Boolean matrix product by an acyclic network using only the operations of binary conjunction and disjunction requires at least IJK conjunctions and IJ(K-1) disjunctions for the product of matrices of sizes I x K and K x J. Furthermore any two such networks having these minimum numbers of operations are equivalent using only the commutativity of both operations and the associativity of disjunction.

1. Introduction.

The product of an I x K Boolean matrix A with a K x J Boolean matrix B (I x K x J Boolean product) is the I x J matrix C defined by

$$c_{ij} = \bigvee_{k=1}^{K} a_{ik} \wedge b_{kj} \quad \text{for} \quad i = 1,\ldots, I;\ j = 1,\ldots,J.$$

We consider the computation of Boolean products by acyclic logical networks with binary conjunctions ($\wedge$-gates) and binary disjunctions ($\vee$-gates) as the logic elements. These are called monotone networks because of the properties of the basic elements. An example of a monotone network for 1 x 2 x 1 product is given in figure 1. It has four inputs, a_{11}, a_{12}, b_{11}, b_{21}, and one output, c_{11}. In general, for an I x K x J product there would be IK + KJ inputs and IJ outputs.

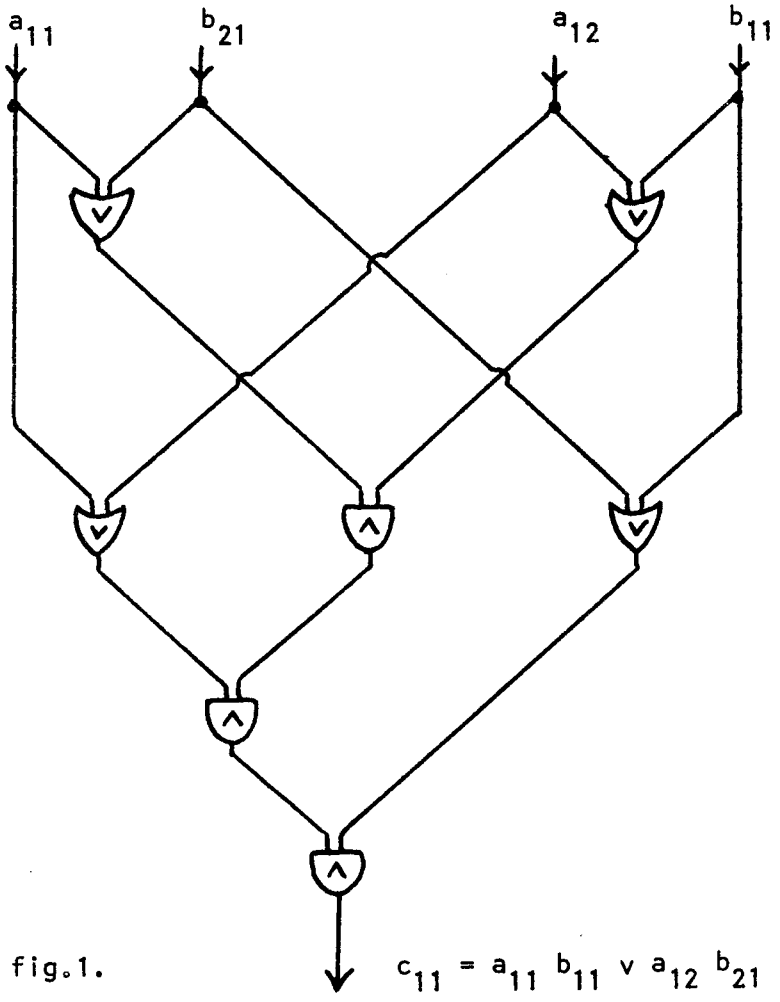

fig.1. $c_{11} = a_{11}\, b_{11} \vee a_{12}\, b_{21}$

Of course the form of the definition of matrix product suggests a straightforward network with IJK $\wedge$-gates, the results of which are combined using IJ(K-1) v-gates. The principal results of this paper are that these numbers are each minimal and that any minimal monotone network for Boolean product is of such form. Several of the ideas for our proofs come from a recent paper of Pratt [3], in which he proves an $N^3/2$ lower bound on the number of $\wedge$-gates for N x N x N product.

These results are of particular interest in juxtaposition with the construction described in [1] of (non-monotone) networks for NxNxN Boolean product using $\wedge$,v and <u>complementation</u> as basic elements which require only $O(N^{\log_2 7}.(\log N)^{1+\epsilon})$ such elements.

A corollary of our results slightly extends some of the results in [2] concerning the (min, +)-product of real matrices, i.e. 'minimum' and '+' replace '+' and 'x' respectively in the usual definition of real matrix product. This is because any (min, +) network for matrix product becomes a network for Boolean product when each 'min' and each '+' is replaced by 'v' and '$\wedge$' respectively.

2. Notation and preliminaries.

From now on we use <u>network</u> to mean a monotone network which computes I x K x J Boolean matrix product for some fixed non-zero I,J,K. Each 'wire' or connecting arc of a network has naturally associated with it a monotone Boolean function of the input variables. The functions associated with the pair of input wires to any gate we call its <u>arguments</u> and the output function its <u>result</u>. We shall often use mere juxtaposition to denote conjunction, e.g. '$a_{11}\ b_{11}$' for '$a_{11} \wedge b_{11}$', and for brevity write 1 for <u>true</u> and 0 for <u>false</u>. A Boolean function is identified with the set of argument values which yield the function value '<u>true</u>', so we could write, for example,

$$0 \subseteq a_{11}\ b_{11} \subseteq a_{11} \subseteq a_{11} \vee b_{11} \subseteq 1.$$

We denote by U, the sequence of all input variables $(a_{11},\ldots, a_{IK}, b_{11},\ldots, b_{KJ})$, and use $x, x_1, x_2,\ldots$ to denote arbitrary elements of U. $s, s', s_1, s_2,\ldots$ will be used for arbitrary monotone functions. Two elementary properties of monotone functions are stated without proof.

P1. If $s_1 \vee s_2 \supseteq x_1\, x_2$ then $s_1 \supseteq x_1\, x_2$ or $s_2 \supseteq x_1\, x_2$

P2. If $s_1\, s_2 \subseteq x_1 \vee\ldots\vee x_k$ then $s_1 \subseteq x_1 \vee\ldots\vee x_k$ or $s_2 \subseteq x_1 \vee\ldots\vee x_k$

3. Refinement.

Extremely inefficient networks can be designed. The refining process described in this section consists of applying local transformations to networks in order to remove redundant gates whilst preserving its input/output behaviour. For example, if $s_1 \subseteq s_2$ then an $\wedge$-gate with arguments s_1, s_2 can be eliminated since its result is merely s_1. This simplification, and many others, can be expressed by this obvious duplication rule,

R1 : If g_1, g_2, are gates (or an input and a gate) with the same function and g_2 does not precede g_1 in the network then g_2 may be eliminated and all connections to the output of g_2 made to g_1 instead.

It is convenient to allow the constants 0, 1, as inputs to a network, but any use of these constants is clearly eliminable by applications of R_1.

More interesting refinement rules depend on knowledge of the output functions of the network. We introduce here this important notion of specific refinement. A fairly simple transformation allows the removal from a function of terms which are 'useless'. For example, '$a_{11} \vee b_{12}\, b_{21}$'

may be replaced by 'a_{11}' since the term '$b_{12}\ b_{21}$' cannot contribute anything essential to the final result. We define a function (set), Dross, which contains all such 'useless' terms.

$$\text{Dross} = \bigvee_{(i,k)\neq(i',k')} a_{ik}\ a_{i'k'} \vee \bigvee_{(k,j)\neq(k',j')} b_{kj}\ b_{k'j'} \vee \bigvee_{\substack{i,j \\ k\neq k'}} a_{ik}\ b_{k'j}$$

In the language of assignment statements for programs,

R2 : For any $x \in U$, if $x \subset s \subseteq x \vee \text{Dross}$ then $s := x$.

Note that the restriction of x to input variables is not necessary for the correctness of R_2, but just to ensure that the application of R_2 eliminates at least one gate.

The correctness of R_2 may be easily established by following the effect of the changes forward through the network. At each stage, only 'dross' is removed.

The next set of specific refinements requires some justification. They all identify functions which are so extensive that they can be replaced by 1. For all i,j, we use c_{ij} to denote the function $\bigvee_k a_{ik}\ b_{kj}$.

Lemma.

For all $i \neq i'$, $j \neq j'$, and all i'', j'', and for all monotone functions s, if $s(U,\ a_{i1}\ b_{1j} \vee a_{i'1}\ b_{1j'}) = c_{i''j''}$ then $s(U,\ 1) = c_{i''j''}$.

Proof.

Suppose the Lemma is false, then there is a valuation α on the inputs such that, under α

$$c_{i''j''} = s(U,\ a_{i1}\ b_{1j} \vee a_{i'1}\ b_{1j'}) = 0 \quad \text{but} \quad s(U,\ 1) = 1$$

Let α be a maximal such valuation, i.e. if any input value is changed from 0 in α to 1, the value of $C_{i''j''}$ changes from 0 to 1. Under α, we must have

$$a_{i1}\, b_{1j} \vee a_{i'j}\, b_{1j'} = 0$$

Without loss of generality suppose $a_{i1} = 0$ and either (i) $a_{i'1} = 0$, or (ii) $b_{1j'} = 0$. Since α is maximal, changing a_{i1} to 1 changes $c_{i''j''}$. Therefore $i = i''$ and $b_{1j} = 1$. In case (i) we can deduce similarly that $i' = i''$ which yields a contradiction. In case (ii), we have on the one hand that $b_{1j''} = 1$ and $b_{1j'} = 0$, therefore $j' \neq j''$, while on the other hand changing $b_{1j'}$ to 1 changes $c_{i''j''}$, therefore $j' = j''$. This contradiction proves the Lemma □

Corollary 1.

If $s(U, s') = c_{i''j''}$ and $a_{i1}\, b_{1j} \vee a_{i'1}\, b_{1j'} \subseteq s'$

then $s(U, 1) = c_{i''j''}$.

Proof.

By monotonicity □

Particular values for s' in applications will be

$a_{i1} \vee a_{i'1}$, $b_{1j} \vee b_{1j''}$, $a_{i1} \vee b_{1j'}$.

Corollary 2.

(i) If $s(U, a_{i1}\, b_{1j} \vee a_{i'1}\, b_{1j}) = c_{i''j''}$ then $s(U, b_{1j}) = c_{i''j''}$.

(ii) If $s(U, a_{i1}\, b_{1j} \vee a_{i1}\, b_{1j'}) = c_{i''j''}$ then $s(U, a_{i1}) = c_{i''j''}$.

Proof.

For (i), there is a monotone function s' such that for all z, $s'(U, z) = s(U, z \wedge b_{1j})$. Corollary 1 is applied with $z = a_{i1} \vee a_{i'1}$; similarly for (ii) □

Corollary 1 justifies the following refinement rules, where we assume $i \neq i'$ and $j \neq j'$ always

R3 : If $a_{i1} \vee a_{i'1} \subseteq s$ then s := 1

R4 : If $b_{1j} \vee b_{1j'} \subseteq s$ then s := 1

R5 : If $a_{i1} \vee b_{1j} \subseteq s$ then s := 1 .

Corollary 2 yields

R6 : If $a_{i1}\, b_{1j} \vee a_{i'1}\, b_{1j} \subseteq s \subset b_{1j}$ then $s := b_{1j}$

R7 : If $a_{i1}\, b_{1j} \vee a_{i1}\, b_{1j'} \subseteq s \subset a_{i1}$ then $s := a_{i1}$

4. Outline of main proof.

A first approach might be to look for a 1-1 mapping from triples of indices (i, j, k) to $\wedge$-gates, since one might suppose there to be at least one $\wedge$-gate which 'essentially' computes $a_{ik} \wedge b_{kj}$. However there seems not to be any 'natural' such correspondence. An example like figure 1, which allows no specific refinement, suggests some of the difficulties. The proof we give here relies on finding such a mapping from <u>pairs</u> (i, j) to $\wedge$-gates, corresponding to the term $a_{i1} \wedge b_{1j}$. These $\wedge$-gates are then eliminated after fixing the values of the inputs, $a_{i1} = 1$ for all i and

$b_{1j} = 0$ for all j. The resulting network is a valid network for I x (K-1) x J Boolean product, so a new set of at least I x J $\wedge$-gates corresponding to the terms $a_{12} \wedge b_{2j}$ can be found, and so on. The procedure for $\vee$-gates is similar. The arguments are greatly simplified by assuming that the refinement rules have been applied wherever possible.

The mapping for $\wedge$-gates is determined by defining for each i,j, a predicate Q_{ij} on the functions associated with the wires of the network. The set of initial occurrences of Q_{ij}, denoted by $I(Q_{ij})$, consists of those gates whose result satisfies Q_{ij} but neither of whose arguments satisfies Q_{ij}. Q_{ij} is such that no input variable satisfies the predicate, but c_{ij} does. This guarantees that $I(Q_{ij})$ is non-empty. We further show that the sets $I(Q_{ij})$ are disjoint for distinct (i,j), that $I(Q_{ij})$ contains only $\wedge$-gates, and that the valuation $a_{i1} = 1$, $b_{1j} = 0$ allows all of $I(Q_{ij})$ to be eliminated.

For $\vee$-gates, we have corresponding predicates R_{ij}. Provided that K > 1, the sets $I(R_{ij})$ are disjoint, non-empty and contain only $\vee$-gates. The same valuation allows these $\vee$-gates to be eliminated.

5. Lower Bounds.

Theorem 1.

Every $(\wedge,\vee)$-network for I x K x J Boolean product contains at least IJK $\wedge$-gates and IJ (K-1) $\vee$-gates.

Proof.

We may suppose that the network has the minimal total number of gates and therefore no non-trivial application of a refinement rule is possible.

For all i, j, define

$$Q_{ij}(s) \leftrightarrow a_{i1}\, b_{1j} \subseteq s \;\&\; a_{i1} \not\subseteq s \;\&\; b_{1j} \not\subseteq s$$

$I(Q_{ij})$ is the set of initial occurrences of Q_{ij} as defined in section 4. Suppose gate g is in $I(Q_{ij})$ with arguments s_1, s_2. If it is an $\vee$-gate then $a_{i1} \not\subseteq s_1$ and $b_{1j} \not\subseteq s_1$, since $Q_{ij}\ (s_1 \vee s_2)$. Therefore $a_{i1}\, b_{1j} \not\subseteq s_1$, since $\neg\ Q_{ij}(s_1)$. The same holds for s_2 and so by P1 in section 2, $a_{i1}\, b_{1j} \not\subseteq s_1 \vee s_2$ which contradicts $Q_{ij}\ (s_1 \vee s_2)$. Therefore g must be an $\wedge$-gate. It is now easy to show that $a_{i1} \subseteq s_1$ and $b_{1j} \subseteq s_2$ or vice versa. Without loss of generality, assume the former. Thus the valuation $a_{i1} = 1$ would allow the elimination by R1 of all gates in $I(Q_{ij})$.

Suppose g is in $I(Q_{ij}) \cap I(Q_{i'j'})$, then either

$$a_{i1} \vee a_{i'1} \subseteq s_1 \quad \text{and} \quad b_{1j} \vee b_{1j'} \subseteq s_2$$

or

$$a_{i1} \vee b_{1j'} \subseteq s_1 \quad \text{and} \quad b_{1j} \vee a_{i'1} \subseteq s_2$$

If $(i,j) \neq (i', j')$, then at least one of R3, R4, R5, is applicable in each case, which contradicts the assumption of minimality. Thus the sets $I(Q_{ij})$ are disjoint.

If $K > 1$, for all i,j, define

$$R_{ij}\ (s) \leftrightarrow a_{i1}\, b_{1j} \subseteq s \subseteq A_i \vee b_{1j} \;\&\; s \not\subseteq b_{1j}$$

$$\text{where } A_i = \bigvee_{k \neq 1} a_{ik}$$

Suppose g is in $I(R_{ij})$ with arguments s_1, s_2. If it is an $\wedge$-gate then $s_1 \not\subseteq A_i \vee b_{1j}$ and $s_2 \not\subseteq A_i \vee b_{1j}$ since $R_{ij}\ (s_1 \wedge s_2)$ but $\neg R_{ij}\ (s_1)$ and $\neg R_{ij}\ (s_2)$. But P2 now implies that $s_1 \wedge s_2 \not\subseteq A_i \vee b_{1j}$ which is a contradiction. Hence g is an $\vee$-gate, and so both s_1, s_2 are

contained in $A_i \vee b_{1j}$. By P1, at least one of s_1, s_2, say s_1, contains $a_{i1}\, b_{1j}$ and so $s_1 \subseteq b_{1j}$, since $\neg R_{ij}(s_1)$. Thus the valuation $b_{1j} = 0$ would allow the elimination of all gates in $I(R_{ij})$.

Suppose g is in $I(R_{ij}) \cap I(R_{i'j'})$, then $s_1 \subseteq b_{1j} \wedge (A_{i'} \vee b_{1j'})$, and so $j = j'$ since $a_{i1}\, b_{1j} \subseteq s_1$. Since $R_{i'j}(s_1 \vee s_2)$, we have $s_2 \not\subseteq b_{1j}$ and therefore $a_{i'1}\, b_{1j} \not\subseteq s_2$ because $\neg R_{i'j}(s_2)$. Hence $a_{i'1}\, b_{1j} \subseteq s_1$. So $a_{i1}\, b_{1j} \vee a_{i'1}\, b_{1j} \subseteq s_1 \subseteq b_{1j}$. If $i \neq i'$ then from R6 we have $s_1 = b_{1j}$. Since

$$b_{1j} \subseteq s_1 \vee s_2 \subseteq (A_i \vee b_{1j}) \wedge (A_{i'} \vee b_{1j}) \subseteq b_{1j} \vee \text{Dross},$$

R2 implies that $s_1 \vee s_2 = b_{1j}$, and so g can be eliminated by R1. This contradiction shows that the sets $I(R_{ij})$ are disjoint.

For all i,j, no input variable satisfies Q_{ij} or R_{ij}, while c_{ij} satisfies Q_{ij} and if $K > 1$ it also satisfies R_{ij}. Thus if $K > 1$, all the $I(Q_{ij})$ and $I(R_{ij})$ are non-empty and if we fix $a_{i1} = 1$ for all i and $b_{1j} = 0$ for all j, all the gates in $I(Q_{ij})$ and $I(R_{ij})$ can be eliminated. By disjointness, this is a total of at least IJ $\wedge$-gates and IJ $\vee$-gates. The network which remains is a valid network for I x (K-1) x J Boolean product since the function at the (i,j) output is now $\bigvee_{k>1} a_{ik}\, b_{kj}$. If $K = 1$ then we eliminate at least IJ $\wedge$-gates by the valuation. The theorem is therefore established by an inductive argument □

6. Characterization of optimal networks.

Theorem 2.

Any network for I x K x J Boolean matrix product using the minimal numbers of $\wedge$-gates and $\vee$-gates, computes $a_{ik} \wedge b_{kj}$ for all i,j,k, directly using IJK $\wedge$-gates and then for each i,j, computes $\bigvee_k a_{ik} \wedge b_{kj}$ with K-1 $\vee$-gates using a total of IJ(K-1) $\vee$-gates. Thus any two minimal networks are interconvertible using only the commutativity of $\wedge$ and $\vee$ and the associativity of $\vee$.

Proof.

Suppose we have a network with the minimal numbers of $\wedge$-gates and v-gates, then <u>only</u> the gates in $I(Q_{ij})$ or $I(R_{ij})$ for some i,j can be eliminated by the valuation of $a_{i1} = 1$ for all i and $b_{1j} = 0$ for all j. Therefore in particular a_{i1} may appear as an argument only in the gates of $I(Q_{i'j})$ or $I(R_{i'j})$ for some i', j. The analysis in the proof of Theorem 1 shows that the latter case is not possible, and so a_{i1} can be an argument of $\wedge$-gates only. By the symmetries of Boolean matrix product, irrespective of the structures we have superimposed for the sake of our proofs, <u>every</u> input variable is an argument of $\wedge$-gates only. Now it is easy to show for such a network that no gate has a result s with $x \subseteq s$ for an input x. Hence from the proof of Theorem 1 we see that the $\wedge$-gate of $I(Q_{ij})$ has arguments a_{i1} and b_{1j}, and therefore again by symmetry the IJK $\wedge$-gates of a minimal network have argument pairs (a_{ik}, b_{kj}) for all i, j, k. Since the IJ final outputs must be computed directly from these IJK results using IJ(K-1) v-gates the conclusion of the Theorem is now a trivial deduction □

7. Conclusion.

Several of the techniques used in these proofs may have wider applications. Regarding the 'specific refinements', we find that knowing certain properties of the output functions of a network allows us to make unexpected local simplifications which would not be valid in general. It would be interesting to know whether this phenomenon extends to non-monotone networks. The method of examining 'initial occurrences' of suitably chosen predicates is very convenient where applicable. Of course the difficulty lies in choosing the right predicates, since they are not intuitively obvious.

The results presented here give a satisfactory answer to questions about the complexity of monotone networks for Boolean matrix product. A closely related computation is to find the transitive closure of a square Boolean matrix. The complexity is known to be of the same order as that of matrix product but there is still a considerable gap between the upper and lower bounds. I should expect a complete solution to this problem to be very much harder to obtain.

Acknowledgement.

Michael Fischer has been of great help in clarifying several proofs.

References

[1] M.J. Fischer and A.R. Meyer, Boolean matrix multiplication and transitive closure, Conf. Record 12th Symp. on Switching and Automata Theory, IEEE 1971, pp. 129-131.

[2] L.R. Kerr, The effect of algebraic structure on the computational complexity of matrix multiplication, Ph.D. thesis, Dept. of Computer Science, Cornell University, June 1970.

[3] V.R. Pratt, The power of negative thinking in multiplying Boolean matrices, Proceedings of 6th ACM Symposium on Theory of Computing, (1974), 80-83.