Thus, a function such as that of our example, which exhibits two partitions at first, will show four or three partitions after the x_1 column is doubly complemented. It should be observed here that such a situation is brought about by doubly complementing only the x_1 column, and not any other column. This is because all other columns have both 0's and 1's in the rows of the other subpartition (non-indicator) of a partition. Hence we must modify the previous algorithm[1] as follows. As before, it is assumed that the function under test is a unity-ratio function.

Step 1: Write the unity-ratio function in the ordered partitioned tabular form. Compute the number of rows of each partition.

Step 2: Check if the number of occurrences of rows of weight α equals $\binom{n}{\alpha}$ for all α's. If this is not satisfied by a single α, go to Step 3. If yes, the function is a totally symmetric function (tsf). Write its designation with α-numbers and variables of symmetry. Stop.

Step 3: Detect column/columns having all 0 and/or all 1 entries in one or more partitions. If such columns are detected go to Step 5. If not go to the next step.

Step 4: Detect column/columns having all 0 and/or all 1 entries in one or more indicator subpartitions. If such columns are detected go to the next step. If not, the function is not a tsf. Stop.

Step 5: In case more than one partition or indicator subpartition has indicated all 0 and/or all 1 columns, check that they are not contradictory. If not, go to the next step. If yes, the function is not a tsf. Stop.

Step 6: Doubly complement the detected columns, calculate the new row weights. Check that the number of occurrences of rows of row weight α equals $\binom{n}{\alpha}$. If this is not satisfied by even a single α, then go to the next step. If yes, then the function is a tsf. Write its designation with α-numbers and variables of symmetry. Stop.

Step 7: Check if the all 0 and/or all 1 columns have been detected by partitions or subpartitions. If by partitions, then the function is not a tsf. Stop. If by subpartitions, then go to the next step.

Step 8: Doubly complement all detected columns except that of x_1. Calculate the new row weights. Check that the number of occurrences of rows of row weight α equals $\binom{n}{\alpha}$ for all α's. If this is not satisfied by even a single α, then the function is not a tsf. Stop. If yes the function is a tsf. Write its designation with α-numbers and variables of symmetry. Stop.

Comments on "The Relationship Between Multivalued Switching Algebra and Boolean Algebra Under Different Definitions of Complement"

GEORGE EPSTEIN

***Abstract*—This correspondence identifies the definitions of the above paper[1] in mathematical terms. Both algebraic and logical terms are given.**

***Index Terms*—Complemented pseudosupplement, involution, negated affirmation, pseudocomplement, strong negation, weak negation.**

Of the four definitions given in the above paper,[1] the first three definitions have mathematical terms that are given below as representative, but not exhaustive, of those existing in the literature.

Definition 1:

$$\bar{x} = < \begin{matrix} 0, & \text{if } x \neq 0 \\ p, & \text{if } x = 0. \end{matrix}$$

This definition in algebraic terms is that of a pseudocomplement, and in logical terms is that of a weak negation. The pseudocomplement of x is the largest element x^* such that $x \cdot x^* = 0$. Since the strong rule $\bar{\bar{x}} = x$ does not hold, but the weak rule $\bar{\bar{\bar{x}}} = \bar{x}$ does hold, this is referred to as a weak negation. In a previous paper [1], this appears as $C_0(x)$, and in a recent abstract [2] and paper [3] as $\neg x$ or $\sim x$.

Definition 2:

$$\bar{x} = < \begin{matrix} p, & \text{if } x \neq p \\ 0, & \text{if } x = p. \end{matrix}$$

This definition in algebraic terms is that of a complemented pseudosupplement, and in logical terms is that of a negated affirmation. The pseudosupplement of x is the largest complemented element $!\,x$ such that $!\,x \leqslant x$. Hence this definition gives the complement of the pseudosupplement of x. If affirmation is the logical term corresponding to pseudosupplement, then this is the negation of an affirmation [2], [3]. In [1] this appears as: $\overline{C_p(x)} = C_0(C_p(x))$, and in a recent abstract [2] and paper [3] as $\neg(!\,x)$ or $\sim(\Box x)$.

Definition 3: $\bar{x} = p - x$, where the minus sign represents arithmetic subtraction.

This definition in algebraic terms is that of involution, and in logical terms is that of strong negation. There is a wide variety of mathematical terms for this definition, which was investigated early by Lukasiewicz. It may be thought of as a mirror type of reflection. Since the strong rule $\bar{\bar{x}} = x$ holds, this is referred to as a strong negation. In [1] this appears as $\beta(x)$.

For reviews leading to references other than those that follow concerning these mathematical terms, see the headings *Boolean algebras, generalizations of,* and *Many-valued logic,* in the "Index of Reviews by Subjects," *J. Symbolic Logic,* vol. 35, pp. 654–708, Dec. 1970.[2]

References

[1] G. Epstein, "The lattice theory of post algebras," *Trans. Amer. Math. Soc.*, vol. 95, pp. 300–317, May 1960.
[2] G. Epstein and A. Horn, "A propositional calculus for affirmation and negation with linearly ordered matrix," *J. Symbolic Logic* (Abstract), vol. 37, p. 439, June 1972.
[3] G. Epstein, "Multiple-valued signal processing with limiting," in *Conf. Rec., 1972 Symp. Theory and Applications of Multiple-Valued Logic Design*, pp. 33–45.

Manuscript received May 31, 1972; revised March 27, 1973.
The author is with ITT Gilfillan, Van Nuys, Calif., and with the Department of Mathematics, University of California, Los Angeles, Calif. 90024.
[1] S. Y. H. Su and A. A. Sarris, *IEEE Trans. Comput.*, vol. C-21, pp. 479–485, May 1972.
[2] Assistance in the selection of these headings was provided helpfully by H. Enderton of the University of California, Los Angeles.

On Efficient Computation of Matrix Chain Products

SADASHIVA S. GODBOLE

***Abstract*—It is pointed out that the number of scalar multiplications (additions) required to evaluate a matrix chain product depends on the sequence in which the associative law of matrix multiplication is applied. An algorithm is developed to find the optimum sequence that minimizes the number of scalar multiplications. A program is written for use on the CDC 6600 computer to implement this algorithm and also to carry out the chain product according to the optimum sequence. Several examples are included to illustrate the algorithm. The saving in computation and improvement in accuracy that can result from the use**

Manuscript received August 1, 1972; revised April 2, 1973.
The author is with Babcock & Wilcox Company, Lynchburg Research Center, Lynchburg, Va. 24505.

TABLE I

		A_1		A_2		A_3		A_4		A_5			
	n_0		n_1		n_2		n_3		n_4		n_5		
	5		10		15		8		4		10	Cost	Sequence
Step 1			1		0		0		0			750	n_1
			0		1		0		0			1200	n_2
			0		0		1		0			480	n_3
			0		0		0		1			320	n_4
Step 2			1		1		0		0			1350	n_1, n_2
			1		0		1		0			1230	n_1, n_3
			0		1		1		0			1080	n_3, n_2
			1		0		0		1			1070	n_1, n_4
			0		1		0		1			1520	n_2, n_4
			0		0		1		1			1080	n_3, n_4
Step 3			1		1		1		0			1280	n_3, n_2, n_1
			1		1		0		1			1670	n_1, n_2, n_4
			1		0		1		1			1830	n_1, n_3, n_4
			0		1		1		1			1480	n_3, n_2, n_4
Step 4			1		1		1		1			1480	n_3, n_2, n_1, n_4

of this algorithm can be quite significant for chain products of large arrays and in iterative solutions of matrix equations involving chain products.

Index Terms—**Associative law of matrix multiplication, dynamic programming, finite-word-length computation, matrix chain product, truth table.**

I. Introduction

This correspondence is addressed to the following questions arising in connection with the evaluation of matrix chain products of the form

$$\underset{n_0\times n_1}{A_1} \times \underset{n_1\times n_2}{A_2} \times \underset{n_2\times n_3}{A_3} \times\cdots\times \underset{n_k\times n_{k+1}}{A_{k+1}} . \qquad (1)$$

1) Does the sequence in which this product is evaluated affect the amount of total computation required?

2) If the answer to 1) is yes, which is the optimum sequence for evaluating such a product with a minimum computation?

As it turns out, the answer to 1) is yes, while the answer to 2) can be found by using the algorithm developed in the next section.

II. Algorithm

Suppose the product to be evaluated is

$$\underset{5\times 10}{A_1} \times \underset{10\times 10}{A_2} \times \underset{10\times 2}{A_3} .$$

If the product is computed as $(A_1(A_2A_3))$, 300 scalar multiplications (additions) are needed, while if the same is done as $((A_1A_2)A_3)$, 600 such operations are required. The following algorithm determines the most efficient way of computing a given product of the form (1).

Algorithm

The given chain product (1) is evaluated by k matrix multiplications. Each such operation $\left(\underset{n\times m}{P} \times \underset{m\times r}{Q}\right)$ may be considered as elimination of vertex m from the set $\{n, m, r\}$ by nmr scalar multiplications (additions), later referred to as the cost of eliminating m. Thus the problem can be restated as an equivalent problem of eliminating vertices n_1, $n_2, \cdots, n_k$ from the set $\{n_0, n_1, n_2, \cdots, n_k, n_{k+1}\}$ at minimum cost. The algorithm described next can be better understood by reading it in conjunction with Example 1.

The algorithm consists of generating a truth table for k variables (like Table I for $k = 4$) and then going through Steps 1 through k, the ith step, $1 \leqslant i \leqslant k$, being described as follows.

ith Step: The object of this step is to examine various ways of eliminating i vertices and to obtain the corresponding costs and sequences. This amounts to processing all the rows of the truth table having i 1's and $(k - i)$ 0's. More specifically, there are $\binom{k}{i} = k!/i!(k - i)!$ such rows whose processing involves mapping from the set of i 1's in a given row to the minimum cost and the sequence of vertex elimination (matrix multiples) that achieves this minimum cost. Another way of viewing this processing is optimally inserting i parenthesis pairs in the given matrix chain product. For example, row 4 in Step 3 of Table I corresponds to the three matrix multiplies indicated by the parentheses in $A_1((A_2(A_3A_4))A_5)$ and the cost (number of scalar multiplications) given by cost $= n_2n_3n_4 + n_1n_2n_4 + n_1n_4n_5 = 1480$.

The rows processed in this step are of two types, namely: Type 1, characterized by two or more groups of consecutive 1's separated by one or more 0's (e.g., row 2 of Step 3 in Table I) and Type 2, characterized by i consecutive 1's (e.g., row 4 of Step 3 in Table I). The processing of these two types of rows is described next.

Type 1: Such a row is decomposed into a minimum number of rows of Type 2 encountered in the earlier steps. The cost of the row under consideration is then calculated as the sum of the costs of the rows in the decomposition.[1] Likewise, the optimum sequence of the row under consideration is formed by adjoining the sequences of the rows in the decomposition. For example, the cost and optimum sequence of row 2 of Step 3 in Table I is based on the costs and optimum sequences of row 1 of Step 2 and row 4 of Step 1.

Type 2: One of the 1's in this row, say that corresponding to the vertex j, is replaced by a 0. The resulting row is one of the Type 1 rows, say r_j, processed in the previous step. Let the cost and sequence of row r_j be C_j and S_j, respectively. The cost of eliminating vertex j, after all the vertices in row r_j have been eliminated, is added to C_j to obtain a new cost $\mathcal{C}_j$. Similarly S_j is extended by adjoining j at the end to obtain a new sequence $\mathcal{S}_j$. $\mathcal{C}_j$ and $\mathcal{S}_j$ are possible candidates for the cost and sequence of the row under consideration. j is given all the possible values and the minimum $\mathcal{C}_j$ and $\mathcal{S}_j$ are assigned to the row being considered. For example, row 4 of Step 3 in Table I is processed by considering rows $3(r_{n_4})$, $5(r_{n_3})$, and $6(r_{n_2})$ of Step 2, calculating the corresponding $\mathcal{C}$'s and $\mathcal{S}$'s ($\mathcal{C}_{n_4} = 1480$, $\mathcal{S}_{n_4} = [n_3, n_2, n_4]$, $\mathcal{C}_{n_3} = 2320$, $\mathcal{S}_{n_3} = [n_2, n_4, n_3]$, $\mathcal{C}_{n_2} = 2580$, $\mathcal{S}_{n_2} = [n_3, n_4, n_2]$) and assigning the minimum cost $\mathcal{C}_{n_4}$ and $\mathcal{S}_{n_4}$ to row 4 of Step 3.

The cost and sequence found in the kth step is the desired information.

This algorithm represents an application of Bellman's method of dynamic programming in which the optimum procedure at Step i (the

[1] The cost of a row means the minimum cost of eliminating the vertices represented by 1's in the row.

ith matrix multiply) are calculated recursively from the optimum procedures for Steps $1, 2, \cdots, i-1$ $(i \geq 2)$ [1], [2].

III. Software

The above algorithm is programmed for use on the CDC 6600 computer. If desired, the program can also compute the chain product according to the optimum sequence. The program can handle chain products of up to 11 matrices whose combined total number of elements is about 5000 or less. It takes about 42 000 (octal) core to compile and about 38 000 (octal) core to execute. The CP time required for compiling is about 1.85 s, while that required to execute the case $k = 3$ and matrix sizes 5×10, 10×15, 15×8, 8×10 is about 2.1 s (includes computation of the product).

IV. Examples

The following examples illustrate the algorithm and bring out a few interesting facts noted in the next section.

Example 1: Suppose the following matrix product is to be computed.

$$\underset{5\times 10}{A_1} \times \underset{10\times 15}{A_2} \times \underset{15\times 8}{A_3} \times \underset{8\times 4}{A_4} \times \underset{4\times 10}{A_5}.$$

Table I shows the various steps of the algorithm. Thus the optimum sequence for evaluating the given product is $((A_1(A_2(A_3A_4)))A_5)$ and the corresponding (minimum) number of scalar multiplications (additions) is 1480. The computer solution is shown as follows.

```
DIMENSIONS OF MATRICES ENTERING THE PRODUCT

       N 0     N 1     N 2     N 3     N 4     N 5
       5       10      15      8       4       10

OPTIMUM ELIMINATION SEQUENCE: N 3,N 2,N 1,N 4
CORRESPONDING NO. OF SCALAR MULTIPLICATIONS=        1480
```

Example 2: Given the same matrices as in the above example (this will be assumed in the other examples also) and the desired product $A_1A_2A_3A_4$, the optimum sequence and cost are, respectively, $(A_1(A_2 \cdot (A_3A_4)))$ and 1280.

Example 3: If the desired product is $A_1A_2A_3$, the optimum sequence and cost are $((A_1A_2)A_3)$ and 1350, respectively.

Example 4: If the desired product is $A_1A_2A_3P$ where P is an 8×10 matrix, the optimum sequence and cost are $(((A_1A_2)A_3)P)$ and 1750, respectively.

V. Some Observations

Based on the preceding examples, the following interesting observations can be made.

Observation 1: The minimum cost of computing product (1) may be lesser than that of computing $A_1A_2 \cdots A_k$ (see Examples 2 and 3).

Observation 2: If the product (1) and its proper subproduct $A_iA_{i+1} \cdots A_j$ (call it P), $k+1 \geq j > i \geq 1$, are to be evaluated, it may sometimes be more efficient to compute the two separately rather than evaluating P and $A_1A_2 \cdots A_{i-1}PA_{j+1} \cdots A_{k+1}$. Suppose both $P = A_4A_5$ and $A_1A_2A_3A_4A_5$ are to be evaluated. The total cost based on evaluating P and $A_1A_2A_3P$ is $320 + 1750 = 2070$ (see Example 4), while that based on evaluating P and $A_1A_2A_3A_4A_5$ is $320 + 1480 = 1800$ (see Example 1).

VI. Conclusion

It has been pointed out that the number of scalar multiplications (additions) required to evaluate a matrix chain product like (1) depends on the sequence in which the associative law of matrix multiplication is applied. An algorithm is developed and programmed on the CDC 6600 computer to find out the optimum sequence of matrix multiplications. The program can also evaluate the product accordingly. In addition to computing products like (1) at minimum cost, the algorithm can give most accurate results in the case of finite-word-length computations. These advantages can be more significant for chain products of large arrays and in iterative solution of matrix equations involving chain products. The examples included illustrate the algorithm and bring out some interesting facts.

References

[1] R. Bellman, Ed., *Mathematical Optimization Techniques.* Berkeley, Calif.: Univ. California Press, 1963.
[2] S. E. Dreyfus, *Dynamic Programming and the Calculus of Variations.* New York: Academic, 1965.

A Simply Solvable Class of Simultaneous Linear Equations

JOSEPH C. HASSAB

Abstract—**A solution is presented for a class of a very large system of linear algebraic equations. The given set of simultaneous equations is initially transformed to yield an equivalent system of equations in triangular form whose solution is straightforward. With the present technique, the number of operations on the computer is minimized while the convergence to a unique solution is guaranteed.**

Index Terms—**Linear transformation, minimized computer operations, simply solvable equations, triangular matrices.**

Introduction

In this correspondence, a solution is presented for a class of a large system of simultaneous linear equations. This class of equations may arise from the algebraization of elliptic partial differential equations [1, p. 6] or from their conversion into equivalent integral equations and their subsequent numerical treatment [2, p. 89]. The obtained set of simultaneous equations is transformed into an equivalent triangular set where recursive calculation yields the solution.

Simply Solvable System of Equations

Suppose we wish to solve an $n \times n$ linear system that has been converted to the form [1], [2]:

$$x = Ax + y \tag{1}$$

where x and y are $n \times 1$ column matrices. If we can express the matrix A as the matrix sum

$$A = yC + D \tag{2}$$

where y, C, and D are given, C is a $1 \times n$ row matrix, and D is an upper triangular $n \times n$ matrix, then the algebraic equation (1) is shown to be simply solvable.

Letting $\gamma = Cx + 1$ and setting $x = \gamma Z$ in (1) and (2), we get $x = \gamma y + Dx$ or $Z = y + DZ$.

Since D is a triangular matrix, Z is determined recursively with only $n(n+1)/2$ operations. Solving for γ in terms of Z, we have $\gamma = C\gamma Z + 1$, or $\gamma = 1/(1 - CZ)$ and the unknown $x(x = \gamma Z)$ has been determined finally. If the Z system is not solvable or $CZ = 1$, then the method must be modified.[1] Otherwise, the original problem has been solved in about $n^2/2$ operations as contrasted [3, p. 245] with the usual $n^3/3$ operations used in Gauss elimination, $(2/3)n^3$ multiplications plus n square roots in Householder triangularization $(4/3)n^3$ multiplications plus $\frac{1}{2}n^2$ square roots in Givens triangularization, and n^2 operations for each iteration in a Gauss–Seidel solution.

Manuscript received May 31, 1972; revised April 2, 1973.
The author is with the Naval Underwater Systems Center, Newport, R. I.
[1] Pointed out by the referees.