On The Complexity of Matrix Multiplication

by

Robert L. Probert

Department of Applied Analysis &
Computer Science

CS-73-27

University of Waterloo
Waterloo, Ontario, Canada

ON THE COMPLEXITY OF MATRIX MULTIPLICATION

by

Robert L. Probert

A Thesis

Presented to the

University of Waterloo

in Partial Fulfillment of

the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in Mathematics

Department of Applied Analysis and Computer Science

Faculty of Mathematics

September, 1973

ABSTRACT

On The Complexity of Matrix Multiplication

In this thesis, we examine the difficulty of computing any set of bilinear forms, in particular the product of two matrices. This difficulty is expressed in terms of the number of multiplications required (multiplicative complexity) and the number of additions and subtractions required (additive complexity) by any algorithm which does not exploit the commutativity of multiplication of matrix elements We show that the multiplicative complexity of (m, n, p) products (matrix products of the form $A_{m \times n} B_{n \times p}$) is identical to the multiplicative complexity of (u, v, w) products where (u, v, w) **is one of the six permutations of (m, n, p) . Thus, $(2, n, 2)$ products require $\lceil 7n/2 \rceil$ multiplications, whereas $(2, 3, 3)$ products require 15 multiplications. A straightforward method is** presented for deriving algorithms for related computations from a given algorithm such that no more multiplications are employed. A lower bound is obtained on multiplicative complexity of (m, n, p) products, of $m_x(d_0 + d_1 - 1)$ multiplications, where $m_x = \max\{m, n, p\}$ and d_0, d_1 are the two other dimensions. Evidence is presented that multiplicative complexity decreases for fixed dimension product as the three dimensions approach equality.

In the second half of the thesis, the additive complexity of (m, n, p) products using a fixed set of or number of multiplications is studied. A simple graph-theoretic model is given which exactly represents the addition/subtraction steps used by bilinear matrix multiplication algorithms. Operations on the graph representation of an algorithm for (m, n, p) products produce the appropriate 5 related algorithms with all steps specified. Additive complexity is found to be greatest when the middle dimension is largest. The model enables solving for the additive complexity of a problem in terms of a discovered additive complexity of a related problem. By this additive symmetry, inner products and matrix-vector products require n-1 and mn-m additions or subtractions respectively. Additive symmetry also yields a proof of the exact additive complexity of (2, 2, 2) computations using 7 multiplications steps. The main result is that 15 additions/subtractions and 7 multiplications are necessary and sufficient to compute the product of matrices of order 2. Finally, we show that the graph representation model is useful for studying the additive complexity of linear schemes which compute a restricted class of linear forms.

ACKNOWLEDGEMENTS

The author is extremely grateful to his supervisor, Professor Patrick C. Fischer for his guidance, encouragement and many helpful suggestions, especially in Section 5.3 in the analysis of recursive implementations of α_W .

The author wishes to express his appreciation also to Professor Ian Munro for his reading of many illegible preliminary drafts, and for his continual encouragement towards further research. As well, the author's gratitude is due to Professor Charles Fiduccia for his constructive criticisms of some of the manuscript and for many stimulating conversations.

Thanks are particularly due to Mrs. Mary Wang who not only performed an outstanding job of decoding and typing the manuscript, but also looked after all the paper processing concomitant with distributing this thesis.

The criticisms and suggestions of the members of the Thesis Examination Committee, Professors A. Borodin, J. Edmunds, P. Fischer, I. Munro, and E. Robertson are also gratefully acknowledged.

Finally, I would like to thank my wife Lois for her patience and understanding when long hours at this work became increasingly necessary and increasingly long, and above all, for her constant encouragement and support.

Research for this work was aided considerably by field trips supported by National Research Council Grant A5549.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

The complexity of matrix multiplication is the subject of this thesis. In particular, we study the difficulty of computing the product of non-square matrices when using a member of a particular class of algorithms or schemes. The complexity or cost of a computation will be measured as either the number of multiplication "steps", or the total number of addition and subtraction "steps" used in the computation. We will be interested in searching for less costly algorithms and in finding exactly how many of each of multiplication "steps" and addition or subtraction "steps" are necessary. The major results involve complexity relationships between six related matrix multiplication problems. Given any one problem, the five related matrix multiplication problems are called the five symmetric problems of the given one. Thus, the theorem which explains the relationship among the exact numbers of multiplication "steps" used for symmetric problems is called the Multiplicative Symmetry Theorem. Similarly, the relationship between the total number of addition and subtraction "steps" used for symmetric problems is given by the Additive Symmetry Theorem.

In this introductory chapter, we outline the present state of research on the problem, state the scope and goals of this work, and give a brief list of notational conventions.

1.1 A Brief History

A comprehensive survey of the development of the study of arithmetic problems from a computational complexity viewpoint is presented in [K1]. We present here only those results which have immediate relevance to the study of the additive and multiplicative complexity of matrix multiplication. For more information see [B2], [K3] or [K4]

One of the first such results is contained in a 1954 paper by Ostrowski [O1].

Theorem 1.1.1: Any computation of $\sum_{i=0}^{n} r_i a_i$, where $r_0 \neq 0, \cdots , r_n \neq 0$, which employs only addition, subtraction, and multiplication steps of the form $M_i = u_i v_i$ where one of u_i, v_i is independent of the parameters $a_0, \cdots, a_n$ and the other factor is either, one of $\{a_0, \cdots, a_n\}$, or the result of computing a previous step, must use at least n additions or subtractions.

Computations which obey these restrictions are called totally linear computations. Subsequent research has been conducted by Kirkpatrick [K1] , Morgenstern [M1, M2, M3], and others into the problem of computing linear forms using essentially totally linear schemes. A linear form is basically a set of sums each of the form $\sum_{i=1}^{n} c_i x_i$ where the x_i's are indeterminates and the c_i's take on scalar values. In Chapter VI, we define the class of linear forms

L_{mn} of n sums in m indeterminates $x_1, \cdots, x_m$ and show that a simple computation graph representation of algorithms computing these linear forms captures the additive complexity of the associated computation. However, the bulk of the thesis is concerned with treating the complexity of matrix multiplication.

We refer to the computation of the product of two matrices $A_{m\times n}B_{n\times p} = Y_{m\times p}$ as the computation of an <u>(m, n, p) product</u> where the lefthand matrix is $(m\times n)$, and the righthand matrix is an $(n\times p)$ matrix. Then, the multiplicative complexity, <u>$\mathcal{M}(m, n, p)$</u> of computing (m, n, p) products is the smallest number of multiplications of the form $P_1 \cdot P_2$, where P_1, P_2 are possibly polynomials in the matrix elements $\{a_{ij} | a_{ij} \in A\} \cup \{b_{ij} | b_{ij} \in B\}$, which can be used to compute AB = Y (A, B, Y are matrix variables, not particular matrices).

In general, an algorithm α for (m, n, p) product will not be multiplicatively optimal. We denote the set of t multiplications used by any algorithm α for (m, n, p) products by $M = \{M_1, \cdots, M_t\}$. Then, the additive complexity of computing an (m, n, p) product by means of a fixed set M of t multiplications is denoted <u>$\mathcal{A}(m, n, p, M)$</u> . Then, the smallest possible number of addition/subtraction steps which form and combine a set of t multiplications to compute (m, n, p) products is denoted <u>$\mathcal{A}(m, n, p, t)$</u> .

For example, (m, n, p) products are computed by the classical algorithm in mnp multiplication steps and mp(n - 1) additions/subtractions. Thus, $\mathcal{M}(m, n, p) \leq mnp$ and $\mathcal{A}(m, n, p, t) \leq mp(n - 1)$ where $t = mnp$.

One of the first results on the multiplicative complexity of matrix multiplication was presented by Winograd [W1, W2].

An arithmetic scheme for computing (m, n, p) products is the sequence of operations $f_i = f_j$ 'op' f_k where 'op' is one of the four binary operations, '+', '-', '×', '÷' . As well, $j,k < i$ and each f is an indeterminate, a scalar value, or the result of an operation which appears earlier in the sequence. Finally, each element of the (m×p) product matrix appears as some f_i .

By employing an argument based on the number of independent columns in the matrix X , Winograd proved [W2]:

Theorem 1.1.2: Any arithmetic scheme for computing the matrix-vector product $X_{m\times n}y_{n\times 1}$ for general X and y requires mn multiplications/divisions. In our notation, $\mathcal{M}(m, n, 1) \geq mn$.

Thus, the classical method of computing the product of a matrix by a vector is an optimal arithmetic scheme with respect to the number of multiplications/divisions used.

The question arose whether multiplying matrices of order n could be accomplished using fewer multiplications then employed by the classical algorithm (which uses n^3 multiplications).

Winograd [W2] found an algorithm which computes (n, n, n) products in approximately half the multiplicative cost of the standard algorithm and about twice the additive cost:

Form each inner product $p = \sum_{i=1}^{n} a_i b_i$ as

$p = \sum_{i=1}^{n/2} (a_{2i} + b_{2i-1})(a_{2i-1} + b_{2i}) - a_{2i}a_{2i-1} - b_{2i}b_{2i-1}$. By inspection, this computes (n, n, n) products using $\frac{n^3}{2} + n^2$ multiplications and $2n^3$ additions/subtractions. Note that the improvement in multiplicative cost is gained at the expense of an increase in the total number of arithmetics used.

We further refine the class of algorithms being studied by not allowing the operation of division. Such algorithms are called polynomial algorithms and all algorithms will be understood to be polynomial algorithms for the remainder of the thesis unless otherwise stated.

At this stage, Winograd and Strassen observed that results on the multiplicative complexity of a particular matrix multiplication problem could be used to yield upper bounds on the multiplications required to compute general (n, n, n) products [W2, S1].

Theorem 1.1.3: If for some k there is an algorithm which computes (k, k, k) products in ℓ multiplications without exploiting the commutativity of multiplications of matrix elements (which may then themselves be matrices), (n, n, n) products can be computed in $\mathcal{O}\left(n^{\log_k(\ell)}\right)$ total arithmetic operations.

As a corollary, Winograd proved that if $\mathcal{M}(k, k, k) < \frac{k^3}{2}$, then there is some $d < 3$ such that (n, n, n) products can be computed in $\mathcal{O}(n^d)$ arithmetics. Strassen [S1] discovered such a d by finding a 7-multiplication algorithm for $(2, 2, 2)$ products which does not exploit commutativity (we denote this algorithm by α_S); thus, one such d is $\log 7 \approx 2.81$ (all logorithms in this thesis and in most of the literature on algorithm analysis are taken to base 2).

By analysis of his algorithm Strassen also showed [S1]

<u>Theorem 1.1.4</u>: Fewer than $4.7\ n^{\log 7}$ total arithmetic operations are required to compute (n, n, n) products.

If we examine Strassen's algorithm α_S we will observe that all multiplications are of the form $\mathcal{L}\{a_{ij}\} \cdot \mathcal{L}\{b_{ij}\}$. Thus, α_S does not depend on the commutativity of multiplication of matrix elements. In Chapter II we define a class NC of such algorithms; algorithms outside this class are not studied at all in this thesis. Winograd's corollary to Theorem 1.1.3 justifies this position in the following sense: for any scheme at all which computes (n, n, n) products in t multiplication steps, there exists an algorithm in NC for (n, n, n) products using approximately $2t$ multiplication operations. In asymptotic terms, this difference is insignificant. As well, the added restriction on structure seems vital for proving non-trivial lower bounds.

Hopcroft and Kerr [H1] obtained $\mathcal{M}(2, 2, 2) = 7$ by proving (for algorithms in NC)

<u>Theorem 1.1.5</u>: $\mathcal{M}(m, 2, n) \leq \lceil (3mn + \max\{m, n\})/2 \rceil$; $\mathcal{M}(2, 2, n) = \lceil 7n/2 \rceil$; $\mathcal{M}(3, 2, 3) = 15$.

We illustrate the general method of Hopcroft and Kerr for (m, 2, n) products in Section 3.1 where a 26-multiplication algorithm is given for (4, 2, 4) products.

Thus, α_S is multiplicatively optimal for (2, 2, 2) products over all algorithms in NC . For this particular problem, Winograd [W3] has shown that even algorithms designed to exploit commutativity must use at least 7 multiplication steps.

1.2 Bilinear Chains and Fast Algorithms

In [F1] C. Fiduccia catalogues multiplicatively fast algorithms for computing the products of matrices which have special forms, and conjectures $\mathcal{M}(n, n, n) = 2n^2 + \mathcal{O}(n)$ multiplications. In the smae paper, Fiduccia points out that the number of independent rows should be considered as well as columns to obtain lower bounds on the multiplicative complexity of matrix-vector multiplication. In [F2], Fiduccia defines bilinear chains, a subclass of algorithms in NC . Multiplication steps in a K-bilinear chain which computes $A_{m\times n}B_{n\times p}$ are of the form $\mathcal{L}_K\{a_{ij}\}\cdot\mathcal{L}_K\{b_{ij}\}$. $\mathcal{L}_K\{x_i\}$ is merely a linear combination over K of elements x_i . Then, any (m, n, p) product can be encoded as a matrix-vector product for complexity analysis purposes. Fiduccia proves [F2]

Theorem 1.2.1: One can compute Xy (matrix-vector products) with t multiplications by a K-bilinear chain if and only if X - D = CUB where B, C, D are fixed matrices and U is a t×t diagonal matrix with entries in $\mathcal{L}_K\{x_{ij}\}$.

Thus, searching for faster bilinear chains for (m, n, p) products is equivalent to encoding the problem as a matrix-vector product Xy and then decomposing X into a product of 3 simple matrices. Fiduccia also points out that we can equivalently decompose X into the sum of t matrices of rank 1 and verify such decompositions by inspection, thus avoiding considering products at all.

By showing that NC contains no faster algorithms than those which are essentially bilinear chains, and applying Theorem 1.2.1, we show in Chapters II and III (and also in [P1])

Multiplicative Symmetry Theorem (Theorem 2.3.4): Over the class of algorithms in NC, if (u, v, w) is any permutation of (m, n, p), $\mathcal{M}(u, v, w) = \mathcal{M}(m, n, p)$. As well, given any algorithm in NC which computes (m, n, p) products using t multiplications, we can construct an algorithm in NC for (u, v, w) products using no more than t multiplications.

At about the same time that this theorem was derived [P1], Hopcroft and Musinski [H2] and Dobkin [D1], using different techniques, independently discovered and proved essentially the same result.

As a corollary of the Multiplicative Symmetry Theorem and Theorems 1.1.5 and 1.1.2, we have

Corollary 1.2.2: $\mathcal{M}(m, 1, n) = \mathcal{M}(1, m, n) = mn$.

$$\mathcal{M}(2, n, 2) = \mathcal{M}(n, 2, 2) = \lceil 7n/2 \rceil .$$

$$\mathcal{M}(3, 3, 2) = \mathcal{M}(2, 3, 3) = 15 .$$

Kirkpatrick [K2] showed using independence arguments that $\mathcal{M}(m, n, p) \geq m(n + p - 1)$. Thus, we have

Corollary 1.2.3: $\mathcal{M}(m, n, p) \geq m_x (d_0 + d_1 - 1)$ where $m_x = \max \{m, n, p\}$, d_0, d_1 are the other 2 dimensions. The best lower bound known for $\mathcal{M}(m, n, p)$ is given by Dobkin [D1]:

Theorem 1.2.4: $\mathcal{M}(m, n, p) \geq mn + mp + np - (m+n+p) + 1$. Thus, $\mathcal{M}(n, n, n) \geq 3n^2 - 3n + 1$ is the best lower bound known on the multiplicative complexity of general matrix multiplication.

The state of the art regarding the additive complexity of matrix multiplication is even less advanced. Winograd [W3] has shown

Theorem 1.2.5: Any polynomial scheme to compute $(m, n, 1)$ (matrix-vector) products must employ at least $m(n - 1)$ additions/ subtractions. Thus, the classical algorithm is additively optimal.

In Chapter IV we give a graph-theoretic representation of addition/subtraction steps used by bilinear chains to compute (m, n, p) products. We show that the additive complexities of the 6 related problems are fixed once a set of multiplication is chosen for any one problem. In Chapter V, this intimate additive relationship is employed to yield the following theorem:

Theorem 1.2.6: $\mathcal{A}(2, 2, 2, 7) = 15$, i.e. bilinear chains which compute $(2, 2, 2)$ products using only 7 multiplications must use 15 addition/subtraction steps, and this bound is achievable.

In section 5.1, Theorem 1.2.5 in proved as an immediate corollary of the results in Chapter IV on additive symmetry.

Finally, the best lower bound known on the additive complexity of general (m, n, p) products is due to Kirkpatrick [K1] and is $(m + p - 1)(n - 1)$ by independence arguments. We make a strong case in this thesis for referring to additive complexity of (m, n, p) products only when the number of multiplications which can be used is a parameter of the problem as well as m, n, p .

1.3 Notational Conventions

The following table lists in approximately alphabetical order each symbol used followed by a brief description of its meaning and the number of the section(s) in which the symbol is defined. For a proper definition of the symbol and its uses, the reader should refer to the designated section.

Symbol	Meaning	Defined in Sec.
$A, B, C, \cdots$	matrix variables	
$a_{ij}, b_{ij}, c_{ij}, \cdots$	matrix element variables	
$\mathcal{A}(m, n, p, M)$	the smallest possible number of addition/subtraction steps which can be used to compute (m, n, p) products using the multiplications given in the set M	1.1
$\mathcal{A}(m, n, p, t)$	the smallest possible number of addition/subtraction steps which can be used to compute (m, n, p) products using a set of t multiplications	1.1
α	an algorithm for computing (m, n, p) matrix products	
α^*	a symmetric algorithm of α, i.e. if α computes (m, n, p) products, α^* computes (n, m, p) products	4.2
$\hat{\alpha}, \alpha', \alpha''$	this notation merely signifies that $\hat{\alpha}, \alpha', \alpha''$ were derived somehow (usually by the methods of 3.1) from α	
α_S	the algorithm discovered by Strassen to compute $(2, 2, 2)$ products	4.1

Symbol	Meaning	Defined in Sec.
α_W	the algorithm discovered by Winograd to compute (2, 2, 2) products	5.3
α_H	the algorithm discovered by Hopcroft and Kerr to compute (m, 2, n) products	6.3
γ_i	an edge in graph G_i	
ΓG_i	the set of all edges of G_i	
$\mathcal{C}_i$	the connection matrix for the component G_i of an addition flow representation F^i	4.2
$\mathfrak{c}_{jk}$	the element in the j-th row and k-th column of $\mathcal{C}$, a connection matrix for some graph G . $\mathfrak{c}_{jk}$ = the coefficient of the term represented by the k-th source vertex in the sum represented by the j-th sink vertex of G .	4.2
$\lceil x \rceil$	ceiling of x : the smallest integer $\geq$ x	
E	E(A) is the set of elements which form the entries of the matrix A	2.1
$\overset{E}{=}$	$A \overset{E}{=} B$ where A, B are matrices if and only if E(A) = E(B)	2.1
F_α	the unique addition flow representation of the algorithm α .	4.1
F*	the symmetric representation of F . If $F = \langle G_1, G_2, G_3 \rangle$, $F^* = \langle G_1^T, G_3^D, G_2^D \rangle$	4.2
F^T	the transposed representation of F . If $F = \langle G_1, G_2, G_3 \rangle$, $F^T = \langle G_2^T, G_1^T, G_3^T \rangle$	4.3

Symbol	Meaning	Defined in Sec.
F_S, F_W	the addition flow representations of α_S, α_W respectively	
$\lfloor x \rfloor$	floor of x : the largest integer $\leq x$	
G_i	the i-th component in an addition flow representation $F = <G_1, G_2, G_3>$, an acyclic, directed, multi-sink, multi-source flow graph	4.1
G_α	the addition flow graph for α , a linear algorithm which computes linear forms	6.2
G_i^T	the graph G_i with its source nodes relabelled such that each label a_{ij} is replaced by the label a_{ji} . If $i = 3$, the sink nodes are relabelled as above.	4.1, 4.3
G^D	the directional dual of G , i.e., G with all edge directions reversed	4.1
$H(\gamma)$	the head of the directed edge γ	4.2
i,j,k	subscript variables	
ℓ	a linear form, i.e., a set of distinct, non-trivial sums	6.2
ℓ_S	the linear form of the 4 final sums of elements in M_S which α_S computes	6.2
ℓ_W	the 4 final sums of multiplications in M_W which α_W computes	6.2
λ,μ	isomorphisms which label each vertex in any addition flow representation F	4.1
L_{mn}	the class of all linear forms of n sums over m indeterminates	6.2
$\mathcal{L}_K\{a_{ij}\}$	the set of all combinations $q_{ij}a_{ij}$ where $q_{ij} \in K$	2.1
log x	logorithm to the base 2 of x	

Symbol	Meaning	Defined in Sec.
(m, n, p) product	a matrix product of the form $A_{m\times n}$ times $B_{n\times p}$	
$\mathcal{M}(m, n, p)$	the smallest number of multiplication steps which can be used by an algorithm in NC to compute (m, n, p) products	
M_i	the i-th multiplication step of an algorithm	2.1
M_α	the set of multiplications used by α	2.1
M_S	the set of multiplications used by α_S	4.1
M_W	the multiplication set used by α_W	5.3
M_H	the set of multiplications used by α_H to compute (3, 2, 3) products	6.3
M^L, M^R	the set of lefthand and righthand factors respectively, of the multiplications in M	
m_x, d_0, d_1	the largest, the two remaining ones respectively of the three matrix product dimensions m, n, p	3.2
NC	the class of all algorithms α which compute (m, n, p) products without exploiting commutativity of matrix multiplication	1.1, 2.1
P	the set of all paths from a particular source vertex to a chosen sink vertex in a flow graph	4.2
p	one such path in P	
ϕ	a chain of primitive instructions each of which is a binary operation other than division, or a scalar multiplication	2.1

Symbol	Meaning	Defined in Sec.
ϕ_i	the i-th step in the algorithm	2.1
Q	the field of rationals	
R_i	the source set of G_i	4.1
S_i	the set of sink vertices in G_i	4.1
t	the number of multiplications used by a matrix multiplication algorithm	
τ	the group of all transformation which act on matrix multiplication algorithms to give new algorithms, generated by four types of elementary row and column operations	5.2
T	some transformation in τ	5.2
$T(\gamma)$	the tail of the directed edge γ	4.2
v	a vertex in a graph	
VG	the set of vertices in a graph	
w_γ	the weight assigned to the edge γ in flow graph G	4.1, 4.2
X, Y, Z	matrix variables	
x, y, z	vector variables	

CHAPTER II

THE MULTIPLICATIVE SYMMETRY THEOREM[1]

In this chapter we introduce two classes of matrix multiplication algorithms and demonstrate their "equivalence". This equivalence is utilized to obtain the main result, the Multiplicative Symmetry Theorem which states that the multiplicative complexity of (m, n, p), (n, m, p), (p, m, n), (m, p, n), (n, p, m) and (p, n, m) matrix products is identical. As immediate corollaries, we can extend various known non-trivial achievable lower bounds on the multiplicative complexity of matrix multiplication problems.

2.1 Algorithms and Bilinear Chains

Basically, there are two possible approaches for proving lower bounds on multiplicative complexity depending on the definition of the class of algorithms under discussion. Typical of the first approach is that of Winograd [W1] in which an algorithm α is defined as follows:

1 This chapter was presented at the Symposium on Complexity of Sequential and Parallel Numerical Algorithms, May 16-18, 1973 at Carnegie-Mellon University, Pittsburgh, Pennsylvania, U.S.A.

Let the number of steps in α be denoted by N and label the steps $1,2,\cdots,N$. Let Q represent any field (for example, the field of rational numbers).

Denote by $e_\alpha(j)$ the <u>evaluation</u> of α at the j-th step, i.e. the expression computed by α at step j.

An <u>algorithm</u> α is defined as follows: Either $e_\alpha(j) \in Q \cup \{a_{11},\cdots,a_{mn},b_{11},\cdots,b_{np}\}$ where the a_{ij}'s and b_{ij}'s are indeterminates, or $e_\alpha(j) = e\ (j_1)$ 'op' $e_\alpha(j_2)$ where $j_1,j_2 < j$ and 'op' is either '+', '-', or '×' ; division is not allowed. Then, α is said to compute the product matrix $Y_{m\times p}$ of $A_{m\times n}B_{n\times p}$ if $\exists j_1,j_2,\cdots,j_{mp}$ such that $e_\alpha(j_k) = y_{rs}$ where $k = (r-1)p + s$.

In other words, each evaluation consists of a scalar from Q, an indeterminate, or the sum, difference, or product of earlier evaluations. The algorithm computes the right answer if each element in the product matrix appears at some steps as an evaluation. Lower bounds on multiplicative complexity are obtained by using independence arguments as in [W1], [W3], [F1].

Note that this definition of a computation permits intermediate evaluations of the form $P_1(a_{ij}, b_{ij})\cdot P_2(a_{ij}, b_{ij})$ where P_1,P_2 are polynomials of arbitrary order in the indeterminates with rational coefficients. Obviously, such evaluations are capable of exploiting the commutativity of the multiplication of matrix elements, i.e. $a_{ij}b_{k\ell} = b_{k\ell}a_{ij}$ for all $1 \le i \le m$, $1 \le j, k \le n$, $1 \le \ell \le p$.

Conceptually, we may think of the indeterminates as taking on transcendental values; thus, since multiplication occurs over the real field, it is a commutative operation.

Since in this thesis, however, we consider only those algorithms for matrix multiplication which do not exploit commutativity of multiplication, we modify Winograd's definition of algorithm and consider only algorithms in the class NC defined as follows:

An algorithm α is in the class NC (for Not Commutativity - exploiting) if and only if α computes product elements without assuming that intermediate product terms $a_{ij}b_{k\ell}$ commute. Thus for each y_{rs}, all terms $a_{r\ell}b_{\ell s}$ are computed in the sequence of evaluations which α uses to compute y_{rs} (terms of the form $b_{ij}a_{k\ell}$ may appear as well; however, they must cancel out in computations of the y_{rs}'s).

Unless otherwise stated, we will use "the multiplicative complexity of a computation" to refer to the minimum number of multiplication steps required to perform the computation by an algorithm in NC . As well, it is understood that multiplication of an evaluation by a scalar from Q is not to be considered a multiplication step; for example, $e_{\alpha}(k) = r_{ij}e_{\alpha}(\ell)$ where $\ell < k$ is not counted in determining the multiplicative complexity of an algorithm α if $r_{ij} \in Q$.

Example: The following algorithms both compute the matrix product of two (3×3) matrices "faster" than the usual brute-force method which uses $3^3 = 27$ multiplications. The first method belongs to NC , the second, a typical inner-product algorithm, presumes commutativity of multiplication of matrix elements and hence is not in NC .

Method 1: To compute $A_{3\times 3}B_{3\times 3}$, proceed as follows.

Let
$$A_1 = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}, \quad B_1 = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix},$$

$$A_2 = (a_{13} \; a_{23} \; a_{33})^T, \quad B_2 = (b_{31} \; b_{32} \; b_{33}) .$$

Then, use the algorithm given in [H1] to compute A_1B_1 in 15 multiplications (which is optimal). Compute A_2B_2 in 9 multiplications (also optimal) and set $AB = A_1B_1 + A_2B_2$ at a total cost of 24 multiplications.

Method 2: The well-known "fast" inner product algorithm of Winograd ([W1], [W4]) uses only three multiplications fewer than brute force. For example, if $C_{3\times 3} = AB$, compute c_{ij} , $1 \leq i, j \leq 3$, as $(a_{i1} + b_{2j})(a_{i2} + b_{1j}) - a_{i1}a_{i2} - b_{1j}b_{2j} + a_{i3}b_{3j}$. Each c_{ij} requires 2 unique multiplications, namely $(a_{i1} + b_{2j})(a_{i2} + b_{1j})$ and $a_{i3}b_{3j}$. There are three multiplications each of the form $a_{i1}a_{i2}$, and $b_{1j}b_{2j}$. Therefore, the total number of multiplications in Method 2 is $(9 \times 2) + 3 + 3 = 24$ multiplications. Thus, exploiting commutativity in this way affords us no saving over Method 1, an algorithm in NC .

In fact, no algorithm for multiplying 3×3 matrices has been found, including algorithms which exploit commutativity, which employs fewer than 24 multiplication steps!

The second approach to the problem is due to Fiduccia [F2] and characterizes a matrix multiplication algorithm α as a <u>bilinear chain</u> ϕ .

We will often refer to the matrix-vector product which corresponds to a given matrix multiplication problem. In order to make precise this correspondence, we define:

the <u>tensor product</u> $A_{m\times n} \otimes B_{r\times s}$ as $C_{\underline{mr}\times\underline{ns}}$ where

$$c_{i+k,j+p} = a_{ij}b_{k+1,p+1} ,$$

and the <u>direct sum</u> $A_{m\times n} \oplus B_{r\times s}$ as $C_{(m+r)\times(n+s)} = \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$.

Then, if I_s stands for the identity matrix with s 1's , we have $I_s \otimes A = A \oplus \cdots \oplus A$, s times. Finally, if $B_{r\times s}$ has the s columns $b_1,\cdots,b_s$, define $\kappa(B)$ as

$$\kappa(B) = [b_1^T,\cdots,b_s^T]^T \text{ , i.e. an } rs\text{-column vector.}$$

<u>Lemma 2.1.1</u>(Fiduccia): If B has s columns, let $d = \kappa(B)$, $C = I_s \otimes A$. Then, the set of entries of AB is identical to the set of entries of Cd .

Essentially, this lemma has reduced the matrix multiplication problem to a matrix-vector multiplication problem. Since this idea of entry equivalence is central, denote the set of elements of the matrix result AB by E(AB) . Define $AB \overset{E}{=} Cd$ iff $E(AB) = E(Cd)$.

Let R be a ring with a unit and let K be a subring of the centre of R such that $ar = ra$ for all (a, r) in $K \times R$. Let $X = (x_{ij})$ be a matrix variable which ranges over a non-empty subset S of $R^{m\times n}$ and $y = (y_1, \cdots, y_n)^T$ be a vector variable over $R^{n\times 1} = R^n$. α is an <u>algorithm</u> for the matrix-vector product Xy if α computes E(Xy) from $E(X) \cup E(y)$ for any pair (X, y) in $S \times R^n$. Define $L_K(E(X))$ as the set of all linear combinations $\sum_{i=1}^{m\cdot n} w_i x_i$ of E(X) (Here, X is $(m\times n)$, with fixed w_i's in K .)

A <u>K-chain</u> ϕ for E(Xy) is a finite sequence $\phi_1, \cdots$ such that for each $z \in E(Xy)$ there is a p such that $\phi_p = z$ where each ϕ_k is either in $E(X) \cup E(y)$ or

$\phi_k = r\phi_i$ where $r \in K$, or

$\phi_k = \phi_i$ 'op' ϕ_j for $i,j < k$ and 'op' $\in$ {'+', '-', '×'} .

The chain ϕ is <u>K-bilinear</u> iff whenever $\phi_k = \phi_i \times \phi_j$, $\phi_i \in L_K(E(X))$ and $\phi_j \in L_K(E(y))$.

In this chapter and the next, we will study arbitrary algorithms in NC . However, in studying the additive complexity of computing a set of bilinear forms (Chapters IV, V and VI), we will make various restrictions upon the algorithms under consideration. Unless otherwise stated, all these algorithms will be K-bilinear chains for arbitrary K . However, the lower bounds on $\mathcal{Q}(2, 2, 2, 7)$ are proved for K-bilinear chains where $K = \{0, 1, -1\}$ and then extended to hold for K-bilinear chains where $K \subseteq Z$, the ring of integers.

2.2 Equivalence of Bilinear Chains and NC-Algorithms

It can be seen that NC properly contains the class of bilinear chains. First, every bilinear chain is an algorithm which does not exploit commutativity and therefore is in NC . Also, algorithms in NC may compute polynomials of arbitrary degree in the indeterminates whereas this is not possible for bilinear chains. However, we are able to demonstrate that this extra capability of algorithms in NC provides no saving in multiplicative complexity for the matrix multiplication problem. Thus, for our purposes, the class of bilinear chains and NC are computationally equivalent.

The following two lemmas are straightforward modifications of results for algorithms which exploit commutativity given in [W2].

Lemma 2.2.1: If α is any algorithm in NC which computes (m, n, p) matrix products and $M_1,\cdots,M_t$ are the results of the multiplications in α , then the partial result computed at the jth step, $e_\alpha(j)$, is of the following form:

$$q + \sum_{i=1}^{t} q_i M_i + \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} a_{ij} + \sum_{i=1}^{n} \sum_{j=1}^{p} s_{ij} b_{ij}$$

where q and the q_i's , r_{ij}'s , and s_{ij}'s are all in Q .

Proof: Obviously, $e_\alpha(1) \in Q \cup \{a_{11},\cdots,b_{np}\}$ and therefore is of the required form. Suppose for all steps $j < \ell$, $e_\alpha(j)$ is of the required form. If $e_\alpha(\ell)$ is a multiplication, then $e_\alpha(\ell) = M_k$ for some k . Otherwise, $e_\alpha(\ell) = e_\alpha(j_1) \pm e_\alpha(j_2)$ for

$j_1, j_2 < \ell$, i.e.

$$e_\alpha(\ell) = (q^{(1)} \pm q^{(2)}) + \sum_{i=1}^{t} (q_i^{(1)} \pm q_i^{(2)}) M_i$$
$$+ \sum_{i=1}^{m} \sum_{j=1}^{n} (r_{ij}^{(1)} \pm r_{ij}^{(2)}) a_{ij} + \sum_{i=1}^{n} \sum_{j=1}^{p} (s_{ij}^{(1)} \pm s_{ij}^{(2)}) b_{ij}$$

which is of the required form.

Lemma 2.2.2: Let α be an algorithm in NC which computes (m, n, p) products, and t multiplication steps. Then, there exists an algorithm α' with no more than t multiplication steps, such that each multiplication is of the form:

$$\left(\sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} a_{ij}\right)\left(\sum_{i=1}^{n} \sum_{j=1}^{p} s_{ij} b_{ij}\right)$$

and such that α' computes the same matrix product. (Recall that the number of multiplications in an algorithm does not count multiplication by a scalar from Q , e.g. $r_{ij} a_{ij}$ is not counted as a multiplication if r_{ij} is in Q .)

Proof: Let the functions L_i , i = 0,1,2,3, of polynomials in the a_{ij}'s and b_{ij}'s be defined as follows:

$$L_i : Q[a_{11}, \cdots, a_{mn}, b_{11}, \cdots, b_{np}] \text{ into } Q[a_{11}, \cdots, b_{np}]$$

such that if $u \in Q[a_{11},\cdots,b_{np}]$, then $L_0(u)$ = the constant term of u ,

$L_1^A(u)$ = the term in u which is linear in the a_{ij}'s , i.e.

$$\sum_{i=1}^{m}\sum_{j=1}^{n} q_{ij}a_{ij} \quad ,$$

$L_1^B(u)$ = the term in u which is linear in the b_{ij}'s ,

$L_2(u)$ = the term in u of the form

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{\ell=1}^{p} s_{ijk\ell}a_{ij}b_{k\ell} \quad ,$$

and $L_3(u)$ what remains, i.e.

$$L_3(u) \;=\; u - L_0(u) - L_1^A(u) - L_1^B(u) - L_2(u) \quad .$$

Suppose $e_\alpha(j) = e_\alpha(j_1)\cdot e_\alpha(j_2) = u\cdot v$.

$$\begin{aligned} e_\alpha(j) \;&=\; L_0(u)\cdot L_0(v) + L_0(u)(v - L_0(v)) \\ &\quad + (u - L_0(u))L_0(v) + (u - L_0(u))(v - L_0(v)) \quad . \end{aligned}$$

The only multiplication which is counted is the final one. Thus, we can construct a new algorithm α_0 such that α_0 has the same number of multiplications as α , computes the same product, and if $e_{\alpha_0}(j) = u\cdot v$, then $L_0(u) = L_0(v) = 0$. Therefore, if $e_{\alpha_0}(j) = u\cdot v$, then $L_2(u\cdot v) = L_1^A(u)\cdot L_1^B(v)$.

By Lemma 2.2.1, if $e_{\alpha_0}(j_0) = \sum_{\ell=1}^{n} a_{i\ell} b_{\ell j}$, (1)

then, $e_{\alpha_0}(j_0) = q_0 + \sum_{i=1}^{t} q_i M_i + \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} a_{ij} + \sum_{i=1}^{n} \sum_{j=1}^{p} s_{ij} b_{ij}$ (2)

Applying L_2 to (1) and (2), we have $\sum_{\ell=1}^{n} a_{i\ell} b_{\ell j} = \sum_{i=1}^{k} q_i L_2(M_i)$.

Then, set $M_i = u_i \cdot v_i$. Thus,

$$\sum_{\ell=1}^{n} a_{i\ell} b_{\ell j} = \sum_{i=1}^{t} q_i L_2(u_i \cdot v_i) = \sum_{i=1}^{t} q_i L_1^A(u_i) \cdot L_1^B(v_i) \quad .$$

Note that $q = (q_1, \cdots, q_t)$ is a function of i and j .

Hence, to compute y_{ij} for $1 \leq i \leq m$, $1 \leq i \leq p$, an algorithm α' suffices which computes the t multiplications $L_1^A(u_i) \cdot L_1^B(v_i)$. Note that no multiplications are required to compute $L_1^A(u_i)$ and $L_1^B(v_i)$, and since $\alpha \in NC$ and α contains $u_i \cdot v_i$, $L_1^A(u_i) = \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} a_{ij}$ and $L_1^B(v_i) = \sum_{i=1}^{n} \sum_{j=1}^{p} s_{ij} b_{ij}$. Thus, the algorithm α' has only multiplications of the required form and computes the same matrix product as α .

The above lemma is a formal proof of the intuitively obvious observation made in [H1], namely that since constant terms and terms higher than quadratic do not appear in the product to be computed, these terms need not appear in intermediate calculations. Thus, an equivalent algorithm exists with the same number of multiplications whose multiplication steps are the product of a linear sum of a_{ij}'s with a linear sum of b_{ij}'s .

But this algorithm satisfies the definition of a bilinear chain. Thus, we have

<u>Lemma 2.2.3</u>: For any algorithm $\alpha \in NC$ which computes $A_{m\times n}B_{n\times p}$ using only t multiplication steps, there exists a bilinear chain which computes $E(Xy)$ where $X = I_p \otimes A$ and $y = (b_1^T,\cdots,b_p^T)^T$ using no more than t multiplications, and conversely.

Hence, the two distinct models of computations are equivalent for the purpose of studying multiplicative complexity of matrix multiplication.

In the next section, we exploit this equivalence to prove the invariance of multiplicative complexity over symmetric computations.

2.3 The Multiplicative Symmetry Theorem

To prove the main result that the 6 related matrix products are equally multiplicatively complex, we use a theorem of Fiduccia [F2]:

Theorem 2.3.1: There is a K-bilinear chain for $E(Xy)$ with t multiplication steps iff there is a K-bilinear chain for $E(X^T z)$, where z ranges over R^m, with t multiplication steps.

The consequences of this result are illustrated in Chapter III; for now, we need only the statement of the theorem to prove a result about symmetry in NC .

Lemma 2.3.2: Matrix products of the form $A_{m\times n}B_{n\times p}$ require t multiplications by algorithms in NC iff products of the form $C_{n\times m}D_{m\times p}$ require t multiplications to be computed by an algorithm in NC .

Proof: By Lemma 2.2.3, there is an algorithm $\alpha \in$ NC which computes $A_{m\times n}B_{n\times p}$ in t multiplications iff there is a bilinear chain using only t multiplications which computes $E(Xy)$, where Xy is the corresponding matrix-vector product.

By Theorem 2.3.1, this chain exists iff there is a bilinear chain for $E(X^T z)$ with t multiplication steps where z ranges over R^m . Note that X^T is of the form $I_p \otimes C_{n\times m}$ and z is of the form $(d_1^T, \cdots, d_p^T)^T$.

Again by Lemma 2.2.3, this chain exists iff there is an algorithm α' NC which computes $E(C_{n\times m}D_{m\times p})$ where C,D are matrix variables (as are A, B, X) with exactly t multiplication steps.

Hence, (m, n, p) products require the same number of multiplications as (n, m, p) products.

<u>Lemma 2.3.3</u>: For any algorithm α with t multiplication steps which computes matrix products of the form $A_{m\times n}B_{n\times p}$, there exists an algorithm α' which computes matrix products of the form $C_{p\times n}D_{n\times m}$ using the same number of multiplication steps.

<u>Proof</u>: Note that simply computing $C_{p\times n}D_{n\times m}$ as $(D^TC^T)^T$ via algorithm α will not work since we are not allowed to assume element multiplication is commutative; element products $d_{ji}c_{\ell k}$ do not necessarily equal $c_{\ell k}d_{ji}$. What we can do, however, is to collect the multiplications M_i of algorithm α where α is applied to a computation of the form $A_{m\times n}B_{n\times p}$. The matrices A and B are D^T and C^T respectively. For each M_i in α, construct the "reverse" $\hat{M}_i$ multiplication as follows: If

$$M_i = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}a_{ij}\right)\left(\sum_{i=1}^{n}\sum_{j=1}^{p} s_{ij}b_{ij}\right), \text{ then set}$$

$$\hat{M}_i = \left(\sum_{i=1}^{n}\sum_{j=1}^{p} s_{ij}b_{ij}\right)\left(\sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}a_{ij}\right).$$ Now, if we combine these

multiplications exactly as α does, we get an element of the product matrix $C_{p\times n}D_{n\times m} = Z_{p\times m}$.

More formally, an element z_{ij} in CD is found as follows: If α computes z_{ji} (in AB) as $\sum_{i=1}^{t} r_i M_i$ where $r_i \in K = \{0, 1, -1\}$, then let α' compute z_{ij} (in CD) as $\sum_{i=1}^{t} r_i \hat{M}_i$. The example in the next section will illustrate this procedure.

The main result on the symmetry of the general matrix multiplication problem follows immediately by alternating applications of Lemmas 2.3.2 and 2.3.3.

<u>Theorem 2.3.4</u> (Multiplicative Symmetry Theorem): Matrix products of each of the following forms have the same multiplicative complexity, i.e. require the same number of multiplications to compute:

$$(n\times m)(m\times p),\quad (p\times m)(m\times n),\quad (m\times p)(p\times n),$$
$$(n\times p)(p\times m),\quad (p\times n)(n\times m),\quad (m\times n)(n\times p)\ .$$

Theorem 2.3.4 can be used to extend known results for lower bounds for matrix products. For example, we can extend the results of Hopcroft and Kerr [H1] as follows:

<u>Corollary 2.3.5</u>: Algorithms which compute matrix products of the following forms must contain at least 15 multiplication steps:

$$(3\times 2)(2\times 3),\quad (2\times 3)(3\times 3),\quad (3\times 3)(3\times 2)\ .$$

<u>Corollary 2.3.6</u>: Algorithms which compute matrix products of the form $(2\times2)(2\times n)$, $(n\times2)(2\times2)$, or $(2\times n)(n\times2)$ require $\lceil 7n/2 \rceil$ multiplications.

If we examine the proofs of Theorem 2.3.1 and Lemma 2.3.3, we have the following result:

<u>Corollary 2.3.7</u>: Given any algorithm α which computes (m, n, p) products, we can construct bilinear chains of equal multiplicative complexity which compute (p, n, m), (n, p, m), (m, p, n), (p, m, n) and (n, m, p) products.

Thus, the lower bounds of Corollaries 2.3.5 and 2.3.6 are achievable.

CHAPTER III

IMPLICATIONS OF MULTIPLICATIVE SYMMETRY

In Sections 3.1 and 3.2 we examine two straightforward applications of the Symmetry Theorem:

1) transforming fast algorithms for a particular problem into fast algorithms for symmetric computations - specifically, constructing from a fast algorithm for (4, 2, 4) products fast algorithms of equal multiplicative complexity for (2, 4, 4) and (4, 4, 2) products, and

2) deriving a new lower bound on the multiplicative complexity of non-square (m, n, p) matrix multiplication, namely the product of max {m, n, p} and one less than the sum of the two smaller dimensions.

In Section 3.3, we ruminate on the possible behaviour of lower bounds for the general matrix multiplication problem in light of the Multiplicative Symmetry Theorem.

3.1 Deriving Symmetric Algorithms - An Example

Corollary 2.3.7 asserted that given an algorithm α, which computes (m, n, p) products using t multiplication steps, we can construct algorithms which compute (n, m, p), (p, m, n), (m, p, n), (n, p, m) and (p, n, m) products using exactly to multiplication steps. Clearly, to illustrate the procedure it suffices to derive algorithms for (n, m, p) and (p, m, n) products from α, since the remaining algorithms can be obtained by repeating the procedure.

We first present a 26-multiplication algorithm α for $(4, 2, 4)$ products obtained from the construction given in [H1] for fast algorithms which compute $(m, 2, n)$ products. Then, as an example of the general technique, we adapt the techniques of Fiduccia [F2] (c.f. Theorem 2.3.1) to obtain a 26-multiplication algorithm α' (the symmetric algorithm to α) for $(2, 4, 4)$ products, and then apply Lemma 2.3.3 to derive from α' a 26-multiplication algorithm α'' which computes $(4, 4, 2)$ products. In this example of deriving algorithms from an (m, n, p) algorithm, m happens to be the same as p; the technique, however, will be seen to be independent of the values of m, n, p.

The following 26-multiplication algorithm α computes matrix products of the form $A_{4\times 2}B_{2\times 4}$. The letters A_i, B_j, C_k, etc. refer to the structure of the multiplication in relation to the seven multiplications in some variant of Strassen's algorithm. In their paper, Hopcroft and Kerr defined a group of product-preserving transformations of algorithms for matrix multiplication. Thus, the seven letters designate the seven equivalence classes of multiplications in Strassen's scheme.

The multiplications of α are listed below.

A_1	$a_{12}(b_{11} + b_{21})$	E_2	$(a_{12} + a_{22})(b_{21} + b_{22})$
B_1	$(a_{11} - a_{12})b_{11}$	F_2	$(a_{11} + a_{21})(b_{11} + b_{12})$
C_1	$(a_{21} - a_{22})b_{22}$	G_2	$(a_{12} + a_{31} - a_{11})(b_{11} + b_{21} - b_{23})$
D_1	$a_{21}(b_{12} + b_{22})$	A_3	$(a_{32} - a_{22})(b_{13} + b_{23} - b_{12} - b_{22})$
E_1	$a_{32}b_{23}$	B_3	$(a_{31} + a_{22} - a_{32} - a_{21})(b_{13} - b_{12})$
F_1	$a_{31}b_{13}$	C_3	$(a_{31} + a_{42} - a_{41} - a_{32})(b_{23} - b_{24})$
G_1	$(a_{21} + a_{12})(b_{11} - b_{22})$	D_3	$(a_{31} - a_{41})(b_{13} + b_{23} - b_{14} - b_{24})$
A_2	$a_{42}(b_{14} + b_{24})$	E_3	$(a_{42} + a_{22})(b_{24} + b_{22})$
B_2	$(a_{41} - a_{42})b_{14}$	F_3	$(a_{41} + a_{21})(b_{14} + b_{12})$
C_2	$(a_{31}+a_{12}-a_{11}-a_{32})(b_{23}-b_{21})$	G_3	$(a_{21} + a_{32} - a_{22})(b_{13} - b_{12} - b_{22})$
D_2	$(a_{31}-a_{11})(b_{13}+b_{23}-b_{11}-b_{21})$		

C_4	$(a_{11}+a_{21}+a_{42}-a_{41}-a_{12}-a_{22})(b_{21}+b_{22}-b_{24})$
D_4	$(a_{11}+a_{21}-a_{41})(b_{11}+b_{12}+b_{21}+b_{22}-b_{14}-b_{24})$
G_4	$(a_{21} + a_{42})(b_{14} - b_{22})$
G_5	$(a_{31} - a_{41} + a_{42})(b_{14} + b_{24} - b_{23})$
G_6	$(a_{11}+a_{21}-a_{41}+a_{42})(b_{14}-b_{21}-b_{22}+b_{24})$

Table 3.1.1: Multiplications in α

Each diagonal element is computed from two multiplications; symmetric off-diagonal elements are computed in pairs from three additional multiplications. For example, if we let $C_{4\times4}$ denote the product matrix $A_{4\times2}B_{2\times4}$, then the elements c_{11}, c_{22}, c_{12} , and c_{21} of C are computed from the multiplications A_1, B_1, C_1, D_1, E_2, F_2, and G_1 as follows:

$$\begin{aligned}
c_{11} &= a_{11}b_{11} + a_{12}b_{21} = A_1 + B_1 \\
c_{22} &= a_{21}b_{12} + a_{22}b_{22} = -C_1 + D_1 \\
c_{12} &= -B_1 - D_1 + F_2 - G_1 \\
&= -a_{11}b_{11} + a_{12}b_{11} - a_{21}b_{12} - a_{21}b_{22} + a_{11}b_{11} + a_{11}b_{12} \\
&\quad + a_{21}b_{11} + a_{21}b_{12} - a_{21}b_{11} + a_{21}b_{22} - a_{12}b_{11} + a_{12}b_{22} \\
&= a_{11}b_{12} + a_{12}b_{22} \\
c_{21} &= -A_1 + C_1 + E_1 + G_1 \\
&= a_{21}b_{11} + a_{22}b_{21}
\end{aligned}$$

The algorithm uses 8 multiplications to compute the four diagonal elements, as well as 18 multiplications to compute the six pairs of off-diagonal elements. The total number of multiplications used is therefore $8 + 18 = 26$.

The next step is to transform α into α' via Lemma 2.3.2 such that α' computes $(2, 4, 4)$ products using exactly 26 multiplications. To do this, we first must characterize α in terms of the following decomposition theorem:

Theorem 3.1.1 (Fiduccia): There exists a K-bilinear chain ϕ for $E(Xy)$ with t multiplication steps iff there exist fixed matrices V,W with elements in K and a $(t\times t)$ diagonal matrix U with elements in $\mathcal{L}_K(E(X))$ such that

$$X = WUV$$

Actually, Fiduccia's result is slightly different, characterizing X as $WUV + H$. Our class of algorithms allows operations on zero matrices. Thus, $X = 0$ implies $U = 0$ which in turn implies $H = 0$, yielding Theorem 3.1.1.

To make use of this theorem we first must find the matrix-vector product which corresponds to a $(4, 2, 4)$ product. By Lemma 2.1.1, this is Xy where X and y are defined as follows.

Let

$$X = I_4 \otimes A_{4\times 2} = \begin{pmatrix} A & . & . & . \\ . & A & . & . \\ . & . & A & . \\ . & . & . & A \end{pmatrix}$$

(where '.' stands for the appropriate sized 0 matrix)

Furthermore, let $y = \kappa(B) = (b_{11},b_{21},b_{12},b_{22},b_{13},b_{23},b_{14},b_{24})^T$. Then, Xy has the product form $(16\times 8)(8\times 1)$.

Now, we construct the (26×26) matrix U by placing along its diagonal the contents of the left bracket of each multiplication of α in order as it appears in Table 3.1.1, and zeros everywhere else. In other words,

$$U = \begin{pmatrix} A_1^L & & & & 0 \\ & B_1^L & & & \\ & & \cdot & & \\ & & & \cdot & \\ 0 & & & \cdot & G_6^L \end{pmatrix},$$

where M_i^L is obtained from M_i by deleting the entries from $L_K(E(y))$, i.e. M_i^L is the left factor of multiplication M_i . For example, $A_1 = a_{12}(b_{11} + b_{21})$. Hence $A_1^L = a_{12}$.

The matrix W will specify which M_i's will be used to calculate an element of Xy and exactly how these intermediate results will be composed. The matrix V will operate on y to provide the elements from $L_K(E(y))$ which make up the righthand component of each original M_i . For $X_{16\times 8}y_{8\times 1}$, the matrix $V_{26\times 8}$ will be:

$$V = \begin{array}{c} \left(\begin{array}{cccccccc}
1 & 1 & . & . & . & . & . & . \\
1 & . & . & . & . & . & . & . \\
. & . & . & 1 & . & . & . & . \\
. & . & 1 & 1 & . & . & . & . \\
. & . & . & . & . & 1 & . & . \\
. & . & . & . & 1 & . & . & . \\
1 & . & . & -1 & . & . & . & . \\
. & . & . & . & . & . & 1 & 1 \\
. & . & . & . & . & . & 1 & . \\
. & -1 & . & . & . & 1 & . & . \\
-1 & -1 & . & . & 1 & 1 & . & . \\
. & 1 & . & 1 & . & . & . & . \\
1 & . & 1 & . & . & . & . & . \\
1 & 1 & . & . & . & -1 & . & . \\
. & . & -1 & -1 & 1 & 1 & . & . \\
. & . & -1 & . & 1 & . & . & . \\
. & . & . & . & . & 1 & . & -1 \\
. & . & . & . & 1 & 1 & -1 & -1 \\
. & . & . & 1 & . & . & . & 1 \\
. & . & 1 & . & . & . & 1 & . \\
. & . & -1 & -1 & 1 & . & . & . \\
. & 1 & . & 1 & . & . & . & -1 \\
1 & 1 & 1 & 1 & . & . & -1 & -1 \\
. & . & . & -1 & . & . & 1 & . \\
. & . & . & . & . & -1 & 1 & 1 \\
. & -1 & . & -1 & . & . & 1 & 1
\end{array}\right) \end{array}
\begin{array}{l}
A_1 \\ B_1 \\ C_1 \\ D_1 \\ E_1 \\ F_1 \\ G_1 \\ A_2 \\ B_2 \\ C_2 \\ D_2 \\ E_2 \\ F_2 \\ G_2 \\ A_3 \\ B_3 \\ C_3 \\ D_3 \\ E_3 \\ F_3 \\ G_3 \\ C_4 \\ D_4 \\ G_4 \\ G_5 \\ G_6
\end{array}$$

where the letters on the right indicate which multiplication of α has its right component equal to that row multiplied by y . For example, multiplying the first row of V by the column vector y yields

$$(1,1,0,\cdots,0)(b_{11},b_{21},\cdots,b_{24})^T$$

$$= b_{11} + b_{21} \quad ,$$

the right side of multiplication A_1 , as expected. In fact, the product $V\underset{\sim}{y}$ is a 26-element column vector which contains the right-hand components of each multiplication of α in order. Thus, the product UVy yields all the multiplications of α in the order in which they are listed in Table 3.1.1. The (16×26) matrix W is given below. The $4(j-1) + i$ the row of W encodes the combinations of the multiplications in α which are used to compute z_{ij} in the (16×1) vector $Xy = (z_{11},z_{21},z_{31},\cdots,z_{44})$.

$$W = \left(\begin{array}{cccccccccccccccccccccccccc}
1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
-1 & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & -1 & 1 & \cdot & -1 \\
\cdot & -1 & \cdot & -1 & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & -1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & -1 & \cdot & \cdot \\
1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot \\
\cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & 1 & \cdot & -1 & \cdot & 1 \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{array}\right)$$

$$\begin{array}{cccccccccccccccccccccccccc}
A_1 & B_1 & C_1 & D_1 & E_1 & F_1 & G_1 & A_2 & B_2 & C_2 & D_2 & E_2 & F_2 & G_2 & A_3 & B_3 & C_3 & D_3 & E_3 & F_3 & G_3 & C_4 & D_4 & G_4 & G_5 & G_6
\end{array}$$

For example, to compute z_{23} in Xy we need to scan the $4(3 - 1) + 2 = 10$ th row of W to see which multiplications of α are required and, also, to see how these multiplications are combined. Row 10 suggests

$$\begin{aligned} z_{23} &= D_1 + E_1 - A_3 + G_3 \\ &= a_{21}(b_{12} + b_{22}) + a_{32}b_{23} - (a_{32} - a_{22})(b_{13} + b_{23} - b_{12} - b_{22}) \\ &\qquad + (a_{21} + a_{32} - a_{22})(b_{13} - b_{12} - b_{22}) \\ &= a_{21}b_{13} + a_{22}b_{23} \qquad \text{as required.} \end{aligned}$$

Multiplying the 10th row by U results in the following 26-element vector u :

$$(0,0,0,D_1^L,E_1^L,0,\cdots,0,-A_3^L,0,\cdots,0,G_3^L,0,\cdots,0)$$

Multiplying u by Vy yields

$$D_1^L(b_{12}+b_{22}) + E_1^L(b_{23}) - A_3^L(b_{13}+b_{23}-b_{12}-b_{22}) + G_3^L(b_{13}-b_{12}-b_{22})$$

which is exactly the previous computation of z_{23} .

Thus, $Xy = WUVy$ and this leads directly to the required decomposition of X ; namely, let y range over the subset $\{e_1,\cdots,e_8\}$ of R^8 , then we have

$$X = XI = (WUV)I = WUV .$$

For example, $x_{12} = a_{12} = x_{54} = x_{96} = x_{13,8}$; hence, the decomposition should assign a_{12} to each of these four x_{ij}'s .

$$\begin{aligned} x_{12} &= (w_{11},w_{12},\cdots,w_{1,26})U(v_{12},v_{22},\cdots,v_{26,2})^T \\ &= (1\cdot a_{12},1\cdot(a_{11} - a_{12}),0,\cdots,0)(1,0,\cdots,-1)^T \\ &= a_{12} \end{aligned}$$

$$\begin{aligned} x_{54} &= (w_{15},w_{52},\cdots,w_{5,26})U(v_{14},v_{24},\cdots,v_{26,4}) \\ &= -a_{21} + 1(a_{21} + a_{12}) \\ &= a_{12} = x_{12} \quad \text{as expected.} \end{aligned}$$

Similarly, $x_{96} = x_{13,8} = a_{12}$, and in general, $x_{i+4k,j+2k} = x_{ij}$ for $k = 0,1,2,3$.

Thus, we have decomposed our encoding X of the matrix multiplicand A into three component matrices W,U,V such that U is a diagonal matrix, and W,V contain only elements in $K = \{0, 1, -1\}$.

Now, we are able to complete step two of the example, namely to derive an algorithm α' to compute a $(2, 4, 4)$ product. For now we have a decomposition of matrices of the form (8×16) ; $X^T = (WUV)^T = V^TUW^T$ because of the nature of $U,V,$ and W . More precisely,

$$I_4 \otimes A^T = X^T = V^TUW^T .$$

There is a small problem here, however, in that we do not wish a decomposition of the particular matrix A^T, rather a decomposition of all (2×4) matrices, based on α. This problem is easily solved by replacing each a_{ij} in U by a_{ji}; therefore, call this modification of U, U'. To understand this, note that (let $v'_{ij} = v_{ji}$, $w'_{ij} = w_{ji}$)

$$\begin{aligned}
&(v'_{21}, v'_{22}, \cdots, v'_{2,26})U(w'_{11}, \cdots, w'_{26,1})^T \\
&= (v_{12}, v_{22}, \cdots, v_{26,2})U(1,1,0,\cdots,0)^T \\
&= (1,0,\cdots,-1)(a_{12},(a_{11} - a_{12}),\cdots,0)^T \\
&= a_{12}
\end{aligned}$$

where the actual element in the second row, first column of $I_4 \otimes A_{2\times4}$ is a_{21}. Replacing U by U', however we have

$$\begin{aligned}
&(v'_{21}, \cdots, v'_{2,26})U'(w'_{11}, \cdots, w'_{26,1})^T \\
&= (1,0,\cdots,-1)(a_{21},(a_{11} - a_{21}),\cdots,0)^T \\
&= a_{21} \quad \text{as required.}
\end{aligned}$$

Thus, to compute a matrix product $A_{2\times4}B_{4\times4}$, find the equivalent matrix-vector product Xy where $X = I_4 \otimes A_{2\times4}$, $y = \kappa(B) = (b_{11}, b_{21}, b_{31}, b_{41}, b_{12}, b_{22}, \cdots, b_{34}, b_{44})^T$ as $Xy = V^TU'W^Ty$. Note that each element in the 8-element column vector $z = Xy$ is a unique entry in the product matrix $C_{2\times4} = A_{2\times4}B_{4\times4}$. c_{ij} is then found by computing the $2(j-1) + i$ the entry in z. This, in turn, is obtained by multiplying the product of the $2(j-1) + i$ th row of V^T with U' by W^Ty.

We now compute $c_{23} = a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} + a_{24}b_{43}$ to illustrate the procedure. First, note that c_{23} is the $2(3 - 1) + 2 = 6$th entry in z . Thus, to see which multiplications are involved, scan the 6th row of V^T , i.e. the 6th column of V ; 1's occur in the columns of V^T which correspond to elements of U' : E_1,C_2,D_2,A_3,C_3,D_3 , -1's in those columns corresponding to G_2,G_5 and 0's elsewhere (M_i in U' is obtained from M_i^L in U by replacing each a_{jk} by a_{kj}). Therefore, let the vector

$$\begin{aligned} u' &= (v'_{61},v'_{62},\cdots,v'_{6,26})U' \\ &= (0,\cdots,a_{23},\cdots,a_{13}+a_{21}-a_{11}-a_{23},a_{13}-a_{11},0,0, \\ &\quad -(a_{21}+a_{13}-a_{11}),a_{23}-a_{22},0,a_{13}+a_{24}-a_{14}-a_{23},a_{13}-a_{14},\cdots, \\ &\quad -(a_{13}-a_{14}+a_{24}),0) \end{aligned}$$

As before, W^Ty yields a 26-element vector whose entries are the righthand components of the multiplications to be used by α' ; thus, the set of multiplications used by α' to compute $A_{2\times4}B_{4\times4}$ is exactly $E(U'W^Ty)$. In particular,

$$\begin{aligned} c_{23} &= u'W^Ty \\ &= a_{23}(b_{31}+b_{23}+b_{33}+b_{34}) + (a_{13}+a_{21}-a_{11}-a_{23})b_{13} + (a_{13}-a_{11})\cdot \\ &\quad (-b_{13}) - (a_{21}+a_{13}-a_{11})(b_{31}-b_{13}) + (a_{23}-a_{22})(-b_{23}) + \\ &\quad (a_{13}+a_{24}-a_{14}-a_{23})b_{34} + (a_{13}-a_{14})(-b_{43}) - (a_{13}-a_{14}+a_{24})(b_{34}-b_{43}) \\ &= a_{23}b_{33} + a_{21}b_{13} + a_{22}b_{23} + a_{24}b_{43} \qquad \text{as required.} \end{aligned}$$

Thus, if we let α' be the algorithm which combines the multiplications given by $E(U'W^Ty)$ according to the entries in V^T, we have accomplished the second stage of the example, i.e. we have constructed an algorithm α' which computes $(2, 4, 4)$ products using exactly the same number of multiplications as α uses to compute $(4, 2, 4)$ products. Moreover, the construction is well-defined once α is known. We should also note here that the number of additions/subtractions employed by algorithms for symmetric problems is not necessarily constant. The effert on additions/subtractions is explained in Chapter IV.

The third and final step in this example, is to find α'' which computes $(4, 4, 2)$ products from the algorithm α' just constructed for the $(2, 4, 4)$ case. Clearly, if we assumed all multiplications commute, we could easily apply α' to the transposed multiplication problem and transpose the result as an algorithm with no more multiplications than α'. However, this is taboo in NC. But, by the techniques of Lemma 2.3.3 we are able to construct the required α'' from α'.

Suppose we wish α'' to compute $C_{4\times4}D_{4\times2} = Z_{4\times2}$. Let $A_{2\times4} = D^T$ and $B_{4\times4} = C^T$. Since $\alpha'' \in NC$, we must not assume each intermediate product $d_{ij}c_{k\ell}$ in the computation of AB commutes. However, we can sum these products as if they did commute. In fact, if α'' sums the reversals of multiplications in $U'W^Ty$ exactly as

α' sums the originals, the result computed by α'' will be the reversal of α''s computation and, therefore, the desired result.

For example, let $C' = AB$, $Z = CD$. Then, to compute $z_{32} = c_{31}d_{12} + c_{32}d_{22} + c_{33}d_{32} + c_{34}d_{42}$, collect those multiplications M_i which α' uses to compute c'_{23} and form the corresponding reverse multiplications as in Lemma 2.3.3 (the multiplications M_i are given in Table 3.1.1).

Summing the reverse multiplications $\hat{M}_i$ as α' sums the M_i, we obtain

$$\begin{aligned}
&(b_{31}+b_{23}+b_{33})a_{23} + b_{13}(a_{13}+a_{21}-a_{11}-a_{23}) + (-b_{13})(a_{13}-a_{11}) \\
&- (b_{31}-b_{13})(a_{21}+a_{13}-a_{11}) + (-b_{23})(a_{23}-a_{22}) + b_{34}(a_{13}+a_{24}-a_{14}-a_{23}) \\
&+ (-b_{43})(a_{13}-a_{14}) - (b_{34}-b_{43})(a_{13}-a_{14}+a_{24}) \\
&= b_{33}a_{23} + b_{13}a_{21} + b_{23}a_{22} + b_{43}a_{24}
\end{aligned}$$

Finally, substiting $a_{ij} = d_{ji}$, $b_{ij} = c_{ji}$, we get

$$\begin{aligned}
&c_{33}d_{32} + c_{31}d_{12} + c_{32}d_{22} + c_{34}d_{42} \\
&= z_{32} \qquad \text{as required.}
\end{aligned}$$

Since all we modified in α' is the order of the multiplicands, α'' uses exactly as many multiplications, and additions/subtractions to compute $(4, 4, 2)$ products as α' does to calculate $(2, 4, 4)$ products.

Thus, we have 26-multiplication algorithms for the symmetric problems: (4, 2, 4), (2, 4, 4) and (4, 4, 2) matrix multiplication. It is not known if these algorithms are optimal; obviously, by the Symmetry Theorem, if any of α , α' , α'' is optimal, they all are.

Note the two references to additive complexity of α' in terms of that of α , and of α'' in terms of α' . We have mentioned that the additive complexity of algorithms for (m, n, p) and (p, n, m) products is invariant. This fact, together with the exact relationship between the additive complexities of (m, n, p) and (n, m, p) products is thoroughly investigated in Chapter IV.

Since we have utilized no specific features of this example to illustrate the algorithm construction process, the process itself is completely general, i.e. we have demonstrated a simple method for building algorithms for (n, m, p) and (p, m, n) products from an algorithm for (m, n, p) products such that the new algorithms are of exactly the same multiplicative complexity as the original. By combining the above processes in appropriate ways, we can obtain algorithms of equal cost for the remaining symmetric problems, namely (m, p, n), (n, p, m), and (p, n, m) products.

3.2 A New Lower Bound for Non-square Products

In this section by applying the Multiplicative Symmetry Theorem, we show that the product of the maximum dimension with one less than the sum of the other two dimensions yields a new lower bound on the multiplicative complexity of non-square matrix multiplication.

First, we note the following theorem due to Kirkpatrick [K2]:

Theorem 3.2.1: Any algorithm over $Q[a_{11},\cdots,a_{mn},b_{11},\cdots,b_{np}]$, where Q , a_{ij} , b_{ij} are defined as before, which computes the matrix product $A_{m\times n}B_{n\times p}$, must employ at least $m(n + p - 1)$ multiplication steps.

Actually, Kirkpatrick's theorem is phrased in terms of independent variables in a field and active *-operations, but if we restrict the class of algorithms to those which do not use division, we obtain Theorem 3.2.1.

Corollary 3.2.2: If we consider only algorithms in NC , an (m, n, p) product requires $m(n + p - 1)$ multiplications.

This is an obvious corollary, but is included to emphasize the difference in models.

As an immediate consequence of the Multiplicative Symmetry Theorem (2.3.4) we have

<u>Theorem 3.2.3</u>: Any algorithm in NC which computes (m, n, p) products must use at least $m_x(d_0 + d_1 - 1)$ multiplications where m_x is the largest dimension, d_0 and d_1 are the two smaller dimensions.

<u>Proof</u>: By Theorem 2.3.4, the same number of multiplication steps is required to compute (m, n, p) and (m_x, d_0, d_1) products where m_x, d_0, d_1 are any distinct choice of m, n, p.

<u>Remark 3.2.4</u>: The choice $m = m_x$, $n = d_0$, $p = d_1$ where m_x is the largest dimension, and d_0, d_1 are the two smaller dimensions in the unordered triple $[m, n, p]$ gives the best lower bound obtainable by the method of Theorem 3.2.1. We need only observe that

$$\begin{aligned} d_0(m_x + d_1 - 1) &= m_x d_0 + d_0(d_1 - 1) \\ &\leq m_x d_0 + m_x(d_1 - 1) \quad \text{since } d_1 \geq 1,\ m_x \geq d_0 \\ &= m_x(d_0 + d_1 - 1) \ . \end{aligned}$$

And similarly, $d_1(m_x + d_0 - 1) \leq m_x(d_0 + d_1 - 1)$.

As an example of this, observe that $(2, 4, 4)$ products require at least $4(2 + 4 - 1) = 20$ multiplications, whereas Kirkpatrick's lower bound is $2(4 + 4 - 1) = 14$ multiplications. This lower bound may not be optimal; the best algorithm to date, α'

in Section 3.1, uses 26 multiplications. We can, however, show that Theorem 3.2.3 does not necessarily produce achievable lower bounds by examining computations of $(2, n, 2)$ products for $n \geq 3$. Kirkpatrick's lower bound is $2(n + 2 - 1) = 2n + 2$ multiplications. The new lower bound is $n(2 + 2 - 1) = 3n$ multiplications, an improvement. However, by Corollaries 2.3.6 and 2.3.7 of the Multiplicative Symmetry Theorem, $\lceil 7n/2 \rceil$ multiplications is an achievable lower bound. In particular, for $(2, 3, 2)$ products, Theorem 3.2.3 yields a lower bound of 9, but 11 multiplications are required. Clearly, further refinements of the bound of Theorem 3.2.3 are required.

3.3 Other Ramifications of Multiplicative Symmetry

The previous sections illustrated some immediate applications of the Symmetry Theorem, namely a construction for deriving a variety of equicomplex algorithms for five related problems from a known algorithm for a particular problem. Extensions of the relatively few known lower bounds on non-square matrix multiplication problems were also made. As well, multiplicative symmetry presents less concrete implications for our understanding of lower bounds.

For example, Theorem 2.3.4 corrects an invalid intuitive feeling that one dimension in particular is the primary contributor to the multiplicative complexity of (m, n, p) products; for example, n, the number of terms involved in brute-force inner-product multiplication, or m, the first dimension, as in Theorem 3.2.1.

Suppose we denote the multiplicative complexity of (m, n, p) products by a lower bound function $\mathcal{M}(m, n, p)$. Note that multiplicative symmetry implies that $\mathcal{M}$ is a symmetric function.

That $\mathcal{M}$ is not linear in the product $m \cdot n \cdot p$ of the dimension variables is well known and can be proved by examining the special problem of (n, n, n) products. By Strassen's result [S1], $\mathcal{M}(n, n, n) \leq c \cdot n^{2.82}$ where c is a constant. Thus, for suitably large n, $\mathcal{M}(n, n, n) < n^3$ which proves $\mathcal{M}$ is not simply linear in the dimension product.

We can gain further insight into the nature of $\mathcal{M}$ by fixing m, n, p in examples such as the following.

Example 3.3.1: (8, 2, 2), (16, 2, 1), and (4, 4, 2) products all have 32 as the product of their dimensions. By [W1], $\mathcal{M}(16, 2, 1) = 32$. However, by Corollaries 2.3.6 and 2.3.7, $\mathcal{M}(8, 2, 2) = 28$. Finally, since α'' in Section 3.1 computes (4, 4, 2) products in 26 multiplications, $\mathcal{M}(4, 4, 2) \leq 26$.

Upon closer inspection, we note that with the dimension product fixed, the greater the disparity among the sizes of the dimensions, the greater the number of multiplications which are required. Put another way, the more symmetric two matrices A and B are in size, the lower the multiplicative complexity. This suggests the following

Conjecture 3.3.2: Given any pair of triples of integers m_1, n_1, p_1 and m_2, p_2, n_2 such that $m_1n_1p_1 = m_2n_2p_2 = d$, then if $c = \sqrt[3]{d}$, $|m_1-c| + |n_1-c| + |p_1-c| > |m_2-c| + |n_2-c| + |p_2-c|$ implies $\mathcal{M}(m_1, n_1, p_1) > \mathcal{M}(m_2, n_2, p_2)$.

In order to avoid degenerate cases, we restrict m_i, n_i, p_i to be integers ≥ 2 . This is necessary since independence arguments have shown that if 1 dimension, say m , is 1 , then $\mathcal{M}(m, n, p) = np$. Thus, in our example, let $m_1 = 32$, $n_1 = p_1 = n_2 = 1$, $m_2 = 16$, $p_2 = 2$. Let $c = \sqrt[3]{32}$. Then

$|m_1-c| + |n_1-c| + |p_1-c| > |m_2-c| + |n_2-c| + |p_2-c|$ and yet $\mathcal{M}(m_1, n_1, p_1) = \mathcal{M}(m_2, n_2, p_2) = 32$.

A consequence of the conjecture would be

<u>Corollary 3.3.3</u>: If $mnp = d^3$, $\mathcal{M}(m, n, p) = \mathcal{M}(d, d, d)$ implies $m = n = p = d$.

Thus, for a fixed dimension product, square matrix multiplication would be least multiplicatively complex.

We feel that $\mathcal{M}(m, n, p)$ allocates weight to each dimension variable corresponding to its size in relation to the other dimension variables. This is supported by Theorem 3.2.3, but as yet is an open problem.

Theorem 1.2.4, the best lower bound known, supports Corollary 3.3.3, since its bound is essentially $mn + np + mp$. By employing partial differentiation, we can show that $\frac{1}{m} + \frac{1}{n} + \frac{1}{p}$ is minimized for $m = n = p$ with mnp fixed. Since mnp is a fixed product, $mn + np + mp$ is also minimized for $m = n = p$.

The next two chapters will focus on the additive complexity of matrix multiplication.

CHAPTER IV

THE ADDITIVE COMPLEXITY OF MATRIX MULTIPLICATION

In this chapter, we investigate the nature of the additive complexity of computing (m, n, p) products using a fixed number t (perhaps $\mathcal{M}$(m, n, p)) of multiplications. For this purpose, we present a simple graph-theoretic model which exactly characterizes each step in an (m, n, p) computation; in particular, the model encodes addition/subtraction steps in a natural way. For purposes of transforming representations of computations into representations of related computations, the model contains certain canonical features which allow us to obtain relations between the additive complexity of symmetric computations (for example, of (m, n, p) and (n, m, p) products). In section 3.1, we noted that α'' for (p, m, n) products uses exactly the same number of additions/subtractions as α' for (n, m, p) products. Generalizing these results, we are able to relate the additive complexities of the five associated computations to the additive complexity of a given (m, n, p) computation. This result is called the Additive Symmetry Theorem.

Note that the term "algorithm" used in reference to additive complexity will not refer to an arbitrary member of NC , but rather to a K-bilinear chain where K , unless otherwise specified, is a subring of the centre of any ring with a unit. Again, the reader should be able to persuade himself that the results on additive complexity as well as those on multiplicative complexity hold for computations of any set of bilinear forms, although the results are described solely in terms of matrix multiplication.

4.1 Additive Complexity: Addition Flow Representations

As noted in Chapter I, by the additive complexity of a matrix multiplication problem, we mean the least possible number of additions and subtractions necessary to multiply matrices using no more than t multiplications. To investigate the additive complexity of (m, n, p) computations using any set of t multiplications, we will wish to examine the number of additions/subtractions used by a particular algorithm for (m, n, p) products. Thus, we will wish to optimize the number of additions/subtractions used to form the factors in a fixed set of multiplications and to combine the calculated products to form the product elements.

We present a natural graph-theoretic model of computation which lends itself to this type of investigation. Essentially, the model consists of an ordered triple of three graphs $<G_1, G_2, G_3>$ such that G_1 encodes the addition/subtraction steps used to form lefthand factors of the fixed set of multiplication steps, G_2 encodes the addition/subtraction steps used to form the corresponding righthand factors, and G_3 represents exactly how to combine the results of the multiplications to form the elements in the product matrix. More formally:

Given an algorithm α which computes (m, n, p) products using the t multiplications $M = \{M_1, M_2, \cdots, M_t\}$, an addition flow representation of α , denoted F_α , is an ordered triple

$<G_1, G_2, G_3>$ where each G_i is an acyclic, multi-source, multi-sink flow graph with vertex and edge sets denoted VG_i, ΓG_i, respectively. As well, if the source and sink set of each G_i is denoted R_i, S_i, respectively, then

1) $|R_1| = mn$, $|R_2| = np$, $|S_3| = mp$,

2) $|S_1| = |S_2| = |R_3| = t$,

3) there exists an isomorphism λ which labels the vertices in $VG_1 \cup VG_2 \cup VG_3$ as follows: if the ith evaluation in α is denoted $e_\alpha(i)$, and α computes the product $A_{m\times n}B_{n\times p} = Y_{m\times p}$, then

   a) $R_1 = \{\lambda(a_{11}),\cdots,\lambda(a_{mn})\}$, $R_2 = \{\lambda(b_{11}),\cdots,\lambda(b_{np})\}$,

   b) $e_\alpha(j) = e_\alpha(j_1)\cdot e_\alpha(j_2)$ if and only if $\lambda(e_\alpha(j)) \in R_3$ (in other words, the t multiplication steps, $M = \{M_1,M_2,\cdots,M_t\}$ are represented by the t vertices in R_3),

   c) $S_1 = M^L$, $S_2 = M^R$ (i.e. S_1, S_2 represent the set of left-hand and righthand factors respectively of the multiplications $M = \{M_1,\cdots,M_t\}$),

   d) $S_3 = \{\lambda(y_{11}),\lambda(y_{12}),\cdots,\lambda(y_{mp})\}$,

   e) for each evaluation $e_\alpha(j)$ of the form $e_\alpha(j_1) \pm e_\alpha(j_2)$ there exist unique edges γ_1, γ_2 in one of the graphs such that γ_1 has edge weight 1, γ_2 has edge weight 1 or -1 depending on whether $e_\alpha(j_2)$ is added to, or subtracted from $e_\alpha(j_1)$. As well, the "tail" of edge γ_1 , written $T(\gamma_1)$ is the vertex representing $e_\alpha(j_1)$. Similarly, $T(\gamma_2) = \lambda(e_\alpha(j_2))$. The "heads" of γ_1, γ_2, written $H(\gamma_1)$, $H(\gamma_2)$ are the same vertex, namely $\lambda(e_\alpha(j))$,

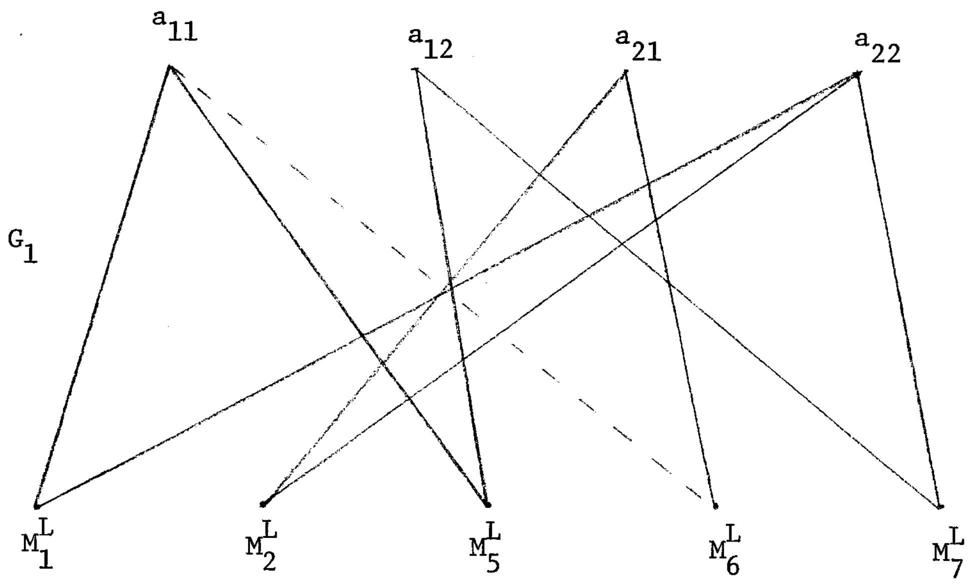
and f) for each evaluation $e_\alpha(j)$ of the form $r \cdot e_\alpha(k)$ where r is a scalar, there is a unique edge γ with edge weight r, such that $H(\gamma) = \lambda(e_\alpha(j))$ and $T(\gamma) = \lambda(e_\alpha(k))$.

Basically, an addition flow representation is a modification of the "computation tree" of an algorithm. To illustrate the differences, we present a typical computation graph representation for Strassen's algorithm α_S given in [S1] for computing $A_{2\times2}B_{2\times2} = Y_{2\times2}$ (α_S is K-bilinear where $K = \{0, 1, -1\}$) :

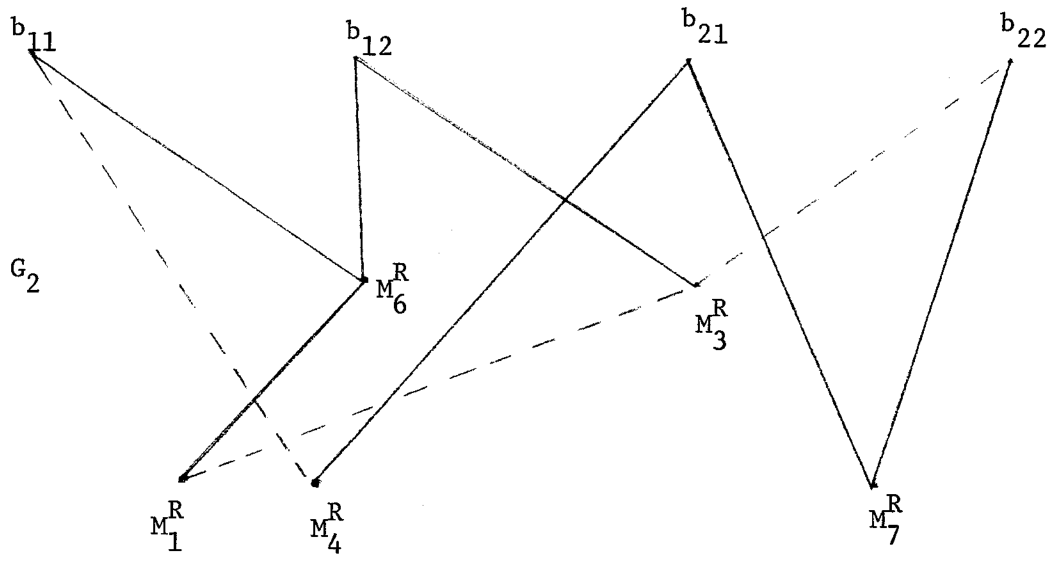
The 7 multiplications, $\{M_1,\cdots,M_7\} = M_S$, used by α_S are

M_1	$(a_{11} + a_{22})(b_{11} + b_{22})$	M_5	$(a_{11} + a_{12})b_{22}$
M_2	$(a_{21} + a_{22})b_{11}$	M_6	$(-a_{11} + a_{21})(b_{11} + b_{12})$
M_3	$a_{11}(b_{12} - b_{22})$	M_7	$(a_{12} - a_{22})(b_{21} + b_{22})$
M_4	$a_{22}(-b_{11} + b_{21})$		

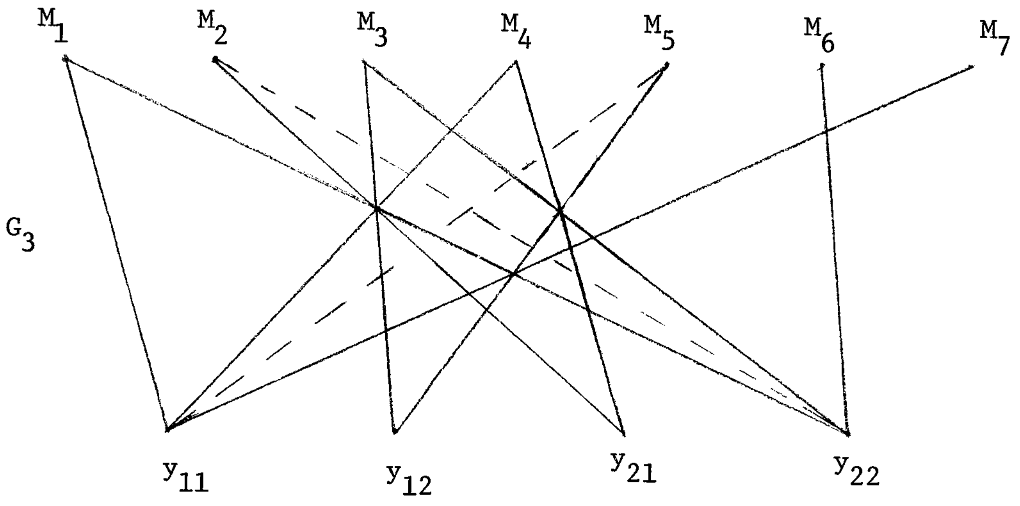
We may think of α_S as computing (2, 2, 2) products in three stages, forming lefthand and righthand factors of the multiplications in M_S , calculating the actual values of the M_i , and finally combining these values to yield the y_{ij} , $1 \leq i,j \leq 2$. Accordingly, we could represent the first stage by graphs G_1,G_2 as follows:



In this graph, as in any graphs presented in the remainder of the thesis, all edges are directed downwards. Note we do not include $M_3^L = a_{11}$ and $M_4^L = a_{22}$ in the sink set S_1 of G_1. This indicates that α_S uses no intermediate calculations to form these factors. A dashed line indicates an edge weight of -1. Thus, α_S computes M_6^L as $-a_{11}+a_{21}$.



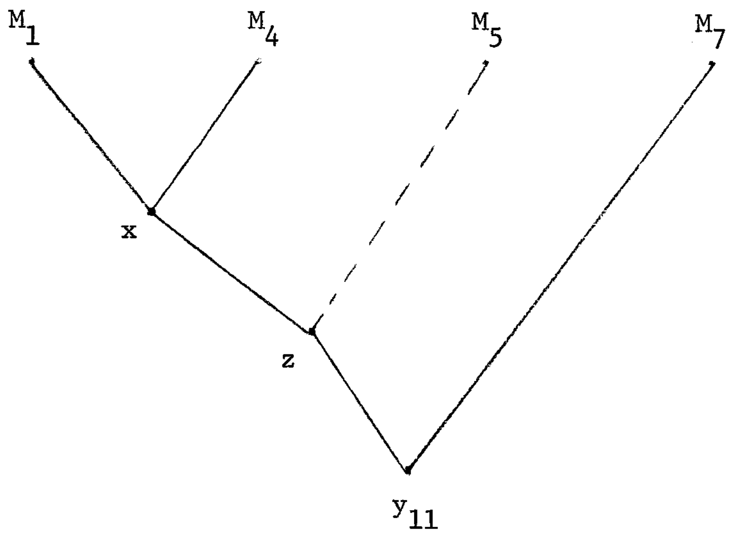
This graph represents a slightly devious way α_S could compute $M_1^R = b_{11} + b_{22}$. As illustrated, α_S computes $M_6^R = b_{11} + b_{12}$, then $M_3^R = b_{12} - b_{22}$, and finally $M_1^R = M_6^R - M_3^R$. Once again, since α_S needs no intermediate calculations to compute $M_2^R = b_{11}$ and $M_5^R = b_{22}$, vertices representing M_2^R and M_5^R do not appear in the above "computation graph". Finally, suppose we let the values of the seven multiplications in M_S be represented by seven vertices in a final "computation graph", G_3 . Then, if α_S computes the y_{ij}'s from these values exactly as given in [S1], G_3 would be drawn as



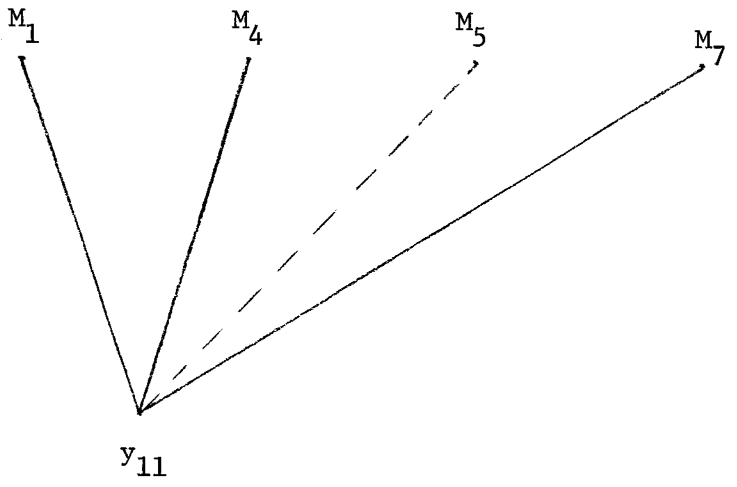
For example, α_S computes y_{11} as $M_1 + M_4 - M_5 + M_7$. Clearly, since each evaluation in an algorithm is defined as at most a binary operation, the computation of y_{11} by α_S actually involves three steps, namely:

$$x = M_1 + M_4 ,$$
$$z = x - M_5 ,$$
$$y_{11} = z + M_7 .$$

This sequence of operations should properly be represented by

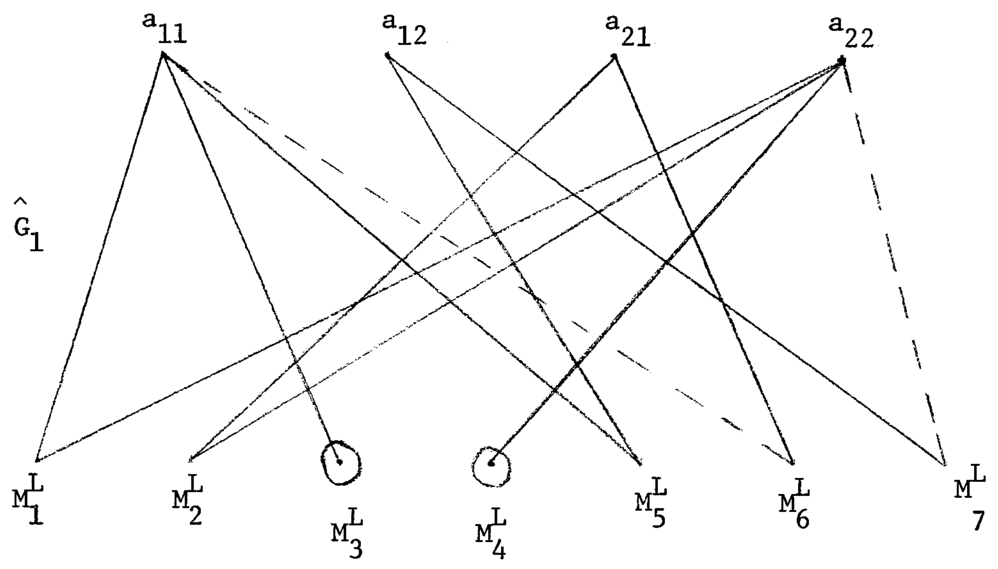


However, since the meaning is clear, we will represent such a left to right sequence of operations as in G_3 , i.e. as

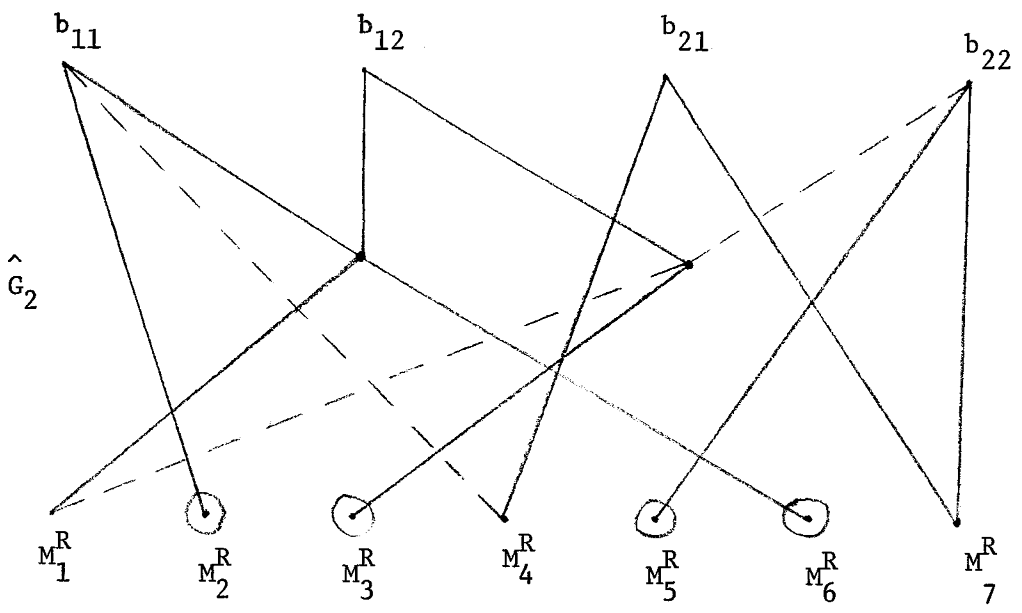


Note that the above representation $<G_1, G_2, G_3>$ of algorithm α_S is not an addition flow representation of α_S for the following reason. Although $R_3 = \{M_1, \cdots, M_t\} = M_S$ as required, $S_1 \subsetneq M_S^L$, $S_2 \subsetneq M_S^R$. Before justifying the definition of the model by proving the main result of this section, namely that given any algorithm α which computes (m, n, p) products using t multiplication steps, there is an algorithm $\hat{\alpha}$ which computes (m, n, p) products in t multiplications using no more additions/subtractions than α, such that $\hat{\alpha}$ has a unique addition flow representation, we illustrate part of the technique by constructing from $<G_1, G_2, G_3>$, the addition flow representation $\hat{F} = <\hat{G}_1, \hat{G}_2, \hat{G}_3>$ of an algorithm $\hat{\alpha}_S$ which computes $(2, 2, 2)$ products by the 7 multiplications in M_S, and uses exactly the same number of additions/subtractions as α_S. As well, $\hat{\alpha}_S$ will be K-bilinear.

Clearly, G_3 satisfies the definition of the third graph of an addition flow representation. Therefore, set $\hat{G}_3 = G_3$. G_1 is merely G_1 with $\hat{S}_1 = S_1 \cup \{M_3^L, M_4^L\}$. Thus, we have



where we have inserted 2 edges, one from a_{11} to M_3^L, and one from a_{22} to M_4^L. By the definition of an addition flow representation, this modification corresponds to inserting 2 trivial evaluations into α_S of the form $e_\alpha(j) = 1 \cdot e_\alpha(k)$ where $e_\alpha(k) = a_{11}$, or $e_\alpha(k) = a_{22}$. Clearly, the modified algorithm $\hat{\alpha}_S$ uses no more additions/subtractions than α_S to compute lefthand factors. Similarly, we can modify G_2 by introducing trivial scalar multiplications of intermediate calculations into the algorithm.



Again trivial scalar multiplications corresponding to the circled sink vertices $M_2^R, M_3^R, M_5^R, M_6^R$, are added to algorithm α_S with no increase in the number of addition/subtraction steps used to form righthand factors. Clearly, $\hat{\alpha}_S$ (α_S with the above scalar multiplications added) computes (2, 2, 2) products using the multiplications in M_S and no more addition/subtraction steps. Also, since $\hat{S}_1 = M_S^L$, $\hat{S}_2 = M_S^R$, $\hat{F} = <\hat{G}_1, \hat{G}_2, \hat{G}_3>$ is an addition flow representation for $\hat{\alpha}_S$.

This is a simple example, but the method is clearly generalizable to any representation $<G_1, G_2, G_3>$ for which

$S_1 \subsetneq M^L$, $S_2 \subsetneq M^R$, or $S_3 \subsetneq \{y_{11},\cdots,y_{mp}\}$. Before we can apply this simple construction, however, we need to show that there is no loss in generality in studying the additive complexity of (m, n, p) products with a fixed number t of multiplications, over only those algorithms which have addition flow representations. Accordingly, we prove

<u>Theorem 4.1.1</u>: For any K-bilinear α which computes (m, n, p) products in t multiplications and <u>a</u> additions/ subtractions, there is K-bilinear chain $\hat{\alpha}$ which performs the same computation in t multiplications and no more than <u>a</u> addition/ subtraction steps and $\hat{\alpha}$ has an addition flow representation.

<u>Proof</u>: **Ideally, we would like to show that NC-** algorithms are not additively faster than K-bilinear chains; however, this equivalence remains an open problem. We concurnwith A. Borodin who conjectured the two classes of algorithms are not additively equivalent. However, it may be possible to show that any computation in NC of a set of bilinear forms which uses t multiplication steps and <u>a</u> addition/subtraction steps can be simulated by a K-bilinear chain (where $K \subseteq Q$, the field of scalars for the NC-computation) using t multiplication steps and about 3<u>a</u> addition/subtraction steps.

Since α is K-bilinear, we let $\hat{\alpha}$ be α with undesirable properties removed, by the same technique as in the construction of the addition-flow representation of $\hat{\alpha}_S$.

By the bilinearilty of $\hat{\alpha}$ (where $\hat{\alpha}$ is initially defined as α ; the set $\hat{M}$ of multiplications of $\hat{\alpha}$ initially defined as M), the computation performed by $\hat{\alpha}$ may be divided into four disjoint stages:

1) form lefthand factors of the t multiplications in $\hat{M}$,

2) form righthand factors of the t multiplications,

3) compute the t values of $\hat{M} = \{\hat{M}_1, \cdots, \hat{M}_t\}$,

4) compute the mp product elements $\{y_{11}, \cdots, y_{mp}\}$

from the multiplications in $\hat{M}$. Thus, we can partially define $\langle G_1, G_2, G_3 \rangle$ by $R_1 = \{a_{11}, \cdots, a_{mn}\}$, $R_2 = \{b_{11}, \cdots, b_{np}\}$, $R_3 = \{\hat{M}_1, \cdots, \hat{M}_t\} = \hat{M}$. However, other modifications to $\hat{\alpha}$ may be required to enable the construction of an addition flow representation $\hat{F}$ for $\hat{\alpha}$.

First, $\hat{\alpha}$ may not contain non-essential intermediate calculations, for example, α_0 may compute an evaluation $e_{\alpha_0}(j)$ which is not a product element and yet is not re-used in any subsequent evaluation. However, by the construction of $\hat{\alpha}$, such evaluations are deleted. Note that $\hat{\alpha}$ may contain inefficient or unnecessary evaluations. For example, one could compute $\hat{M}_j = a_{11}$ by forming $e_{\hat{\alpha}}(j_1) = a_{11} + a_{21}$, $e_{\hat{\alpha}}(j) = e_{\hat{\alpha}}(j_1) - a_{21}$, where a single trivial scalar multiplication $e_{\hat{\alpha}}(j) = 1 \cdot a_{11}$ would suffice. However, this is necessary to allow the study of general algorithms. Thus, we can now ensure that S_1, S_2, S_3 may be defined such that $S_1 \subseteq (\hat{M})^L$, $S_2 \subseteq (\hat{M})^R$, $S_3 \subseteq \{y_{11}, \cdots, y_{mp}\}$.

Secondly, $\hat{\alpha}$ may use an evaluation which is either a lefthand factor, a righthand factor, or a product element as an intermediate calculation in the subsequent computation of another lefthand factor, righthand factor, or product element respectively. Then, by adding trivial scalar multiplications to $\hat{\alpha}$ (as was done to construct $\hat{\alpha}_S$ from α_S in the illustration) we create a new evaluation which can be represented by a new vertex in the appropriate sink set (c.f. G_2 in the example). Therefore, modify $\hat{\alpha}$ accordingly.

Finally, if some $(\hat{M}_j)^L$ or $(\hat{M}_k)^R$ is a single a_{ij} or b_{ij} element respectively, and $\hat{\alpha}$ does not yet contain a corresponding evaluation, insert a trivial scalar multiplication corresponding to this term. Repeat this procedure until $\hat{\alpha}$ contains all such trivial evaluations. Then, each such evaluation can be represented as a vertex in the appropriate sink set of an addition flow representation (c.f. M_3^L, M_4^L in $\hat{G}_1$ of the example).

We claim the resulting $\hat{\alpha}$ (no addition/subtraction steps have been added in these final modifications, only trivial scalar multiplications) has an addition flow representation. But this is obvious from the bilinearity of $\hat{\alpha}$ and the above constructions.

Thus, the theorem is proved.

By Lemma 2.2.3 we have

<u>Corollary 4.1.2</u>: Given any algorithm $\alpha \in NC$ which computes (m, n, p) products using t multiplication steps (where multiplication by an element of a ring K with unit is not counted)

there exists a K-bilinear chain which computes (m, n, p) products in no more than t multiplication steps (but possibly many more additions/subtractions) which has an addition flow representation.

By Theorem 4.1.1, we need not study the additive complexity of computations of sets of bilinear forms over all possible bilinear algorithms, but rather only over all bilinear chains which have an addition flow representation. This restriction will allow us to treat computations by operating on flow representations in very simple ways.

Thus, in the remainder of the thesis, unless otherwise specified, we use the term "algorithms" to mean "algorithms which have an addition flow representation". In other words, we will focus on **bilinear chains to study additive complexity.**

Note that an algorithm has a unique addition representation.

Lemma 4.1.3: The addition flow representation F_α of an algorithm for (m, n, p) products is unique.

Proof: Suppose $F_\alpha = \langle G_1, G_2, G_3 \rangle$, and $F'_\alpha = \langle G'_1, G'_2, G'_3 \rangle$ are addition flow representations for α with representation isomorphisms λ, μ respectively. Then, for $1 \leq i \leq 3$, $\lambda \circ \mu^{-1}$ maps VG'_i onto VG_i . As well, each pair of isomorphic (under $\lambda \circ \mu^{-1}$) vertices represent a unique evaluation in α , and so, adjacencies and edge weights are preserved by $\lambda \circ \mu^{-1}$. Therefore, except for a possible relabelling of the edges in their component graphs, F_α and F'_α are identical.

Although an algorithm has a unique representation, the same representation can represent many algorithms depending on the method of traversing the graph, since each computation involves a path from source vertices. We define the <u>standard algorithm with representation $F = \langle G_1, G_2, G_3 \rangle$</u> to be the algorithm which follows the four stages of computation in order. In any G_i we define the <u>height</u> of an intermediate vertex v to be the length of the longest path from any source vertex to that vertex v. Then, the standard algorithm computes the intermediate calculations represented by intermediate vertices in ascending order of height in a left to right order among vertices of equal height for G_1, then G_2, and finally G_3. The multiplications represented by R_3 are calculated before G_3 is processed.

Thus, instead of studying the class of algorithms which compute (m, n, p) products, we investigate without loss of generality, the number of additions/subtractions represented by addition flow representations. For ease of reference, we denote the number of addition/subtraction steps in an algorithm by <u>nadds (α)</u>. <u>nadds (G_i)</u>, <u>nadds (F)</u> denote the number of additions/subtractions represented by graph G_i and representation F respectively.

Clearly,
$$\begin{aligned} \text{nadds}(\alpha) &= \text{nadds}\ (F_\alpha) \\ &= \sum_{i=1}^{3} \text{nadds}\ (G_i) \ . \end{aligned}$$

The number of additions/subtractions represented at a particular non-source vertex v in some G_i is clearly indegree (v) - 1 . Thus we have

Lemma 4.1.4: The number of additions/subtractions used by algorithm α with addition flow representation $F_\alpha = \langle G_1, G_2, G_3 \rangle$ is $\sum_{i=1}^{3} [|\Gamma G_i| - |VG_i| + |R_i|]$.

Proof: $$\text{nadds }(\alpha) = \sum_{i=1}^{3} \text{nadds }(G_i)$$

And, $$\text{nadds }(G_i) = \sum_{v \in VG_i - R_i} (\text{inval }(v) - 1)$$

since inval (v) = 0 for $v \in R_i$, $$= \sum_{v \in VG_i - R_i} \text{inval }(v) - \sum_{v \in VG_i - R_i} 1$$

since $\sum_{v \in VG_i - R_i} \text{inval }(v) = |\Gamma G_i|$, $$= |\Gamma G_i| - |VG_i| + |R_i|$$

and the lemma is proved.

Thus, the number of addition/subtraction steps represented by $\hat{F}_S$ is

$$\sum_{i=1}^{3} [|\Gamma G_i| - |VG_i| + |R_i|]$$

$$= (12 - 11 + 4) + (14 - 13 + 4) + (12 - 11 + 7)$$

$$= 18 \quad \text{additions/subtractions as expected.}$$

4.2 Symmetric Addition Flow Representations

The values of the source and sink sets in a flow representation exactly specify the parameters of the problem. For example, given a canonical representation $F = \langle G_1, G_2, G_3 \rangle$, the standard algorithm with representation F computes (m, n, p) products using t multiplications, where $t = |R_3|$,

$$n = \left(\frac{|R_1| \cdot |R_2|}{|S_3|} \right)^{\frac{1}{2}}, \quad m = \frac{|R_1|}{n}, \quad p = \frac{|R_2|}{n} .$$

Therefore, it is not surprising to find that operations on addition flow representations produce flow representation for matrix multiplication problems with modified parameters.

One such operation on F is taking the symmetric representation F^* to F.

Given an addition flow representation $F = \langle G_1, G_2, G_3 \rangle$, the symmetric addition flow representation F^* is defined by $F^* = \langle G_1^T, G_3^D, G_2^D \rangle$ where G_1^T is G_1 transposed, i.e. each vertex in G_1 is associated with a vertex in G_1^T where each occurrence of a_{ij} in the vertex name is replaced by a_{ji} for $1 \leq i \leq m$, $1 \leq j \leq n$. G_i^D denotes the directional dual of G_i with relabelling of the sink and source sets appropriate to the position of G_i^D within the ordered triple F^* (the directional dual of G is G with edge directions reversed). To illustrate how the problem parameters are altered, suppose we are given an addition flow representation $F = \langle G_1, G_2, G_3\rangle$ of an algorithm α which computes (m, n, p) products using t multiplications. Then, $|S_1| = |S_2| = |R_3| = t$, $|R_1| = mn$, $|R_2| = np$, and $|S_3| = mp$. We shall show that the standard algorithm with addition flow representation $F^* = \langle G_1^T, G_3^D, G_2^D\rangle$ computes (n, m, p) products using t multiplications.

First, however, consider how the parameters are modified by the symmetric canonical representation. Clearly, F^* would correspond to algorithms using t multiplications since if F^* is written $\langle G_1^*, G_2^*, G_3^*\rangle$, $|S_1^*| = |S_1| = t$, $|S_2^*| = |R_3| = t$, and $|R_3^*| = |S_2| = t$. Also, R_1^* represents the entries of an $(n\times m)$ matrix, since a_{ij} in R_1 is replaced by a_{ji} in R_1^*. Since $|R_2^*| = |S_3| = mp$ and R_2^* contains a b_{ij} for each y_{ij} in S_3, R_2^* represents the entries of an $(m\times p)$ matrix. Finally, S_3^* represents the entries of an $n\times p$ matrix since $|S_3^*| = |R_2| = np$ and each b_{ij} in R_2 is replaced by y_{ij} in S_3^*. Thus, the symmetric addition flow representation has appropriate structural properties for the study of symmetric computations.

A <u>symmetric algorithm</u> α^* to a given algorithm α which computes (m, n, p) products using t multiplications is an algorithm constructed from α by the techniques given in Section 3.1. In other words, by Lemma 2.2.3 and Theorem 3.1.1 we can decompose a given α into matrix products $WUVy$. Then, α^* is an algorithm which computes (n, m, p) products in t multiplications as $V^T U' W^T y$ where all matrices are defined and illustrated in Section 3.1.

Immediately from the definitions we have

Lemma 4.2.1: Every addition flow representation has a unique symmetric flow representation. Every algorithm has a symmetric algorithm associated with it, and this symmetric algorithm will generally be not unique.

The last assertion follows from the fact that the constructions of Section 3.1 do not constrain addition, subtraction, or scalar multiplication steps.

We proceed to show that the standard algorithm with canonical representation $F^*_\alpha = \langle G_1^T, G_3^D, G_2^D \rangle$ is a symmetric algorithm to the algorithm α with canonical addition flow representation $F_\alpha = \langle G_1, G_2, G_3 \rangle$.

One of the invariant properties of flow graphs G_i under transposition and directional dual is the connectedness of a particular vertex pair in $R_i \times S_i$. We tabulate this information in a matrix τ_i defined as follows:

Given a canonical addition flow $F = \langle G_1, G_2, G_3 \rangle$, for each component graph G_i of F we define a connection matrix τ_i with dimensions $|S_i| \times |R_i|$ where τ_{jk} has value $\sum_{p \in \wp} \prod_{\gamma \in P} w_\gamma$ where

$\mathcal{P}$ is the set of all paths P from r_k to s_j $(r_k \in R_i\ ,\ s_j \in S_i)$, p is the set of edges γ which form P , and w_γ is the scalar weight assigned to edge γ . For purposes of generality, we allow w_γ to take on arbitrary non-zero values; however, we need consider only values of 1 or -1 . If w_γ is not specified, it is assumed to be 1 . If $\mathcal{P}$ is empty, τ_{jk} is set to 0 .

To illustrate this simple concept, consider τ_{12} in the connection matrix $\mathcal{C}_2$ for $\hat{G}_2$ of the addition flow representation for $\hat{\alpha}_S$ given in Section 4.1. There are 2 paths from $r_2 = b_{12}$ to $S_1 = M_1^R$, namely $\gamma_1 \cdot \gamma_3$ and $\gamma_2 \cdot \gamma_4$.

Therefore

$$\begin{aligned} \tau_{12} &= w_{\gamma_1} \cdot w_{\gamma_3} + w_{\gamma_2} \cdot w_{\gamma_4} = 1 \cdot 1 + 1 \cdot (-1) \\ &= 0 \ . \end{aligned}$$

If we examine $M_1^R = b_{11} + b_{22}$, we see that the above computation of τ_{12} serves only to indicate that no b_{12} term appears in M_1^R .

Using the notion of connection matrices, we can now prove

<u>Lemma 4.2.2</u>: Let α be an algorithm which computes (m, n, p) products using t multiplication steps. Let $F_\alpha = \langle G_1, G_2, G_3 \rangle$ be the unique addition flow representation for α . If X,y are defined as in Section 3.1, then the matrices W, U, V such that $Xy = WUVy$ may be found directly from F_α.

<u>Proof:</u> Suppose α computes $A_{m\times n}B_{n\times p}$. By Section 3.1, $X = I_p \otimes A$, $y = \kappa(B)$.

Let x_1 be the mn row vector $(r_1^{(1)},\dots,r_{mn}^{(1)})$ where $r_i^{(j)} \in R_j$. Let $\mathfrak{C}_i$ be the connection matrix for G_i .

Then, $\mathfrak{C}_1 x_1$ yields the left sides of the multiplications $M_1,\dots,M_t$ in order. Therefore, let $U_{t\times t}$ be given by

$$u_{ii} = (\mathfrak{c}_{i1}^{(1)}, \mathfrak{c}_{i2}^{(1)}, \dots, \mathfrak{c}_{i,mn}^{(1)})x_1$$

$$u_{ij} = 0 \quad \text{for } i \neq j .$$

Similarly, $V_{t\times np} = \mathfrak{C}_2$ where the $r_j^{(2)}$ are specified in order of occurrence in $\kappa(B)$. Finally, $W_{mp\times t} = \mathfrak{C}_3$ with the same type of restriction on $s_j^{(3)}$. Clearly, as in Theorem 3.1.1, $Xy = WUVy$.

Thus, by simply altering the connection matrices appropriately, we can obtain an addition flow representation whose associated algorithm is a symmetric algorithm to α . The following lemma demonstrates that symmetric addition flow representations modify connection matrices in the required manner.

<u>Lemma 4.2.3:</u> Let α be an algorithm which computes $A_{m\times n}B_{n\times p}$ using t multiplications. Let $F = \langle G_1, G_2, G_3\rangle$ be the addition flow representation of α . Then, the symmetric flow representation $F^* = \langle G_1^T, G_3^D, G_2^D\rangle$ represents an algorithm α^* which is a symmetric algorithm to α (i.e. computes (n, m, p) products in t multiplications).

<u>Proof</u>: Let (n, m, p) products be represented by products $C_{n\times m}D_{m\times p} = Y'$. Let $X = I_p \otimes C_{n\times m}$, $y = \kappa(D)$.

Recall that by the definition of the symmetric flow, for $a_{ij} \in R_1$, $c_{ji} \in R_1^*$, $R_2^* = S_3$ with each y_{ij} in S_3 replaced by d_{ij} , and $S_3^* = R_2$ with b_{ij} replaced by y'_{ij} . S_1^*, S_2^*, R_3^* and S_1, R_3, S_2 have identical corresponding entries. Thus, for $F^* = \langle G_1^*, G_2^*, G_3^*\rangle$, the connection matrices are $\mathfrak{C}_1, \mathfrak{C}_3^T, \mathfrak{C}_2^T$ where $\mathfrak{C}_1, \mathfrak{C}_2, \mathfrak{C}_3$ are connection matrices for G_1, G_2, G_3 in F respectively.

By Lemma 4.2.2, Xy , where $E(Xy) = E(Y')$, may be computed as $W^*U^*V^*y$ where

$$u_{ii}^* = (\mathfrak{C}_{i1}^{(1)}, \mathfrak{C}_{i2}^{(1)}, \ldots, \mathfrak{C}_{i,mn}^{(1)})x_1'$$

where x_1' is x_1 with a_{ij} replaced by $\mathfrak{C}_{ji}$, $u_{ij}^* = 0$ for $i \neq j$. But this U^* is just U calculated from α as in Section 3.1 with transposed entries, and by our conventions, is denoted U' . By Lemma 4.2.2, $W^* = \mathfrak{C}_2^T$ and $V^* = \mathfrak{C}_3^T$. Therefore the standard algorithm α^* with symmetric flow representation F^* computes Xy as $W^*U^*V^*y = \mathfrak{C}_2^T U' \mathfrak{C}_3^T y$. But α computed $E(A_{m\times n}B_{n\times p})$ as $WUVy$ where $W = \mathfrak{C}_3$, $V = \mathfrak{C}_2$. Therefore α^* computes $E(Xy)$ as $\mathfrak{C}_2^T U' \mathfrak{C}_3^T y = V^T U' W^T y$. Therefore α^* is a symmetric algorithm to α as required.

This completes the justification of addition flows as a model for analysing the complexity of symmetric computations. The above results show that the addition flow models contain all the information about multiplicative complexity contained in previous models (such as the matrix product decomposition of Section 3.1) as well as encoding all the non-multiplication steps in algorithms.

The next section focuses on the additive complexity of symmetric addition flows.

4.3 The Additive Symmetry Theorem

Thus far, only Lemma 4.1.5 has dealt specifically with additive complexity. Before we proceed with the main result, we need to present an addition flow analogue to the construction of Lemma 2.3.3.

A transposed addition flow representation F^T of a canonical representation F of an algorithm α which computes (m, n, p) products in t multiplications, where $F = \langle G_1, G_2, G_3 \rangle$, is given by $F^T = \langle G_2^T, G_1^T, G_3^T \rangle$ where G_3^T is G_3 with each y_{ij} in S_3 replaced by y_{ji} and G_1^T, G_2^T treated similarly.

By an argument based on connection matrices as in the previous section, or merely by inspection, we have

Lemma 4.3.1: If α computes (m, n, p) products in t multiplications and a additions/subtractions, then the standard algorithm with addition flow representation $(F_\alpha)^T$ computes (p, n, m) products in t multiplication and a addition/subtraction steps .

Comparing symmetric flows we obtain

Lemma 4.3.2: Let α be an algorithm for computing (m, n, p) products in t multiplications and a additions. Then, the standard algorithm with canonical flow representation $(F_\alpha)^*$ computes (n, m, p) products using t multiplications and $a + (m - n)p$ additions/subtractions.

<u>Proof:</u> Let $F = \langle G_1, G_2, G_3 \rangle$ be the addition flow representation of α , $F^* = \langle G_1^*, G_2^*, G_3^* \rangle$ be the symmetric flow representation to F . Then, if α^* is the standard algorithm with canonical representation F^* ,

$$D = \text{nadds } (\alpha^*) - a = \sum_{i=1}^{3} [\text{nadds } (G_i^*) - \text{nadds } (G_i)]$$

Clearly, $\text{nadds } (G_1^*) = \text{nadds } (G_1)$. Therefore

$$D = \sum_{i=2}^{3} [\text{nadds } (G_i^*) - \text{nadds } (G_i)]$$

Expanding $\text{nadds } (G_i)$ by Lemma 4.1.5, we have

$$D = \sum_{i=2}^{3} [|\Gamma G_i^*| - |VG_i^*| + |R_i^*|] - \sum_{i=2}^{3} [|\Gamma G_i| - |VG_i| + |R_i|]$$

But, $|\Gamma G_2^*| = |\Gamma G_3|$, $|\Gamma G_3^*| = |\Gamma G_2|$. Also, $|VG_2^*| = |VG_3|$, $|VG_3^*| = |VG_2|$. Therefore

$$D = |R_2^*| + |R_3^*| - |R_2| - |R_3|$$

But $|R_2^*| = |S_3| = mp$, and $|R_3^*| = |S_2| = t$. Therefore

$$\begin{aligned} D &= mp + t - np - t \\ &= (m - n)p \quad \text{as required.} \end{aligned}$$

The following lemma provides the final connection between addition flow representation operators, and derivations of equicomplex algorithms for symmetric problems from α for (m, n, p) products.

Lemma 4.3.3: Suppose α is an algorithm for (m, n, p) products with addition flow representation $F_\alpha = \langle G_1, G_2, G_3 \rangle$. Then, if $\hat{\alpha}$ is any algorithm for one of the 5 symmetric computations ((n, m, p), (p, m, n), (m, p, n), (n, p, m), (p, n, m) products) which is derived from α by the techniques of Section 3.1, there exists an addition flow representation F_0 which is found by taking a sequence of transposed or symmetric representations of F_α such that α_0 , the standard algorithm represented by F_0 , performs the same computation as $\hat{\alpha}$ (i.e., computes the same sums of the same multiplication as $\hat{\alpha}$).

Proof: By Lemma 4.2.3, if we derive a (v, u, w) product algorithm α_i from an algorithm α_j for (u, v, w) products by the technique in Section 3.1, then, the standard algorithm α_k with addition flow representation $(F_\alpha)^*$ has an identical set of multiplication steps to that of α_i and, as well, computes the same sums of multiplications as α_i . By Lemma 4.3.1, and an examination of the construction of Section 3.1 based on Lemma 2.3.3, the analogous result occurs when we derive an algorithm α_i for (w, v, u) products from an algorithm α_j for (u, v, w) products. In other words, the standard algorithm with representation $(F_{\alpha_j})^T$ uses the same multiplication steps as α_i and computes the same sums of these multiplications as α_i .

But any derivation in Section 3.1 is simply an alternation of constructing symmetric and transposed algorithms. This, the corresponding alternating sequence of constructing symmetric and transposed flow representations of F_α results in the desired canonical representation F_0 .

Thus, we have a strict predicatable relationship between the number of additions/subtractions in a derived addition flow representation and the number of additions/subtractions in the original algorithm. A stronger form of this statement is

<u>Theorem 4.3.4</u> (Additive Symmetry): Let α be an additively optimal t-multiplication (bilinear) algorithm for (m, n, p) products with (unique) addition flow representation $F_\alpha = \langle G_1, G_2, G_3 \rangle$. Then, let $\hat{\alpha}$ be an algorithm derived from α to compute (u, v, w) products, one of the 5 symmetric problems to (m, n, p) products. Let $\hat{F}$ be the addition flow representation derived from F_α according to the derivation of $\hat{\alpha}$ from α . Then, the standard algorithm α_0 with representation $\hat{F}$, is additively optimal over the class of all algorithms in NC for (u, v, w) products which use the t multiplications of $\hat{\alpha}$.

<u>Proof</u>: By Lemma 4.3.3, α_0 computes (u, v, w) products using the t multiplication steps of $\hat{\alpha}$. By alternating applications of Lemmas 4.3.1 and 4.3.2 according to the derivation of $\hat{F}$ from F_α , we see that the value nadds (α) - nadds (α_0)

is calculable and equal to x , say. Clearly, this process is reversible, i.e., assume nadds $(\hat{\alpha})$ < nadds (α_0) , or in other words that α_0 is not additively optimal. Then, the cost of performing the "reverse" operations on representations to get an algorithm α_1 for (m, n, p) products is $(-x)$. In other words,

$$\begin{aligned} \text{nadds}\ (\alpha_1) &= \text{nadds}\ (\hat{\alpha}) - (-x) \\ &< \text{nadds}\ (\alpha_0) + x \\ &= \text{nadds}\ (\alpha) - x + x \\ &= \text{nadds}\ (\alpha) \end{aligned}$$

Therefore, nadds (α_1) < nadds (α) . But this contradicts α's additive optimality. Therefore α_0 for (u, v, w) products is additively optimal.

Thus, if any of the 6 related algorithms are additively optimal, they all are. If we are given an algorithm α which is not known to be additively optimal for its set of multiplications, we may be able by transforming F_α to either prove its optimality by the Additive Symmetry Theorem or to find an improvement of a derived algorithm, and pass this improvement back to α via a sequence of operations on addition flow representations. For this purpose, a table of the corresponding additive complexities are given below.

Corollary 4.3.5: Suppose a given algorithm α computes (m, n, p) products using $\underline{a}$ additions and subtractions. The additive costs of each of the derived representations of F_α is:

$$\begin{aligned}
(p, n, m) &\longrightarrow a \\
(n, p, m) &\longrightarrow a + (p - n)m \\
(m, p, n) &\longrightarrow a + (p - n)m \\
(p, m, n) &\longrightarrow a + (m - n)p \\
(n, m, p) &\longrightarrow a + (m - n)p
\end{aligned}$$

Proof: The respective additive costs follow from appropriately alternating applications of Lemmas 4.3.1 and 4.3.2. For example, if F_α is the addition flow representation of α, by Lemma 4.3.1, $(F_\alpha)^T$ represents the computation of (p, n, m) products using $\underline{a}$ additions/subtractions. By Lemma 4.3.2, $((F_\alpha)^T)^*$ represents the computation of (n, p, m) products using $(p - n)m$ more addition/subtraction steps, i.e. altogether using $a + (p - n)m$ additions and subtractions. The remaining 3 cases are proved similarly.

Implications of this result are presented in the next chapter.

CHAPTER V

IMPLICATIONS OF ADDITIVE SYMMETRY

In this chapter we apply Additive Symmetry to obtain immediately (previously known) additive complexities for inner products and matrix-vector products of $n-1$ and $mn-n$ additions/subtractions respectively. The main result is that 15 addition/subtraction steps are necessary and sufficient to compute $(2, 2, 2)$ products by 7-multiplication algorithms. ("necessary" holds only for $K \subseteq Z$).

For convenience, we denote the additive complexity of computing (m, n, p) products employing a set $M = \{M_1, \cdots, M_t\}$ of t multiplications by $\mathcal{A}(m, n, p, M)$.

Note that M implicitly contains sufficient information to yield the parameters m, n, p of the matrix multiplication problem in the correct order. For example, since M computes each element in the product matrix, the lefthand factors of multiplications in M encode the first two parameters as the highest row and the highest column index, respectively, on individual terms. Thus, we can denote the additive complexity of a related problem by $\mathcal{A}(u, v, w, M)$ where (u, v, w) is a permutation of (m, n, p) , since M encodes all the parameters of the original problem ((m, n, p) products). More precisely, $\mathcal{A}(u, v, w, M)$ is the additive complexity of computing (u, v, w) products using a set $\hat{M}$ of multiplications derived from M by the method illustrated in Section 3.1.

Finally, for any positive integer t , $\mathcal{A}(m, n, p, t)$ denotes the additive complexity of computing (m, n, p) products using no more than t multiplications. In particular, if $t = \mathcal{M}(m, n, p)$, then $\mathcal{A}(m, n, p, t)$ is the number of addition/ subtraction steps required by any multiplicatively optimal algorithm for (m, n, p) products.

5.1 Immediate Applications of Additive Symmetry

By Lemma 4.3.3, rather than studying algorithms derived from a given algorithm α, we can restrict our investigation to addition flow representations derived from the representation F of α. Moreover, a nice feature of flow representations is that the operations $*$, T are clearly reversible. In other words, $(F^*)^* = (F^T)^T = F$. Thus, the order of derivation of an addition flow representation $\hat{F}$ (derived from F) does not affect the additive cost of the computation represented by $\hat{F}$ relative to the additive cost represented by F. We illustrate this in the following example.

Example 5.1.1: Given a (bilinear) α which computes (m, n, p) products using the t multiplications in $M = \{M_1, \cdots, M_t\}$ and $\underline{a}$ addition/subtraction steps, we wish to determine the additive cost of a derived algorithm $\hat{\alpha}$ for (p, m, n) products. Let F be the addition flow representation of α. Then, the standard algorithm for $\hat{F}$ where $\hat{F}$ is either $(F^*)^T$ or $(((F^T)^*)^T)^*$ computes (p, m, n) products using t multiplications. If $\hat{F}$ is computed as $(F^*)^T$, then

$$\begin{aligned} \text{nadds}\,(\hat{F}) &= \text{nadds}\,[(F^*)^T] \\ &= \text{nadds}\,(F^*) \quad \text{(by Lemma 4.3.1)} \\ &= a + (m - n)p \quad \text{(by Lemma 4.3.2)}\,. \end{aligned}$$

If $\hat{F}$ is $(((F^T)*)^T)*$, then

$$\begin{aligned} \text{nadds}\ (\hat{F}) &= \text{nadds}\ [(((F^T)*)^T)*] \\ &= \text{nadds}\ [((F^T)*)^T] + (m - p)n && \text{(by Lemma 4.3.2)} \\ &= \text{nadds}\ [(F^T)*] + (m - p)n && \text{(by Lemma 4.3.1)} \\ &= \text{nadds}\ (F^T) + (p - n)m + (m - p)n && \text{(by Lemma 4.3.2)} \\ &= \text{nadds}\ (F) + (p - n)m + (m - p)n && \text{(by Lemma 4.3.1)} \\ &= a + (m - n)p && \text{as before.} \end{aligned}$$

Thus, the additive cost of the derived algorithm $\hat{\alpha}$ for (p, m, n) products is $a + (m - n)p$ and is independent of the order in which $\hat{\alpha}$ is derived, as expected.

As well, we can easily use the Additive Symmetry Theorem (4.3.4) to show that the size of the inner dimension in any of the six related or symmetric matrix multiplicative problems directly affects the additive complexity.

Lemma 5.1.2: For any set $M = \{M_1, \cdots, M_t\}$ of t multiplications which can be used (by K-bilinear chains) to compute (m, n, p) products, $\mathcal{A}(u, v, w, M)$ is largest for all permutations (u, v, w) of (m, n, p) when $v = \max\{m, n, p\}$, and smallest when $v = \min\{m, n, p\}$.

<u>Proof</u>: Let $v = \max\{m, n, p\}$. Then, since $u \leq v$ and $w \leq v$, $(u - v)w \leq 0$ and $(w - v)u \leq 0$. Let $\mathcal{A}(u, v, w, M) = \underline{a}$ additions/subtractions. By Corollary 4.3.5,

$$\begin{aligned}
\mathcal{A}(w, v, u, M) &= a\ , \\
\mathcal{A}(v, u, w, M) &= \mathcal{A}(w, u, v, M) \\
&= a + (u - v)w \\
&\leq a\ , \\
\text{and} \quad \mathcal{A}(u, w, v, M) &= \mathcal{A}(v, w, u, M) \\
&= a + (w - v)u \\
&\leq a \quad \text{as required.}
\end{aligned}$$

Similarly, if $v = \min\{m, n, p\}$, $(u - v)w \geq 0$ and $(w - v)u \geq 0$. Let $\mathcal{A}(u, v, w, M) = a$. Therefore, by Corollary 4.3.5,

$$\mathcal{A}(v, u, w, M),\ \mathcal{A}(w, u, v, M),\ \mathcal{A}(u, w, v, M),\ \mathcal{A}(v, w, u, M) \geq a\ .$$

This lemma supports the intuitive feeling that for a fixed number of multiplications, the size of the inner product directly influences the additive complexity of a matrix multiplication problem. In other words, for a given (possibly optimal) set of multiplications, derived computations of a few additively complicated product elements have higher additive complexity than derived computations of a large number of product elements each of which are sums of relatively few terms.

A final application of Additive Symmetry is to any problem which is one of the five symmetric problems to a computation which has a known additive complexity. In particular, the following application of Corollary 4.3.5 was suggested to the author by C. Fiduccia.

There are some matrix multiplication computations, for example, computing $(n, 1, 1)$ and $(m, 1, n)$ products, which can be carried out without employing any additions or subtractions. Since 0 is a trivial lower bound on the additive complexity of any matrix multiplication problem, computations which achieve this lower bound are optimal. Then, Corollary 4.3.5 yields the additive complexity of any symmetric problem.

For example, Winograd [W2] and others have shown that the obvious algorithms for $(n, 1, 1)$ and $(m, 1, n)$ products are multiplicatively optimal.

Let M^1, M^2 be the respective multiplication sets. Then, $\mathcal{A}(n, 1, 1, M^1) = \mathcal{A}(n, 1, 1, n) = 0$, and similarly, $\mathcal{A}(n, 1, n, M^2) = \mathcal{A}(m, 1, n, mn) = 0$. Hence we have

<u>Lemma 5.1.3</u>: $(1, n, 1)$ (inner) products and $(m, n, 1)$ or $(1, n, m)$ (matrix vector) products require $n-1$ and $mn-n$ additions/subtractions respectively.

<u>Proof</u>: By Corollary 4.3.5,

$$\mathcal{A}(1, n, 1, M') = \mathcal{A}(n, 1, 1, M^1) + (n-1)1$$
$$= 0 + (n-1) = n-1 \quad \text{as required.}$$

Similarly,

$$\begin{aligned}\mathcal{A}(1,\ n,\ m,\ M^2) &= \mathcal{A}(n,\ 1,\ n,\ M^2) + (n-1)m \\ &= 0 + (mn - n) \qquad \text{as required.}\end{aligned}$$

The above additive complexities are not new, and have been proved by [illegible] independence arguments. However, they are immediate from the additive symmetry results.

In the next two sections, we obtain the additive complexity of (2, 2, 2) products for any K-bilinear algorithm, where K is a subring of the centre of Z (the integers).

5.2 Transformations of Fast (2, 2, 2) Algorithms

Recall from Chapter I that a fast algorithm for (m, n, p) products is a K-bilinear chain which uses fewer than mnp multiplications to compute (m, n, p) products. Thus, fast $(2, 2, 2)$ algorithms are multiplicatively optimal by Theorem 1.1.5 and use exactly 7 multiplications.

Following [H1] and [H2], we define a group τ of transmations on algorithms which compute $A_{2\times2}B_{2\times2} = Y_{2\times2}$.

Let Z be the ring of integers. Then, τ is the group of transformations $T : \{a_{ij}\} \to \mathcal{L}_Z\{a_{ij}\}$

and $\{b_{ij}\} \to \mathcal{L}_Z\{b_{ij}\}$,

which is generated by transformations that act on A and/or B in one of 4 ways:

1) interchange 2 rows of A , 2 columns of B , or both columns of A and both rows of B ;

For $i \neq j$,

2) add (subtract) row i of A to (from) row j of A ;

3) add (subtract) column i of B to (from) column j of B ;

or 4) add (subtract) column i of A to (from) column j of A and subtract (add) row j of B from (to) row i of B .

Intuitively, if α is an algorithm for $(2, 2, 2)$ products, each multiplication M_i of α will be of the form $\mathcal{L}_Z\{a_{ij}\}\cdot\mathcal{L}_Z\{b_{ij}\}$. This restriction is necessary to prove lower bounds.

For any transformation $T \in \tau$, $T(\alpha)$ is the algorithm obtained from α by replacing each multiplication $M_i = M_i^L \cdot M_i^R$ by $T(M_i) = T(M_i^L) \cdot T(M_i^R)$.

For purposes of generality, we may note that τ may be defined as the group of transformations over the computation $A_{m\times 2} B_{2\times n} = Y_{m\times n}$ with no change in the above definition. However, we will only be interested in the case $m = n = 2$.

For notational convenience we define <u>type-1</u> and <u>type-2</u> multiplications.

First, as in [H1], define an equivalence relation between multiplications as

$$M_i \equiv_\tau M_j \quad \text{if} \quad \exists T \in \tau \quad \text{such that} \quad T(M_i^L) = (M_j^L) \quad .$$

Then, <u>a type-1 multiplication</u> is any M_i which is τ-equivalent to a_{11} .

A <u>type-2 multiplication</u> is any M_i which is τ-equivalent to $a_{11}+a_{22}$.

Note that these definitions invite a generalization to type-n multiplications (equivalent to $a_{11} + a_{22} + \cdots + a_{nn}$) in algorithms for general (n, n, n) products. However, the general case appears too difficult for present techniques.

In any case, for (m, 2, n) products, we can classify all possible multiplications as being either type-1 or type-2. An informal classification scheme is to check M_i^L for "diagonality", i.e.

$$M_i^L = a_{11} + a_{12} - a_{21} \quad \text{is diagonal, whereas}$$
$$M_i^L = a_{11} + a_{12} + a_{21} - a_{22} \quad \text{is not.}$$

Diagonality implies that we cannot simultaneously affect all terms of M_i^L by a transformation in τ. Therefore, diagonal multiplications are type-2; all others are type-1.

Thus, an essential observation about the group τ is that its transformations must preserve the type of multiplications (by the group property). In [H2], Hopcroft and Musinski prove the following theorem:

Theorem 5.2.1: Given a 7-multiplication (fast) $\{0, 1, -1\}$-bilinear α which computes (2, 2, 2) products, there exists a transformation T in τ such that $T(\alpha) = \alpha_S$, i.e. if M is the set of multiplications of α, $T(M) = M_S$. Thus, Strassen's algorithm is unique to within a transformation of τ.

Moreover, if we examine M_S (given in Section 4.1), we note that $M_1 = (a_{11} + a_{22})(b_{11} + b_{22})$ is type-2 and all 6 other M_i's are type-1 multiplications. By Theorem 5.2.1 then, any fast

(7-multiplication) algorithm for (2, 2, 2) products must contain 6 type-1 multiplications, and 1 type-2 multiplication, namely $T^{-1}(m_1)$. This observation is central to the analysis in the next section of the additive complexity of fast (2, 2, 2) computations.

In order to justify relating Theorem 5.2.1 to bilinear chains with integer coefficients in Lemma 5.3.13, we note

Corollary 5.2.2: Given any K-bilinear algorithm α , where K is a subring of the centre of Z , the field of integers, such that α computes (2, 2, 2) products using 7 multiplications and $\underline{a}$ addition/subtraction steps; there exists a {0, 1, -1}-bilinear algorithm $\hat{\alpha}$ which computes (2, 2, 2) products using 7 multiplications and no more than $\underline{a}$ addition/subtraction steps.

Proof: By the definition of K-bilinear chains in Section 2.1, each product element y_{ij} is computed as $\sum_{k=1}^{7} g_k M_k$ where $\{M_1, \cdots, M_7\}$ is the set of 7 multiplications used by α . Taking all q_k modulo 2 (possibly changing signs, but not adding operands in the sum) yields $y_{ij} = \sum_{k=1}^{7} q'_k M_k$ where $q'_k \in \{0, 1, -1\}$. Now, we can rewrite each M_k as $\mathcal{L}_{K'}\{a_{ij}\} \cdot \mathcal{L}_{K'}\{b_{ij}\}$ as before, by using as coefficients, the original coefficients modulo 2, possibly with sign changes, where K' = {0, 1, -1} . Call this new bilinear algorithm $\hat{\alpha}$ - clearly $\hat{\alpha}$ uses no more than $\underline{9}$ additions/subtractions as required.

Finally, note that the transformations generating τ act as linear transformations on sums of a_{ij}'s and b_{ij}'s . Therefore, for any constant c and $\ell(A) \in \mathcal{L}_K\{a_{ij}\}$, $T \in \tau$ implies that

$$T(c \cdot \ell(A)) = c \cdot T(\ell(A)) \quad .$$

Then, by the group property of τ , and since multiplications $M_i, M_j \in M_S$ where $i \neq j$ cannot be written as $M_i = c \cdot M_j$ for any constant c , we have

<u>Lemma 5.2.3</u>: For any K-bilinear chain α for (2, 2, 2) products which employs the 7 multiplications in $M_\alpha = \{M_1, \cdots, M_7\}$, $i \neq j$ implies that $M_i^L \neq c \cdot M_j^L$ for any constant c .

5.3 The Additive Complexity of Fast (2, 2, 2) Computations

Winograd [W4] and Hopcroft and Kerr [H1] have shown that Strassen's [S1] algorithm α_S is multiplicatively optimal for computing (2, 2, 2) products. A natural related problem is whether α_S is additively optimal.

By inspection, α_S uses 18 addition/subtraction steps: 5 additions/subtractions to form each of M_S^L and M_S^R (lefthand and righthand factors respectively of the 7 multiplications in α_S), and 8 additions/subtractions to compute y_{ij}, $1 \leq i, j \leq 2$, from the 7 multiplications. In fact we note that since each M_S^L and M_S^R involve 5 different sums, at least 5 addition/subtraction steps are required for computing each set M_S^L, M_S^R of expressions. We generalize this notion in the following obvious lemma.

We define a sum to be an expression of the form $S = \sum_{i=1}^{n} c_i x_i$ where $n \geq 1$, each c_i is a non-zero constant, and the x_i's are non-zero, distinct terms.

S is a trivial sum if $n = 1$, and a non-trivial sum otherwise.

Two sums S_1, S_2 are distinct if neither sum is a constant multiple of the other.

Then, we have

Lemma 5.3.1: At least a addition/subtraction steps are required to compute a distinct, non-trivial sums by any scheme which employs only addition, subtraction, or multiplication by a constant.

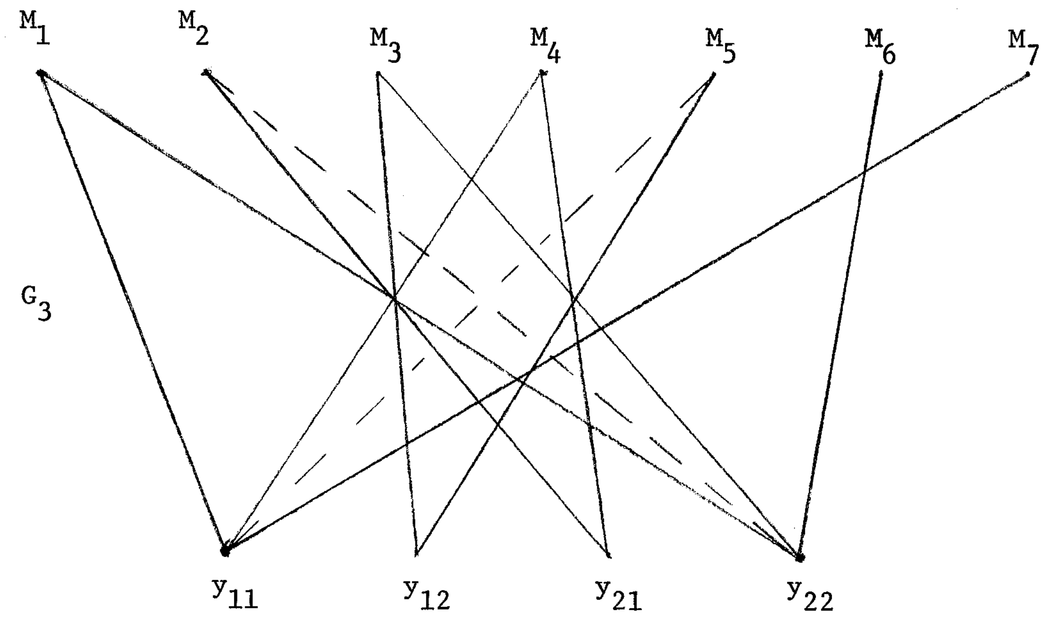
<u>Proof</u>: Each non-trivial sum cannot be obtained by multiplication of a constant by a term. Each sum requires at least 1 addition or subtraction. Since the <u>a</u> sums are distinct, <u>a</u> addition/subtraction steps are required.

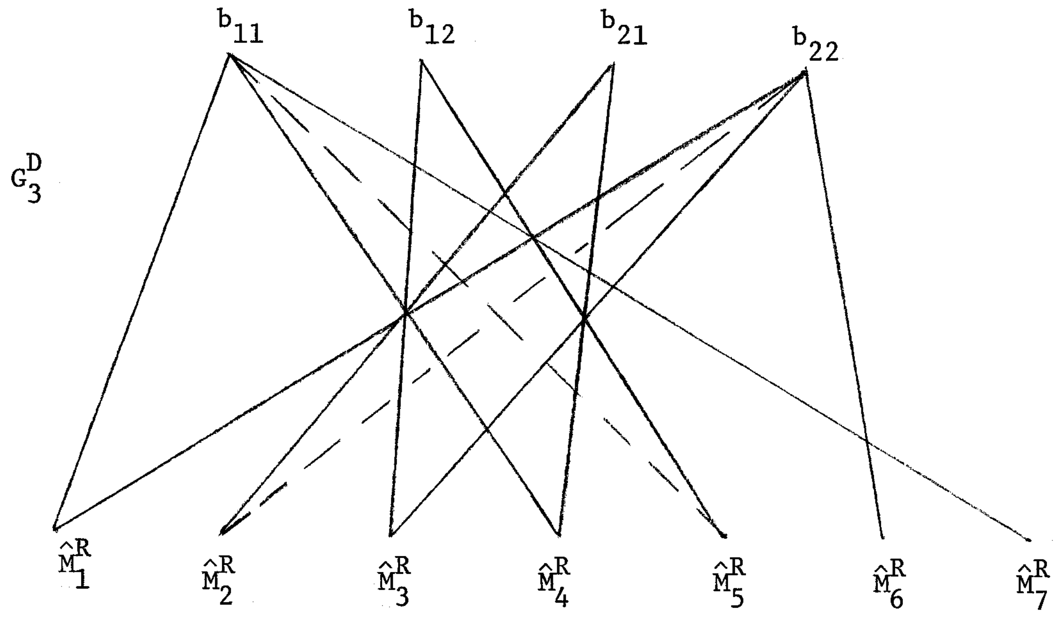
Thus, at least 10 addition/subtraction steps are required to form lefthand and righthand factors of the multiplications in M_S.

Let $F_S = <G_1, G_2, G_3>$,be ~~t~~the addition flow representation of α_S (G_1, G_2, G_3 are drawn in Section 4.1). By the Additive Symmetry Theorem (4.3.4), we have

<u>Lemma 5.3.2</u>: If G_i is a component in an addition flow representation $<G_1, G_2, G_3>$, $i = 2$ or 3 , then nadds (G_i) is minimum if and only if nadds (G_i^D) is minimum.

In particular, G_3 of F_S represents an additively optimal computation if and only if G_3^D represents an additively optimal computation. G_3 and G_3^D are drawn below.


M_1
M_2
M_3
M_4
M_5
M_6
M_7
G_3
y_{11}
y_{12}
y_{21}
y_{22}


b_{11}
b_{12}
b_{21}
b_{22}
G_3^D
$\hat{M}_1^R$
$\hat{M}_2^R$
$\hat{M}_3^R$
$\hat{M}_4^R$
$\hat{M}_5^R$
$\hat{M}_6^R$
$\hat{M}_7^R$

$$\text{nadds } (G_3^D) = |\Gamma G_3| - |VG_3| + |S_3|$$
$$= 5 \ .$$

By inspection, G_3^D represents the computation of

$\{(b_{11} + b_{22}), (b_{21} - b_{22}), (b_{12} + b_{22}), (b_{11} + b_{21}), (-b_{11} + b_{12}), b_{22}, b_{11}\}$,

a set of 5 non-trivial, distinct sums, using 5 addition/subtraction steps. Therefore, G_3^D represents an additively optimal computation by Lemma 5.3.1. Hence, G_3 represents an additively optimal computation by Lemma 5.3.2.

But G_3 represents a scheme to compute y_{ij} , $1 \leq i,j \leq 2$, from multiplications in M_S . Therefore, 8 addition/subtraction steps are required for any computation of y_{ij}'s from multiplications in M_S , and we have

<u>Lemma 5.3.3</u>: $\mathcal{A}(2, 2, 2, M_S) = 18$.

In other words, for Strassen's [S1] choice of multiplications, the obvious algorithm α_S is additively optimal. To test whether M_S is as good a good a choice of multiplications for (2, 2, 2) products as possible, by Theorem 5.2.1 we need examine only sets of multiplications which are transformations of M_S .

During such investigations, a fast algorithm for (2, 2, 2) products which used only 16 addition/subtraction steps was found. Winograd [W5] improved upon this by discovering essentially the following 15-addition/subtraction algorithm α_W .

Compute $A_{2\times2}B_{2\times2} = Y_{2\times2}$ as follows:

Form

$$\begin{array}{llll} s_1 \leftarrow a_{21} + a_{22} & & s_5 \leftarrow b_{12} - b_{11} \\ s_2 \leftarrow s_1 - a_{11} & & s_6 \leftarrow b_{22} - s_5 \\ s_3 \leftarrow a_{11} - a_{21} & & s_7 \leftarrow b_{22} - b_{12} \\ s_4 \leftarrow a_{12} - s_2 & & s_8 \leftarrow s_6 - b_{21} \end{array}$$

using 8 additions and subtractions.

Then calculate the following 7 multiplications:

$$M_W \quad \begin{array}{llll} M_1 \leftarrow s_2 \cdot s_6 & & M_4 \leftarrow s_3 \cdot s_7 \\ M_2 \leftarrow -a_{11} \cdot b_{11} & & M_5 \leftarrow s_1 \cdot s_5 \\ M_3 \leftarrow a_{12} \cdot b_{21} & & M_6 \leftarrow s_4 \cdot b_{22} \\ & & M_7 \leftarrow a_{22} \cdot s_8 \end{array}$$

Finally, using only 7 more additions/subtractions, form

$$\begin{array}{llll} s_{10} \leftarrow M_3 - M_2 & & s_{14} \leftarrow s_{13} + M_6 \\ s_{11} \leftarrow M_1 - M_2 & & s_{15} \leftarrow s_{12} - M_7 \\ s_{12} \leftarrow s_{11} + M_4 & & s_{16} \leftarrow s_{12} + M_5 \\ s_{13} \leftarrow s_{11} + M_5 & & \end{array}$$

Then,

$$\begin{array}{llll} y_{11} = s_{10} & & y_{12} = s_{14} \\ y_{21} = s_{15} & & y_{22} = s_{16} \ . \end{array}$$

Accordingly, we have

Lemma 5.3.4: Strassen's [S1] algorithm α_S is not additively optimal for fast (2, 2, 2) computations.

Corollary 5.3.5: $\mathcal{A}(2, 2, 2, 7) \leq \mathcal{A}(2, 2, 2, M_W) \leq 15$.

Thus, there are sets of 7 multiplications which can be formed and combined to yield (2, 2, 2) products using only 15 addition/subtraction steps.

The reader may verify that the set M_W of multiplications may be written $M_W = T(M_S)$ where $T \in \tau$ is defined as follows $(1 \leq i,j \leq 2)$:

Let
$$T_0 : \begin{aligned} a_{i1} &\to a_{i1} - a_{i2} \\ b_{2j} &\to b_{2j} + b_{1j} \end{aligned} \quad , \qquad T_1 : \begin{aligned} a_{i2} &\to a_{i2} + a_{i1} , \\ b_{1j} &\to b_{1j} - b_{2j} , \end{aligned}$$

$$T_2 : \begin{aligned} a_{i1} &\to a_{i2} \\ a_{i2} &\to a_{i1} \\ b_{1j} &\to b_{2j} \\ b_{2j} &\to b_{1j} \end{aligned} \quad , \qquad \begin{aligned} T_3 : b_{i1} &\to b_{i1} - b_{i2} \\ \\ T_4 : b_{i2} &\to b_{i1} \\ b_{i1} &\to b_{i2} \quad . \end{aligned}$$

Then, set $T = T_4 \cdot T_3 \cdot T_2 \cdot T_1 \cdot T_0$.

Note that T maps the product elements y_{ij} , $1 \leq i,j \leq 2$, computed by α_S into the following form:

$$\begin{aligned} y_{i1} &\to y_{i2} \\ y_{i2} &\to y_{i1} - y_{i2} \quad . \end{aligned}$$

Consequently, any algorithm for $(2, 2, 2)$ products which uses the multiplications in M_W must compute the product elements in a manner essentially different from α_S. For example, α_W uses only 7 addition/subtraction steps to form the y_{ij}'s from its set of multiplications; α_S uses 8 additions and subtractions to form the y_{ij}'s from M_S.

By employing α_W recursively as in [S1], we can obtain a new upper bound on the number of total arithmetics necessary for computing (n, n, n) products.

<u>Lemma 5.3.6</u>: Two matrices of order n can be multiplied in approximately $4.57\, n^{\log 7}$ total arithmetic operations (all logarithms, unless otherwise specified, are base 2; $\log 7$ is approximately 2.8).

<u>Proof</u>: Let $C_M(n)$, $C_A(n)$ be the multiplicative and additive cost respectively, of multiplying two $n\times n$ matrices where $n = m\cdot 2^k$ using algorithm α_W. Then, $C_M(m\cdot 2^k) = 7\, C_M(m\cdot 2^{k-1})$ and $C_M(m) = m^3$. Therefore, $C_M(m\cdot 2^k) = m^3 7^k$.

$C_A(m\cdot 2^k) = 7\cdot C_A(m\cdot 2^{k-1}) + 15(m\cdot 2^{k-1})^2$ and $C_A(m) = m^2(m-1)$.

$$\begin{aligned}
\text{Therefore,}\qquad C_A(m\cdot 2^k) &= 7^k m^2(m-1) + 15\, m^2 \sum_{i=1}^{k} 7^{k-i}\, 4^{i-1} \\
&= 7^k m^2(m-1) + \frac{15}{4}\, m^2 7^k \sum_{i=1}^{k} \left(\frac{4}{7}\right)^i \\
&= 7^k m^2(m-1) + 5\, m^2 7^k - 5\, m^2 4^k \\
&= m^2((m+4)7^k - 5\cdot 4^k)
\end{aligned}$$

Thus, the total arithmetic cost of using α_W is $m^2((2m + 4)7^k - 5\cdot 4^k)$. Then, imbed matrices of order n into matrices of order $m2^k$ by setting $k = \lfloor \log n - 4 \rfloor$, $m = \lceil n2^{-k} \rceil$. Then, the total number of arithmetics needed is $< 7^k m^2 \, 2(m + 2)$

$$\leq 7^k(n2^{-k} + 1)^2 \cdot 2(n2^{-k} + 3)$$
$$= \frac{7^k}{2^{3k}} (n^2 + 2n2^k + 2^{2k})(2n + 6\cdot 2^k)$$
$$= 2n^3(\tfrac{7}{8})^k + 10\, n^2(\tfrac{7}{4})^k + 14\, n(\tfrac{7}{2})^k + 6\cdot 7^k$$

Since $16\; 2^k \leq n$, $14\, n(\frac{7}{2})^k + 6\cdot 7^k \leq .898\, n^2(\frac{7}{4})^k$. Therefore the total number of arithmetics is

$$\leq 2\, n^3(\tfrac{7}{8})^k + 10.898\, n^2(\tfrac{7}{4})^k$$
$$= \left[2(\tfrac{8}{7})^{\log n-k} + 10.898(\tfrac{4}{7})^{\log n-k}\right]7^{\log n}$$
$$\leq \max_{4\leq t\leq 5} \left[2(\tfrac{8}{7})^t + 10.898(\tfrac{4}{7})^t\right]n^{\log 7}$$
$$= 4.57388\, n^{\log 7}$$

since the expression is convex in this region (this value occurs at $t = 4$).

Thus, implementing α_W recursively rather than α_S saves about $.13\, n^{\log 7}$ arithmetic operations. Indeed, if we implement α_W in a slightly different manner [F3] than done for α_S in [S1], we arrive at a saving of about $.15\, n^{\log 7}$ operations. This is achieved by letting $k = \lfloor \log n - 5.04 \rfloor$, $m = \lceil n \cdot 2^{-k} \rceil$ $(5.04 = \log 33)$.

This kind of a priori implementation must necessarily be less than optimal most of the time, since m , and k are chosen without considering any numerical properties of n . For instance, if n is a large power of two, we can search for optimal choices of m , and k by tabulating the coefficients of $n^{\log 7}$ as follows.

For $n = 2^p$ very large, the additive cost of employing α_W recursively is essentially $m^2(m + 4)7^k$. Then, if we examine only values of m which are powers of 2, say $m = 2^i$, $i \geq 0$,

$$\begin{aligned} C_A(2^p) &= m^2(m + 4)7^{p-i} \\ &= \frac{m^2(m + 4)}{7^i} n^{\log 7} \quad . \end{aligned}$$

Similarly, $C_M(2^p) = \frac{m^3}{7^i} n^{\log 7}$ and the total arithmetic cost is $C_A(2^p) + C_M(2^p)$.

Table 5.3.7 gives the additive, multiplicative, and total arithmetic cost of multiplying $2^p \times 2^p$ matrices using α_W for $m = 2^i$, $0 \leq i \leq 8$, as a coefficient of $n^{\log 7} = 7^p$.

Choice of m	Add. Cost	Mult. Cost	Total Arith. Cost
1	5.0	1.0	6.0
2	3.43	1.14	4.57
4	2.61	1.31	3.92
8	2.24	1.49	3.73
16	2.13	1.71	3.84
32	2.19	1.95	4.14
64	2.37	2.23	4.60
128	2.63	2.55	5.18
256	2.96	2.91	5.87

Table 5.3.7

Cost of Implementing α For $m \cdot 2^k = n = 2^p$ Matrix Multiplication As m Varies ($k = \log n - \log m$)

By inspection of Table 5.3.7, $m = 1$ minimizes $C_M(2^p)$, $m = 16$ minimizes $C_A(2^p)$, and $m = 8$ minimizes the total number of arithmetics.

The questions naturally arise whether this observation holds when we allow any choice, not only 2^i, for m and whether these choices of m minimize costs for arbitrary (n, n, n) products.

To answer these questions we perform a general analysis of the multiplicative and total arithmetic cost functions, $m^3 7^k$ and $m^2((2m + 4)7^k - 5 \cdot 4^k)$, as suggested by P. Fischer [F3], in an effect to discover the best dynamic (as opposed to a priori) strategy for choosing m.

<u>Lemma 5.3.8</u>: Given arbitrary n, the multiplicative cost of computing (n, n, n) products by α_W is minimized by recursing $(m_1 = \lceil n/2 \rceil, \cdots, m_{i+1} = \lceil m_i/2 \rceil)$ until $m_i = 1, 3, 5, 9, 13$ or 17.

<u>Proof</u>: (1) Suppose at some stage m is even. Then, we wish to compare $C_M(m \cdot 2^k)$ and $C_M(\frac{m}{2} \cdot 2^{k+1})$ to discover whether we should continue recursing.

$$\begin{aligned} C_M(m \cdot 2^k) &= m^3 \cdot 7^k \\ &> \frac{7}{8} m^3 \cdot 7^k \qquad \text{for all } m \geq 1 \\ &= C_M(\tfrac{m}{2} \cdot 2^{k+1}) \ . \end{aligned}$$

Therefore, it is multiplicatively less costly to compute (n, n, n) products with $m' = \frac{m}{2}$, $k' = k+1$, by α_W whenever m is even.

(2) Suppose m is an odd number at some stages of recursing, and $m \geq 23$, i.e. $m = 23 + x$ for even x .

$$\begin{aligned} C_M(m \cdot 2^k) &= m^3 \cdot 7^k \\ &= (23 + x)^3 \cdot 7^k \\ &= (12{,}167 + 1587\, x + 69\, x^2 + x^3) \cdot 7^k \ . \end{aligned}$$

The multiplicative cost of using α_W with $m' = \frac{m+1}{2}$, $k' = k+1$,is

$$\begin{aligned} C_M(m' \cdot 2^{k'}) &= \frac{7}{8} (m + 1)^3 \cdot 7^k \\ &= (12{,}096 + 1512\, x + 63\, x^2 + \frac{7}{8}\, x^3) \cdot 7^k \end{aligned}$$

which is less than $C_M(m \cdot 2^k)$ for all x .

Combining (1) and (2), we have that $C_M(m \cdot 2^k)$ can be improved upon by recursing one level for $m \geq 23$, and for all even m .

(3) Suppose m is odd and $m \leq 23$. Then, we wish to compare $C_M(m \cdot 2^k)$ and $C_M((\frac{m+1}{2}) \cdot 2^{k+1})$, i.e. $m^3 \cdot 7^k$ and $\frac{7}{8} (m + 1)^3 \cdot 7^k$ respectively. Also, this tests only whether an improvement is obtained by recursing just one level. Clearly, by (1), if $\frac{m+1}{2} \cdot 2^{k+1}$ is an improvement, and $\frac{m+1}{2}$ is even, then $\frac{m+1}{4} \cdot 2^{k+2}$ will be a further improvement. Table 5.3.9 lists various

choices for m' which could be used to compute (n, n, n) products where $n \leq m \cdot 2^k$ and $m = 2i + 1$, $1 \leq i \leq 10$. Beside these choices are given the associated multiplicative costs as the coefficient of 7^k .

	m'	k'	$C_M(m \cdot 2^k)$	$C_M(m' \cdot 2^{k'})$
m = 3	2	k+1	27	56
	1	k+2		49
m = 5	3	k+1	125	189
m = 7	4	k+1	343	448
	2	k+2		392
	1	k+3		343
m = 9	5	k+1	729	875
m = 11	6	k+1	1331	1512
	3	k+2		1323
m = 13	7	k+1	2197	2401
m = 15	8	k+1	3375	3584
	1	k+3		2401
m = 17	9	k+1	4913	5103
m = 19	10	k+1	6859	7000
	5	k+2		6125
m = 21	11	k+1	9261	9317
	3	k+3		9261

Table 5.3.9

Lowest Multiplicative Cost Obtained by Using α_W with Various Choices of m' (best choice(s) underlined)

Thus, for example, it is less costly to use α_W to multiply matrices of order $3 \cdot 2^{k+2}$ than to multiply matrices of order $11 \cdot 2^k$ $(k \geq 0)$. In particular, it is multiplicatively optimal to compute (11, 11, 11) products as (12, 12, 12) products when α_W (or α_S) is used. By inspection of Table 5.3.9, the lemma is proved.

Note that there are two values 7, and 21 for which recursing further leads to an equally costly implementation. Thus, we could have equivalently asserted that a multiplicatively optimal dynamic strategy is to continue recursing on m' as long as m' is even $m' \geq 23$, or $m' = 11, 15$ or 19. Also, by (1) in Lemma 5.3.8, $C_M(2^p)$ is minimized for an implementation of α_W with $m = 1$.

Similarly, by considering the total arithmetic cost function $C_T(m \cdot 2^k) = m^2((2m + 4)7^k - 5 \cdot 4^k)$, we can prove

<u>Lemma 5.3.10</u>: The total number of arithmetic operations involved in computing (n, n, n) products by α_W is minimized by an implementation $n \leq m \cdot 2^k$, where m is found by iterating:

$m_1 = \lceil n/2 \rceil, \ldots, m_{i+1} = \lceil m_i/2 \rceil$ until

1) $m_i \leq 13$,

2) m_i is odd and $m_i \leq 33$,

or 3) m_i is 31 .

Proof: The lemma is proved by showing that recursing even one level obtains an improvement for $m \geq 37$ and for $m = 2i$, $i \geq 7$. Recursing two levels obtains an improvement for $m = 35$ and $m = 31$. For example, computing (n, n, n) products with $m = 31$ uses $(63{,}426)7^k - (4805)4^k$ total arithmetics. Computing by α_W with $m' = 8$, $k' = k+2$ uses $(62{,}720)7^k - (5120)4^k$ total arithmetics, fewer for all $k \geq 0$.

As a corollary of Lemmas 5.3.8 and 5.3.10, we have

Corollary 5.3.11: Given any integer p, α_W computes $(2^p, 2^p, 2^p)$ products with the fewest multiplications and fewest total arithmetics for $m = 1$, and $m = 8$ respectively.

Saving half the total arithmetics using even the best dynamic strategy is an extremely slow process, crossing over at approximately $p = 10$ for $n = 2^p$. We can obtain a uniform lower bound, however, on the total arithmetic cost of using α_W with the best dynamic strategy.

Assume that n falls somewhere in the interval between $2^{p-1}+1$ and 2^p for $p \geq 6$. There are 16 different possible choices for m; therefore, divide the interval into 16 sub-intervals each of which is associated with a unique value of n obtained by recursing using best dynamic strategy. The sub-intervals, together with associated m and k values are:

$$2^{p-1} + 1 \leq n \leq 33 \cdot 2^k = 2^{p-1} + 2^{p-6} \quad , \quad m = 33, \quad k = p-6$$

$$2^{p-1} + 2^{p-6} + 1 \leq n \leq 2^{p-1} + 2^{p-5} \quad , \quad m = 17, \quad k = p-5$$

$$2^{p-1} + 2^{p-5} + 1 \leq n \leq 2^{p-1} + 2 \cdot 2^{p-5} \quad , \quad m = 9 \ , \quad k = p-4$$

$$2^{p-1} + 2 \cdot 2^{p-5} + 1 \leq n \leq 2^{p-1} + 3 \cdot 2^{p-5} \quad , \quad m = 19, \quad k = p-5$$

and so on, until

$$2^{p-1} + 13 \cdot 2^{p-5} + 1 \leq n \leq 2^{p-1} + 14 \cdot 2^{p-5} \quad , \quad m = 15, \quad k = p-4$$

$$2^{p-1} + 14 \cdot 2^{p-5} + 1 \leq n \leq 2^p \quad , \quad m = 8 \ , \quad k = p-3 \ .$$

The number of total arithmetics employed is constant throughout each sub-interval; therefore, when this number is expressed as a coefficient times $n^{\log 7}$, the coefficient will be largest at the lower end of each sub-interval (and conversely, smallest at the upper end).

Since we are using the strategy of Lemma 5.3.10, $m \leq 33$. Thus, as $n \leq m \cdot 2^k$ increases arbitrarily, so does k. The asymptotic cost with respect to total arithmetics of using α_W is

$$m^2(2m + 4)7^k \ .$$

The coefficient of $n^{\log 7}$ in this cost is

$$m^2(2m + 4)7^{k-\log n} \ .$$

For example, for $m = 33$, and arbitrarily large $n \leq m \cdot 2^k$, the worst case occurs when $n = 2^{p-1}+1$, $k = p-6$. Then, $\log n \approx p-1$ (all logs are to base 2). The total number of arithmetic operations used is

$$\begin{aligned}
&33^2(70)7^{p-6} \\
&= \frac{76230\ 7^{p-6}}{7^{\log n}} \cdot 7^{\log n} \\
&= \frac{76230}{7^5}\ n^{\log 7} \\
&= 4.5356\ n^{\log 7} \quad , \text{ about } 4.54\ n^{\log 7} \ .
\end{aligned}$$

For $m = 19$, $k = p-5$, $n = 2^{p-1} + 2 \cdot 2^{p-5} + 1$, the coefficient of $n^{\log 7}$ in the number of total arithmetics is

$$\frac{19^2(42)}{7^{4.17}} = 4.5368 \text{ or about } 4.54.$$

The other values of m yield coefficients of $n^{\log 7}$ which are smaller than 4.54. Thus, we have

<u>Theorem 5.3.12</u>: Using best dynamic strategy, α_W can be used recursively to multiply two matrices of order n in about $4.54\ n^{\log 7}$ total arithmetic operations in the worst case.

The final result in this section is that α_W computes (2, 2, 2) products in an additively optimal manner over all possible fast algorithms.

First, we show that α computes lefthand and righthand sides of the multiplications in as few addition/subtraction steps as possible, namely 8. Let $K = \{0, 1, -1\}$, Z be the integers.

Lemma 5.3.13: If α is a fast, K-bilinear chain for (2, 2, 2) products with addition flow representation $F = \langle G_1, G_2, G_3 \rangle$, then nadds $(G_1) \geq 4$.

Proof: Let M be the set of multiplications of α and recall that M_S is the set of multiplications in [S1] used by α_S for (2, 2, 2) products. Then, by Theorem 5.2.1, there is a transformation $T \in \tau$ such that $T(M) = M_S$. Since, τ is a group, there is also a transformation $T_0 = T^{-1}$ in τ such that

$$M = T_0(M_S) .$$

Thus, $M = \{T_0(M_1), T_0(M_2), \cdots, T_0(M_7) \mid M_i \in M_S\}$. By the definition of τ in Section 5.2, T_0 preserves the type of a multiplication. Therefore α has 1 type-2 multiplication, namely $T_0(M_1)$, and 6 type-1 multiplications, $T_0(M_i)$, $2 \leq i \leq 7$. Also, since τ is a group, $T_0(M_i) \neq T_0(M_j)$ for $i \neq j$.

Now, each T is a linear transformation. Therefore, since $M_1^L = M_3^L + M_4^L$,

$$T_0(M_1^L) = T_0(M_3^L) + T_0(M_4^L) .$$

Similarly,

$$T_0(M_1^L) = T_0(M_5^L) - T_0(M_7^L)$$

and $$T_0(M_1^L) = T_0(M_2^L) - T_0(M_6^L) \quad .$$

Since $T_0(M_1^L)$ is a type-2 sum (τ-equivalent to $a_{11} + a_{22}$), $T_0(M_1^L)$ is the sum or difference of 2 or 3 terms.

Suppose $T_0(M_1^L)$ contains 3 terms. Since $T_0(M_1^L) = T_0(M_3^L) + T_0(M_4^L)$, at least one of $T_0(M_3^L)$, $T_0(M_4^L)$ must contain at least 2 terms, i.e. at least one is a non-trivial sum.

Similarly, at least one of $T_0(M_5^L)$, $T_0(M_7^L)$ and one of $T_0(M_2^L)$, $T_0(M_6^L)$ must be a non-trivial sum. Therefore, at least 3 of the type-1 sums in M^L are distinct non-trivial sums. Since the type-2 sum is non-trivial as well, M^L contains at least 4 non-trivial sums. By Lemma 5.3.1, nadds $(G_1) \geq 4$.

Suppose $T_0(M_1^L)$ contains 2 terms. $T_0(M_1^L)$ can be written in only 1 way as the sum or difference of 2 trivial sums. Therefore, at most one of the following pairs of sums,

$$[T_0(M_3^L),\ T_0(M_4^L)] \quad ,$$

$$[T_0(M_5^L),\ T_0(M_7^L)] \quad ,$$

$$[T_0(M_2^L),\ T_0(M_6^L)]$$

can be a pair of single a_{ij} terms.

Therefore at most 2 of the 6 type-1 sums can be trivial sums. Therefore at least 4 of the type-1 sums are non-trivial. As well, $T_0(M_1^L)$ is a non-trivial sum, and all $T_0(M_i^L)$ are distinct by Lemma 5.2.3.

Therefore M^L contains 5 non-trivial, distinct sums. Therefore by Lemma 5.3.1, nadds $(G_1) \geq 5$. Thus, in either case, nadds $(G_1) \geq 4$ as required.

<u>Corollary 5.3.14</u>: If $F = \langle G_1, G_2, G_3 \rangle$ represents a fast K-bilinear chain for (2, 2, 2) products, nadds $(G_2) \geq 4$.

<u>Proof</u>: Assume nadds $(G_2) \leq 3$. Since F represents a fast (2, 2, 2) algorithm, by Lemma 4.3.1 $F^T = \langle G_2^T, G_1^T, G_3^T \rangle$ does as well. But nadds (G_2^T) = nadds $(G_2) \leq 3$, contradicting Lemma 5.3.13. Therefore nadds $(G_2) \geq 4$ as required.

By inspection of the proof of Lemma 5.3.13 and by considering a transposed representation noting that reversing matrix indices does not affect the number of distinct non-trivial sums, we have

<u>Corollary 5.3.15</u>: If $F = \langle G_1, G_2, G_3 \rangle$ represents a fast K-bilinear chain α for (2, 2, 2) products, nadds (G_1) = nadds (G_2) = 4 only if the type-2 multiplication of α is the product of a sum involving exactly 3 a_{ij} terms and a sum involving exactly 3 b_{ij} terms.

Thus, α_W forms lefthand and righthand factors of its 7 multiplications in an additive optimal manner over all possible sets of 7 multiplications, meeting the lower bound of 8 addition/subtraction steps given by Lemma 5.3.13 and Corollary 5.3.14. We can easily show now that α_W combines the multiplications in M_W to form the product elements y_{ij} , $1 \leq i,j \leq 2$, in an additively optimal way over all K-bilinear chains for (2, 2, 2) products. In other words, M_W is as good as any set of 7 multiplications for (2, 2, 2) computations.

<u>Lemma 5.3.16</u>: If $F = \langle G_1, G_2, G_3 \rangle$ represents a fast K-bilinear chain for (2,2,2) products, then nadds $(G_3) \geq 7$.

<u>Proof</u>: Consider the symmetric representation $F^* = \langle G_1^T, G_3^D, G_2^D \rangle$ to F . F* represents a fast algorithm for (2, 2, 2) products by Lemma 4.2.3. Therefore, by Corollary 5.3.14, nadds $(G_3^D) \geq 4$. However, by Lemma 4.1.5.

$$\text{nadds } (G_3^D) = |\Gamma G_3| - |VG_3| + |S_3|$$

$$\therefore \quad |\Gamma G_3| - |VG_3| + |S_3| \geq 4$$

$$\therefore \quad (|\Gamma G_3| - |VG_3| + |R_3|) + |S_3| \geq 4 + |R_3| \ .$$

Now, $|\Gamma G_3| - |VG_3| + |R_3| = \text{nadds } (G_3)$ by Lemma 4.1.5, and for fast (2, 2, 2) products, $R_3 = 7$, $S_3 = 4$.

$\therefore$ nadds $(G_3) + 4 \geq 4 + 7$. $\therefore$ nadds $(G_3) \geq 7$ as required.

Thus, from the above and Corollary 5.2.2, we have

<u>Corollary 5.3.17</u>: Any fast Z-bilinear chain to compute (2, 2, 2) products must use at least 15 addition/subtraction steps, or employing our notation, $\mathcal{A}(2, 2, 2, 7) \geq 15$.

Since α_W meets this lower bound on additions/subtractions, α_W is an additively optimal fast algorithm for (2, 2, 2) products. Combining Corollaries 5.3.5 and 5.3.17, we have the main result.

<u>Theorem 5.3.18</u>: $\mathcal{A}(2, 2, 2, 7) = 15$, i.e. 15 additions and subtractions are necessary and sufficient to compute the product of matrices of order 2 using no more than 7 multiplications.

Thus, as an immediate corollary, we have

<u>Corollary 5.3.19</u>: The obvious way of computing (2, 2, 2) products (8 multiplications, 4 additions) is optimal with respect to total arithmetics.

Of course, this does not hold for (n, n, n) products; the obvious method uses $2n^3 - n^2$ total arithmetics, whereas by Lemma 5.3.6, $4{\cdot}57n^{2.82}$ total arithmetic operations suffice.

The additive complexity of the more general problem of computing (m, 2, n) products by Z-bilinear chains is an open problem, as is the additive complexity of fast (2, 2, 2) computations by algorithms in NC .

As a final application of the addition flow representation model, note that the standard algorithm for each of the five representations derivable from F_W is an additively optimal fast algorithm for (2, 2, 2) products. As well, we can employ the model and apply the results in Section 5.2 to characterize all additively optimal fast (2, 2, 2) algorithms. We do not include the arguments involved, since they involve tedious case by case analysis.

Recall that by Corollary 5.3315, any additively optimal algorithm contains exactly 1 multiplication which is the product of a sum of 3 a_{ij} terms and a sum of 3 b_{ij} terms. We can also show that exactly 2 multiplications must be trivial products, one non-trivial multiplication must have a trivial lefthand factor, one non-trivial multiplication must have a trivial righthand factor, and so on. Similarly, by considering symmetric representations, we can characterize the manner in which additively optimal fast algorithms combine their multiplications to form the product elements. A generalization of the method may be useful for more general problems.

CHAPTER VI

FINAL OBSERVATIONS

In this final chapter, we observe that changes in the number of input parameters for matrix multiplication do not necessarily affect the multiplicative complexity of the problem, and show that fixing the number of inputs and at the same time increasing the size of the product of the three dimensions causes a strict increase in complexity. The second section introduces the topic of linear algorithms and modifies the definition of addition flow graph (c.f. Section 4.1) to obtain a useful model for the computation of linear forms by such algorithms. In the last section, we attempt to define the relationship between the number of multiplications employed and additive complexity. As well, the additive complexity of fast computations of $(3, 2, 3)$ and $(3, 3, 3)$ products is investigated.

6.1 Multiplicative Complexity: The Number of Inputs

Some simple problems in analysis of algorithms such as polynomial evaluation [B1] have been shown to have multiplicative complexity linear or at least less than quadratic in the number of input parameters of the problem. This is likely to be dependent on the linear format of these problems. The number of inputs in (m, n, p) matrix product computations is $n(m + p)$; a natural question is whether the multiplicative complexity of matrix multiplication is a linear function of this number.

The best lower bound known (Theorem 1.2.4) for (m, n, p) products namely,

$$n(m + p) + mp - (m + n + p) + 1$$

is essentially of this form. By Remark 3.2.4, and the multiplicative symmetry theorem (2.3.4), we know that the multiplicative complexity of problems with $n(m + p)$, $m(n + p)$, and $p(m + n)$ inputs is identical. Thus, rather than fixing the dimension product as in Section 3.3, we may fix the number of inputs to a matrix multiplication problem and study the effect on multiplicative complexity of varying the product of the dimensions.

First, recall the well-known technique of block multiplication of matrices.

<u>Lemma 6.1.1</u>: $\mathcal{M}(k_1m, k_2n, k_3p) \leq k_1k_2k_3\mathcal{M}(m, n, p)$.

<u>Proof</u>: First, we show that $\mathcal{M}(m, kn, p) \leq k\mathcal{M}(m, n, p)$. Given matrices $A_{m\times kn}$, $B_{kn\times p}$, let C_ℓ be the submatrix of A given by

$$c_{ij} = a_{i,n(\ell-1)+j} \quad \text{for} \quad 1 \leq \ell \leq k .$$

Similarly, let D_ℓ be given by

$$d_{ij} = b_{n(\ell-1)+i,j} \quad .$$

Then, compute $A \cdot B$ as

$$\sum_{\ell=1}^{k} C_\ell D_\ell$$

using $k\mathcal{M}(m, n, p)$ multiplications. Thus, $\mathcal{M}(k_1m, k_2n, k_3p) \leq k_2\mathcal{M}(k_1m, n, k_3p)$. But, by Theorem 2.3.4, $\mathcal{M}(k_1m, n, k_3p) = \mathcal{M}(n, k_1m, k_3p)$. Therefore by performing block multiplication, we have

$$\begin{aligned} \mathcal{M}(k_1m, k_2n, k_3p) &\leq k_2k_1\mathcal{M}(n, m, k_3p) \\ &= k_2k_1\mathcal{M}(m, n, k_3p) \end{aligned}$$

Repeating the above, $\mathcal{M}(k_1m, k_2n, k_3p) \leq k_1k_2k_3\mathcal{M}(m, n, p)$ as required.

For example, as a corollary of Theorem 1.1.5 and the above lemma, we have

<u>Corollary 6.1.2</u>: $\mathcal{M}(m, 2n, p) \leq n\left\lceil\frac{3pm + \max\{m, p\}}{2}\right\rceil$.

Now, consider (km, n, kp) and (m, kn, p) products. Clearly each involves the same number of inputs, namely $kn(m + p)$, although the dimension products k^2mnp , $kmnp$ differ (we assume $k \geq 2$) . We have

Lemma 6.1.3: $\mathcal{M}(km, n, kp) > \mathcal{M}(m, kn, p)$ for all $k \geq 2$ and $k \geq n$.

Proof: Assume $n = 1$. Then, by Theorems 1.1.2 and 2.3.4, $\mathcal{M}(km, 1, kp) = k^2mp$. By Lemma 6.1.1, $\mathcal{M}(m, k\,1, p) \leq k\mathcal{M}(m, 1, p) = kmp$. Thus, the result holds for $n = 1$ and all $k \geq 2$.

Assume $n \geq 2$. Then, by Theorem 1.2.4,

$$\mathcal{M}(km, n, kp) \geq k^2mp + kn(m + p) - k(m + p) - n + 1$$

For $k \geq n$, this is $\geq knmp + k(n - 1)(m + p) - n + 1$

$> knmp$.

But, by Lemma 6.1.1, $\mathcal{M}(m, kn, p) \leq k\mathcal{M}(m, n, p) \leq kmmp$. Thus, $\mathcal{M}(km, n, kp) > \mathcal{M}(m, kn, p)$ for all $k \geq n \geq 2$.

This restriction, $k \geq n$, is very undesirable, since it seems likely that it is necessary only that $k \geq 2$ to obtain this strict increase in multiplicative complexity. However, the restric restriction was necessitated by the lack of a better lower bound than Theorem 1.2.4.

6.2 On the Additive Complexity of Linear Forms

Morgenstern [M1, M2, M3], Kirkpatrick [K1], and others have studied the problem of computing various sets of sums using algorithms which employ only addition, subtraction, and multiplication by a constant. In this section we present examples which indicate that addition flow graphs are useful for studying the complexity of a small subclass of such computations.

Recall the definition in Section 5.3 of a sum S as an expression $\sum_{i=1}^{n} c_i x_i$ where the x_i's are indeterminates and the c_i's are non-zero constants.

Then, a linear form ℓ is a set of distinct, non-trivial sums (this definition is somewhat anomalous - c.f. 5.3 for the definition of "distinct, non-trivial" sums). The class of linear forms of n sums in m indeterminates is denoted $L_{m,n}$.

For example, in the computation of $(2, 2, 2)$ products by Strassen's algorithm, if $M_S = \{M_1, M_2, \cdots, M_7\}$ is regarded as a set of indeterminates, then α_S computes the linear form $\ell_S = \{(M_1 + M_4 - M_5 + M_7), (M_3 + M_5), (M_2 + M_4), (M_1 + M_3 - M_2 + M_6)\}$ of 4 sums in 8 addition/subtraction steps. Here, $\ell_S \in L_{7,4}$.

Algorithms which employ only addition, subtraction, and multiplication by a constant to compute linear forms will be referred to as linear algorithms.

Clearly, given a linear form ℓ , we can analyze a linear algorithm α for computing ℓ by studying its corresponding addition flow graph G_α , since linear algorithms correspond exactly to non-multiplication stages of matrix multiplication algorithms. Thus we can define: the restricted addition flow graph, denoted G_α , of a linear algorithm α which computes a linear form $\ell \in L_{m,n}$ is the directed acyclic graph defined as for addition flow representations (cf. Section 4.1). The m source vertices represent the m indeterminates, the n sink vertices represent the sums in ℓ , and intermediate vertices represent intermediate calculations by α (as in addition flow representations). Since this model is a submodel of addition representations, we can restrict our study of the additive complexity of linear forms to the study of restricted addition flow graphs.

The additive complexity of a linear form ℓ , denoted $\mathcal{A}(\ell)$, is the fewest number of addition and subtraction steps which can be used by a linear algorithm to compute ℓ . Thus, by Lemma 5.3.2, for example, $\mathcal{A}(\ell_S) = 8$.

Note that the flow graphs defined in Chapter 4 are not necessarily restricted addition flow graphs. Since linear forms contain no trivial sums by definition, each sink vertex of a flow graph for computing any linear form must be connected to at least one vertex (possibly itself) which has invalence greater than one. Thus, for example, $\hat{G}_1$ in section 4.1 is a flow graph which is not a restricted flow graph.

If α is a linear algorithm for computing $\ell \in L_{m,n}$, let G_α be the restricted addition flow graph for α. By the same reasoning as above, G_α^D (the directional dual of G_α) will be a restricted addition flow graph only if the sink set of G_α^D contains no vertices connected only to vertices with invalence less than 2, i.e. only if the source set R_α of G_α contains no vertices which are connected only to vertices with outvalence less than 2.

Accordingly, define G_α^* to be G^D with such vertices and all edges adjacent to those vertices removed. Then, G_α^* represents the computation of a set of non-trivial sums. However, since these sums are not necessarily distinct, G_α^* may not be a restricted addition flow graph.

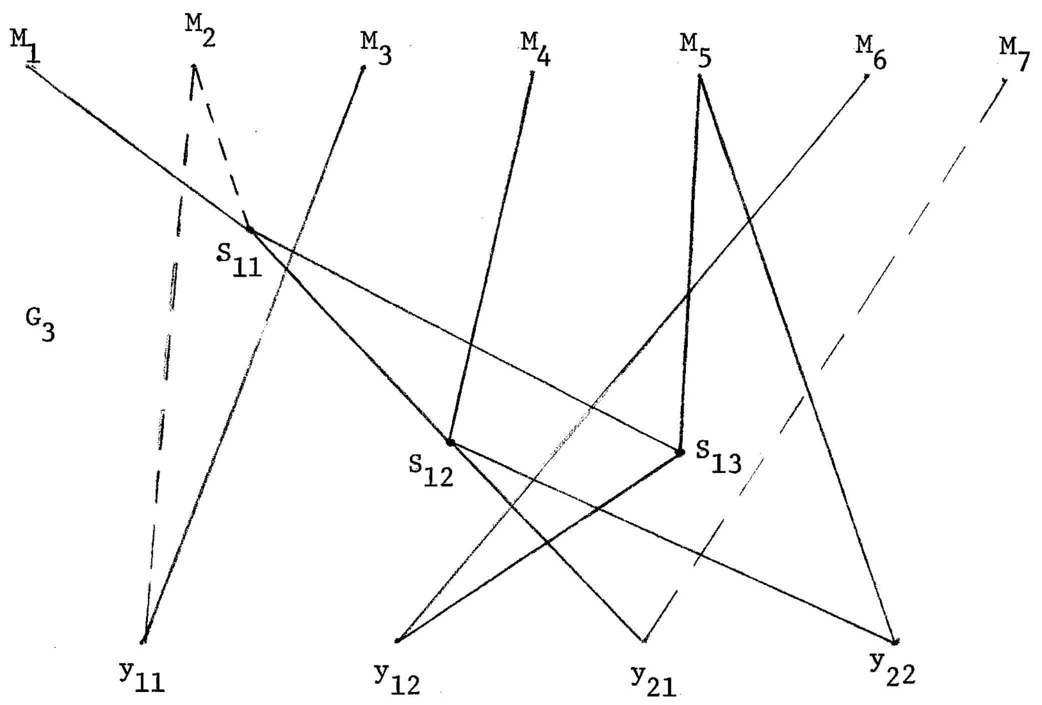
Therefore, given G_α, let C be its $(n \times m)$ connection matrix. Let $\hat{C}$ be C with all columns which contain only 1 non-zero element removed. Then, $C^* = (\hat{C})^T$ and the number of distinct, non-trivial sums represented by G_α^* is

p = the number of pairwise distinct rows of C^*

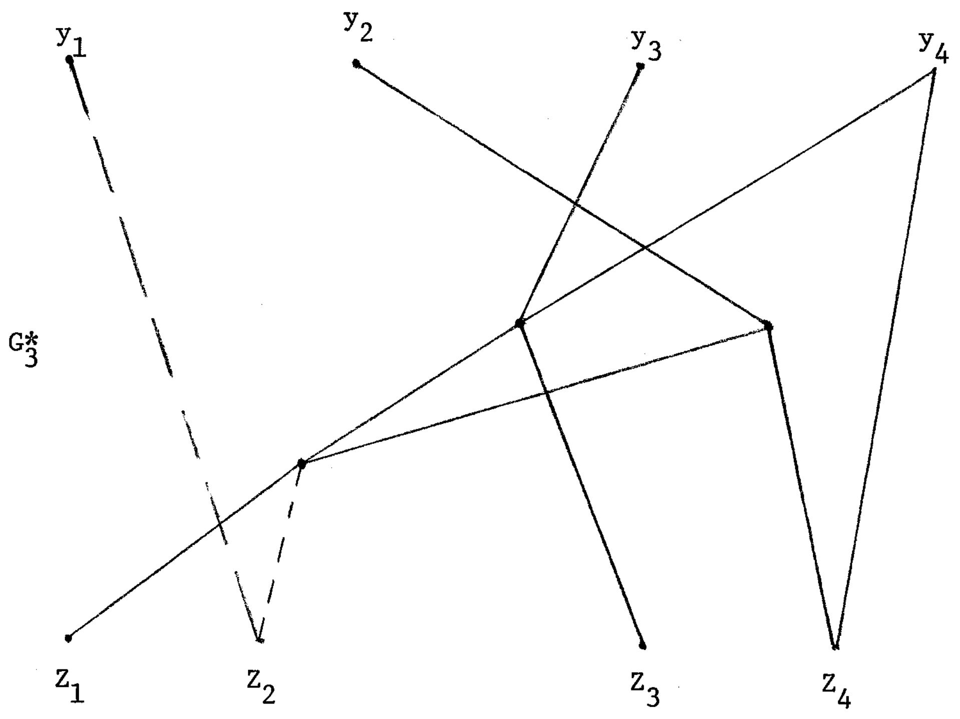
= the number of pairwise distinct columns of $\hat{C}$.

Thus, G_α^* represents a computation of a linear form, denoted ℓ^*, such that $\ell^* \in L_{n,p}$.

To illustrate the approach, consider the following flow graph G_3 of the addition representation F_W for α_W given in Section 5.3.



Clearly, G_3 represents the computation of the linear form ℓ_W consisting of the 4 distinct, non-trivial sums labelled y_{11}, y_{12}, y_{21}, y_{22} of the 7 indeterminates labelled $M_1, M_2, \cdots, M_7$. Or, we could write $\ell_W = \{(M_3 - M_2), (M_1 - M_2 + M_5 + M_6), (M_1 - M_2 + M_4 - M_7), (M_1 - M_2 + M_4 + M_5)\}$. Since M_3 , M_6 , M_7 in R_3 are connected only to vertices with outvalence ≤ 1 , they will not appear in G_3^* given below.



Then, ℓ_W^*

$$= \{(y_2 + y_3 + y_4),\ (-y_1 - y_2 - y_3 - y_4),\ (y_3 + y_4),\ (y_2 + y_4)\}$$

By analogy to results in Chapters 4 and 5, we have

<u>Lemma 6.2.1</u>: If α computes $\ell \in L_{m,n}$ and has restricted addition flow graph G_α, then nadds (G_α) is minimum if and only if nadds (G_α^*) is minimum (c.f. Lemma 5.3.2).

<u>Corollary 6.2.2</u>: If $\ell \in L_{m,n}$ then $\mathcal{A}(\ell^*) = \mathcal{A}(\ell) + n - m$.

Proof: Let α be an optimal linear algorithm for computing ℓ . Let G be the restricted flow graph of α . Then, $\mathcal{A}(\ell)$ = nadds (α) = nadds $(G) = |\Gamma G| - |VG| + |R|$. By Lemma 6.2.1,

$$\begin{aligned} \mathcal{A}(\ell^*) &= \text{nadds } (G^*) \\ &= |\Gamma G^*| - |VG^*| + |R^*| \\ &= |\Gamma G| - |VG| + |R| + |S| - |R| \\ &= \text{nadds } (G) + |S| - |R| \end{aligned}$$

since $\ell \in L_{m,n}$, $\quad = \mathcal{A}(\ell) + n - m$ as required.

Corollary 6.2.3: $\ell \in L_{m,n}$ and $\ell^* \in L_{n,p}$ implies

$$\mathcal{A}(\ell) \geq m + p - n$$

Proof: By Lemma 5.3.1 and the definition of a linear form,

$$\mathcal{A}(\ell^*) \geq p \ .$$

By Corollary 6.2.2, $\mathcal{A}(\ell) = \mathcal{A}(\ell^*) + m - n$

$$\geq m + p - n \ .$$

In other words, at least $m+p-n$ addition/subtraction steps are required to compute a set of n non-trivial distinct sums from m indeterminates, where p is the number of distinct columns in the modified connection matrix $\hat{C}$. For example, ℓ_S is a set of 4 sums of 7 indeterminates. The corresponding $\hat{C}$ has 5 distinct columns; thus, $\ell^*_S \in L_{4,5}$. By Corollary 6.2.3, $\mathcal{A}(\ell_S) \geq 7 + 5 - 4 = 8$. Since α_S uses only 8 additions/subtractions, $\mathcal{A}(\ell_S) = 8$. Similarly, for ℓ_W , $m = 7$, $n = 4$, but $p = 4$. Therefore, $\mathcal{A}(\ell_W) \geq 7 + 4 - 4 = 7$. Since α_W achieves this lower bound, $\mathcal{A}(\ell_W) = 7$.

Thus, this trivial lower bound is quite sufficient for treating some simple linear forms. For larger forms, in particular for forms for which $n \geq m$, the bound is useless. Of course, in the special case of computing product matrix elements from a set of t multiplications, the number of multiplications will always be larger than the number of elements of the product matrix by Theorem 1.2.4. Hence, we can apply Corollary 6.2.3 to the last stage in the computation of matrix products. This is attempted in the next section.

6.3 An Additive/Multiplicative Complexity "Trade-off"

Until Strassen [S1] discovered a means to multiply $n \times n$ matrices in asymptotically less than n^3 total arithmetic operations, many researchers thought that the total arithmetic cost was invariant, and in fact was $O(n^3)$. Thus, a saving in multiplicative cost achieved by an algorithm such as Winograd's [W2] "inner-product" algorithm, would be offset by a proportionate (and usually greater) increase in additive cost. Then, the classical algorithm, which employs $2n^3-n^2$ total operations, would be essentially optimal with respect to total arithmetics.

Of course, this is not the case, since using a Strassen - like implementation of α_W we need only $4.57\, n^{\log 7}$ total arithmetics by Lemma 5.3.6, and by a dynamic implementation strategy, only $4.54\, n^{\log 7}$ (Corollary 5.3.12).

By Lemma 5.3.10, the total number of arithmetic operations is less when recursively employing α_W than when employing the classical method to multiply two matrices of order n for all $n \geq 35$, $n = 31$, or n even and $n \geq 14$. To see this, set $m = n$, $k = 0$. Then, the total arithmetic cost of using α_W with this implementation is

$$\begin{aligned} &m^2((2m + 4)7^k - 5.4^k) \\ &= n^2(2n - 1) \quad , \end{aligned}$$

the cost of computing (n, n, n) products with the classical algorithm.

For example, the total arithmetics used by a classical scheme to multiply two 14×14 matrices is ($n = 14$)

$$n^2(2n - 1) = 5292 \text{ total arithmetic operations.}$$

By Lemma 5.3.6, employing α_W with $m = 7$, $k = 1$ costs

$$m^2((2m + 4)7^k - 5.4^k) = 5194 \text{ total arithmetics.}$$

Thus, even for matrices as small as 14×14, total arithmetic cost is not independent of the number of multiplications used, but can actually be reduced by decreasing multiplicative cost.

Of course, recursively implementing α_W costs asymptotically fewer arithmetic operations ($4.54\, n^{2.8}$) than the number of addition operations alone ($n^3 - n^2$) used by the classical method to compute (n, n, n) products. Thus, from some value of n onwards, a reduction in multiplication costs achieved by a recursive implementation of an algorithm (here, α_W) will result in a concomitant saving in additive cost. In fact, the classical algorithm uses no more additions/subtractions than an implementation of α_W for $n \leq 24$, or n odd and $n \leq 45$. Thus, for these "small" computations a complexity trade-off may exist between multiplicative and additive complexity.

To give the simplest example, $\mathcal{A}(2, 2, 2, 8) = 4$ whereas $\mathcal{A}(2, 2, 2, 7) = 15$ by Theorem 5.3.18.

The best lower bound known on the number of additions/subtractions required by algorithms to compute (m, n, p) products using any number of multiplications was proved by Kirkpatrick [K1] using independence arguments and, as stated in Section 1.2, is

$$(m + p - 1)(n - 1) \quad . \qquad (1)$$

When $m = p = n$, this bound becomes

$$2n^2 - 3n + 1 \quad . \qquad (2)$$

In the remainder of this section we consider the multiplicative/additive trade-off involved in computing (3, 2, 3) and (3, 3, 3) products by known bilinear algorithms.

The classical costs of computing these products are 18 multiplications and 9 additions/subtractions for (3, 2, 3) products and 27 multiplications and 18 additions/subtractions for (3, 3, 3) products. Hopcroft and Kerr [H1] give the following algorithm, α_H , which computes (3, 2, 3) products using 15 multiplications and 43 additions/subtractions and they show that α_H is multiplicatively optimal. The 15 multiplication steps of α_H are

M_1	$(a_{11} - a_{12})b_{11}$	M_6	$(a_{32} - a_{31})b_{23}$
M_2	$a_{12}(b_{11} + b_{21})$	M_7	$(a_{11} + a_{21})(b_{12} + b_{22} + b_{11} + b_{21})$
M_3	$a_{21}b_{12}$	M_8	$(a_{11} + a_{21} - a_{12})(b_{11} + b_{21} + b_{22})$
M_4	$a_{22}b_{22}$	M_9	$(a_{11} + a_{21} - a_{12} - a_{22})(b_{21} + b_{22})$
M_5	$a_{31}(b_{13} + b_{23})$	M_{10}	$(a_{22} + a_{32})(b_{22} + b_{12} + b_{13} + b_{23})$

M_{11} $(a_{22} + a_{32} - a_{31})(b_{12} + b_{13} + b_{23})$

M_{12} $(a_{22} + a_{32} - a_{31} - a_{21})(b_{12} + b_{13})$

M_{13} $(a_{12} + a_{31})(b_{11} - b_{23})$

M_{14} $-(a_{12} + a_{32})(b_{21} + b_{23})$

M_{15} $(a_{11} + a_{31})(b_{11} + b_{13})$

Denote by M_H the set $\{M_1, M_2, \cdots, M_{15}\}$. The first stage of α_H , forming M_H^L , and the second stage, forming M_H^R , each require and can be done in 11 addition/subtraction steps. The computation of the y_{ij} , where $1 \leq i,j \leq 3$, from the multiplications in M_H , may be thought of as the computation of a linear form

$$\begin{aligned} \ell_H = \{&(M_1 + M_2), (-M_1 - M_4 + M_8 + M_9), (-M_2 - M_6 + M_{13} - M_{14}), \\ &(-M_2 - M_3 + M_7 - M_8), (M_3 + M_4), (-M_4 - M_5 + M_{10} - M_{11}), \\ &(-M_1 - M_5 - M_{13} + M_{15}), (-M_3 - M_6 + M_{11} - M_{12}), (M_5 + M_6)\} \\ = \{&y_{11}, y_{21}, y_{31}, y_{12}, y_{22}, y_{32}, y_{13}, y_{23}, y_{33}\} \ . \end{aligned}$$

Since ℓ_H contains 9 sums in 15 "indeterminates", and $\hat{C}$ contains 9 distinct column, by Corollary 6.2.3,

$$\mathcal{A}(\ell_H) \geq 15 + 9 - 9 = 15 \ .$$

Since 21 additions/subtractions are used, we cannot be sure whether α_H is additively optimal even for the above choice of multiplications, although the above argument proves

Lemma 6.3.1: $\mathcal{A}(3, 2, 3, M_H) \geq 37$.

Of course, there is likely a better choice of multiplications than M_H for computing (3, 2, 3) products. By (1), $\mathcal{A}(3, 2, 3, t) \geq 5$ where t is any positive integer. If we set $t = \mathcal{M}(3, 2, 3) = 15$, then it is obvious that some multiplications in an additively optimal 15-multiplication algorithm must appear in more than one of the 9 sums. In fact, the reader can show that at least 3 multiplications must contribute to more than one sum. Thus, the final stage of an optimal (3, 2, 3) computation involves at least

$$15 + 3 - 9 = 9 \quad \text{addition/subtraction steps.}$$

Since additions/subtractions are employed as well to form some lefthand and righthand factors of the multiplications in a multiplicatively optimal algorithm, we conjecture

$$\mathcal{A}(3, 2, 3, 15) \geq 33 \quad .$$

The value of $\mathcal{A}(3, 2, 3, M_H)$ and $\mathcal{A}(3, 2, 3, 15)$ are interesting open problems.

Now, consider (3, 3, 3) products. The bound (2) gives 10 addition/subtraction steps as a lower bound on $\mathcal{A}(3, 3, 3, t)$ for any t .

Let M be the set of multiplications used in Method 1 in Section 2.1 to compute (3, 3, 3) products. Then, noting that 9 additions/subtractions are necessary to add the elements of the intermediate 3×3 matrices in method 1, we have

Remark 6.3.2: $\mathcal{A}(3, 3, 3, M) = \mathcal{A}(3, 2, 3, M_H) + \mathcal{A}(3, 1, 3) + 9 \geq 37 + 0 + 9 = 46$.

There likely are additively better 24-multiplication algorithms for $(3, 3, 3)$ products, although $\mathcal{A}(3, 3, 3, 24)$ is unknown. If we let $t = \mathcal{M}(3, 3, 3)$, then the values of t , and $\mathcal{A}(3, 3, 3, t)$ are open problems.

We can apply our knowledge of addition flow representations (c.f. Chapters IV and V) to specifically attack the complexity of (n, n, n) products as follows. Recall in Section 5.3, that $\mathcal{A}(2, 2, 2, 7)$ was achieved when an algorithm α_W was found with addition flow representation $F_W = \langle G_1, G_2, G_3\rangle$ where G_3 was given in Section 6.2 and G_1, G_2 can easily be drawn. The "nice" feature of F_W is that

$$\text{nadds}\ (G_1) = \text{nadds}\ (G_2) = \text{nadds}\ (G_3^D) ,$$

so that whatever sequence of operations $*, T$ on representations were applied to F_W , the resulting representation F had the property that $\text{nadds}\ (F) = \text{nadds}\ (F_W)$. Accordingly, we make

<u>Conjecture 6.3.3</u>: Given an algorithm α which computes (n, n, n) products in t multiplications, such that $F = \langle G_1, G_2, G_3 \rangle$, $\mathcal{A}(n, n, n, t) = \text{nadds}\ (\alpha)$ implies

$$\text{nadds}\ (G_1) = \text{nadds}\ (G_2) = \text{nadds}\ (G_3^D) .$$

Note that the converse does not hold; F_S is a suitable counter example. If this conjecture is valid, we have

<u>Conjectured Corollary 6.3.4</u>: $\mathcal{A}(n, n, n, t) \geq 4t - 3t_1 - n^2$ where t_1 is the number of multiplications which c contribute to the computation of exactly one product element.

<u>Proof</u>: Assume α is an additively optimal algorithm over all algorithms which compute (n, n, n) products in t multiplication steps. Let $F = \langle G_1, G_2, G_3 \rangle$ be the addition flow representation of α . Then, by Corollary 6.2.3,
$\text{nadds}\ (G_3) \geq t + t - t_1 - |S_3| = 2t - t_1 - n^2$ where t_1 is the sume of the number of columns in the connection matrix C containing only one non-zero entry, and the number of redundant (non-distinct) columns in $\hat{C}$.

$$\begin{aligned} \text{nadds}\ (G_1) &= \text{nadds}\ (G_2) = \text{nadds}\ (G_3^D) \\ &= \text{nadds}\ (G_3) + |S_3| - |R_3| \\ &\geq 2t - t_1 - n^2 + n^2 - t \\ &= t - t_1 . \end{aligned}$$

Therefore

$$\begin{aligned} \text{nadds}\ (\alpha) &= \text{nadds}\ (F_\alpha) \\ &= \sum_{i=1}^{3} \text{nadds}\ (G_i) \\ &\geq t - t_1 + t - t_1 + 2t - t_1 - n^2 \\ &= 4t - 3t_1 - n^2 \qquad \text{as required.} \end{aligned}$$

If true, Corollary 6.3.4 implies that an asymptotic lower bound on additions/subtractions is the number of multiplications employed. For example, if we use all n^3 multiplications of the classical method to compute (n, n, n) products, set $t = t_1 = n^3$. Then, by Corollary 6.3.4, $\mathcal{A}(n, n, n, n^3) \geq 4t - 3t_1 - n^2 = n^3 - n^2$ as expected. Compare lower bound (2) which essentially states that the asymptotic additive complexity is bounded below by $2n^2$. By Theorem 1.2.4, $\mathcal{M}(n, n, n) \geq 3n^2 - 3n + 1$. Let $t = \mathcal{M}(n, n, n)$. Then, Corollary 6.3.4 yields $\mathcal{A}(n, n, n, t) \geq t \approx 3n^2$, which would be an improvement on (2). Unlike (2), as well, Corollary 6.3.4 bounds the additive complexity of small computations when only fast algorithms are employed.

If we reconsider $(3, 3, 3)$ computations, we know that $19 \leq t = \mathcal{M}(3, 3, 3) \leq 24$. Then, estimating that $t_1 \leq 8$, we can conjecture, using Corollary 6.3.4, that

$$\begin{aligned}\mathcal{A}(3,\ 3,\ 3,\ t) &\geq 4t - 3t_1 - 3^2 \\ &\geq 4 \cdot 19 - 3 \cdot 8 - 9 \\ &= 43\end{aligned}$$

In any case, the above observations suggest that additive complexity depends on the number of multiplications used. This dependence is much more pronounced for small matrix multiplication problems. Thus, we suggest the term "additive complexity of matrix multiplication", especially when applied to small matrix products, must be qualified by one of the following phrases:

1) "with respect to the set M of multiplications",

or 2) "with respect to any set of t multiplications".

BIBLIOGRAPHY

[B1] Borodin, Allan and Munro, Ian, "Effecient Evaluation of Polynomial Forms", University of Waterloo Technical Report CSTR-1013 (1972).

[B2] Borodin, Allan B. and Munro, Ian, "Notes on Efficient and Optimal Algorithms", Universities of Toronto and Waterloo (1972).

[D1] Dobkin, David and Brockett, Roger W., "On the Optimal Evaluation of a Set of Bilinear Forms", Proc. of Fifth Annual Symposium on Theory of Computing, 88-95 (1973).

[F1] Fiduccia, Charles, "Fast Matrix Multiplication", Proc. of Third Annual Symposium on Theory of Computing, 45-49 (1971).

[F2] Fiduccia, Charles, "On Obtaining Upper Bounds on the Complexity of Matrix Multiplication", Complexity of Computer Computations, Plenum Press (1972).

[F3] Fischer, Patrick C., Personal Communication.

[H1] Hopcroft, John E. and Kerr, L.B., "On Minimizing the Number of Multiplications Necessary for Matrix Multiplication", Cornell University Technical Report 69-44 (1969).

[H2] Hopcroft, John E. and Musinski, Jean, "Duality Applied to the Complexity of Matrix Multiplication and Other Bilinear Forms", Proc. of Fifth Annual Symposium on Theory of Computing, 73-87 (1973).

[K1] Kirkpatrick, David, "On the Additions Necessary to Compute Certain Functions", University of Toronto Technical Report No. 39 (1972).

[K2] Kirkpatrick, David, Personal Communication.

[K3] Knuth, Donald E., "Seminumerical Algorithms", The Art of Computer Programming, Vol. 2, Addison-Wesley (1969).

[K4] Karp, Richard, "Notes on Computational Complexity", University of California at Berkeley (1971).

[M1] Morgenstern, J., "Algorithmes Linéaires", Compt. Rend. Acad. Sci. 272, 1059-1060.

[M2] Morgenstern, Jacques, "On Linear Algorithms", Theory of Machines and Computations, 59-66, Academic Press, New York (1971).

[M3] Morgenstern, Jacques, "Note on a Lower Bound of the Linear Complexity of the Fast Fourier Transform", JACM, Vol. 20, No. 2, 305-306 (1973).

[M4] Munro, Ian, "Problems Related to Matrix Multiplication", Computational Complexity, 137-151, Algorithmics Press, New York (1973).

[O1] Ostrowski, A.M., "On Two Problems in Abstract Algebra Connected with Horner's Rule", Studies Presented to R. von Mises, Academic Press, New York, 40-48 (1954).

[P1] Probert, Robert L., "On the Complexity of Symmetric Computations", University of Waterloo Technical Report CS-73-02 (1973).

[S1] Strassen, Volker, "Gaussian Elimination is not Optimal", Numer. Math. 13, 354-356 (1969).

[S2] Strassen, Volker, "Evaluation of Rational Functions", Complexity of Computer Computations, Plenum Press (1972).

[W1] Winograd, Shmuel, "On the Algebraic Complexity of Inner Product", IBM Research Report RC-2729 (1969).

[W2] Winograd, Shmuel, "On the Number of Multiplications Necessary to Compute Certain Functions", Communications on Pure and Applied Mathematics, Vol. 23, 165-179 (1970).

[W3] Winograd, Shmuel, "On the Algebraic Complexity of Functions", IBM Research Report (1970).

[W4] Winograd, Shmuel, "On Multiplication of 2×2 Matrices", Linear Algebra and Its Applications 4, 381-388 (1971).

[W5] Winograd, Shmuel, Personal Communication.