ADDITION REQUIREMENTS FOR MATRIX AND TRANSPOSED MATRIX PRODUCTS

Michael Kaminski[†], David G. Kirkpatrick[‡],
and Nader H. Bshouty[§]

Technical Report 86-17

October 1986

Abstract

Let M be an $s \times t$ matrix and let M^T be the transpose of M. Let $\mathbf{x}$ and $\mathbf{y}$ be t– and s–dimensional indeterminate column vectors, respectively. We show that any linear algorithm A that computes $M\mathbf{x}$ has associated with it a natural dual linear algorithm denoted A^T that computes $M^T\mathbf{y}$. Furthermore, if M has no zero rows or columns then the number of additions used by A^T exceeds the number of additions used by A by exactly s–t. In addition, a strong correspondence is established between linear algorithms that compute the product $M\mathbf{x}$ and bilinear algorithms that compute the bilinear form $\mathbf{y}^T M\mathbf{x}$.

Key words. arithmetic complexity, linear forms, linear algorithms, matrix multiplication, graphs of algorithms, bilinear forms, bilinear algorithms.

---

† Faculty of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel. A part of the paper was written while the author was visiting Department of Computer Science of the University of Waterloo, where it was supported by NSERC Grant No. A0254.

‡ Department of Computer Science, the University of British Columbia, Vancouver, B.C., Canada, V6T 1W5. This research was supported in part by NSERC Grant No. A3583.

§ Faculty of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel.

1. Introduction

Many numerical computations involve evaluating the product of a given matrix by a vector of indeterminates. Obviously, the number of arithmetical operations used in the evaluation of such a product depends on the organization of the computation.

In this paper we study the complexity, specifically the number of additions, of linear algorithms that compute a set of linear forms defined by a given matrix M. Our central result establishes a kind of duality between algorithms that compute the set of forms associated with M and algorithms that compute the set of forms associated with M^T, the transpose of M. This gives a tight relationship between the complexities of these related computational tasks. This relationship leads to new upper bounds on the complexity of matrix vector products when the coefficients of M are drawn from a small set. It also permits simple derivations of lower bounds on the number of additions necessary to evaluate certain sets of linear forms.

The duality result also implies a strong relationship between linear algorithms for computing the matrix vector product $M\mathbf{x}$ and bilinear algorithms for computing the bilinear form $\mathbf{y}^T M\mathbf{x}$. This correspondence is reminiscent of the correspondence between bilinear algorithms for computing a set of bilinear forms and trilinear algorithms for computing a trilinear form defined by the corresponding three dimensional tensor (cf. [3]).

This paper is organized as follows. In Section 2, we give a graph theoretic definition of linear algorithms. This definition suggests a natural notion of transposed algorithms. Theorem 1 establishes a relationship between the computations of a linear algorithms and its transpose. Section 3 explores some of the applications of Theorem 1 for bounding the arithmetic complexity of certain matrix vector products. Section 4 establishes the correspondence between linear algorithms for computing $M\mathbf{x}$ and bilinear

algorithms for computing $\mathbf{y}^T M\mathbf{x}$.

2. Linear algorithms and transposed linear algorithms

In their conventional definition (cf. [2,6,8]) linear algorithms are presented as a sequence of elementary computation steps - a straight-line program. While this definition is appealing because of its correspondence with low level programs, it imposes more structure than is necessary on the underlying process, possibly disguising certain fundamental attributes such as inherent symmetries. We choose to present linear algorithms as labeled directed acyclic graphs. The correspondence between our definition and conventional definitions is straightforward; essentially all topological sortings of the vertices of our directed acyclic graph algorithms give equivalent straight line programs. One of the advantages of moving to a graph-theoretic definition is the natural way which it permits (in fact, suggests) the definition of a transposed algorithm - literally, the original algorithm "run backwards".

Throughout this section R denotes an arbitrary communicative ring with additive and multiplicative units 0 and 1, respectively.

Definition 1. A *linear algorithm* A over R is a triple (V,E,λ) where

i) (V,E) is a directed acyclic graph with vertex set V and edge set E; and

ii) $\lambda: E \rightarrow R-\{0\}$.

We denote by V^I (respectively, V^O) the subset of V consisting of vertices of in-degree (respectively, out-degree) 0. V^I (respectively, V^O) corresponds to the input (respectively, output) vertices of algorithm A.

Let $|V^I|=t$ and $|V^O|=s$ and suppose that $\beta:\{1,...,t\} \rightarrow V^I$ and $\gamma:\{1,...,s\} \rightarrow V^O$ are arbitrary bijections (i.e. indexings of V^I and V^O, respectively). Hereafter, we take β and γ to

be fixed for A. However, different choices for β and γ amount to nothing more than renamings (or reorderings) of inputs and outputs.

Let $X=(x_1,...)$ denote a sequence of distinct indeterminates over R. Denote by $R\{X\}$ the set of linear forms in the indeterminates X with coefficients from R. $\mathbf{x}_i$ denotes the column vector $(x_1,x_2,...,x_i)^T$.

Definition 2. Let $A=(V,E,\lambda)$ be a linear algorithm over R. With each vertex $v\in V$ we can associate an element $f(v)\in R\{X\}$, called the ***linear form associated with*** v, as follows:

$$f(v) = \begin{cases} x_{\beta^{-1}(v)} & v\in V^I \\ \sum_{(w,v)\in E} \lambda(w,v) f(w) & v\in V-V^I \end{cases}$$

Definition 3. A sequence $v_1,...,v_k$, describes a ***path*** in the algorithm $A=(V,E,\lambda)$ if

i) $v_i\in V$, $1\le i\le t$; and

ii) $(v_i,v_{i+1})\in E$, $1\le i<t$.

The path $v_1,...,v_t$ is said to *connect* vertex v_1 to vertex v_t. The ***weight*** of path $P=v_1,...,v_t$, denoted $w(P)$, is given by

$$w(P) = \begin{cases} 1 & t=1 \\ \prod_{j=2}^{t} \lambda(v_{j-1},v_j) & \textit{otherwise.} \end{cases}$$

Let $\mathbf{P}_A(v,w)$ denote the set of all paths connecting vertex v to vertex w in algorithm A. If $P=v_1,...,v_t$, then P^T denotes the sequence $v_t,...,v_1$.

Paths and their associated weights allow us to give an alternative characterization of the linear form associated with an arbitrary vertex in an algorithm A.

Lemma 1. Let $A=(V,E,\lambda)$ be a linear algorithm over R with $|V^I|=t$. For each vertex

$v \in V$, $f(v)=(\eta_1,\ldots,\eta_t)\mathbf{x}_t$ where

$$\eta_i = \sum_{P \in \mathbf{P}_A(\beta(i),v)} w(P).$$

Proof. Straightforward induction on the depth of v, i.e. the length of the longest path connecting a vertex in V^I to v.

□

Definition 4. We say that algorithm $A=(V,E,\lambda)$ *computes* a set of forms $\{f_1,\ldots,f_s\} \subseteq R\{X\}$ if for each f_i there exists a $v \in V^O$ such that $f_i=f(v)$. Suppose that $|V^I|=t$ and $|V^O|=s$. We denote by F_A the vector $(g_1,\ldots,g_s)^T$, where $g_j=f(\gamma(j))$ (i.e. g_j is the linear form associated with the jth output vertex of A). The ***matrix associated with algorithm*** A, denoted M_A, is the $s \times t$ matrix over R defined by $M_A\mathbf{x}_t=F_A$.

Lemma 2. The $(i,j)th$ entry of M_A is given by

$$M_A(i,j) = \sum_{P \in \mathbf{P}_A(\beta(i),\gamma(j))} w(P).$$

Proof. This is immediate from Lemma 1.

□

Definition 5. Let $A=(V_1,E_1,\lambda_1)$ be any linear algorithm over R. We denote by A^T the linear algorithm over R obtained from A by reversing all of the edges in E. More formally, $A^T=(V_2,E_2,\lambda_2)$ where $V_2=V_1$, $E_2=\{(v,w)|(w,v) \in E_1\}$ and $\lambda_2(v,w)=\lambda_1(w,v)$, for all $(w,v) \in E_1$. We refer to A^T as the *transpose* of algorithm A.

It should be clear that $V_2^I=V_1^O$ and $V_2^O=V_1^I$. Suppose that β_2 and γ_2 are chosen so that $\beta_2=\gamma_1$ and $\gamma_2=\beta_1$ (i.e. the input vertices of A^T are indexed in the same way as the

output vertices of A, and vice versa). Then we have the following correspondence between a linear algorithm and its transpose and their associated matrices.

Theorem 1. Let A be any linear algorithm over R. Then $M_{A^T}=(M_A)^T$.

Proof. Since $P\in\mathbf{P}_A(\beta_1(i),\gamma_1(j))$ if and only if $P^T\in\mathbf{P}_{A^T}(\beta_2(j),\gamma_2(i))$, it follows from Lemma 2 that $M_A(i,j)=M_{A^T}(j,i)$ for all i,j.

□

3. Multiplying a vector by a matrix and its transpose

Theorem 1 expresses a kind of duality between computations of a linear algorithm and its transpose. In this section, we explore some of the implications of this duality in studying the complexity of matrix vector products.

Definition 6. If $A=(V,E,\lambda)$ is a linear algorithm define $\alpha(A)$ and $\xi(A)$ as $\alpha(A)=|E|-|V|+|V^I|$ and $\xi(A)=|\{e\in E|\lambda(e)\neq 1\}|$. $\alpha(A)$ (respectively, $\xi(A)$) gives the number of additions (respectively, scalar multiplications) used by algorithm A.

Note.

1. If $\{g_1,\ldots,g_s\}\subseteq R\{X\}$ and A is a linear algorithm that computes $\{g_1,\ldots,g_s\}$ then there exists a linear algorithm A' with $\alpha(A')\leq\alpha(A)$ and $\xi(A')\leq\xi(A)$ for which $F_{A'}=(g_1,\ldots,g_s)^T$.
2. If A is a linear algorithm with $|V^I|=t$ and $|V^O|=s$ then $\alpha(A^T)=\alpha(A)+s-t$ and $\xi(A^T)=\xi(A)$.

Definition 7. Let $G=(g_1,\ldots,g_s)^T$ where $g_i\in R\{X\}$, $1\leq i\leq s$, (i.e. G is a vector of s linear forms over R). We define,

$$\mu(G)=min\{\alpha(A)|F_A=G\}.$$

Thus, any linear algorithm A for which $F_A=G$ uses at least $\mu(G)$ additions.

Theorem 2. Let M be an $s\times t$ matrix over R without zero rows or columns. Then,

$$\mu(M^T\mathbf{x}_s)=\mu(M\mathbf{x}_t)+s-t.$$

Proof. Suppose algorithm A computes $M\mathbf{x}_t$. Then without loss of generality, $M_A=M$. But, by theorem 1, $M_{A^T}=(M_A)^T$ and hence algorithm A^T computes $M^T\mathbf{x}_s$. Since M has no zero rows or columns, it follows that $|V^I|=t$ and $|V^O|=s$. Hence, $\alpha(A^T)=\alpha(A)+s-t$. It follows that $\mu(M^T\mathbf{x}_s)\leq\mu(M\mathbf{x}_t)+s-t$, and by symmetry $\mu(M\mathbf{x}_t)\leq\mu(M^T\mathbf{x}_s)+t-s$.

□

Savage [7] points out that conventional algorithms for matrix vector products can be improved when the coefficient set of the matrix is small. Specifically,

Theorem 3. (Savage)

Let M be an $s\times t$ matrix over R with at most r distinct coefficients. Then

$$\mu(M\mathbf{x}_t)\leq\begin{cases} O(st/log_r s) & t\geq log_r s \\ O(r^t) & t<log_r s \end{cases}$$

As a corollary to theorem 2 and theorem 3, we have the following extension of theorem 3.

Corollary 1. Let M be an $s\times t$ matrix over R with at most r distinct coefficients. Then

$$\mu(M\mathbf{x}_t) \leq \begin{cases} O(st/\log_r s) & s \geq t \geq \log_r s \\ O(r^t) & t < \log_r s \\ O(st/\log_r t) & t \geq s \geq \log_r t \\ O(t) & s < \log_r t \end{cases}$$

Proof. If $s \geq t$ apply theorem 3 directly. Otherwise, apply theorem 3 to the computation of $M^T\mathbf{x}_s$ and convert the result using theorem 2.

□

Note. The improvements in number of additions provided by the bound of Corollary 1 do not come at the expense of an increase in the number of scalar multiplications. Indeed, Savage's algorithm uses tr scalar multiplications and the algorithm implicit in Corollary 1 uses $\min\{sr,tr\}$ scalar multiplications.

Corollary 1, in turn, provides a new variant of Kronrod's algorithm [1] for Boolean matrix products.

Corollary 2. Let P and Q be $s \times t$ and $t \times u$ Boolean matrices. The product PQ can be computed in $O(\frac{stu}{\max\{\log_2 s, \log_2 t, \log_2 u\}})$ additions, provided $s,u \geq \log_2 t$, and $O(tu)$ additions, otherwise.

Theorem 2 can also be used to simplify certain lower bound arguments.

Corollary 3. (cf. [2, p. 14])

If M is a $1 \times t$ matrix over $R-\{0\}$ then $\mu(M\mathbf{x}_t)=t-1$

Proof. M^T is a $t \times 1$ matrix and hence $\mu(M^T\mathbf{x}_1)=0$. The result follows from Theorem 2.

□

Corollary 4. ([5]) Let M be a $2\times t$ matrix over $R-\{0\}$ such that any two columns of M are linearly independent. Then any linear algorithm that computes $M\mathbf{x}_t$ requires $2t-2$ additions.

Proof. Obviously $M^T\mathbf{x}_2$ requires t additions, since to obtain a linear form which is independent from any already computed requires an addition.

□

Note. Both Corollary 3 and Corollary 4 are known to hold for more general families of algorithms. (cf. [2,5]).

Corollary 4 can be generalized as follows. Let R be a principal ideal ring. Define an equivalence relation $\sim$ on R^2 by $\mathbf{u}\sim\mathbf{v}$ if and only if the vectors $\mathbf{u}$ and $\mathbf{v}$ are linearly independent.

Corollary 5. If M is a $2\times t$ matrix without zero rows or columns and r is the number of equivalence classes (under $\sim$) of the columns of M not containing a zero element then any linear algorithm that computes $M\mathbf{x}_t$ requires $r+t-2$ additions.

4. Bilinear algorithms and bilinear forms

In this section we describe some of the implications of the results of section 3 for the computation of bilinear forms using bilinear algorithms.

Definition 8. A *bilinear algorithm* (cf. [3]) B over R is a triple (A_1,A_2,A_3) where

i) $A_i=(V_i,E_i,\lambda_i)$ is a linear algorithm over R, $1\leq i\leq 3$; and

ii) $|V_1^O|=|V_2^O|=|V_3^I|$.

Suppose $|V_1^I|=t$, $|V_2^I|=s$ and $|V_3^I|=r$. B is said to compute the set of bilinear forms

$$(M_{A_3})^T[(M_{A_2}\mathbf{y}_s)\cdot(M_{A_1}\mathbf{x}_t)]$$

where $\cdot$ denotes outer (componentwise) product.

Definition 9. The set of coefficients $\{m_{ijk}\}$, $1\le i\le t$, $1\le j\le s$, $1\le k\le r$, is *degenerate* if there exists an i such that $m_{ijk}=0$, $1\le j\le s$, $1\le k\le r$, or there exists a j such that $m_{ijk}=0$, $1\le i\le t$, $1\le k\le r$, or there exists a k such that $m_{ijk}=0$, $1\le i\le t$, $1\le j\le s$.

Theorem 4. Suppose that $\{m_{ijk}\}$ is a non-degenerate set of coefficients and that the set of bilinear forms $\{\sum_{i=1}^{t}\sum_{j=1}^{s} m_{ijk}x_iy_j\}k=1,\ldots,r$ can be computed by a bilinear algorithm in α additions. Then the dual sets $\{\sum_{i=1}^{t}\sum_{k=1}^{r} m_{ijk}x_iz_k\}j=1,\ldots,s$ and $\{\sum_{j=1}^{s}\sum_{k=1}^{r} m_{ijk}y_iz_k\}i=1,\ldots,t$ can be computed by bilinear algorithms in $\alpha+r-s$ and $\alpha+r-t$ additions respectively.

Proof. It is easy to confirm that if $\{\sum_{i=1}^{t}\sum_{j=1}^{s} m_{ijk}x_iy_j\}k=1,\ldots r$ is computed by the bilinear algorithm (A_1,A_2,A_3) then $\{\sum_{i=1}^{t}\sum_{k=1}^{r} m_{ijk}x_iz_k\}j=1,\ldots,s$ (respectively, $\{\sum_{j=1}^{s}\sum_{k=1}^{r} m_{ijk}y_jz_k\}i=1,\ldots,t$) is computed by the bilinear algorithm (A_1,A_3^T,A_2^T) (respectively (A_3^T,A_2,A_1^T)). The result follows from Theorem 2.

□

Note. The proof above also demonstrates that the numbers of non-scalar (respectively, scalar) multiplications required by these three dual sets are identical. See [3] for a more comprehensive treatment of non-scalar multiplications in this setting.

Definition 10. A bilinear algorithm (A_1,A_2,A_3) is *elementary* if $M_{A_3}=(1,1,\ldots,1)^T$. An elementary bilinear algorithm computes the single bilinear form $(M_{A_2}\mathbf{y}_s)^T(M_{A_1}\mathbf{x}_t)$. It is with no loss of generality that we assume that bilinear algorithms that compute a single

bilinear form are elementary.

Definition 11. Let $A_1=(V_1,E_1,\lambda_1)$ and $A_2=(V_2,E_2,\lambda_2)$ be linear algorithms over R with $|V_1^O|=|V_2^I|$. The *composition* of A_1 and A_2, denoted $A_2 \circ A_1$, is the linear algorithm formed from A_1 and A_2 by identifying the output vertices of A_1 with the input vertices of A_2 in a 1–1 fashion. It should be clear that $M_{A_2 \circ A_1}=M_{A_2}M_{A_1}$.

Theorem 5. Let M be an $s\times t$ matrix over R. Let $B=(A_1,A_2,A_3)$ be any elementary bilinear algorithm which computes the bilinear form $\mathbf{y}_s^T M\mathbf{x}_t$. Then $(A_2)^T \circ A_1$ computes $M\mathbf{x}_t$.

Proof. By definition $M=(M_{A_2})^T M_{A_1}$. But, by Theorem 1, $(M_{A_2})^T=M_{(A_2)^T}$. Thus $M=M_{(A_2)^T \circ A_1}$.

□

Theorem 5 demonstrates that any bilinear algorithm for computing the bilinear form $\mathbf{y}_s^T M\mathbf{x}_t$ can be transformed simply into a linear algorithm that computes $M\mathbf{x}_t$. (In particular, a bilinar form $\mathbf{y}_s^T M\mathbf{x}_t$ can be evaluated optimally - among bilinear algorithms - by an algorithm that first computes $M\mathbf{x}_t$). In fact, many different bilinear algorithms will transform to the same linear algorithm. The converse of Theorem 5 also holds. That is, any linear algorithm that computes $M\mathbf{x}_t$ can be transformed into many different - but equally efficient - bilinear algorithms that compute $\mathbf{y}_s^T M\mathbf{x}_t$.

Theorem 6. Let M be an $s\times t$ matrix over R and suppose that algorithm A computes $M\mathbf{x}_t$. Then if $A=A_2 \circ A_1$, the elementary bilinear algorithm $B=(A_1,(A_2)^T,A_3)$ computes the bilinear form $\mathbf{y}_s^T M\mathbf{x}_t$.

Proof. By definition $M=M_{A_2 \circ A_1}$. But B computes $\mathbf{y}_s^T(M_{(A_2)^T})^T M_{A_1}\mathbf{x}_t$. Since $(M_{(A_2)^T})^T=M_{A_2}$, by Theorem 1, the result follows from the definition of composition.

□

Note that if A is any linear algorithm then any *cut* of A, that is any minimal set of vertices of A that intersects any path connecting input and output vertices of A, defines two linear algorithms A_1 and A_2 (formed by edges on the input and output sides of the cut respectively) such that $A=A_2 \circ A_1$. Thus, to each cut C of A there corresponds a distinct bilinear algorithm computing $\mathbf{y}_s^T M_A \mathbf{x}_t$. The number of additions used by each such bilinear algorithm is

$$\alpha(A_1)+\alpha((A_2)^T)+|C|-1$$

$$=\alpha(A_1)+\alpha(A_2)+s-1$$

$$=\alpha(A)+s-1.$$

Hence, the conventional ways of computing $\mathbf{y}_s^T M\mathbf{x}_t$, by evaluating $\mathbf{y}_s^T(M\mathbf{x}_t)$ or $(\mathbf{y}_s^T M)\mathbf{x}_t$, are just two of many equivalent (in terms of additions and scalar multiplications) possibilities.

Note. Analogs of Theorems 4, 5 and 6 also hold for the computation of N-linear forms using multilinear algorithms (cf. [4]).

References

[1] V.L. Arlazarov, E.A. Dinic, M.A. Kronrod and I.A. Faradžev, *On economical construction of the transitive closure of an oriented graph,* Soviet Math. Dokl., 11 (1970), pp. 1209-1210.

[2] A. Borodin, J.I. Munro, *The Computational Complexity of Algebraic and Numeric Problems,* Americal Elsevier Publishing Co., 1975.

[3] R. Brockett, D. Dobkin, *On the Optimal Evaluation of a Set of Bilinear Forms,* Linear Alg. Appl. 19 (1978), 207-235.

[4] D. Dobkin, *On the optimal evaluation of a set of N-linear forms,* Proc. 14th Annual IEEE Symp. on Switching and Automata Theory (1973), 92-102.

[5] D.G. Kirkpatrick, *On the additions necessary to compute certain functions.* Proc. 4th Annual ACM Symp. on Theory of Computing (1972), 94-101.

[6] J. Morgenstern, *The Linear Complexity of Computation,* J. ACM 22 (1975), 184-194.

[7] G.E. Savage, *An algorithm for computation of linear forms,* SIAM J. Comp. 3 (1974), 150-156.

[8] V. Strassen, *Vermeidung von Divisionen,* J. Reine Angew. Math. 264 (1973), 184-202.