Time–Space Tradeoffs for Algebraic Problems on General Sequential Machines*

KARI ARRAHAMSON

Department of Computer Science, Washington State University, Pullman, Washington 99164–1210

Received May 15, 1987; revised July 19, 1989

This paper establishes time–space tradeoffs for some algebraic problems in the branching program model, including convolution of vectors, integer multiplication, matrix–vector products, matrix multiplication, matrix inversion, computing the product of three matrices, and computing \( PAQ \) where \( P \) and \( Q \) are permutation matrices. The lower bounds apply to general sequential models of computation. Although the lower bounds are for a more general model, they are as large as the known bounds for straight-line programs (even improving the known straight-line bounds for matrix multiplication) except for the case of computing \( PAQ \), for which non-oblivious algorithms can outperform oblivious ones, and integer multiplication, where our lower bound is a polylogarithmic factor below the known straight-line bound. Some of the tradeoffs are proved for expected time and space, where all inputs are equally likely. © 1991 Academic Press, Inc.

1. INTRODUCTION

Straight-line programs, and the related pebble game, have been used extensively in demonstrating time–space tradeoffs. Results are known for problems including sorting [18], convolution, matrix–vector products and discrete Fourier transform [18], binary integer multiplication [17], matrix multiplication and inversion, iterated matrix multiplication [9, 11, 16], and others [8, 13–15, 19, 22].

An alternative approach, based on branching programs, has been developed by Borodin et al. [7] and Borodin and Cook [4], and is applied to sorting in both cases. (The former restricts attention to comparison algorithms, but the latter does not.) Yesha [24] uses branching programs to establish time–space tradeoffs for matrix multiplication and discrete Fourier transform. Other problems to which this technique has been applied include element distinctness [6, 23], finding unique elements [3], and a string matching problem [1]. The consequences of very severe space restrictions are investigated in [5, 21].

Unlike straight-line programs, which model oblivious algorithms, branching programs model algorithms that make decisions on the fly. Lower bounds for

* A preliminary version of this paper appeared in in the 27th Annual Symposium on Foundations of Computer Science.
branching programs, as they are defined here, apply to a general sequential model of computation. Consequently, results for branching programs are preferred, when they can be had, over comparable results for straight-line programs. The preference is strongest for problems whose algorithms could benefit, even only apparently, from non-oblivious behavior. An example is matrix multiplication over $\mathbb{F}_2$, where one could conceivably (although not in reality) avoid effort by exploiting zeros.

This paper demonstrates that the branching program model is capable of supporting strong results for algebraic problems. The results are summarized in the next section. Features of the results are

1. With the exception of bounds for integer multiplication and computing $P A Q$, our lower bounds are at least as strong as known results for straight-line programs.

2. In the case of matrix multiplication, our lower bound is stronger than the previously known lower bound for the more restrictive straight-line programs, when all of the results are restricted to matrices over a field. The reason for this surprising fact is discussed after its proof.

3. For the problem of computing $P A Q$, where $P$ and $Q$ are permutation matrices, the ability to make decisions on the fly does help. Our bounds for branching programs are closer to the actual cost for random access machines than results for straight-line programs for this problem.

4. Our results for matrix multiplication, convolution, and for computing the product of three matrices are tight to within a constant factor within the branching program model, for a wide range of time and space values. Consequently, any stronger results must rely on features of real algorithms not modeled by branching programs.

5. Some of our lower bounds (those that are proved directly) apply to expected time and space. Results proved by reduction are for the worst case.

2. SUMMARY AND RELATED RESULTS

The results concern multiple input, multiple output functions over a field. Throughout, $\mathcal{F}$ is any field, $\mathcal{D} = \{d_1, ..., d_\delta\}$ is a finite subset of $\mathcal{F}$, and $\delta = |\mathcal{D}| \geq 2$. The functions accept $n$ inputs in $\mathcal{D}$, and produce $m$ outputs in $\mathcal{F}$, for some $n$ and $m$. For this summary, $S$ and $T$ represent space and time, respectively, required to solve the given problem. Although the same symbols $S$ and $T$ are used for both the straight-line and the branching program models, space and time are somewhat different in the two models. The major difference is that, in the straight-line model, space is measured in words, where each word can hold one member of $\mathcal{F}$, while in the branching program model, space is measured at the bit level. As a result, there
is a factor of log δ in the results for branching programs that does not appear in results for straight-line programs. Details of the branching program model can be found in Section 3.

For straight-line programs, only worst case complexity makes sense. For branching programs, there is an obvious notion of expected time complexity, and a less obvious but meaningful notion of expected space. Some of our lower bounds for branching programs apply to expected space and time. Symbols $S$ and $T$ represent expected space and time, respectively, where all inputs are equally likely.

We begin by studying matrix-vector products, since results for such products can be applied to a variety of problems. Since the results on matrix-vector products depend on technical definitions, they are not summarized in this section.

The next two results concern convolution and the related problem of binary integer multiplication. Both are known to require $ST = \Omega(n^2)$ in the straight-line model [17, 18]. Our results are that $ST = \Omega(n^2 \log \delta)$ for convolution of $n$ component vectors over $\mathcal{D}$, and $ST = \Omega(n^2/\log \delta)$ for multiplication of two $n$ bit binary integers, in the branching program model. The constant implicit in the $\Omega$, as in all to follow, is independent of $n$, $\mathcal{F}$, and $\mathcal{D}$. Yesha [24] has shown that the related problem of computing the discrete Fourier transform of an $n = p - 1$ component vector over $GF(p)$, for $p$ prime, requires $ST = \Omega(n^2)$. We strengthen Yesha's result to expected space and time, and remove the restriction on the field.

The next problems are the related problems of computing the product $ABC$ of three $n \times n$ matrices over $\mathcal{D}$, and of inverting an upper triangular matrix. For computing $ABC$, we can show $ST = \Omega(n^4 \log \delta)$, which is tight to within a constant factor. For inversion, a worst case bound of $ST = \Omega(n^4 \log \delta)$ is shown, strengthening Ja'Ja's result [11] of $ST = \Omega(n^4)$ for matrix inversion over a field in the straight-line model.

The next function is $n \times n$ matrix multiplication, restricted to input matrices over $\mathcal{D}$. In the straight-line model, Grigoryev [9] and Ja'Ja' [11] have shown that $ST = \Omega(n^2)$. Ja'Ja's result holds for any ring. In the branching program model, Yesha [24] has shown that, for every $n$, there is a finite field $\mathcal{F}$, of size $\Theta(n)$, such that $n \times n$ matrix multiplication over $\mathcal{D}$ requires $ST = \Omega(n^3)$. Both of the above results are strengthened here, when restricted to a field. For any field $\mathcal{F}$ and any finite subset $\mathcal{D} \subseteq \mathcal{F}$ of size $\delta \geq 2$, $n \times n$ matrix multiplication over $\mathcal{D}$ requires $ST = \Omega(n^6 \log \delta)$ in the branching program model. When $T = O(n^2)$, our bound gives $S = \Omega(n^2 \log n)$, while previous bounds give $S = \Omega(n)$. Our bound can be achieved to within a constant factor within the model, assuming $\mathcal{D} = \mathcal{F}$, for any $S$ satisfying $\log(\delta n) \leq S \leq n^2 \log \delta$.

The next result concerns computation of the matrix product $PAQ$, where $P$ and $Q$ are $n \times n$ permutation matrices, and $A$ is a fixed $n \times n$ matrix, all of whose elements are distinct. It follows from results of Savage [16] and Vuillemin [20] that any straight-line program to compute $PAQ$ requires $ST = \Omega(n^4)$. But there is a random access program that computes $PAQ$ in $ST = O(n^3 \log^2 n)$, for a wide range of $S$ and $T$. A lower bound of $ST = \Omega(n^3)$ is shown to hold for branching programs, and $ST = O(n^3 \log n)$ is achievable in that model.
3. Branching Programs

This section describes branching programs, and contains a key lemma, based on the ideas of Borodin and Cook, for establishing time-space tradeoffs.

The branching program model is a generalization of the decision tree model. It has been described and justified elsewhere [1, 4, 7, 21, 24], so the description here will be short.

Let \( f : \mathcal{D}^n \rightarrow \mathcal{F}^m \), where \( \mathcal{D} = \{d_1, \ldots, d_b\} \). A \( b \)-way branching program \( P \) that computes \( y = f(x) \) is a directed acyclic graph with a single source (a node with in-degree zero), and possibly many sinks (nodes with out-degree zero). Each non-sink has out-degree \( \delta \), and is labeled by a query of the form \( \text{"x}_i = ?" \) for some \( 1 \leq i \leq n \). Each arc is labeled by a response to a query. If non-sink \( u \) is labeled by query \( \text{"x}_i = ?" \), then the \( \delta \) arcs exiting \( u \) are labeled with the \( \delta \) possible responses \( \text{"x}_i = d_1", \ldots, \text{"x}_i = d_{\delta}" \).

Each node is also labeled by a possibly empty set of outputs, each of the form \( \text{"y}_j = a" \), where \( a \in \mathcal{F} \) and \( 1 \leq j \leq m \).

Each input vector \( x \) defines a computation in \( P \), namely the unique directed path from the source to a sink whose responses are consistent with \( x \). The outputs are just those that appear along the computation.

The worst case time complexity \( T \) of \( P \) is the length of the longest computation. When a probability distribution is assigned to the inputs, the expected time complexity \( \bar{T} \) is defined in the obvious way.

Suppose \( P \) has \( k \) nodes. The (worst case) space complexity \( S \) of \( P \) is defined as \( \log_2 k \). (All logarithms in this paper are to base 2.)

We would also like a notion of expected space. Since the branching program model is primarily used for proving lower bounds, we choose a notion that gives at least as small a value for expected space as any other reasonable notion. A numbering of \( P \) is a 1-1 function assigning one of the integers \( 1, \ldots, k \) to each of the nodes of \( P \). (The number of a node represents the encoding of the corresponding state of a machine.) Relative to a particular numbering, the space used by \( P \) on input \( x \) is the logarithm of the largest number of a node on the computation for input \( x \). The expected space complexity \( S \) is defined as the minimum, over all numberings, of the expected space used on a random input (relative to the numbering and to a given probability distribution).

With two restrictions, lower bounds for branching programs apply, to within a constant factor, to general sequential models of computation, including multi-tape Turing machines and logarithmic cost random access machines. The first restriction is that space is assumed to be at least logarithmic. The second is that each output component must be produced as a unit. For example, if numbers are represented in binary, then it is not permissible to produce one bit of \( y_1 \), followed by one bit of \( y_2 \), then a second bit of \( y_1 \).

It is important to note that the definition of space complexity of Turing machines and random access machines counts only work space. The input is assumed to be given in a random access read-only input memory, and the output is placed in a random access write-only output memory, and neither input nor output space is
counted. Such a definition is a natural one, and permits sublinear space algorithms. For more detail, see [10, 12]. Discussions of the relationship between branching programs and other models for worst case complexity can be found in [1, 4, 24].

Proofs for average case complexity are nearly identical to those for the worst case. Say that branching program $P$ is in normal form if its nodes are organized into levels, where for each arc $(u, v)$, $u$ is in level $i$ and $v$ is in level $i + 1$, for some $i$.

Any branching program can be put into normal form by replicating the entire program at each level, and altering the arcs in the obvious way. If the worst case time is $T$, then the number of nodes is increased by a factor of $T$, which adds log $T$ to the worst case space. Since it is obvious that $S \geq \log T$, we have only increased the worst case space by a constant factor. For the problems considered here, there is an obvious lower bound on the best case time $T_{\text{best}}$, which is within a polynomial of any reasonable worst case time $T$. Since $S \geq \log T_{\text{best}}$, converting to normal form also increases the expected space by only a constant factor. Therefore, we can assume that all branching programs are in normal form.

The Borodin-Cook technique is probabilistic. For each function, a probability distribution is assigned to the input vectors. Typically, but not necessarily, all inputs are considered equally likely. All probability calculations for a given function are understood to be relative to its associated input distribution.

The key lemma relates properties of branching programs for a given function to properties of the more tractable tree programs. A tree program is just a branching program with a tree structure. A tree program is non-redundant if no computation contains two nodes with the same query. Following two definitions, the lemma is stated in two forms, one for worst case, the other for expected case. Since the proof for worst case appears in [1, 4, 7, 24], and is similar to the for expected case, only the version for expected case is proved here.

**Definition 3.1.** A computation $\pi$ $k$-solves vector $x$ (w.r.t. function $f$) iff $\pi$ produces at least $k$ outputs, all of the query responses on $\pi$ are consistent with $x$, and all of the outputs are correct (w.r.t. $f(x)$). A tree program $\tau$ $k$-solves $x$ iff the computation of $\tau$ on $x$ $k$-solves $x$.

**Definition 3.2.** Let $f: \mathcal{D}^n \rightarrow \mathcal{F}^m$ be a function and $I$ be an input probability distribution for $f$. Let $\alpha$ and $\beta$ be positive integers. For any non-redundant tree program $\tau$, define $C_{f, I}(\tau, \beta)$ as the probability that $\tau$ $\beta$-solves a random input $x$ (w.r.t. $f$). $C_{f, I}(\tau, \beta)$ provides a measure of the portion of the entire solution of $f$ covered by $\tau$. Define $C_{f, I}(\alpha, \beta)$ as the maximum, over all depth $\alpha$ non-redundant tree programs $\tau$, of $C_{f, I}(\tau, \beta)$. In what follows, the input distribution $I$ will be left implicit, and we will refer to $C_f(\alpha, \beta)$.

**Lemma 3.3.** Let $f: \mathcal{D}^n \rightarrow \mathcal{F}^m$, and let $\alpha$ and $\beta$ be positive integers. Let $P$ be a normal form branching program computing $f$. 
(i) If \( P \) has worst case time \( T \) and space \( S \), with \( \alpha \leq n \leq T \) and \( \beta \leq \lceil m\alpha/2T \rceil \), then \( S \geq -\log C_f(\alpha, \beta) \).

(ii) If \( P \) has expected time \( \bar{T} \) and expected space \( \bar{S} \), with \( \alpha \leq n \leq \bar{T} \) and \( \beta \leq \lceil m\alpha/4\bar{T} \rceil \), then \( \bar{S} \geq -(1/3) \log C_f(\alpha, \beta) - 1 \).

Proof. We prove part (ii) only. Suppose that the nodes of \( P \) have been numbered in such a way as to minimize the expected space complexity, relative to the given numbering. Truncate \( P \) at depth \( t = 2\bar{T} \), and call the truncated program \( P' \). With probability at least \( 1/2 \), \( P' \) completes the solution of a random input. Break \( P' \) into \( s = \lfloor (t + 1)/(\alpha + 1) \rfloor \leq 2t/\alpha \) disjoint stages, where, for \( i = 1, \ldots, s \), stage \( i \) consists of levels \( (i - 1)(\alpha + 1) \) to \( i(\alpha + 1) - 1 \). Each stage is a branching program of depth \( \alpha \), except that it may have many sources. In the obvious way, unwind the program within each stage, duplicating shared nodes, so that each stage becomes a collection of disjoint trees.

Now some operations are performed on the tree programs comprising the modified stages. In a tree program, it cannot hurt to insist that outputs be given only at the leaves. Push all of the outputs down to the leaves, preserving the input/output behavior of the trees. Also, in a tree program it cannot help to make redundant queries. Eliminate redundant queries from the trees, preserving their input/output behavior, and pad the trees to a uniform depth of \( \alpha \) by adding superfluous but non-redundant queries as needed. Any numbers can be assigned to the new nodes, since we are only concerned with the numbers of the root nodes. Call the resulting program \( P'' \).

Imagine running \( P'' \) on a random input \( x \). With probability at least \( 1/2 \), \( m \) outputs are produced. Since there are at most \( 2t/\alpha \) stages, with probability at least \( 1/2 \) some stage must produce at least \( \lceil m\alpha/2t \rceil \geq \beta \) outputs on input \( x \). So

\[
\Pr(\text{some tree in some stage } \beta\text{-solves } x) \geq \frac{1}{2}.
\]  

(3.1)

The root of each tree in each stage of \( P'' \) corresponds to a node in \( P \). Let \( \tau_i \) be the tree in \( P'' \) whose root is numbered \( i \), for each \( i \), if such a tree exists. Note that the computation of \( P'' \) on \( x \) passes through the root of \( \tau_i \) just when the computation of \( P \) on \( x \) passes through the corresponding node. Let \( p_i \) be the probability that, for a random input \( x \), (a) the computation of \( P'' \) on \( x \) passes through the root of \( \tau_i \), and (b) \( \tau_i \) \( \beta \)-solves \( x \). If \( \tau_i \) does not exist, let \( p_i = 0 \).

Let \( u = 2^{35} \) and \( q = \sum_{i \geq u} p_i \). Two facts about the probabilities \( p_i \) are evident. (1) \( p_i \leq C_f(\alpha, \beta) \), by the definition of \( C_f(\alpha, \beta) \). (2) \( q \leq 1/3 \), since otherwise the expected space would exceed \( \bar{S} \). So

\[
\Pr(\text{some tree in some stage } \beta\text{-solves } x) \leq \sum_{i < u} p_i + \sum_{i \geq u} p_i
\leq 2^{35} C_f(\alpha, \beta) + \frac{1}{3}
\]  

(3.2)

Combining inequalities (3.1) and (3.2) yields the desired result. 

\[ \blacksquare \]
4. Matrix–Vector Products

A number of different problems can be expressed as matrix–vector products. As a result, bounds on the complexity of matrix–vector products can go a long way. Tompa [18], for example, using a result of Valiant on matrix vector products, is able to establish time–space tradeoffs for convolution and discrete Fourier transform in the straight-line model.

This section contains a lower bound on the complexity of computing matrix–vector products. Before getting to the lower bound, though, some technical definitions and results on matrices are required.

Recall that, throughout this and subsequent sections, $\mathbb{F}$ is any field, and $\mathcal{D} = \{d_1, \ldots, d_\delta\}$ is any finite subset of $\mathbb{F}$ of size $\delta \geq 2$. The following three lemmas are proved in the appendix.

**Lemma 4.1.** Suppose $\mathcal{F} \subseteq \mathbb{F}^n$ is contained in an affine subspace $\mathcal{A}$ of $\mathbb{F}^n$ of dimension at most $r$. Then $|\mathcal{F}| \leq \delta^r$.

**Definition 4.2.** Let $0 < c < 1/2$. Let $A$ be an $m \times n$ matrix with $m \leq n$. Say that $A$ is $c$-nice iff every $p \times q$ submatrix of $A$ has rank $p$, for $p \leq \lceil c m \rceil$ and $q \geq \lfloor (1-c) n \rfloor$. Say that $A$ is $c$-ok iff every such submatrix of $A$ has rank at least $c p$.

**Lemma 4.3.** There is a constant $\gamma$, where $0 < \gamma < 1/2$, such that at least a fraction $1 - \delta^{-1}(2/3)^m$ of the $n \times n$ matrices over $\mathcal{D}$ are $\gamma$-nice.

**Definition 4.4.** Matrix $A$ is Toeplitz iff $A_{i,j}$ is a function of $j-i$, for each $i$ and $j$. That is, each diagonal is constant.

**Lemma 4.5.** There is a constant $\gamma$, $0 < \gamma < 1/2$, such that at least a fraction $1 - \delta^{-1}(2/3)^m$ of the $n \times n$ Toeplitz matrices over $\mathcal{D}$ are $\gamma$-nice.

The following theorem is the basis for time–space tradeoffs for matrix–vector products. The input distribution is presumed to be uniform over all $n$ component vectors $\mathbf{x}$ over $\mathcal{D}$.

**Theorem 4.6.** Let $0 < c \leq 1/2$, let $A = [a_{i,j}]$ be an $m \times n$ $c$-ok matrix, and let $f: \mathbb{F}^n \rightarrow \mathbb{F}^m$ be the function $f(\mathbf{x}) = A \mathbf{x}$. Suppose $\alpha$ and $\beta$ are positive integers, with $\alpha \leq \lceil c m \rceil$ and $\beta \leq \lceil c m \rceil$. Then $C_f(\alpha, \beta) \leq \delta^{-c \beta}$.

**Proof.** Let $\tau$ be a depth $\alpha$ non-redundant tree program that partially computes $\mathbf{y} = A \mathbf{x}$, in the sense that it computes some of the components of $\mathbf{y}$, and let $\pi$ be an arbitrary computation in $\tau$. Suppose $\pi$ makes at least $\beta$ outputs, and select just $\beta$ of them. For the remainder of this proof, queries, responses, and outputs are those of computation $\pi$.

Suppose $\pi$ $\beta$-solves input $\mathbf{x}$. Each output "$y_k = v$" induces an equation
\(v = \sum_i a_{ki}x_i\), which \(x\) must satisfy. Together the \(\beta\) selected outputs induce a system of equations \(Bx = v\), where \(B\) consists of \(\beta\) distinct rows of \(A\) and \(v\) is a vector of the \(\beta\) selected output values.

Similarly, the \(\alpha\) queries and their associated responses induce a system of equations \(Qx = r\), where matrix \(Q\) consists of \(\alpha\) distinct rows of the \(n \times n\) identity matrix, and \(r\) is a vector of the \(\alpha\) responses. Together, the queries and outputs induce the system \(Cx = w\), where \(C = \binom{G}{\beta}\) is an \((\alpha + \beta) \times n\) matrix and \(w = \binom{r}{\beta}\) is an \((\alpha + \beta)\) component column-vector. We show that \(C\) has rank at least \(\alpha + c\beta\).

Say that a column of \(C\) is unqueried if its first \(\alpha\) components are zero. Let \(C'\) be the submatrix of \(C\) consisting of the last \(\beta\) rows and the \(n - \alpha\) unqueried columns. Then \(C'\) is a \(\beta \times (n - \alpha)\) submatrix of \(A\). Since \(\alpha \leq \lceil cn \rceil\) and \(\beta \leq \lceil cm \rceil\), and \(A\) is \(c\)-ok, \(C'\) has rank at least \(c\beta\). Any \(\lceil c\beta \rceil\) linearly independent columns of \(C'\), together with the \(\alpha\) queried columns, form \(\alpha + \lceil c\beta \rceil\) linearly independent columns of \(C\).

The solutions \(x\) in \(\mathbb{F}^n\) to \(Cx = w\) form an affine space of dimension at most \(n - \alpha - c\beta\). By Lemma 4.1, there are at most \(\delta^{n - \alpha - c\beta}\) solutions in \(\mathbb{D}^n\). Since there are \(\delta^n\) vectors in \(\mathbb{D}^n\), the probability that \(x\) solves a random input is at most \(\delta^{n - \alpha - c\beta}\). Since \(\tau\) has just \(\delta^n\) distinct computations, the probability that \(\tau\) \(\beta\)-solves a random input is at most \(\delta^{n - \alpha - c\beta}\).

**Theorem 4.7.** Let \(c\) be a constant, \(0 < c < 1/2\). Let \(A\) be an \(m \times n\) \(c\)-ok matrix, where \(m \leq n\) and \(cm \geq 1\). Let \(P\) be a branching program that computes \(Ax\) for \(x \in \mathbb{D}^n\). Suppose \(P\) has expected time \(T \geq n\) and expected space \(S\), where all vectors over \(\mathbb{D}\) are equally likely. Then \(ST = \Omega(nm \log \delta)\).

**Proof.** Choose \(\alpha = \lceil cn \rceil\) and \(\beta = \lceil m\alpha/4T \rceil\), and combine Lemma 3.3 and Theorem 4.6.

### 5. Convolution and Integer Multiplication

The result for matrix–vector products leads readily to other results. As an example, consider the discrete Fourier transform (DFT), which is naturally expressed as a matrix vector product. Specifically, let \(x\) be an \(n\) component vector, and suppose that field \(\mathbb{F}\) contains a primitive \(n\)th root of unity \(\omega\). Then the DFT of \(x\) is just the function \(f(x) = Ax\), where \(A\) is the \(n \times n\) matrix given by \(a_{ij} = \omega^{ij}\). (The rows and columns of \(A\) are numbered \(0, ..., n - 1\).) A consequence of a result observed by Yesha [24] is the following.

**Theorem 5.1 (Yesha).** The DFT matrix is \((1/4)\) ok.

An immediate consequence of Theorems 4.7 and 5.1 is a strengthening of Yesha's result for the DFT to expected space and time, and to arbitrary fields with appropriate roots of unity.

**Theorem 5.2.** Suppose that the field \(\mathbb{F}\) has a primitive \(n\)th root of unity. Let \(P\)
be a branching program that computes the DFT of an $n$ component vector over $\mathbb{D}$ in expected time $T$ and expected space $S$. Then $ST = \Omega(n^2 \log \delta)$. Another problem that is fairly naturally expressed as a matrix-vector product is convolution of vectors. Let $\mathbf{u} = (u_0, \ldots, u_{n-1})$ and $\mathbf{v} = (v_0, \ldots, v_{n-1})$ be vectors over $\mathbb{D}$. Then the convolution $\mathbf{w} = (w_0, \ldots, w_{n-1})$ of $\mathbf{u}$ and $\mathbf{v}$ is defined by $w_k = \sum_{i=0}^{n-1} u_i v_{k-i}$, where subscripts are reduced modulo $n$.

Savage [16] defines function $f$ to be a subfunction of $g$ if $f(x) = p(g(q(x)))$, where $p$ only projects and permutes its input in a fixed way, and $q$ only pads and permutes its input, possibly duplicating components, in a fixed way. If $f$ is a subfunction of $g$, then lower bounds for $f$ carry over to $g$ at least in the case of worst case complexity, and sometimes for expected case complexity. We will use that in the proof of the following.

**Theorem 5.3.** If $P$ is a branching program that computes the convolution of two $n$ component vectors over $\mathbb{D}$ in expected time $T \geq n$ and expected space $S$, where all vectors are equally likely, then $ST = \Omega(n^2 \log \delta)$. 

**Proof.** Convolution can be expressed as a matrix vector product. Given a vector $\mathbf{u} = (u_0, \ldots, u_{n-1})$, let $U$ be the $n \times n$ matrix $U_{i,j} = u_i v_j$, where subscripts are reduced modulo $n$. Then the convolution of $\mathbf{u}$ and $\mathbf{v}$ is just $U\mathbf{v}$. Assume that $n$ is even, and view $U$ as a $2 \times 2$ matrix

$$U = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

of $n/2 \times n/2$ blocks. In each of $A$ and $B$, each diagonal contains a distinct element of $\mathbf{u}$. By Lemma 4.5, there is a constant $\gamma > 0$ such that each of $A$ and $B$ is $\gamma$-nice with probability at least $1 - \delta^{-1}(2/3)^n$. Hence, for sufficiently large $n$, $A$ and $B$ are simultaneously $\gamma$-nice with probability at least $1/2$.

Since a constant fraction of the inputs $\mathbf{u}$ leads to both $A$ and $B$ being $\gamma$-nice, the input distribution can be restricted to such vectors $\mathbf{u}$, without increasing the computed expected time and space by more than a constant factor.

But when $A$ and $B$ are both $\gamma$-nice, so is the $n/2 \times n$ matrix $[A \ B]$. The product $[A \ B] \mathbf{x}$ is a subfunction of $U\mathbf{x}$, since, if $\mathbf{y} = U\mathbf{x}$, then $[A \ B] \mathbf{x}$ is just $(y_1, \ldots, y_{n/2})$. Hence Theorem 4.7 applies.

**Theorem 5.4.** Suppose $P$ is a branching program that multiplies two $n$ bit binary numbers in worst case time $T$ and space $S$. Then $ST = \Omega(n^2/\log^2 n)$.

**Proof.** It is well known that convolution of $n$ component vectors over $\mathbb{Z}_2$ is a subfunction of multiplication of two $2n\lceil \log n \rceil$ bit binary numbers, as can be seen as follows. To avoid carry effects, simply pad each digit with $\lceil \log n \rceil - 1$ zeros. To achieve the wrap-around of the definition of convolution, concatenate each vector with itself to form the binary numbers to be multiplied.
Suppose that $\mathcal{F}$ is a finite field, and assume that $\mathcal{F} = \mathcal{F}$, so that intermediate results can be stored in $O(\log \delta)$ space. Then the lower bound for convolution can be met, to within a constant factor, for the entire range of relevant time and space values within the branching program model. At one extreme is the "table lookup" algorithm, which is a tree of time $O(n)$ and space $O(n \log \delta)$. At the other extreme is the naive algorithm, which requires time $O(n^2)$ and space $O(\log n + \log \delta)$. Hybrid algorithms can be constructed that perform block matrix multiplication on the matrix–vector product representation of convolution, using the tree algorithm to multiply blocks, and the naive algorithm to combine the blocks. Such algorithms fill in the middle range of the tradeoff. In fact, for $\delta < n$, the correct extreme for low space is achieved by a slightly hybridized naive algorithm, which uses $k \times k$ blocks, where $k = \log n / \log \delta$, and which takes time $O(n^2 \log \delta / \log n)$ and space $O(\log n)$, and thus matches the lower bound to within a constant factor.

On a more realistic model of computation, one would use the fast Fourier transform algorithm [2], resulting in a polylogarithmic loss in space–time product complexity, and requiring that the appropriate roots of unity exist.

The binary integer multiplication lower bound can be met to within a factor of $O(\log^2 n)$ within the branching program model, for a wide range of time and space values, and to within a polylogarithmic factor in more realistic models by use of the Schönhage–Strassen integer multiplication algorithm [2]. See [12] for a low space algorithm for integer multiplication.

6. THE PRODUCT OF THREE MATRICES AND MATRIX INVERSION

The next result concerns the problem of computing the product $ABC$, where $A$, $B$, and $C$ are $n \times n$ matrices over $\mathcal{F}$. The goal is to express that problem as a matrix–vector product.

**DEFINITION 6.1.** The Kronecker product $A \otimes B$ of $n \times n$ matrices $A$ and $B$ is defined to be the $n^2 \times n^2$ matrix obtained by replacing each element $a_{i,j}$ of $A$ by the matrix $a_{i,j}B$.

The following lemma is proved in the appendix.

**LEMMA 6.2.** Let $0 < c < 1/2$. If $A$ and $B$ are both $c$-nice, then $A \otimes B$ is $c^2$-ok.

For an $n \times n$ matrix $B$, let $B$ be the $n^2$ component column vector obtained by concatenating the transposes of the rows of $B$, in their natural order.

**LEMMA 6.3.** Let $A$, $B$, $C$, and $D$ be $n \times n$ matrices over a commutative ring. The following two equations are equivalent.

(i) $D = ABC$.

(ii) $D = (A \otimes C^T)B$. 

Proof. Let \( D = ABC \) and \( E = (A \otimes C^T) B \). Let \( B_k \) be the \( k \)th row of \( B \), and \( C^j \) be the \( j \)th column of \( C \). Rows and columns are numbered starting at zero. Using lower case letters to denote components of corresponding upper case vectors or matrices and \( \mathcal{J} \) to denote inner product, we obtain:

\[
e_{ni+j} = \sum_{k=0}^{n-1} \mathcal{J}(a_{i,k} C^j, B_k) = \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} a_{i,k} c_{i,l} b_{k,l} = d_{i,j}.
\]

Theorem 6.4. Let \( P \) be a branching program that computes the product \( ABC \), where \( A, B, \) and \( C \) are \( n \times n \) matrices over \( \mathbb{D} \). If \( P \) uses expected time \( \bar{T} \) and expected space \( \bar{S} \), where all matrices over \( \mathbb{D} \) are equally likely, then \( \bar{S} \bar{T} = \Omega(n^4 \log \delta) \).

Proof. Since most matrices are \( \gamma \)-nice, we can restrict the input distribution to inputs where both \( A \) and \( C^T \) are \( \gamma \)-nice, without affecting the expected cost by more than a constant factor. But then Lemmas 6.2 and 6.3 imply that, for each possible choice of \( A \) and \( C \), computing \( ABC \) (where \( B \) is the input) is equivalent to an \( n^2 \times n^2 \) matrix-vector product, where the matrix is \( \gamma^2 \)-ok. Theorem 6.4 follows from Theorem 4.7.

The preceding result immediately applies to the problem of cubing a matrix.

Corollary 6.5. Let \( P \) be a branching program that computes \( A^3 \), where \( A \) is an \( n \times n \) matrix over \( \mathbb{D} \). If \( P \) uses worst case time \( T \) and space \( S \), then \( ST = \Omega(n^4 \log \delta) \).

Proof. This proof is immediate from the following equation, showing that the product of three \( n \times n \) matrices is a subfunction of cubing a \( 4n \times 4n \) matrix.

\[
\begin{pmatrix}
0 & A & 0 & 0 \\
0 & B & 0 & 0 \\
0 & C & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}^3 = \begin{pmatrix}
0 & ABC \\
0 & 0 \\
0 & 0 \\
0 & 0
\end{pmatrix} = \begin{pmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{pmatrix}. \tag{6.1}
\]

The next result concerns the problem of inverting a unit upper triangular matrix. A matrix \( A \) is unit upper triangular if \( a_{i,j} = 0 \) for \( i > j \) and \( a_{i,i} = 1 \) for each \( i \). The standard solution solves \( n \) systems of equations, employing back substitution for each, and uses \( O(n) \) words of space and \( O(n^3) \) arithmetic operations.

Theorem 6.6. Let \( P \) be a branching program that computes \( A^{-1} \), given input \( A \), a unit upper triangular \( n \times n \) matrix over \( \mathbb{D} \). If \( P \) has worst case time \( T \) and space \( S \), then \( ST = \Omega(n^4 \log \delta) \).
Proof. The fact that $ABC$ is a subfunction of $A^{-1}$ follows from the following well known equation relating $4n \times 4n$ matrices.

$$
\begin{pmatrix}
I & -A & 0 & 0 \\
0 & I & -B & 0 \\
0 & 0 & I & -C \\
0 & 0 & 0 & I
\end{pmatrix}
= \begin{pmatrix}
I & A & AB & ABC \\
0 & I & B & BC \\
0 & 0 & I & C \\
0 & 0 & 0 & I
\end{pmatrix}^{-1}
$$

Ja' Ja' [11] shows that the problem of solving an $n \times n$ system of linear equations over a field requires $ST = \Omega(n^3)$ in the straight-line model. His proof applies equally well here.

**Corollary 6.7.** Any branching program that solves an $n \times n$ system of linear equations over $\mathcal{D}$ in worst case time $T$ and space $S$ requires $ST = \Omega(n^3 \log \delta)$.

Proof. This proof is immediate from the fact that it is possible to invert an $n \times n$ matrix by solving $n$ systems of $n$ linear equations.

For the purpose of discussing upper bounds, suppose $\mathcal{D} = \mathcal{F}$. Theorem 6.4 is tight to within a constant factor, over the entire range of relevant time and space values, within the branching program model. The tree algorithm achieves $O(n^2)$ time and $O(n^2 \log \delta)$ space. The standard algorithm can be implemented in $O(n^3)$ time and $O(n \log \delta)$ space, since only one row of $AB$ needs to be stored at any given time, to compute its inner product with each of the columns of $C$. Ja' Ja' and Simon [12] describe an algorithm that computes $ABC$ in $O(\log n + \log \delta)$ space and $O(n^4)$ time. It computes the product as $(AB) C$, but each time an element of $AB$ is needed, it is recomputed. The tree and standard algorithms, as well as the standard and low space algorithms, are easily hybridized to fill in the tradeoff. As was the case for convolution, the correct low space extreme for $\delta < n$ is actually met by a slightly hybridized version of Ja' Ja' and Simon's algorithm, requiring $O(\log n)$ space and $O(n \log \delta / \log \log n)$ time.

The lower bound for inversion of a unit upper triangular matrix is tight for $n^2 \leq T \leq n^3$. Algorithms that match the bound are the tree algorithm, the back substitution algorithm, and hybrids of those. For $T > n^3$, Theorem 6.6 does not appear to be tight. $O(\log^2 n)$ space algorithms are known [12], but they require super-polynomial time.

7. **Matrix Multiplication**

This section deals with the problem of computing the product of two $n \times n$ matrices $A$ and $B$ over $\mathcal{D}$. The proof is similar to the preceding ones, but exhibits one new feature. Suppose that we are interested only in worst case bounds. Then the lower bound proof for convolution, for example, only requires that a $\gamma$-nice
matrix exist. So the probabilistic nature of Lemma 4.3 is unimportant. The following proof, on the other hand, exploits the probabilistic nature of Lemma 4.3 in an essential way, even when it is used only for worst case bounds. There is more on that after the proof.

Let \( \gamma \) be the constant of Lemma 4.3. For the purposes of the following theorem, the input distribution is uniform over \( \gamma \)-nice matrices \( A \) and \( B^T \).

**Theorem 7.1.** Let \( f: \mathcal{D}^{2n^2} \to \mathcal{F}^{n^2} \) be an \( n \times n \) matrix multiplication over \( \mathcal{D} \). Let \( \alpha \) and \( \beta \) be positive integers, and suppose \( \gamma n \geq 1 \) and \( (\alpha/\gamma n)^2 < \beta/2 \). Then \( C_\gamma(\alpha, \beta) \leq 8^2 - \gamma/4 \).

**Proof.** The proof is similar to that for Theorem 4.6, but is more involved. Suppose we have a depth \( \alpha \) non-redundant tree program \( \tau \) that partially computes \( C = AB \), and let \( \pi \) be an arbitrary computation in \( \tau \). For the remainder of this proof, queries, responses, and outputs are those on path \( \pi \).

Suppose \( \pi \) makes at least \( \beta \) outputs. Select just \( \beta \) of them. Say that row \( i \) of matrix \( C = AB \) is heavy if at least \( \gamma n \) queries concern row \( i \) of \( A \). Similarly, say that column \( j \) of \( C \) is heavy if at least \( \gamma n \) queries concern column \( j \) of \( B \). There are at most \( \alpha/\gamma n \) heavy rows or columns in \( C \). A row or column is light if it is not heavy.

It must be the case that either at least \( \beta/4 \) selected outputs fall in light rows of \( C \), or at least \( \beta/4 \) selected outputs fall in light columns of \( C \). For suppose the former is false. Then at least \( 3\beta/4 \) selected outputs fall in at most \( \alpha/\gamma n \) rows. Of those outputs, at most \( (\alpha/\gamma n)^2 \) can fall in heavy columns. Hence, a total of at most \( (\alpha/\gamma n)^2 + \beta/4 \leq 3\beta/4 \) selected outputs fall in heavy columns.

Without loss of generality, assume that at least \( \beta/4 \) selected outputs fall in light columns of \( C \). (Otherwise, consider the equivalent problem \( C^T = B^T A^T \), and rename matrices. In what follows, only the left hand matrix in the product has to be \( \gamma \)-nice.) Call those \( \beta/4 \) outputs light outputs. Express equation \( AB = C \) as

\[
\begin{pmatrix}
A \\
\vdots \\
A
\end{pmatrix}
\begin{pmatrix}
B^1 \\
\vdots \\
B^n
\end{pmatrix}
= 
\begin{pmatrix}
C^1 \\
\vdots \\
C^n
\end{pmatrix},
\]

where \( B^j \) is the \( j \)th column of \( B \), and similarly for \( C \). Let \( B \) and \( C \) be the vector representations of \( B \) and \( C \) appearing in Eq. (7.1). Suppose \( \pi \) makes \( \alpha_1 \) queries about \( A \) and \( \alpha_2 \) about \( B \), where \( \alpha = \alpha_1 + \alpha_2 \), and suppose \( \pi \) \( \beta \)-solves input \( (A, B) \). There are at most \( 6^{n^2 - \alpha_1} \) possible values for \( A \), since \( A \) must be consistent with the \( \alpha_1 \) distinct query responses on \( \pi \). In order for the light outputs to be correct, \( B \) must satisfy at least \( \beta/4 \) equations from system (7.1). In fact, for any particular matrix \( A \), matrix \( B \) must satisfy a system of equations (Eq. (7.1)) \( Q \) \( B = r \), where \( Q \) consists of \( \alpha_2 \) distinct rows of the \( n^2 \times n^2 \) identity matrix (corresponding to the queries about \( B \)), \( r \) is a vector of \( \alpha_2 \) responses, \( A' \) consists of \( \beta/4 \) rows of the block diagonal matrix of Eq. (7.1), and \( c \) consists of \( \beta/4 \) rows of \( C \) (corresponding to the light outputs).
Matrix \( H = (Q_A) \) must have rank at least \( \alpha_2 + \gamma \beta/4 \). To see that, say that a column of \( H \) is unqueried if its first \( \alpha_2 \) entries are zero. Because only light outputs were used, if any row of a given copy of \( A \) in the block diagonal matrix is included in \( H \), then at least \( (1 - \gamma) n \) of that copy's columns are unqueried in \( H \). Since \( A \) is \( \gamma \)-nice, if \( k \) rows of a given copy of \( A \) are included in \( H \), then that copy contributes \( \min(k, \lceil \gamma n \rceil) \) linearly independent unqueried columns to \( H \). There can be no dependencies between columns that intersect different copies of \( A \) in the block diagonal matrix. Thus, at least \( \gamma \beta/4 \) unqueried columns of \( H \) are linearly independent. The \( \alpha_2 \) queried columns bring the total number of linearly independent columns to at least \( \alpha_2 + \gamma \beta/4 \).

By Lemma 4.1, at most \( \delta^{n^2 - \alpha_2 - \gamma \beta/4} \) matrices \( B \) can be paired with a given \( \gamma \)-nice matrix \( A \), and still be consistent with path \( \pi \). Since only \( \delta^{n^2 - \alpha_1} \) values of \( A \) are consistent with \( \pi \), the total number of \( \gamma \)-nice matrix pairs that can be \( \beta \)-solved by \( \pi \) is at most \( \delta^{2n^2 - \alpha - \gamma \beta/4} \). Since there are at least \( (1 - \delta^{-1}) \delta^{n^2} \) \( \gamma \)-nice \( n \times n \) matrices, the probability that \( \pi \) \( \beta \)-solves the pair \((A, B)\) is at most \((\delta/(\delta - 1))^2 \delta^{-\alpha - \gamma \beta/4} \). There are just \( \delta^\alpha \) computations in \( \tau \), so the probability that any of them \( \beta \)-solves a random input is at most \((\delta/(\delta - 1))^2 \delta^{-\alpha - \gamma \beta/4} \leq \delta^{2 - \gamma \beta/4} \).

Now suppose that at most matrices over \( \mathcal{D} \) are equally likely as inputs.

**Theorem 1.2.** Let \( P \) be a branching program that multiplies two \( n \times n \) matrices over \( \mathcal{D} \). If \( P \) has expected time \( T \geq n^2 \) and expected space \( S \), where all matrices over \( \mathcal{D} \) are equally likely, then \( ST^2 = \Omega(n^6 \log \delta) \).

**Proof.** As in preceding proofs, restrict attention to inputs where \( A \) and \( B^T \) are \( \gamma \)-nice. Doing so will only only affect the complexity by a constant factor. Choose \( \alpha = \lceil \gamma^2 n^4/8 T \rceil \) and \( \beta = \lceil n^2 \alpha/4 T \rceil = \Omega(n^6/T^2) \). Then the conditions of Theorem 7.1 are met. Lemma 3.3 implies that \( S \geq (\gamma \beta/4 - 2) \log \delta - 4 = \Omega(n^6 \log \delta/T^2) \).

An alternative to the above proof is to fix one of \( A \) and \( B \), and let the other be the input. Then the proof becomes somewhat simpler. Moreover, the fixed matrix can be chosen to be \( \gamma \)-nice, so it suffices for a \( \gamma \)-nice matrix to exist. Unfortunately, the bound is weakened to \( ST = \Omega(n^3 \log \delta) \). No better bound can be proved when \( A \) is fixed, since then there is a branching program that computes \( AB \), for input \( B \), in \( O(n^2) \) time and \( O(n \log \delta) \) space. Having read and stored a column of \( B \), the program has enough information to output a column of \( AB \). Matrix \( A \) can be stored in the program at no cost in space as it is defined for branching programs.

It is important for both \( A \) and \( B \) to be input to the algorithm. But it is also important to the proof that both \( A \) and \( B^T \) be \( \gamma \)-nice most of the time. So the proof really depends on the probabilistic nature of Lemma 4.3.

The following is immediate from Eq. 6.1.

**Corollary 7.3.** Let \( P \) be a branching program that squares an \( n \times n \) matrix over \( \mathcal{D} \). If \( P \) has worst case time \( T \geq n^2 \) and space \( S \), then \( ST^2 = \Omega(n^6 \log \delta) \).

Assuming that \( \mathcal{D} = \mathcal{F} \), the lower bound of Corollary 7.2 can be achieved for a
broad range of time and space values, within the branching program model. At one
end of the tradeoff is a tree of space \(O(n^2 \log \delta)\) and time \(O(n^2)\). At the opposite
end is the standard algorithm, which requires space \(O(\log n + \log \delta)\) and time
\(O(n^3)\). A hybrid algorithm partitions the matrices into \(k \times k\) blocks, and executes
the standard algorithm on the blocked matrices, using the tree algorithm for multi-
plication of blocks. That algorithm uses time \(O(n^3/k)\) and space \(O(k^2 \log \delta + \log n)\).
In fact, for \(\delta < n\), the low space end of the tradeoff is achieved by a time
\(O(n^3 \sqrt{\log \delta/\log n})\), space \(O(\log n)\) hybrid algorithm.

8. THE PAQ FUNCTION

The problem is to compute the product \(PAQ\), where \(A\) is a fixed \(n \times n\) matrix of
distinct elements (which for definiteness can be assumed to be \(\{0, \ldots, n^2 - 1\}\)), and
\(P\) and \(Q\) are input \(n \times n\) permutation matrices. As input distribution, choose the
uniform one over pairs of permutation matrices.

One algorithm for computing \(C = PAQ\) finds, for each \(i, j = 1, \ldots, n\), the column \(k\)
of the sole 1 in row \(i\) of \(P\), and the row \(l\) of the sole 1 in column \(j\) of \(Q\), producing
output \(C_{i,j} = A_{k,l}\). That algorithm uses space \(O(\log n)\) and time \(O(n^3 \log n)\) on a
logarithmic cost random access machine. By storing the position of the sole 1 in
each row of \(P\) and column of \(Q\), at a factor of \(n\) increase in space, redundant
scanning of \(P\) and \(Q\) can be eliminated, and a factor of \(n\) in time is saved. Hybrid
algorithms fill in a tradeoff of \(ST = O(n^3 \log^2 n)\). The complexity drops to
\(ST = O(n^3 \log n)\) in the branching program model.

**Theorem 8.1.** Let \(n \geq 40\), and let \(f: \{0, 1\}^{2n^2} \rightarrow \{0, \ldots, n^2 - 1\}^{n^2}\) be the PAQ function. Let \(a\) and \(B\) be positive integers, and let \(w = \lceil 15a/n \rceil\). Suppose \(w^2 < B < n^2/4\). Then \(C_f(a, B) < 2^{2-w}\).

**Proof.** Let \(\tau\) be a depth \(x\) non-redundant binary tree program. The weight of a
computation in \(\tau\) is the number of responses in that computation of the form \(x_i = 1\).
A computation is light if its weight is less than \(w\). Otherwise it is heavy. The contributions of light computations and heavy computations are analyzed separately.

First consider heavy computations. Let \(p\) be the probability that a random pair
of permutation matrices (all pairs equally likely) follows a computation in \(\tau\) of
weight at least \(w\). We show that \(p \leq 3 \cdot 2^{-w}\).

Let \(r_k\) be the maximum, over all computations \(\pi\) of weight \(k\), of the probability
that a random pair of permutation matrices \(P\) and \(Q\) are consistent with the
responses on computation \(\pi\). Each time \(\pi\) finds a 1 in a permutation matrix, it
learns a little bit about the positions of the remaining 1's in that matrix. The maximum of \(r_k\) is realized for a computation \(\pi\) that looks for up to \(n\) 1's in \(P\), after
which it looks for 1's in \(Q\). So \(r_k \leq 1/n^k\) for \(k \leq n\), and \(r_k \leq 1/(n! \cdot n^{k-n})\) for \(k > n\),
where \(n^k = n!/(n-k)\) is the descending power of \(n\). Then

\[
p \leq \sum_{w \leq k \leq n} \binom{x}{k} \frac{1}{n^k} + \sum_{n < k \leq 2n} \binom{x}{k} \frac{1}{n! \cdot n^{k-n}}.
\]

(8.1)
The first sum of inequality (8.1) is easily bounded by using \( n^k \geq \exp(\int_{n-k}^{n} \ln(x) \, dx) \geq n^k e^{-k} \). The terms of \( \sum_k \binom{n}{k} e^k n^{-k} \) for \( k \geq w \) are geometrically decreasing at a ratio of at most \( \varepsilon e / wn < e / 15 \), so the first sum is less than \( 2(\varepsilon) e^n n^{-w} < 2(e^2 \alpha / wn)^w \leq 2 \cdot 2^{-w} \). The second sum of inequality (8.1) can be similarly bounded. The terms of \( \sum_k \binom{n}{k} e^k n^{-k} \) for \( k \geq n \) are geometrically decreasing at a ratio of less than \( \varepsilon e / n^2 < 1/2 \), so the second sum is less than \( (1/n!)^n < (e^2 \alpha / n^2)^n < 2^{-n} \). But \( w \leq n \), so \( p < 3 \cdot 2^{-w} \).

Now consider the contribution of light computations. There are \( \sum_{k=1}^{w} \binom{n}{k} < \binom{n}{w} \) computations of weight less than \( w \) in \( \tau \). Suppose a given one of them, \( \pi, \beta \)-solves input \((P, Q)\). Since the elements of \( A \) are distinct, each output can have come from only one place in \( A \), so each output forces the contents of a row of \( P \) and a column of \( Q \) to particular vectors. The number of rows and columns forced can be minimized, and hence the probability of \( \beta \)-solving a random input maximized, if the \( \beta \) outputs occur in a \( \sqrt{\beta} \times \sqrt{\beta} \) block. In that case \( \sqrt{\beta} \) rows of \( P \) and columns of \( Q \) are forced. Since \( \sqrt{\beta} \leq n/2 \), the dependencies between rows and columns are weak, and the probability that a random permutation matrix has given vectors in \( \sqrt{\beta} \) given rows or columns is at most \( (2/n)^{2\sqrt{\beta}} \). So the contribution of light computations to \( C_\beta(\pi, \beta) \) is at most \( (\binom{n}{w})(2/n)^{2\sqrt{\beta}} \leq (e^2 \alpha / w)^w (2/n)^{2w} \leq 2^{-w} \).

So the probability that any computation in \( \tau \) \( \beta \)-solves a random input is at most \( 3 \cdot 2^{-w} + 2^{-w} \), and Theorem 8.1 is established.  

THEOREM 8.2. Let \( P \) be a branching program that computes \( PA \), for \( n \times n \) matrices, in expected time \( T \geq n^2 \) and space \( S \), where all permutation matrices are equally likely. Then \( ST = \Omega(n^3) \).

Proof. Let \( c = 1/(5 \cdot 18^2) \). We can assume that \( T \leq 3cn^3 \) and \( n \geq 40 \). Choose \( a = \lceil vn^4 / T \rceil \) and \( \beta = \lceil n^2 \alpha / 4T \rceil \). Then the conditions of Theorem 8.1 are met. By Lemma 3.3, \( S \geq w - 6 \geq 15\alpha / n - 6 \), so \( ST = \Omega(n^3) \).

APPENDIX

LEMMA 4.1. Suppose \( \mathcal{S} \subseteq \mathbb{R}^n \) is contained in an affine subspace \( \mathcal{A} \) of \( \mathbb{R}^n \) of dimension at most \( r \). Then \(|\mathcal{S}| \leq \delta^{r} \).

Proof. The proof is elementary linear algebra. Let \( \mathcal{A} \) be the solution space of \( Ax = b \), where \( A \) is an \( n \times n \) matrix of rank \( n - r \). Partition \( A \) as

\[
A = \begin{pmatrix}
B & C \\
D & E
\end{pmatrix},
\]

where \( B \) is \((n-r) \times (n-r)\) and \( E \) is \( r \times r \). Presume that the system \( Ax = b \) has been permuted so that \( B \) is nonsingular. Let \( y \) consist of the first \( n - r \) components of the indeterminate vector \( x \), and let \( z \) be the last \( r \) components of \( x \). Let \( d \) consist of the
first \( n - r \) components of \( b \). Then \( Ax = b \) implies that \( By = d - Cz \). So the first \( n - r \) components of \( x \) are determined by the last \( r \) components. There are \( \delta' \) possible values for \( z \).

**Lemma 4.3.** There is a constant \( \gamma \), where \( 0 < \gamma < 1/2 \), such that at least a fraction \( 1 - \delta^{-1}(2/3)^n \) of the \( n \times n \) matrices over \( \mathcal{D} \) are \( \gamma \)-nice. The constant \( \gamma \) is independent of \( n \) and \( \mathcal{D} \).

**Proof:** Given two indices \( i \) and \( j \) and a matrix \( A \), let \( A_{i,j} \) be the \((i,j)\)th entry of \( A \). If \( I \) and \( J \) are index sets, let \( A_{I,J} \) denote the submatrix of \( A \) indexed by rows \( I \) and columns \( J \).

The constant \( \gamma \) will be selected below. For now, suppose we have \( \gamma \). Let \( p = \lceil \gamma n \rceil \) and \( q = \lfloor (1 - \gamma) n \rfloor \). Presume that \( \gamma n > 1 \), since otherwise the lemma is trivial. We must look at all \( p \times q \) submatrices, so start by considering an arbitrary one. Let \( I = \{i_1, \ldots, i_p\} \) and \( J = \{j_1, \ldots, j_q\} \) be its index sets, with \( i_1 < i_2 < \cdots < i_p \) and \( j_1 < j_2 < \cdots < j_q \).

Imagine generating a random \( n \times n \) matrix \( A \) over \( \mathcal{D} \). Start at the lower left hand corner, an generate elements independently by diagonals, moving toward the upper right hand corner. During the process, maintain indices \( i = i_e I \) and \( j = j_e J \), where element \( A_{i,j} \) has not yet been chosen. Also maintain index sets \( R = \{i_1, \ldots, i_p\} \) and \( C \subseteq \{j_1, \ldots, j_q\} \), such that \( A_{R,C} \) is a non-singular \((p-t) \times (p-t)\) submatrix of \( A \). Initially, it suffices to choose \( i = i_p \) and \( j = j_1 \), assuming that a \( 0 \times 0 \) matrix is defined to have determinant 1. Each time element \( A_{i,j} \) is generated, \( s \) is incremented, and sometimes \( t \) is decremented.

Now imagine we have generated part of \( A \), we have indices \( i = i_e I \) and \( j = j_e J \), and we are about to generate element \( A_{i,j} \). See Fig. 1. Let \( R' = R \cup \{i\} \) and \( C' = C \cup \{j\} \). Note that all of the elements of \( A_{R,C}' \), excluding \( A_{i,j} \), are on lower diagonals than \( A_{i,j} \), and so have already been generated. Let \( x = A_{i,j} \) and \( B = A_{R,C}' \). Suppose \( \det B = 0 \). Expanding \( \det B \) by its first row, and setting the result to zero, gives a linear equation in \( x \), with the coefficient of \( x \) being \( \pm \det A_{R} \neq 0 \). Hence, at

\[
\begin{bmatrix}
    C' \\
    \hline
    R \left\{ \begin{bmatrix}
    \vdots \\
    \vdots \\
    A_{i,j} \\
    \vdots \\
    \vdots \\
    \end{bmatrix}
    \right\} R'
    \end{bmatrix}
\]

**Fig. 1.** A snapshot during the proof of Lemma 4.3.
most one choice of \(x\) can cause \(\det B = 0\), and with probability at least \(1 - \delta^{-1}\), \(\det B \neq 0\).

After generating \(A_{i,j}\), increment \(s\), and if \(\det B \neq 0\) then decrement \(t\) and set \(C \leftarrow C'\), recording that a larger non-singular submatrix has been found.

For each column index in \(J\) there will be an opportunity to decrement \(y\), at least until \(t = 0\). Each opportunity yields a success with probability at least \(1 - \delta^{-1}\), independently of previous results. So the probability of failing to find a non-singular \(p \times p\) submatrix of \(A'_i\) is at most the probability of getting fewer than \(p\) successes in \(q\) independent Bernoulli trials, where each trial has success probability \(1 - \delta^{-1}\). The following lemma bounds that probability.

**Sublemma A.1.** Let \(P_{n,k}\) be the probability of fewer than \(k\) successes in \(n\) independent Bernoulli trials, where each trial has success probability \(\geq 1 - \delta^{-1}\), where \(\delta \geq 2\). Then \(P_{n,k} \leq g^{-n}\delta^{k-1}\), where \(g = \delta^2/(2\delta - 1) \leq 2\delta/3\).

**Proof.** By induction on \(n\).

\[
\begin{align*}
n = 0: & \quad P_{0,0} = 0 \leq \delta^{-1} \quad \text{and} \quad P_{0,k} = 1 \leq \delta^{k-1} \quad \text{for} \quad k \geq 1. \\
n > 0: & \\
& \quad P_{n,k} = \delta^{-1}P_{n-1,k} + (1 - \delta^{-1})P_{n-1,k-1} \\
& \quad \leq \delta^{-1}g^{1-n}\delta^{k-1} + (1 - \delta^{-1})g^{1-n}\delta^{k-2} \\
& \quad = g^{-n}\delta^{k-1}g\delta^{-1}(2 - \delta^{-1}) \\
& \quad = g^{-n}\delta^{k-1}
\end{align*}
\]

So, for the selected index sets \(I\) and \(J\), the probability that \(A'_i\) fails to have maximal rank, for a random matrix \(A\), is at most \(g^{-q}\delta^{p-1}\). But \(q = n - p\), so there are \(\binom{n}{p}^2\) pairs of index sets \(I\) and \(J\), and the probability that any \(p \times q\) submatrix of \(A\) fails to have maximal rank is at most \(Q = \binom{n}{p}^2g^{-q}\delta^{p-1}\). A crude Stirling approximation suffices to bound \(Q\). Applying \((\frac{n}{p})^{2p}n!e^{-p}p^{-p}\) and \(g \leq 2\delta/3\) gives \(Q < (ne/p)^{2p}g^{p-n}\delta^{p-1} \leq (ne/p)^{2p}(2/3)^p g^{-n}\delta^{-1}\). To establish Lemma 4.3, we need to show that \(Q \leq \delta^{-1}(2/3)^p\). It suffices to show that \((ne/p)^{2p/n} \leq g\). But \(n/p \geq 1/(2\gamma)\), and \(\lim_{x \to \infty} x^{2/x} = 1\), so a sufficiently small choice of \(\gamma\) brings \((n/p)^{2p/n}\) below \(1.1\). Also, \(2p/n \leq 4\gamma\), so a sufficiently small \(\gamma\) guarantees that \((e\delta)^{2p/n} \leq \max(1.1, \delta/3)\). Then for any \(\delta \geq 2\), \((ne/p)^{2p/n} \leq \delta^{-2}/(2\delta - 1) = g\).

**Lemma 4.5.** There is a constant \(\gamma, 0 < \gamma < 1/2\), such that at least a fraction \(1 - \delta^{-1}(2/3)^n\) of the \(n \times n\) Toeplitz matrices over \(\mathcal{D}\) are \(\gamma\)-nice.

**Proof.** Inspection of the proof of Lemma 4.3 shows that it applies equally well to Toeplitz matrices, since it makes no assumption that elements on a common diagonal are generated independently.

**Lemma 6.2.** Let \(0 < c < 1/2\). If \(A\) and \(B\) are both \(c\)-nice, then \(A \otimes B\) is \(c^2\)-ok.
Proof. Let \( E = A \otimes B \). Rows and columns of \( A, B, \) and \( E \) are numbered starting from zero. Let \( E_i \) be the \( i \)th row of \( E \) and, for \( J \) a set of columns, let \( E_i^J \) be the projection of \( E_i \) onto the columns in \( J \).

Select an arbitrary \( p \times q \) submatrix \( S \) of \( E \), given by index sets \( I \) and \( J \), where \( |I| = p \leq [c^2n^2] \) and \( |J| = q \geq [(1 - c^2)n^2] \). It must be shown that \( S \) has rank at least \( c'^p \). Presume that \( cn \geq 1 \), since otherwise the lemma is trivial.

A block row of \( E \) is that part of \( E \) corresponding to a row of \( A \). The \( i \)th block row of \( E \) consists of rows \( ni, ..., ni + n - 1 \). Let \( \Delta_i = I \cap \{ ni, ..., ni + n - 1 \} \) be the rows of \( S \) that fall in the \( i \)th block row. Choose a set \( \Gamma \subset \{ 0, ..., n - 1 \} \) of size \( \lceil cn \rceil \) so as to maximize \( \sum_{i \in \Gamma} |\Delta_i| \). Then \( \sum_{i \in \Gamma} |\Delta_i| \geq cp \), since any imbalance in the distribution of the rows of \( S \) among the block rows of \( E \) can only increase the number of rows of \( S \) occurring in the most populous \( \lceil cn \rceil \) block rows. For each \( i \in \Gamma \), let \( A_i = A_i \) if \( |\Delta_i| \leq \lceil cn \rceil \), and let \( A_i \) consist of the smallest \( \lceil cn \rceil \) members of \( A_i \) otherwise. Call the rows \( \bigcup_{i \in \Gamma} A_i \) of \( S \) blue rows. There must be a total of at least \( c^2 p \) blue rows, since at least a fraction of \( 1/c \) of the rows in \( \bigcup_{i \in \Gamma} A_i \) are blue. It suffices to show that the blue rows of \( S \) are linearly independent.

Suppose, to the contrary, that the blue rows are linearly dependent. Let \( (c_{i,j} : i \in \Gamma, j \in A_i) \) be constants, not all zero, such that

\[
\sum_{i \in \Gamma} \sum_{j \in A_i} c_{i,j} E^J_{ni+1+j} = 0. \tag{A.1}
\]

Choose \( r \) and \( s \) so that \( c_{r,s} \neq 0 \).

Say that column \( j \) of \( B \) is good if that column is associated with at least \( \lceil (1 - c) n \rceil \) columns of \( S \); that is if \( |\{ i : ni + j \in J \}| \geq \lceil (1 - c) n \rceil \). There are at least \( (1 - c) n \) good columns. (Otherwise more than \( cn \) columns of \( B \) are associated with at most \( (1 - c) n - 1 \) columns of \( S \), and \( S \) has fewer than \( (cn)((1 - c)n - 1) + (1 - c)n^2 < \lceil (1 - c^2)n^2 \rceil \) columns.) Let \( g_i \) be the projection of the \( i \)th row of \( B \) onto the good columns. Since \( B \) is \( c \)-nice, any set of up to \( \lceil cn \rceil \) of the vectors \( g_i \) are linearly independent. In particular, it must be the case that \( \sum_{i \in A_i} c_{r,i} g_i \neq 0 \). So it is possible to choose a good column \( t \) such that

\[
\sum_{i \in A_t} c_{r,i} b_{i,t} \neq 0. \tag{A.2}
\]

Let \( \Phi = \{ i : ni + t \in J \} \) be the columns of \( S \) that are associated with column \( t \) of \( B \). Since column \( t \) is good, \( |\Phi| \geq \lceil (1 - c) n \rceil \).

Let \( u_i \) be the projection of \( E_i \) onto columns \( \{ ni + t : i \in \Phi \} \). Let \( v_i \) be the projection of the \( i \)th row of \( A \) onto the columns of \( \Phi \). Then, from the definition of \( E = A \otimes B, \ u_{ni+t} = b_{j,t}v_i \). Taking a projection of Eq. (A.1),

\[
\theta = \sum_{i \in \Gamma} \sum_{j \in A_i} c_{i,j} u_{ni+j} = \sum_{i \in \Gamma} \left( \sum_{j \in A_i} c_{i,j} b_{j,t} \right) v_i. \tag{A.3}
\]
But $|\Gamma| = \lceil cn \rceil$ and $|\Phi| \geq \lfloor (1 - c) n \rfloor$. So the vectors $v_i$ for $i \in \Gamma$ are the rows of a $p' \times q'$ submatrix of $A$, where $p' = \lceil cn \rceil$ and $q' \geq \lfloor (1 - c) n \rfloor$. Since $A$ is $c$-nice, vectors $v_i$ for $i \in \Gamma$ are linearly independent. Hence, the only way to satisfy Eq. (A.3) is to have $\sum_{r \in \Gamma} c_{i,r} b_{i,r} = 0$ for every $i \in \Gamma$. But $r \in \Gamma$, so Eq. (A.2) must be violated. The supposition that the blue vectors are linearly dependent has lead to a contradiction.

ACKNOWLEDGMENTS

Thanks to David Kirkpatrick for introducing me to Borodin and Cook's technique, to Nicholas Pippenger for pointing out Yeshia's work, and to an anonymous referee for valuable comments. This research was supported in part by the National Sciences and Engineering Research Council of Canada.

REFERENCES