Time-Space Tradeoffs For Branching Programs Contrasted With Those For Straight-Line Programs*

by

Karl Abrahamson

Technical Report 87-19

June 1987

*A preliminary version of this paper appeared in the 27th Annual Symposium on Foundations of Computer Science.

Abstract

This paper establishes time-space tradeoffs for some algebraic problems in the branching program model, including convolution of vectors, matrix multiplication, matrix inversion, computing the product of three matrices and computing PAQ where P and Q are permutation matrices. While some of the results agree well with known results for straight-line programs, one of them (for matrix multiplication) surprisingly is stronger, and one (for computing PAQ) is necessarily weaker. Some of the tradeoffs are proved for expected time and space, where all inputs are equally likely.

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 [13], convolution and discrete Fourier transform [13], binary integer multiplication [12], and matrix multiplication and inversion [7,9,11].

An alternative approach, based on branching programs, has been developed by Borodin et al. [6] and Borodin and Cook [3], and is applied to sorting in both cases. Yesha [16] uses branching programs to establish time-space tradeoffs for matrix multiplication and discrete Fourier transform. Other papers using branching programs as the model of computation include [1,4,5,15].

Unlike straight-line programs, which model oblivious algorithms, branching programs model algorithms which make decisions on the fly. Lower bounds for branching programs 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 Z_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. Our results for convolution and matrix inversion are 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.

3. For the problem of computing PAQ, 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 which 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, $\mathcal{D} = \{d_1, \ldots, d_\delta\}$ 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 $\log \delta$ in the results for branching programs which 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 $\overline{S}$ and $\overline{T}$ represent expected space and time, respectively, where all inputs are equally likely.

The first 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 [12,13]. Our results are that $\overline{S}\,\overline{T} = \Omega(n^2 \log \delta)$ for convolution of n component vectors over $\mathcal{D}$, and $ST = \Omega(n^2/\log^2 n)$ for multiplication of two n bit binary integers, in the branching program model.

The constant implicit in the Ω, as in all to follow, is independent of n, $\mathcal{F}$ and $\mathcal{D}$. Yesha [16] 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.

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 $\overline{S}\,\overline{T} = \Omega(n^4 \log \delta)$. That bound is tight to within a constant factor. For inversion, a worst case bound of $ST = \Omega(n^4 \log \delta)$ is shown. That agrees well with Ja' Ja's result [9] 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 [7] and Ja' Ja' [9] have shown that $ST = \Omega(n^3)$. Ja' Ja's result holds for any ring. In the branching program model, Yesha [16] has shown that, for every n, there is a finite field $\mathcal{E}$, of size $\Theta(n)$, such that $n \times n$ matrix multiplication over $\mathcal{E}$ 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 $\overline{S}\,\overline{T}^2 = \Omega(n^6 \log \delta)$ in the branching program model. The bound can be achieved to within a constant factor within the model, assuming $\mathcal{D} = \mathcal{F}$, for any $\log(\delta n) \leq S \leq n^2 \log \delta$.

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 [11] and Vuillemin [14] that any straight-line program to compute PAQ requires $ST = \Omega(n^4)$. But there is a random access program which computes PAQ in $ST = O(n^3 \log^2 n)$, for a wide range of S and T. A lower bound of $\overline{S}\,\overline{T} = \Omega(n^3)$ 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 [1,3,6,15,16], so the description here will be short.

Let $f: \mathcal{D}^n \rightarrow \mathcal{F}^m$, where $\mathcal{D} = \{d_1, \ldots, d_\delta\}$. A branching program P which computes $\vec{y} = f(\vec{x})$ 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 δ, and is labeled by a *query* of the form "x_i?", for some $1 \leq i \leq n$. Each arc is labeled by a *response* to a query. If nonsink u is labeled by query "x_i?", then the δ arcs exiting u are labeled with the δ possible responses "$x_i = d_1$", ..., "$x_i = d_\delta$".

Each node is also labeled by a possibly empty set of *outputs*, each of the form "$y_j = a$", where $a \in \mathcal{F}$ and $1 \leq j \leq m$.

Each input vector $\vec{x}$ defines a *computation* in P, namely the unique directed path from the source to a sink whose responses are consistent with $\vec{x}$. The outputs are just those which 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 $\overline{T}$ 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.) The expected space complexity $\overline{S}$ is defined as follows. A *numbering* of P is a 1-1 function assigning one of the integers $0, \ldots, k-1$ to each of the nodes of P. 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 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 temporary 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 [8,10]. To within a constant factor, lower bounds for branching programs apply even to Turing machines and logarithmic cost random access machines where only temporary space is counted. Discussions of the relationship between branching programs and other models for worst case complexity can be found in [1,3,16]. 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. As pointed out in [3,6,16], we can assume that branching programs are in normal form. The reason is simply that any branching program can be put into normal form by replicating all of the nodes at each level, in the obvious way, at a cost of adding $\log T$ to the space complexity of a time T program. Since it is clear that $S \geq \log T$, and we are only interested in complexity results to within a constant factor, there is no real loss in converting to normal form.

The Borodin-Cook technique is essentially 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 [3,6,16], and is similar to that for expected case, only the version for expected case is proved here.

Definition 3.1: A computation π *k-solves* vector $\vec{x}$ (w.r.t. function f) iff π produces at least k outputs, all of the query responses on π are consistent with $\vec{x}$, and all of the outputs are correct (w.r.t. f) for input $\vec{x}$. A tree program τ k-solves $\vec{x}$ iff the computation of τ on $\vec{x}$ k-solves $\vec{x}$.

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 α and β be positive integers. For any non-redundant tree program τ, define $C_{f,I}(\tau, \beta)$ as the probability that τ β-solves a random input $\vec{x}$ (w.r.t. f). $C_{f,I}(\tau, \beta)$ provides a measure of the portion of the entire solution of f covered by τ. Define $C_{f,I}(\alpha, \beta)$ as the maximum, over all depth α non-redundant tree programs τ, 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 α and β 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 $\beta \leq \left\lceil \frac{m\alpha}{2T} \right\rceil$, then $S \geq -\log C_f(\alpha, \beta)$.

ii) If P has expected time $\overline{T}$ and expected space $\overline{S}$, with $\alpha \leq n \leq \overline{T}$ and $\beta \leq \left\lceil \frac{m\alpha}{4\overline{T}} \right\rceil$, then $\overline{S} \geq -\log C_f(\alpha, \beta) - 4$.

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 $t = 2\overline{T}$, and call the truncated program P'. With probability at least $\frac{1}{2}$, P' completes the solution of a random input. Break P' into $s = \lceil (t+1)/(\alpha+1) \rceil \leq 2t/\alpha$ 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 α, 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 can't 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 α by adding superfluous but non-redundant queries as needed. Call the resulting program P''.

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

$$\Pr(\text{some tree in some stage } \beta\text{-solves } \vec{x}) \geq \frac{1}{2}. \tag{3.1}$$

The root of each tree in each stage of P'' corresponds to a node in P. Let τ_i be the tree in P'' whose root is numbered i, for each i, if such a tree exists. Let p_i be the probability that, for a random input $\vec{x}$, (a) the computation of P'' on $\vec{x}$ passes through the root of τ_i, and (b) τ_i β-solves $\vec{x}$. Then $p_i \leq C_f(\alpha, \beta)$. Let K be a random variable denoting the largest node number on the computation in P of a random input $\vec{x}$. Then $E(K) = 2^{\overline{S}}$, and by Markov's inequality $\Pr(K \geq 2^{\overline{S}+2}) \leq \frac{1}{4}$. So

$$\begin{aligned} \Pr(\text{some tree in some stage } \beta\text{-solves } \vec{x}) &\leq \sum_i p_i \\ &\leq 2^{\overline{S}+2} C_f(\alpha, \beta) + \frac{1}{4} \end{aligned} \tag{3.2}$$

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

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 [13], 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.

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

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

Definition 4.2: Let $0 < c < \frac{1}{2}$. 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 $p \leq \lceil cm \rceil$ and $q \geq \lfloor (1-c)n \rfloor$. Say that A is *c-ok* iff every such submatrix of A has rank at least cp.

Lemma 4.3: There is a constant γ, where $0 < \gamma < 1/2$, such that at least a fraction $1 - \delta^{-1}\left(\frac{2}{3}\right)^{\gamma n}$ of the $n \times n$ matrices over $\mathcal{D}$ are γ-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 γ, $0 < \gamma < 1/2$, such that at least a fraction $1 - \delta^{-1}\left(\frac{2}{3}\right)^{\gamma n}$ of the $n \times n$ Toeplitz matrices over $\mathcal{D}$ are γ-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 $\vec{x}$ 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 $f: \mathcal{D}^n \rightarrow \mathcal{F}^m$ be the function $f(\vec{x}) = A\vec{x}$. Suppose α and β are positive integers, with $\alpha \leq \lceil cn \rceil$ and $\beta \leq \lceil cm \rceil$. Then $C_f(\alpha, \beta) \leq \delta^{-c\beta}$.

Proof: Let τ be a depth α non-redundant tree program which partially computes $\vec{y} = A\vec{x}$, in the sense that it computes some of the components of

$\vec{y}$, and let π be an arbitrary computation in τ. Suppose π makes at least β outputs, and select just β of them. For the remainder of this proof, queries, responses and outputs are those of computation π.

Suppose π β-solves input $\vec{x}$. Each output "$y_k = v$" induces an equation $v = \sum_i a_{k,i}x_i$, which $\vec{x}$ must satisfy. Together the β selected outputs induce a system of equations $B\vec{x} = \vec{v}$, where B consists of β distinct rows of A and $\vec{v}$ is a vector of the β selected output values.

Similarly, the α queries and their associated responses induce a system of equations $Q\vec{x} = \vec{r}$, where matrix Q consists of α distinct rows of the $n \times n$ identity matrix, and $\vec{r}$ is a vector of the α responses. Together, the queries and outputs induce the system $C\vec{x} = \vec{w}$, where $C = \binom{Q}{B}$ and $\vec{w} = \binom{\vec{r}}{\vec{v}}$. We show that C has rank at least $\alpha + c\beta$.

Say that a column of C is *unqueried* if its first α components are zero. Let C' be the submatrix of C consisting of the last β 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 α queried columns, form $\alpha + \lceil c\beta \rceil$ linearly independent columns of C.

The solutions $\vec{x}$ in $\mathcal{F}^n$ to $C\vec{x} = \vec{w}$ 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 $\mathcal{D}^n$. The probability that π β-solves a random input is thus at most $\delta^{-\alpha-c\beta}$. Since τ has just δ^α distinct computations, the probability that τ β-solves a random input is at most $\delta^{-c\beta}$. ∎

Corollary 4.7: Let A be an $m \times n$ c-nice matrix, where $m \leq n$ and $0 < c \leq \frac{1}{2}$ and $cm \geq 1$. Let P be a branching program which computes $A\vec{x}$ for $\vec{x} \in \mathcal{D}^n$. Suppose P has expected time $\overline{T} \geq n$, and expected space $\overline{S}$, where all vectors over $\mathcal{D}$ are equally likely. Then $\overline{S}\,\overline{T} = \Omega(nm \log \delta)$.

Proof: Choose $\alpha = \lceil cn \rceil$ and $\beta = \lceil m\alpha/4\overline{T} \rceil$, 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 $\vec{x}$ be an n component vector, and suppose that field $\mathcal{F}$ contains a primitive n^{th} root of unity ω. Then the DFT of $\vec{x}$ is just the function $f(\vec{x}) = A\vec{x}$, where A is the $n \times n$

matrix given by $a_{i,j} = \omega^{ij}$. (The rows and columns of A are numbered 0, ..., $n-1$. The same convention will be used for other matrices as well.) A consequence of a result observed by Yesha [16] is the following.

Theorem 5.1 (Yesha): The DFT matrix is $\frac{1}{4}$-ok.

An immediate consequence of Corollary 4.7 and Theorem 5.1 is a strengthening of Yesha's result for the DFT.

Corollary 5.2: Suppose that the field $\mathcal{F}$ has a primitive n^{th} root of unity. Let P be a branching program which computes the DFT of an n component vector over $\mathcal{D}$ in expected time $\overline{T}$ and expected space $\overline{S}$. Then $\overline{S}\,\overline{T} = \Omega(n^2 \log \delta)$.

Another problem which is fairly naturally expressed as a matrix-vector product is convolution of vectors. Let $\vec{u} = (u_0, \ldots, u_{n-1})$ and $\vec{v} = (v_0, \ldots, v_{n-1})$ be vectors over $\mathcal{D}$. Then the convolution $\vec{w} = (w_0, \ldots, w_{n-1})$ of $\vec{u}$ and $\vec{v}$ is defined by $w_k = \sum_{i=0}^{n-1} u_i v_{k-i}$, where subscripts are reduced modulo n.

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

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

$$U = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

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

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

But when A and B are both γ-nice, so is $[A\ B]$. The product $[A\ B]\vec{x}$ is a subfunction of $U\vec{x}$ in the sense that if $\vec{y} = U\vec{x}$, then $[A\ B]\vec{x}$ is just $(y_1, \ldots, y_{n/2})$. Hence Corollary 4.7 applies. ∎

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

Proof: Savage [11] defines function f to be a subfunction of g if $f(\vec{x}) = p(g(q(\vec{x}))$, 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. It is well known that convolution of n component vectors over Z_2 is a subfunction of multiplication of two $2n\lceil \log n \rceil$ bit binary numbers. 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 $\mathcal{D} = \mathcal{F}$, 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 which 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 [10] 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 $\mathcal{D}$. 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 $0 < c < \frac{1}{2}$. If A and B are both c-nice, then $A \otimes B$ is c^2-ok.

For an $n \times n$ matrix B, let $\vec{B}$ 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) $\vec{D} = (A \otimes C^T)\vec{B}$.

Proof: Let $D = ABC$ and $\vec{E} = (A \otimes C^T)\vec{B}$. 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 I to denote inner product,

$$\begin{aligned} e_{ni+j} &= \sum_{k=1}^{n} I(a_{i,k}C^j, B_k) \\ &= \sum_{k=1}^{n}\sum_{l=1}^{n} a_{i,k}c_{l,j}b_{k,l} \\ &= d_{i,j} \end{aligned}$$

■

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

Proof: Since most matrices are γ-nice, we can restrict the input distribution to inputs where both A and C are γ-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 γ^2-ok. Theorem 6.4 follows from Corollary 4.7. ∎

The next result concerns the problem of inverting a unit upper triangular matrix. 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 6.5: Let P be a branching program which computes A^{-1}, given input A, a unit upper triangular $n \times n$ matrix over $\mathcal{D}$. 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.

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

∎

Ja' Ja' [9] 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 6.6: Any branching program which 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: 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 [10] describe an algorithm which 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 $O(n \log \delta / \log n)$ time.

The lower bound for inversion of a unit upper triangular matrix is tight for $n^2 \leq T \leq n^3$. Algorithms which match the bound are the tree algorithm and the back substitution algorithm, and hybrids of those. For $T > n^3$, Theorem 6.5 does not appear to be tight. $O(\log^2 n)$ space algorithms are known [10], 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 γ-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 γ be the constant of lemma 4.3. For the purposes of the following theorem, the input distribution is uniform over γ-nice matrices for A and B^T.

Theorem 7.1: Let $f: \mathcal{D}^{2n^2} \rightarrow \mathcal{F}^{n^2}$ be $n \times n$ matrix multiplication over $\mathcal{D}$. Let α and β be positive integers, and suppose $\gamma n \geq 1$ and $\left(\frac{\alpha}{\gamma n}\right)^2 \leq \beta/2$. Then $C_f(\alpha, \beta) \leq \delta^{2-\gamma\beta/4}$.

Proof: The proof is similar to that for theorem 4.6, but is more involved. Suppose we have a depth α non-redundant tree program τ which partially computes $C = AB$, and let π be an arbitrary computation in τ. For the remainder of this proof, queries, responses and outputs are those on path π.

Suppose π makes at least β outputs. Select just β of them. Say that row i of matrix $C = AB$ is *heavy* if at least γn queries concern row i of A. Similarly, say that column j of C is heavy if at least γn 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 $\left(\frac{\alpha}{\gamma n}\right)^2$ can fall in heavy columns. Hence, a total of at most $\left(\frac{\alpha}{\gamma n}\right)^2 + \beta/4 \leq 3\beta/4$ 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 γ-nice.) Call those $\beta/4$ outputs light outputs. Express equation $AB = C$ as

$$\begin{pmatrix} A & & \\ & \ddots & \\ & & A \end{pmatrix} \begin{pmatrix} B^1 \\ \vdots \\ B^n \end{pmatrix} = \begin{pmatrix} C^1 \\ \vdots \\ C^n \end{pmatrix} \tag{7.1}$$

where B^j (C^j) is the j^{th} column of B (C). Let $\vec{B}$ and $\vec{C}$ be the vector representations of B and C appearing in equation (7.1). Suppose π makes α_1 queries about A and α_2 about B, where $\alpha = \alpha_1 + \alpha_2$, and suppose π β-solves input (A, B). There are at most $\delta^{n^2-\alpha_1}$ possible values for A, since A must be consistent with the α_1 distinct query responses on π. In order for the light outputs to be correct, B must satisfy $\geq \beta/4$ equations from system (7.1). In fact, for any particular matrix A, matrix B must satisfy a system of equations $\binom{Q}{A'}\vec{B} = \binom{\vec{r}}{\vec{c}}$, where Q consists of α_2 distinct rows of the $n^2 \times n^2$ identity matrix (corresponding to the queries about B), $\vec{r}$ is a vector of α_2 responses, A' consists of $\beta/4$ rows of the block diagonal matrix of equation (7.1), and $\vec{c}$ consists of $\beta/4$ rows of $\vec{C}$.

Matrix $H = \binom{Q}{A'}$ must have rank at least $\alpha_2 + \gamma\beta/4$. To see that, say that a column of H is unqueried if its first α_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 γ-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 which intersect different copies of A in the block diagonal matrix. Thus, at

least $\gamma\beta/4$ unqueried columns of H are linearly independent. The α_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 γ-nice matrix A, and still be consistent with path π. Since only $\delta^{n^2-\alpha_1}$ values of A are consistent with π, the total number of γ-nice matrix pairs which can be β-solved by π is at most $\delta^{2n^2-\alpha-\gamma\beta/4}$. Since there are at least $(1-\delta^{-1})\delta^{n^2}$ γ-nice $n \times n$ matrices, the probability that π β-solves the pair (A,B) is at most $\left(\frac{\delta}{\delta-1}\right)^2 \delta^{-\alpha-\gamma\beta/4}$. There are just δ^α computations in τ, so the probability that any of them β-solves a random input is at most $\left(\frac{\delta}{\delta-1}\right)^2 \delta^{-\gamma\beta/4} \leq \delta^{2-\gamma\beta/4}$. ∎

Now suppose that all matrices over $\mathcal{D}$ are equally likely as inputs.

Theorem 7.2: Let P is a branching program which multiplies two $n \times n$ matrices over $\mathcal{D}$. If P has expected time $\overline{T} \geq n^2$ and expected space $\overline{S}$, where all matrices over $\mathcal{D}$ are equally likely, then $\overline{S}\,\overline{T}^2 = \Omega(n^6 \log \delta)$.

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

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 γ-nice, so it suffices for a γ-nice matrix to exist. Unfortunately, the bound is weakened to $\overline{S}\,\overline{T} = \Omega(n^3 \log\delta)$. No better bound can be proved when A is fixed, since then there is a branching program which 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 A and B be γ-nice most of the time. So the proof really depends on the probabilistic nature of lemma 4.3.

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 multiplication 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 α and β be positive integers, and let $w = \lceil 15\alpha/n \rceil$. Suppose $w^2 \leq \beta \leq n^2/4$. Then $C_f(\alpha, \beta) < 2^{2-w}$.

Proof: Let τ be a depth α non-redundant binary tree program. The *weight* of a computation in τ 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) follow a computation in τ of weight at least w. We show that $p \leq 3 \cdot 2^{-w}$.

Let r_k be the maximum, over all computations π of weight k, of the probability that a random pair of permutation matrices P and Q are consistent with the responses on computation π. Each time π 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 π which looks for up to n 1's in P, after which it looks for 1's in Q. So

$r_k \le 1/n^{\underline{k}}$ for $k \le n$, and $r_k \le 1/(n!\, n^{\underline{k-n}})$ for $k > n$, where $n^{\underline{k}} = n!/(n-k)!$ is the descending power of n. Then

$$p \le \sum_{w \le k \le n} \binom{\alpha}{k} \frac{1}{n^{\underline{k}}} + \sum_{n < k \le 2n} \binom{\alpha}{k} \frac{1}{n!\, n^{\underline{k-n}}}. \tag{8.1}$$

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

Now consider the contribution of light computations. There are $\sum_{k=1}^{w-1} \binom{\alpha}{k} < \binom{\alpha}{w}$ computations of weight less than w in τ. Suppose a given one of them, π, β-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 β-solving a random input maximized, if the β 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 $\sqrt{\beta} \le n/2$, 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 $\left(\frac{2}{n}\right)^{\sqrt{\beta}}$. So the contribution of light computations to $C_f(\alpha, \beta)$ is at most $\binom{\alpha}{w}\left(\frac{2}{n}\right)^{2\sqrt{\beta}} \le \left(\frac{e\alpha}{w}\right)^w \left(\frac{2}{n}\right)^{2w} < 2^{-w}$.

So the probability that any computation in τ β-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 which computes PAQ, for $n \times n$ matrices, in expected time $\overline{T} \ge n^2$ and space $\overline{S}$, where all permutation matrices are equally likely. Then $\overline{S}\,\overline{T} = \Omega(n^3)$.

Proof: Let $c = 1/(5 \cdot 18^2)$. We can assume that $\overline{T} \le 3cn^3$ and $n \ge 40$. Choose $\alpha = \lceil cn^4/\overline{T} \rceil$ and $\beta = \lceil n^2\alpha/4\overline{T} \rceil$. Then the conditions of theorem 8.1 are met. By lemma 3.3, $\overline{S} \ge w - 6 \ge 15\alpha/n - 6$, so $\overline{S}\,\overline{T} = \Omega(n^3)$. ∎

Acknowledgement

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

References

[1] K. R. Abrahamson, Generalized string matching, to appear in *SIAM J. on Computing.*

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, Mass., 1974.

[3] A. Borodin and S. Cook, A time-space tradeoff for sorting on a general sequential model of computation, *in* "12th Annual ACM Symposium on Theory of Computing, 1980," pp. 294–301.

[4] A. Borodin, D. Dolev, F. Fich, and W. Paul, Bounds for width two branching programs, *SIAM J. on Computing* **15,2** (1986) 549–560.

[5] A. Borodin, F. Fich, F. Meyer auf der Heide, E. Upfal, and A. Wigderson, A time-space tradeoff for element distinctness, *in* "3rd Annual Symposium on Theoretical Aspects of Computer Science, 1986," pp. 353–358.

[6] A. Borodin, M. J. Fischer, D. G. Kirkpatrick, N. A. Lynch, and M. Tompa, A time-space tradeoff for sorting on non-oblivious machines, *J. Comput. Sys. Sci.* **22,3** (1981) 351–364.

[7] D. Yu. Grigoryev, An application of separability and independence notions for proving lower bounds on circuit complexity, (in Russian), *Notes Scientific Seminar Steklov Mathematical Institute* **60** (1976) 35–48.

[8] J. E. Hopcroft and J. D. Ullman, "Introduction to Automata Theory, Languages and Computation," Addison-Wesley, Reading, Mass., 1979.

[9] J. Ja' Ja', Time-space trade-offs for some algebraic problems, *J. Assoc. Comput. Mach.* **30,3** (1983) 657–667.

[10] J. Ja' Ja' and J. Simon, Some space efficient algorithms, in "17th Allerton Conference on Communication, Control and Computing, 1979," pp. 677–684.

[11] J. E. Savage, Space-time trade-offs for banded matrix problems, *J. Assoc. Comput. Mach.*, **31,4** 422–437.

[12] J. E. Savage and S. Swamy, Space-time tradeoffs for oblivious integer multiplication, *in* H. A. Maurer, editor, Lecture Notes in Computer Science 71, pp. 240–251, Springer-Verlag, New York, 1979.

[13] M. Tompa, Time-space tradeoffs for computing functions, using connectivity properties of their circuits, *J. Comput. Sys. Sci.* **20** (1980) 118–132.

[14] J. Vuillemin, A combinatorial limit to the computing power of VLSI circuits, in "21st Annual IEEE Symposium on Foundations of Computer Science," pp. 294–300, 1980.

[15] I. Wegener, Time-space tradeoffs for branching programs, *J. Comput. Sys. Sci.* **32** (1986) 91–96.

[16] Y. Yesha, Time-space tradeoffs for matrix multiplication and the discrete Fourier transform on any general sequential random-access computer, *J. Comput. Sys. Sci.* **29** (1984) 183–197.

Appendix

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

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

$$A = \left(\begin{array}{c|c} B & C \\ \hline D & E \end{array}\right),$$

where B is $(n-r) \times (n-r)$ and E is $r \times r$. Presume that the system $A\vec{x} = \vec{b}$ has been permuted so that B is nonsingular. Let $\vec{y}$ consist of the first $n - r$ components of the indeterminate vector $\vec{x}$, and let $\vec{z}$ be the last r components of $\vec{x}$. Let $\vec{d}$ consist of the first $n - r$ components of $\vec{b}$. Then $A\vec{x} = \vec{b}$ implies that $B\vec{y} = \vec{d} - C\vec{z}$. So the first $n - r$ components of x are determined by the last r components. There are δ^r possible values for $\vec{z}$. ∎

Lemma 4.3: There is a constant γ, where $0 < \gamma < \frac{1}{2}$, such that at least a fraction $1 - \delta^{-1}\left(\frac{2}{3}\right)^{\gamma n}$ of the $n \times n$ matrices over $\mathcal{D}$ are γ-nice. The constant γ 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 γ will be selected below. For now, suppose we have γ. 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.

Imagine generating a random $n \times n$ matrix A over $\mathcal{D}$. Start at the lower left hand corner, and generate elements independently by diagonals, moving toward the upper right hand corner. During the process, maintain indices $\hat{\imath} = i_t \in I$ and $\hat{\jmath} = j_s \in J$, where element $A_{\hat{\imath},\hat{\jmath}}$ has not yet been chosen. Also maintain index sets $R = \{i_{t+1}, \ldots, i_p\}$ and $C \subseteq \{j_1, \ldots, j_{s-1}\}$, such that A_R^C is a nonsingular $(p - t) \times (p - t)$ submatrix of A. Initially, it suffices to choose $\hat{\imath} = i_p$ and $\hat{\jmath} = j_1$, assuming that a 0×0 matrix is defined to have determinant 1. Each time element $A_{\hat{\imath},\hat{\jmath}}$ is generated, s is incremented, and sometimes t is decremented.

Now imagine we have generated part of A, we have indices $\hat{\imath} = i_t$ and $\hat{\jmath} = j_s$, and we are about to generate element $A_{\hat{\imath},\hat{\jmath}}$. Let $R' = R \cup \{\hat{\imath}\}$ and

$C' = C \cup \{ j \}$. Notice 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 $\pm \det A_R^C \neq 0$. Hence, at 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 nonsingular submatrix has been found.

For each column index in J there will be an opportunity to decrement t, 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 nonsingular $p \times p$ submatrix of A_I^J 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 $g = \delta^2/(2\delta - 1)$.

Proof: By induction on n.

$\underline{n = 0}$: $P_{0,0} = 0 \leq \delta^{-1}$ and $P_{0,k} = 1 \leq \delta^{k-1}$ for $k \geq 1$.

$\underline{n > 0}$:

$$\begin{aligned} P_{n,k} &= \delta^{-1}P_{n-1,k} + (1 - \delta^{-1})P_{n-1,k-1} \\ &\leq \delta^{-1}g^{1-n}\delta^{k-1} + (1 - \delta^{-1})g^{1-n}\delta^{k-2} \\ &= g^{-n}\delta^{k-1}g\delta^{-1}(2 - \delta^{-1}) \\ &= g^{-n}\delta^{k-1} \end{aligned}$$

■

So, for the selected index sets I and J, the probability that A_I^J 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 $Q = \binom{n}{p}^2 g^{-q}\delta^{p-1}$. A crude Stirling approximation suffices to bound Q. Applying $\binom{n}{p} < n^p e^p p^{-p}$ and $g \leq 2\delta/3$ gives $Q < (ne/p)^{2p} g^{p-n}\delta^{p-1} \leq (ne\delta/p)^{2p}\left(\frac{2}{3}\right)^p g^{-n}\delta^{-1}$. To establish lemma 4.3, we need to show that $Q \leq \delta^{-1}\left(\frac{2}{3}\right)^p$. It suffices to show

that $(ne\delta/p)^{2p/n} \leq g$. But $n/p \geq 1/(2\gamma)$, and $\lim_{x\to\infty} x^{2/x} = 1$, so a sufficiently small choice of γ brings $(n/p)^{2p/n}$ below 1.1. Also, $2p/n \leq 4\gamma$, so a sufficiently small γ guarantees that $(e\delta)^{2p/n} \leq \max(1.1, \delta/3)$. Then for any $\delta \geq 2$, $(ne\delta/p)^{2p/n} < \delta^2/(2\delta - 1) = g$. ∎

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

Select an arbitrary $p \times q$ submatrix S of E, given by index sets I and J, where $|I| = p \leq \lceil c^2 n^2 \rceil$ and $|J| = q \geq \lfloor (1 - c^2)n^2 \rfloor$. It must be shown that S has rank at least $c^2 p$. 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 $ni, \ldots, ni + n - 1$. Let $\Delta_i = I \cap \{ni, \ldots, ni + n - 1\}$ be the rows of S which fall in the i^{th} block row. Choose a set $\Gamma \subset \{0, \ldots, n - 1\}$ 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 $\Gamma_i = \Delta_i$ if $|\Delta_i| \leq \lceil cn \rceil$, and let Γ_i consist of the smallest $\lceil cn \rceil$ members of Δ_i otherwise. Call the rows $\bigcup_{i\in\Gamma} \Gamma_i$ blue rows. There must be a total of at least $c^2 p$ blue rows. It suffices to show that the blue rows are linearly independent.

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

$$\sum_{i\in\Gamma} \sum_{j\in\Gamma_i} c_{i,j} E_{ni+j} = \vec{0}. \tag{A.1}$$

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 $\lfloor (1 - c)n \rfloor$ columns of S, that is if $|\{i : ni + j \in J\}| \geq \lfloor (1 - c)n \rfloor$. 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 $(cn)((1-c)n-1)+(1-c)n^2 < \lfloor(1-c^2)n^2\rfloor$ columns.) Let $\vec{g}_i$ 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 $\vec{g}_i$ are linearly independent. In particular, it must be the case that $\sum_{i\in\Gamma_r} c_{r,i}\vec{g}_i \neq \vec{0}$. So it is possible to choose a good column t such that

$$\sum_{i\in\Gamma_r} c_{r,i}b_{i,t} \neq 0 \tag{A.2}$$

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

Let $\vec{u}_i$ be the projection of E_i onto columns $\{\, ni+t : i \in \Phi\,\}$. Let $\vec{v}_i$ be the projection of the i^{th} row of A onto the columns of Φ. Then, from the definition of $E = A \otimes B$, $\vec{u}_{ni+j} = b_{j,t}\vec{v}_i$. Taking a projection of equation A.1,

$$\begin{aligned}\vec{0} &= \sum_{i\in\Gamma}\sum_{j\in\Gamma_i} c_{i,j}\vec{u}_{ni+j} \\ &= \sum_{i\in\Gamma}\left(\sum_{j\in\Gamma_i} c_{i,j}b_{j,t}\right)\vec{v}_i. \end{aligned}\tag{A.3}$$

But $|\Gamma| = \lceil cn\rceil$ and $|\Phi| \geq \lfloor(1-c)n\rfloor$. So the vectors $\vec{v}_i$ 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 $\vec{v}_i$ for $i \in \Gamma$ are linearly independent. Hence, the only way to satisfy equation A.3 is to have $\sum_{j\in\Gamma_i} c_{i,j}b_{j,t} = 0$ for every $i \in \Gamma$. But $r \in \Gamma$, so equation A.2 must be violated. The supposition that the blue vectors are linearly dependent has lead to a contradiction. ∎