An arithmetic complexity lower bound for computing rational functions, with applications to structured and sparse linear algebra

James Demmel

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-82
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-82.html

May 19, 2018

An arithmetic complexity lower bound for computing rational functions, with applications to structured and sparse linear algebra

James Demmel *

May 19, 2018

Abstract

In 1965 Klyuev and Kokovkin-Shcherbak proved that $\frac{n^3}{3} + O(n^2)$ multiplications are necessary to solve an n-by-n system of linear equations. In 1969 Strassen proved $O(n^{\log_2 7})$ multiplications are sufficient. They could both be right because they considered different models of permitted algorithms. Here we propose yet another algorithmic model, closer in spirit to Klyuev and Kokovkin-Shcherbak, but sufficiently more general to be able to provide lower bounds on the number of arithmetic operations required to perform dense, sparse or "structured" one-sided matrix factorizations: The simplest (and overly restrictive) version of our assumption is that each scalar result is computed using only the data on which it depends mathematically. We compare these lower bounds with a variety of algorithms and matrix sparsity patterns and algebraic structures (such as Vandermonde). Depending on the sparsity patterns and structures, conventional algorithms may or may not attain the lower bounds. In some cases when they do not, we present new algorithms that do, for example for the QR decomposition of a sparse matrix.

1 Introduction

Consider computing a rational function $z = f(x)$, where $x \in \mathbb{F}^n$, $z \in \mathbb{F}$ and $\mathbb{F}$ is a field, which may be $\mathbb{R}$ or $\mathbb{C}$. A simple lower bound on the number of binary arithmetic operations required to compute $f(x)$ may be derived by considering the dimension d of the manifold $\mathcal{F} = \{\nabla f(x) | x \in \mathbb{F}^n\}$ [15, 41] determined by the Jacobian ∇f of f: If we perform M binary operations ($\pm$, $\cdot$ or $/$) to compute $f(x)$, then it can depend on at most $M + 1$ different inputs. If $f(x)$ depended on fewer than d inputs, then the dimension of $\mathcal{F}$ could not equal d. Therefore $d - 1$ is a lower bound on the number of arithmetic operations. This idea also appears in the use of transcendence degree to prove lower bounds in [10, Chap 5].

We apply this idea to algorithms from linear algebra. In particular, we want a simple characterization of "non-Strassen-like" matrix algorithms that we can use to prove lower bounds on the number of arithmetic operations required to perform matrix multiplication, or one-sided matrix operations like LU and QR decomposition, for dense, sparse or structured matrices (i.e. matrices whose entries depend on parameters, like Vandermonde matrices $V_{ij} = x_j^{i-1}$). With few assumptions (on the size of the constants appearing in the algorithm, and no divisions allowed), the best

*Computer Science Division and Mathematics Dept., University of California, Berkeley, CA 94720 (demmel@cs.berkeley.edu).

lower bound for dense, unstructured n-by-n matrix multiplication is $\Omega(n^2 \log n)$ operations [36], which no existing algorithm comes close to attaining [44]. In contrast, our bounds will be attained (within small integer factors) by conventional dense algorithms. Furthermore, depending on the sparsity pattern or algebraic structure, they may or may not be attained by conventional algorithms. In some cases when conventional algorithms do not attain the lower bound, we present new algorithms that do.

The first attempt at such lower bounds (predating Strassen's paper [42]) was for solving dense unstructured linear systems [22, 23] where algorithms were limited to adding multiples of rows (or columns) to other rows (or columns). Under this assumption, a lower bound of $\frac{n^3}{3} + O(n^2)$ multiplications for dense, full rank, n-by-n matrices was proven. This bound is attained by conventional algorithms.

We weaken this assumption as much as possible in order to generalize to other matrix factorizations and sparsity patterns: Our main assumption is that the only data used to compute a particular component of the final answer is the data on which it mathematically depends.

For example, in the case of matrix multiplication $C = A \cdot B$, we will assume that the operations that lead to computing $C(:, i)$ (the i-th column of C) only depend on A and $B(:, i)$. This means that we only need a lower bound for matrix-vector multiplication, since $C(:, i) = A \cdot B(:, i)$. Clearly Strassen's algorithm and others like it [44] do not satisfy this, since they form linear combinations of submatrices of B that combine entries of different columns.

Similarly, the i-th columns of L and U in the factorization $A = LU$ only depend on the leading i columns of A. In particular, the i-th columns of L and U can be expressed by applying a linear operator (that depends only on the leading $i - 1$ columns of A) to $A(:, i)$. QR factorization is similar.

We also need to make assumptions about the algebraic independence of the matrix entries. For example, if we know that A is a rank-1 matrix, then multiplication by A can be done with fewer operations than for a general A. The simplest situation will be when all the nonzero matrix entries are algebraically independent. For example, in the case of LU factorization of a dense n-by-n matrix with n^2 algebraically independent entries, one of our lower bounds (see Corollary 5) is $\frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$ multiplications and divisions (abbreviated m/d), which omits only the $\frac{n^2-n}{2}$ m/d's required to divide by all the pivots.

Using these lower bounds, we apply them to various sparse and structured matrices, (such as Vandermonde), and derive new lower bounds. In some cases these lower bound suggest algorithms that are much faster than before. For example, for an "arrow matrix" (nonzero only in the first row, first column and diagonal), LU without pivoting fills in all zero matrix entries leading to an $O(n^3)$ cost. But the lower bound is $(n-1)^2$, and we present an algorithm that attains this lower bound. Of course in this case, one might argue that by reversing the order of the rows and columns, no zero entries are filled in, leading to a lower bound and cost of $\Theta(n)$. We suggest that it would be interesting to apply our lower bound to more complicated sparsity structures, and see how close various matrix reordering schemes come to attaining it.

We also consider the QR decomposition of a sparse m-by-n matrix A with $m > n$. Letting d_i be the number of nonzeros in row i, our lower bound becomes $\Omega(\sum_{i=1}^{m} d_i^2)$. This is in fact attained (modulo lower terms when $m \gg n$) by the Cholesky QR algorithm, which forms $A^T A$ and does its Cholesky factorization $A^T A = R^T R$ to get R. But it is well known that Cholesky QR is not numerically stable if A is ill-conditioned, leading us to ask if there is a stable algorithm that attains the lower bound. We present the first one we know of that does, modulo a "lower order term" that

can be very large. So whether this algorithm is superior to existing sparse QR implementations (see [25] for a survey) remains to be seen.

Besides [10], the other lower bounds work closely related to our is due to Winograd [45, 46], who also provided arithmetic lower bounds on matrix-vector multiplication and related operations. We will see that our lower bounds apply to more general situations, and can in fact be larger (stronger) than his lower bounds.

The rest of this paper is organized as follows. Section 2 presents our 3 models of computation in detail, and uses them to get lower bounds for applying a linear operator to a vector. Section 3 applies this to derive lower bounds for matrix-matrix multiplication, section 4 for LU decomposition, and section 5 for QR decomposition. Sections 4 and 5 also present new optimal algorithms. Section 6 discusses related work, and section 7 lists open problems.

2 Lower Bounds for Applying a Linear Operator to a Vector

We begin with a lower bound for matrix-vector multiplication, or more generally applying a linear operator to a vector. We state our results over a field $\mathbb{F}$, which could be either the real or complex numbers.

2.1 Arithmetic lower bound on $z = B(x) \cdot y$ for independent y_i.

Suppose we want to compute a linear transformation $z = B(x) \cdot y$ where $z \in \mathbb{F}^m$, $x \in \mathbb{F}^r$, $y \in \mathbb{F}^n$ and $B(x)$ is a linear operator that depends on x. Assume that all the entries of x and of y are algebraically independent (we weaken the assumption about y in section 2.2). $B(x)$ may or may not be represented explicitly as an m-by-n matrix, but thinking of $B : \mathbb{F}^r \to \mathbb{F}^{m \times n}$ as a (possibly piecewise-smooth) map, its image $\mathcal{B} = B(\mathbb{F}^r)$ is a manifold (or union of manifolds). Let d be the dimension of this manifold (or highest dimension of any manifold in the union); d will determine our lower bound below.

Our approach using d is closely related to the use of transcendence degree in [10]. We choose to use this quantity to be able to more directly apply known results on the dimension of Stiefel manifolds in our later analysis of the QR decomposition, and to make the extension to dependent y_i in the next section straightforward.

Now we need to be more specific about our models of computation: Consider the DAG (Directed Acyclic Graph) describing the algorithm for computing z. There may be branches (including implicit ones if abs/min/max are used), so we consider only the (sub)DAG executed when $B(x)$ lies in the d-dimensional part of $\mathcal{B}$. This DAG has input nodes (labeled x or y), output nodes (labeled z), and computational nodes ($\pm$, $\cdot$, /, and perhaps unary operations like sqrt). If we write $z_i = \sum_{j=1}^{n} b_{ij}(x) \cdot y_j$, then each $b_{ij}(x)$ is determined (perhaps implicitly) by the DAG.

We distinguish three models of computation, with different assumptions about the DAG and/or counting goals:

Model 1. The operations performed by the DAG must obey the following rules: Computational nodes are only allowed to do the following: (1) perform arbitrary operations when both inputs are labeled c) (meaning a constant in $\mathbb{F}$), in which case the output is labeled c, (2) perform arbitrary operations when one input is labeled x and the other input is either labeled x or c (in which case the output is labeled x), and otherwise (3) compute linear homogeneous functions

of y with coefficients that can depend on x or constants, i.e. of the same functional form as the output z (in which case the output is labeled y_{homo}). In other words, the operations $\{y, y_{homo}\} \pm \{y, y_{homo}\}$, $\{y, y_{homo}\} \cdot \{x, c\}$ and $\{y, y_{homo}\}/\{x, c\}$ are allowed, but the following operations are forbidden: $\{y, y_{homo}\} \cdot \{y, y_{homo}\}$, $\{c, x, y, y_{homo}\}/\{y, y_{homo}\}$, $\{x, c\} \pm \{y, y_{homo}\}$, $\sqrt{\{y, y_{homo}\}}$, etc. Using this model our goal will be to lower bound the number of m/d's.

Model 2. The operations performed by the DAG may be arbitrary. Our goal will be to lower bound the number of multiplications, divisions, additions and subtractions (abbreviated m/d/$\pm$).

Model 3. The operations performed by the DAG may again be arbitrary. Our goal will be to lower bound just the number of m/d's.

Model 1 is most limited in the operations it permits the algorithm to perform, but captures the behavior of the most commonly used algorithms, and allows the largest lower bound to be proven on the number of m/d's: d (see Lemma 1 below). Model 1 is the closest of our three models to [22, 23], and the linear computation sequence in [10, Def. 13.2]. Model 2 allows the most general algorithms, but only allows us to bound the number of m/d/$\pm$'s below by d (see Lemma 2). Model 3 also allows the most general algorithms, and allows us to prove a lower bound of $\lceil d/2 \rceil$ on the number of m/d's (see Lemma 3). This is a strengthening of the results in [45, 46], which we discuss further in section 6.

Lemma 1 *Under the assumptions of Model 1, the number M_1 of m/d's needed to compute $z = B(x) \cdot y$ is at least d.*

Proof: In this simple case we look at all the computational nodes whose outputs are y_{homo}, and have one input labeled x. Thus all such nodes must perform $\{y, y_{homo}\} \{\cdot, /\} x$. The number q of such nodes is a lower bound on the total number of m/d's M_1. Let $\{f_1(x), ..., f_q(x)\}$ be the set of all x inputs to these computational nodes. Thus the final output nodes z can only depend on these q (possibly different) functions of x. This means we can write the mapping $B(x)$ as the composition $B(x) = H(G(x))$, where $G : \mathbb{F}^r \to \mathbb{F}^q$ maps all the inputs to the set $\{f_1(x), ..., f_q(x)\}$, and $H : \mathbb{F}^q \to \mathbb{F}^{m \times n}$ maps $\{f_1(x), ..., f_q(x)\}$ to the (implicitly defined) matrix entries $\{b_{11}(x), ..., b_{mn}(x)\}$. This means that the dimension d of the set $\{B(x) : x \in \mathbb{F}^r\}$ can be at most the dimension of $\{G(x) : x \in \mathbb{F}^r\}$ which is at most q, since all the arithmetic operations are (piecewise) smooth. Altogether, we have $M_1 \geq q \geq d$ as desired. $\square$

Lemma 2 *Under the assumptions of Model 2, the number M_2 of m/d/$\pm$'s needed to compute $z = B(x) \cdot y$ is at least d.*

Proof: The proof is very similar to that of Lemma 1. The outputs of the computational nodes are labeled as follows: (1) If both inputs of the computational node are labeled c, then its output is labeled c. (2) If both inputs of the computational node are labeled x, or one is x and the other c, then its output is labeled x. (3) If at least one input of the computational node is labeled y, then the output is labeled y. Let M_2 be the number of computational nodes with one input labeled x and one input labeled y. By the same argument as in Lemma 1, M_2 is bounded below by the number of x inputs to these computational nodes, which is in turn bounded below by the dimension d of the set $\mathcal{B}$. $\square$

Lemma 3 *Under the assumptions of Model 3, the number M_3 of m/d's needed to compute $z = B(x) \cdot y$ is at least $\lceil d/2 \rceil$.*

Proof: We want to upper bound the number of possibly different functions of x on which intermediate results depend, as a function of the number of m/d's. In particular, we care about terms that depend both on x and y variables, i.e. we will say that $x_1 \cdot y_1$ and $x_1 \cdot y_1 + x_2$ both depend on one function of x, namely x_1. This is because ultimately we want $B(x) \cdot y$, a linear homogeneous function of y, so until x_2 is (possibly) multiplied by a term depending on y, we will not count it. This means that we will count the introduction of a function of x only when a m/d occurs. But it is possible that a m/d introduces dependencies on 2 different functions of x. For example, $(y_1 + x_1) \cdot (y_2 + x_2) = y_1 \cdot y_2 + y_1 \cdot x_2 + y_2 \cdot x_1 + x_1 \cdot x_2$ depends on both x_1 and x_2. In other words, the number q of independent functions of x on which all the outputs can depend will be bounded above by twice the number of m/d's $2M_3$, which in turn is bounded below by d: $2M_3 \geq q \geq d$.

More formally, we may label and count all the nodes of the computational graph as follows. We label the inputs of the computational graph of the program by one of $c(onstant)$, x or y, and the outputs of each computational node as described in the following table. Note that we introduce a new label y^+, to refer to intermediate results like $y_1 + x_1$ which are "carrying" a function of x that could be counted in a subsequent operation. The integer associated with each operation is the number of new functions of x on which the output could possibly depend.

		c		x		y		y^+	
c	m/d	c	0	x	0	y	0	y^+	0
	$\pm$	c	0	x	0	y	0	y^+	0
x	m/d	x	0	x	0	y	1	y^+	1
	$\pm$	x	0	x	0	y^+	0	y^+	0
y	m/d	y	0	y	1	y	0	y	1
	$\pm$	y	0	y^+	0	y	0	y^+	0
y^+	m/d	y^+	0	y^+	1	y	1	y^+	2
	$\pm$	y^+	0	y^+	0	y^+	0	y^+	0

This table describes the counting argument in the first paragraph of this proof in more detail: Note that no $\pm$ can introduce a dependence (the associated table entries are all zero), and the largest integer is otherwise 2. After labeling the computational graph, we add all the integers associated with node, which is at most $2M_3$. We note that cancellation is possible, eg subtracting a y^+ argument from itself yields the constant 0. But at worst this will overcount the possible number of dependencies on functions of x that are introduced, which does not change validity of the lower bound. $\square$

We note that the use of unary operations like sqrt does not change the conclusions of Lemmas 2 or 3 because these cannot introduce any new dependencies ($\sqrt{z}$ depends on the same functions of x as does z).

2.2 Arithmetic lower bound on $z = B(x) \cdot y$ for algebraically related y_i.

Suppose we want to compute a linear transformation $z = B(x) \cdot y$ where the entries of y are not algebraically independent, but "independent enough" to be able to extract the $b_{ij}(x)$ from z. For example, if $y_i = t^i$ where t is a single variable that is algebraically independent of x, then each

entry of $z = B(x) \cdot y$ is a polynomial in t with coefficients $b_{ij}(x)$ that are uniquely determined by z. Obviously many other choices of vectors y have this property. We now extend the lower bounds of Models 2 and 3 to this situation.

Lemma 4 *Assume that the function $z_i(x, y) = \sum_{j=1}^{n} b_{ij}(x) \cdot y_i$ uniquely determines the coefficients $b_{ij}(x)$. Let d be the dimension of $\mathcal{B}$ as before. Then the number M_4 of m/d/± 's needed to compute $z = B(x) \cdot y$ is at least d.*

Proof: The proof is nearly identical to that of Lemma 2. As before, the outputs of the computational nodes are labeled as follows: (1) If both inputs of the computational node are labeled c, then its output is labeled c. (2) If both inputs of the computational node are labeled x, or one is x and the other c, then its output is labeled x. (3) If at least one input of the computational node is labeled y, then the output is labeled y. Let M_4 be the number computational nodes with one input labeled x and one input labeled y. By the same argument as in Lemma 2, M_4 is bounded below by the number of x inputs to this last set of computational nodes, which is in turn bounded below by the dimension d of the set $\mathcal{B}$. □

Similarly, Model 3 is extended as follows:

Lemma 5 *Assume that the function $z_i(x, y) = \sum_{j=1}^{n} b_{ij}(x) \cdot y_i$ uniquely determines the coefficients $b_{ij}(x)$. Let d be the dimension of $\mathcal{B}$ as before. Then the number M_5 of m/d's needed to compute $z = B(x) \cdot y$ is at least $\lceil d/2 \rceil$.*

2.3 Examples

Polynomial Evaluation. Computing $z = \sum_{i=0}^{n} x_i t^i$ is a special case of our result.

Corollary 1 *Under the assumptions of Model 2, the number of m/d/± 's needed to compute $z = \sum_{i=0}^{n} x_i t^i$ is at least $n - 1$. Under the assumptions of Model 3, the number of m/d's needed to compute $z = \sum_{i=0}^{n} x_i t^i$ is at least $\lceil n/2 \rceil$.*

Proof: To apply Model 2, we consider $\sum_{i=1}^{n} x_i t^i$, i.e. we drop the $i = 0$ term. Then Lemma 4 applies immediately, and we get a lower bound of n m/d/±'s. Then clearly at least $n - 1$ m/d/±'s are needed for $\sum_{i=0}^{n} x_i t^i$. Similarly for Model 3, the number of m/d needed to compute $\sum_{i=1}^{n} x_i t^i$ is $\lceil n/2 \rceil$, and the same lower bound holds for $\sum_{i=0}^{n} x_i t^i$. □

In section 6 we compare this to prior results by Motzkin [31] and Pan [33], also described in [45, 46].

Dense and Sparse Matrix-Vector Multiplication. We now apply Lemmas 1 through 3 to a number of basic examples.

Corollary 2 *Consider multiplying $z = B \cdot y$ where B is a m-by-n matrix of $nnz \leq m \cdot n$ algebraically independent entries, the remaining $m \cdot n - nnz$ entries being zero (nnz is short for "number of nonzeros"). Suppose the entries of y are also algebraically independent of one another and of B. Then*

- *Under Model 1, nnz m/d's are needed to compute z.*

- *Under Model 2, nnz m/d/±'s are needed to compute z.*
- *Under Model 3, $\lceil nnz/2 \rceil$ m/d's are needed to compute z.*

Proof: These results follow immediately from Lemmas 1 through 3, since B defines a manifold of dimension nnz. □

We emphasize that the proof applies to a matrix B in the interior of the region where it defines a manifold of dimension d. For example, if the algorithm checked for the special case of $B = 0$, and returned $z = 0$ without doing any arithmetic (but nnz comparisons!) this would not invalidate the result. (It is an open question as to whether counting comparisons in addition to the usual arithmetic operations would eliminate this small set of exceptions.) Analogously, if B is any constant matrix (like the DFT), so that $d = 0$, our lower bound is zero.

These lower bounds are attained (within a factor of 2) by the straightforward algorithm. We note that the same lower bound and proof holds when computing $z = A^{-1} \cdot y$ or $z = f(A) \cdot y$ for any $f(A)$ that defines a manifold of dimension nnz. More generally, when f is not bijective, the dimension of the manifold determined by $f(A)$ depends both on f and the sparsity structure of A; for example $f(A) = A^2$ may be sparser than A, and even identically zero. These lower bounds are generally not thought to be attainable except in special cases, for example $z = A^{-1} \cdot y$ for triangular A.

Low rank matrices.

Corollary 3 *Consider multiplying $z = B \cdot y$ where B is a m-by-n matrix of rank k. Suppose y is algebraically independent as before. Then*

- *Under Model 1, $k(m + n - k)$ m/d's are needed to compute z.*
- *Under Model 2, $k(m + n - k)$ m/d/±'s are needed to compute z.*
- *Under Model 3, $\lceil k(m + n - k)/2 \rceil$ m/d's are needed to compute z.*

Proof: The dimension of the manifold defined by rank-k matrices of dimension m-by-n is $k(m + n - k)$, most easily derived by counting the number of independent nonzero entries in the LU decomposition of such a matrix. The rest follows from Lemmas 1 through 3. □

We note that an algorithm that "attains" this bound in the counting arguments of Lemmas 1 through 3 would factor $B = LU$ and then multiply $B \cdot y = L \cdot (U \cdot x)$. The counting arguments do not count any of the work to compute the LU factorization. We return to the complexity of LU factorization in the next section.

It is possible to get closer to the lower bound by using a randomized (Las Vegas-style) algorithm: Given k, randomly choose k rows and k columns (or a few more for safety); suppose wlog that they are the first k rows and columns. Factor B as

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix} \cdot B_{11}^{-1} \cdot \begin{bmatrix} B_{11} & B_{12} \end{bmatrix}$$

where B_{11} is k-by-k and assumed nonsingular. Then one could straightforwardly multiply

$$B \cdot y = \begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix} \cdot (B_{11}^{-1} \cdot (\begin{bmatrix} B_{11} & B_{12} \end{bmatrix} \cdot y))$$

in $kn + \frac{k^3}{3} + O(k^2) + mk$ multiplications, which is close to the lower bound when k is small enough. For a survey of randomized algorithms for matrix factorizations, see [19]. (Note that the above very simple algorithm cannot always be guaranteed to run in expected polynomial time, for example if B is very sparse.)

Structured matrices. Now we briefly discuss structured matrices, i.e. n-by-n matrices whose entries depend on few, typically $O(n)$, parameters. The literature on fast algorithms for such matrices is large (eg [21, 12]) so we mention just a few examples. Vandermonde matrices are given by $V_{ij} = x_j^{i-1}$, Cauchy matrices are given by $C_{ij} = 1/(x_i + \hat{x}_j)$, and Toeplitz matrices are given by $T_{ij} = x_{i-j}$. All three are examples of *low displacement rank matrices* [16], and depend on n or $2n$ algebraically independent parameters. So by Lemmas 1 through 3, we need $\Omega(n)$ operations to multiply them (or their inverses) by a vector. In fact there are $O(n \log n)$ or $O(n \log^2 n)$ algorithms for multiplying a vector by V, T, C or their inverses; see [18, 17, 1] for the algorithms and generalizations.

In section 6 we compare these results to prior work of Winograd [45, 46].

3 Lower bounds on matrix-matrix multiplication

As we said in the introduction, our goal is to give a complexity lower bound for "non-Strassen-like" matrix operations like matrix multiplication $C = A \cdot B$. In order to apply Models 1, 2 or 3 from section 2, we need to make the following assumptions:

Assumption 0_{MM}**.** The entries of A are algebraically independent of the entries of B.

Assumption 1_{MM}**.** If $i \neq j$, columns i and j of B are algebraically independent.

Assumption 2_{MM}**.** The only data used to compute column j of C are A and column j of B.

Assumption 0_{MM} is a natural generalization of the assumption of the last section that x and y are algebraically independent. Both Assumptions 0_{MM} and 1_{MM} are implied by the stronger assumption that each (nonzero) entry of A and B is algebraically independent. Assumption 2_{MM} eliminates the possibility of Strassen-like algorithms. Assumption 2_{MM} is implied by the stronger assumption that each C_{ij} is computed using only the data on which it mathematically depends, namely the data needed to compute the i-th row of A and j-th column of B. Note that by considering $C^T = B^T \cdot A^T$, Assumption 2_{MM} can be modified to assume that the only data used to compute row j of C are B and row j of A.

Let $Cost_k(A, b)$ be the cost lower bound of multiplying A times b, according to Lemma k, where $k \in \{1, 2, 3, 4, 5\}$. Note that $Cost_k$ may be a lower bound on the number of different kinds of operations, depending on k.

Theorem 1 *Suppose we want to multiply $C = A \cdot B$, where A is m-by-k, B is k-by-n, and C is m-by-n. Then under Assumptions 0_{MM}, 1_{MM} and 2_{MM}, and the further assumptions about the algorithm of Lemma k, $k \in \{1, 2, 3, 4, 5\}$, the number of operations required to compute $C = A \cdot B$ is at least $\sum_{i=1}^{n} Cost_k(A, B(:, i))$.*

Proof: It suffices to prove that the operations counted by Lemma k applied to $A \cdot B(:, i)$ are disjoint, for all i, so we can simply add the lower bounds. In the proofs of Lemmas 1 through 5,

we only counted computational nodes whose outputs were labeled y (or y_{homo}). Recall that y is a function of the vector being multiplied, or $B(:,i)$ in this case. Thus the computational nodes associated with different columns i and j have different labels, and can only be counted once, as desired. $\square$

It is easy to extend all the examples of section 2.3 to matrix-matrix multiplication, adding the lower bounds for each column. For example, multiplying a sparse A with nnz nonzeros times an n-by-m dense matrix B, where all entries are algebraically independent, costs at least $m \cdot nnz$ m/d's using Model 1, and $m \cdot \lceil nnz/2 \rceil$ m/d's using Model 3. The latter bound is attainable asymptotically for dense matrices (see section 6).

More interesting is when B is sparse, because then $Cost_k(A, B(:,i))$ depends only on the dimension of the manifold defined by the columns of A that correspond to nonzero entries of $B(:,i)$. We return to this in the next section when we consider sparse LU decomposition.

Finally, we consider an example of n-by-n structured-matrix-matrix multiplication, say $A \cdot V$ where each A_{ij} is nonzero and algebraically independent, and V is Vandermonde: $V_{ij} = x_j^{i-1}$. Here each column of V depends on the algebraically independent variable x_j, so by Theorem 1 we expect a lower bound of $\Omega(n^3)$. But in fact we can compute $A \cdot V$ in $O(n^2 \log^2 n)$ operations, by instead computing each column of $V^T \cdot A^T$ as a product of a (transposed) Vandermonde matrix and a column vector, i.e. evaluating a degree $n-1$ polynomial at n points, which can be done in $O(n \log^2 n)$ operations, as mentioned in the previous section. In other words there is a big advantage to not computing with each column of V independently, as assumed by Theorem 1.

4 LU decomposition

We reduce finding lower bounds on LU decomposition to the problem of matrix-multiplication as follows. Write

$$\begin{array}{c} \\ i \\ m-i \end{array}\!\!\begin{array}{c} \begin{array}{cc} i & 1 \end{array} \\ \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \end{array} = \begin{array}{c} \\ i \\ m-i \end{array}\!\!\begin{array}{c} \begin{array}{cc} i & m-i \end{array} \\ \begin{pmatrix} L_{11} & 0 \\ L_{21} & I \end{pmatrix} \end{array} \cdot \begin{array}{c} \\ i \\ m-i \end{array}\!\!\begin{array}{c} \begin{array}{cc} i & 1 \end{array} \\ \begin{pmatrix} U_{11} & U_{12} \\ 0 & S \end{pmatrix} \end{array}$$

where S is the Schur complement. Rewrite this as

$$\begin{aligned} \begin{array}{c} \\ i \\ m-i \end{array}\!\!\begin{array}{c} \begin{array}{c} 1 \end{array} \\ \begin{pmatrix} U_{12} \\ S \end{pmatrix} \end{array} &= \begin{array}{c} \\ i \\ m-i \end{array}\!\!\begin{array}{c} \begin{array}{cc} i & m-i \end{array} \\ \begin{pmatrix} L_{11}^{-1} & 0 \\ -A_{21} \cdot A_{11}^{-1} & I \end{pmatrix} \end{array} \cdot \begin{array}{c} \\ i \\ m-i \end{array}\!\!\begin{array}{c} \begin{array}{c} 1 \end{array} \\ \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} \end{array} \\ &= \begin{array}{c} \\ i \\ m-i \end{array}\!\!\begin{array}{c} \begin{array}{c} i \end{array} \\ \begin{pmatrix} L_{11}^{-1} \\ -A_{21} \cdot A_{11}^{-1} \end{pmatrix} \end{array} \cdot \begin{array}{c} \\ i \end{array}\!\!\begin{array}{c} \begin{array}{c} 1 \end{array} \\ \begin{pmatrix} A_{12} \end{pmatrix} \end{array} + \begin{array}{c} \\ i \\ m-i \end{array}\!\!\begin{array}{c} \begin{array}{c} 1 \end{array} \\ \begin{pmatrix} 0 \\ A_{22} \end{pmatrix} \end{array} \\ &\equiv X_i \cdot A_{12} + \begin{bmatrix} 0 \\ A_{22} \end{bmatrix} \end{aligned} \tag{1}$$

In other words, computing the $(i+1)$-st columns of the U factor and the Schur complement S requires applying a linear operator X_i (which depends only on the first i columns of A) to a vector $A_{21} = A(1:i, i+1)$. To get a lower bound, we make assumptions analogous to those of the last section:

Assumption 1_{LU} Columns i and j of A are algebraically independent, for all $i \neq j$.

Assumption 2_{LU} The only data used to compute column i of U and the Schur complement S will be columns 1 through i of A, i.e. the data on which U and S mathematically depends.

Theorem 2 *Suppose we compute the LU decomposition (with or without row pivoting) of the m-by-n matrix A, where $m \geq n$. Then*

- *if LU does not complete early because the matrix is not full rank,*
- *under Assumptions 1_{LU} and 2_{LU}, and*
- *under the further assumptions about the algorithm of Lemma k, $k \in \{1, 2, 3, 4, 5\}$,*

the number of operations required to compute L and U is at least

$$\sum_{i=1}^{n-1} Cost_k(X_i, A(1:i, i+1))$$

where X_i is defined in equation (1).

To apply this theorem when each $A(1:i, i+1)$ is dense, we need to compute the dimension of the manifold defined by the matrix X_i. When $A(1:i, i+1)$ is sparse, so that only some columns of X_i are needed, it is the dimension of the manifold they determine that we need.

4.1 Examples of Lower Bounds and Optimal Algorithms

We start with the simplest possible example, to make sure that our lower bound make sense in an extreme case, and then consider more interesting examples.

LU of a Triangular Matrix.

Corollary 4 *Suppose A is upper triangular or lower triangular. Then at least 0 operations are needed to compute its LU factorization.*

Proof: When A is upper triangular, the matrix X_i is constant (the first i columns of the identity matrix) and so defines a manifold of dimension 0. When A is lower triangular, the vector $A(1:i, i+1)$ is zero, and again the cost of multiplying by it is 0. □

The next example is more interesting:

LU of a Dense Matrix.

Corollary 5 *Suppose A is a dense, nonsingular n-by-n matrix whose entries are all algebraically independent. Then under Assumption 2_{LU}, and the further assumptions about the algorithm of Model k, $k \in \{1, 2\}$, the cost of computing L and U in $A = LU$ is at least $\frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$. Under Assumption 2_{LU} and the further assumptions of Model 3, the cost of computing L and U in $A = LU$ is at least $\frac{1}{2}(\frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6})$.*

Proof: We need to compute the dimension of the manifold defined by the matrix in equation (1). Since A_{11} consists of i^2 algebraically independent parameters, the subdiagonal entries of unit lower triangular matrix L_{11} can take on arbitrary values, so it defines a manifold of dimension $i(i-1)/2$. A_{21} consists of another $i(n-i)$ free parameters, so $A_{21}A_{11}^{-1}$ can also takes on arbitrary values, leading to a total dimension of $i(i-1)/2 + i(n-i)$. Summing $i(i-1)/2 + i(n-i)$ from $i = 1$ to $n-1$ yields the result. □

Applying this corollary for Model $k = 1$, we get a lower bound of $\frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$ multiplications and divisions to compute the LU decomposition of a dense n-by-n matrix. This is exactly how many multiplications conventional Gaussian Elimination performs, excluding dividing the left column of the Schur complement by the pivot at each step. Note that our lower bound does not include this cost of scaling the Schur complement to get L.

LU of a Band Matrix.

Corollary 6 *Let A be an n-by-n nonsingular matrix with bandwidth b, i.e. there are b nonzero diagonals above and below the main diagonal. Then assuming that all the entries of A within the band are algebraically independent, that we do not pivot, that Assumption 2_{LU} holds, and under the further assumptions about the algorithm of Model k, $k \in \{1, 2, 3\}$, the number of operations required to compute L and U in $A = LU$ is at least $nb^2 - O(b^3)$.*

Proof: Computing

$$Cost_k(\begin{matrix} i \\ n-i \end{matrix} \begin{pmatrix} \overset{i}{L_{11}^{-1}} \\ -A_{21} \cdot A_{11}^{-1} \end{pmatrix}, A(1:i, i+1))$$

must be done slightly differently than before, to take the sparsity of $A(1:i, i+1)$ into account: We only need to compute the dimension of the columns of the n-by-i matrix X_i which correspond to nonzero entries of $A(1:i, i+1)$. Excluding the first b steps of the process, the relevant nonzero entries are $A(i-b+1:i, i+1)$ i.e. the rightmost b columns of X_i. To make it easier to count the dimension of the manifold defined by these columns, we write $X_i = [L_{11}^{-1}; -L_{21} \cdot L_{11}^{-1}]$, to make it easier to see that the dimension is the sum of the free parameters in L_{11}^{-1}, namely $b(b+1)/2$, plus the number of free parameters in L_{21}, also $b(b+1)/2$, since L_{21} is upper triangular. Altogether there are $b(b+1)$ free parameters, and summing from $i = b$ to n, we get $nb^2 - O(b^3)$ as claimed. □

Again, $nb^2 + O(b^3)$ is the number of multiplications actually performed by conventional band Gaussian elimination.

One can imagine applying Theorem 2 for an arbitrary sparsity structure. In general, fill-in (which does not occur in the above example of band matrices without pivoting) will cause more operations to be performed than the lower bound.

To illustrate the potential gap between the lower bound and conventional sparse LU, and how this suggests better algorithms, we consider two examples.

LU of an Arrow Matrix. This n-by-n matrix has nonzeros only on the main diagonal, first row and first column. Thus the first step of Gaussian elimination causes the entire matrix to fill-in, and remain filled in throughout the rest of the algorithm, thus doing as much work as on a dense matrix, $n^3/3$ multiplications.

Let us evaluate the lower bound from Theorem 2: Since $A(1:i, i+1)$ is only nonzero in its first entry, we only care about the dimension of the manifold defined by the first column of X_i, namely $n-1$. Thus the lower bound is $(n-1)^2$, much smaller than $n^3/3$.

But the lower bound is (nearly) attainable, by an algorithm that appropriately exploits the sparsity structure of the original matrix, which guarantees that every off diagonal block $L(i:n, 1:i-1)$ of L or off diagonal block $U(1:i-1, i:n)$ of U is rank-one. This suggests the following algorithm, which starts by computing rank-1 representations of subdiagonal entries of $L_{ij} = c(i) \cdot y(j)$ and superdiagonal entries of $U_{i,j} = x(i) \cdot r(j)$:

```
... Algorithm 1: LU factorization of an arrow matrix in O(n^2) operations
... inputs:
...    d(1:n) = diagonal of A
...    r(2:n) = A(1,2:n) = first row of A
...    c(2:n) = A(2:n,1) = first column of A
DU(1) = d(1), x(1) = 1, y(1) = 1/d(1), s(1) = y(1),
for i=2:n,
      DU(i) = d(i) - c(i)*r(i)*s(i-1),
      x(i) = -c(i)*s(i-1),
      y(i) = -r(i)*s(i-1)/DU(i),
      s(i) = s(i-1) + x(i)*y(i),
end for
U(1,1) = DU(1),
for i=2:n,
      U(i,i) = DU(i),
      for j=1:i-1,
            L(i,j) = c(i)*y(j),
            U(j,i) = r(i)*x(j),
      end for
end for
```

Clearly, if we had only computed the rank-1 representations of L and U, the cost would have dropped to $O(n)$, but as long as we ask for an explicit representation of the dense matrices L and U, the lower bound must be $\Omega(n^2)$.

LU of a Broken Arrow Matrix. This n-by-n matrix is nearly the same as the last example, except that $A(2,2) = 0$. One may confirm that A is still generically nonsingular, and there is perfect cancellation when computing the subdiagonal entries of L in columns 3:n-1, so they are identically zero. Similarly the superdiagonal columns of U in rows 3:n-1 are also identically zero. This follows from the fact that if the 3rd through n-th diagonal entries of A were also zero, A would have rank 2, so the first two steps of Gaussian elimination would result in the trailing $(n-2)$-by-$(n-2)$ Schur complement being identically zero. For the general broken arrow matrix, this means that after the first two steps of Gaussian elimination, the Schur complement simply consists of the original diagonal submatrix of A, and no further work is required to factor it.

In other words, we can simply alter the above algorithm for the arrow matrix by stopping after two steps, lowering the cost to $O(n)$. Let us confirm that our lower bound similarly drops to $\Omega(n)$:

One may confirm that the first column of the X_i matrix is nonzero only in its first 2 entries, and so defines a manifold of dimension at most 2, so the sum of all these dimensions is at most $2n$.

Of course the ordering of rows and columns for the above examples is pessimal, maximizing the fill in. Reversing the order of rows and columns of the arrow matrix results in no fill-in during conventional sparse LU factorization, and so a linear time algorithm. Nevertheless, these examples motivate us to compare the lower bound to the work done by sparse solvers on other sparse matrices of interest.

LU of a Vandermonde matrix. A Vandermonde matrix with entries $V_{ij} = x_j^{i-1}$ has algebraically independent columns, each of which depends on a single parameter x_j. Because of the structure of each column, Lemma 4 can be used to bound $Cost_k(X_i, V(1:i, i+1))$ below. Since X_i only depends on i parameters $x_1, ..., x_i$, we get that this cost is i, making the total lower bound $\sum_{i=1}^{n-1} i = n(n+1)/2$. See [21] for a survey of algorithms and their generalizations that attain this bound.

5 QR Decomposition

Let $A = Q \cdot R$ be the QR decomposition of A, where A is m-by-n with $m \geq n$, Q is m-by-n, and R is n-by-n. Then $R = Q^T \cdot A$, and so $R(1:i, i+1) = (Q(:, 1:i))^T \cdot A(:, i+1)$. Here we consider only versions of QR that form R in this (potentially numerically stable) way, by multiplying part of Q in some explicit or implicit way by trailing columns of A, i.e. we exclude forming R by the Cholesky factor of $A^T A$ (to which our previous lower bounds may be applied). In other words, the top i entries of column i of R are gotten by applying a linear operator $(Q(:, 1:i))^T$, that only depends on the first i columns of A, to the $(i+1)$-st column of A.

This is all we need to apply our previous techniques to get a lower bound on *part* of the QR decomposition, which is then also a lower bound on the entire decomposition:

Theorem 3 *If A is m-by-n with $m \geq n$, and*

1. *Columns i and j of A are algebraically independent for $i \neq j$;*
2. *The only data use to compute column i of R will be columns 1 through i of A, i.e. the data on which R mathematically depends;*
3. *QR does not complete early because the matrix is not full rank; and*
4. *under the further assumptions about the algorithm of Lemma k, $k \in \{1, 2, 3, 4, 5\}$,*

the cost computing R is at least

$$\sum_{i=1}^{n-1} Cost_k((Q(:, 1:i))^T, A(:, i+1))$$

The proof is analogous to that of Theorems 1 and 2.

5.1 Examples of Lower Bounds and Optimal Algorithms

QR of Dense Matrices.

Corollary 7 *Suppose A is m-by-n and dense, with $m \geq n$, and all algebraically independent entries. Then the cost to compute R is at least $(3mn^2 - n^3 - 3mn + n)/6$ m/d/$\pm$'s, and half as many m/d's.*

Proof: The dimension of the real Stiefel manifold [15, 41] defined by the m-by-i orthogonal matrix $Q(:, 1:i)$ is $mi - i(i+1)/2$. Summing from $i = 1$ to $n-1$ yields the desired bound. $\square$

This lower bound may be compared to the actual costs of computing the QR decomposition using Householder transformations, which is $2n^2m - \frac{2}{3}n^3$ m/d/$\pm$'s, plus lower order terms. Since we are only bounding the number of operations to compute R, we expect to underestimate the number of multiplications in conventional algorithms by a constant factor (4 in this case). Since there are a large number of implicit ways to represent Q (eg Householder and Givens rotations blocked in a variety of ways) as well as explicit, it appears difficult to state more general lower bounds without further assumptions on the representation.

QR of Sparse Matrices.

Corollary 8 *Suppose A is m-by-n, with $m \geq n$, d_i nonzero entries in row i, and all nonzero entries are algebraically independent Then the cost to compute R is at least $\sum_{i=1}^{m} d_i(d_i - 1)/2$ m/d/$\pm$'s, and half as many m/d's.*

Proof: We proceed rowwise (as will the algorithm below), analyzing how much the cost of computing each column $(Q(:, 1:i))^T \cdot A(:, i+1)$ increases as each row r is added, i.e. we go from computing $(Q(1:r-1, 1:i)^T \cdot A(1:r-1, i+1)$ to $(Q(1:r, 1:i)^T \cdot A(1:r, i+1)$. If $A(r, i+1) = 0$, then clearly the cost remains the same. If $A(r, i+1) \neq 0$, then the cost increases (at least) by the increase in the dimension of the manifold determined by $Q(1:r, 1:i)$ over the manifold determined by $Q(1:r-1, 1:i)$, which is at least the number of nonzeros in $A(r, 1:i)$. In other words, for every nonzero $A(r, i+1)$, we increment the cost by the number of nonzeros in $A(r, 1:i)$. For a fixed r, summing over i yields a cost increment of $1 + 2 + \cdots + (d_r - 1) = d_r(d_r - 1)/2$. Summing over r yields the desired result. $\square$

We note that forming A^TA, and then computing its Cholesky factor R, costs $\sum_{i=1}^{m} d_i(d_i + 1)/2 + O(n^3)$ multiplications; for many matrices $O(n^3)$ is a lower order term (especially if $m \gg n$ and A^TA is sparse enough for sparse Cholesky to be much cheaper). Cholesky QR does not fit our algorithmic model, but this observation still suggests that a sparse QR algorithm with complexity $O(\sum_{i=1}^{m} d_i^2)$ plus "lower order terms" should be possible.

Here is a brief description of the algorithm. We use the following data structure: Consider a graph G with one vertex for each distinct sparsity pattern of a row of A, as well as vertices for sparsity patterns gotten by omitting nonzero entry 1, nonzero entries 1 to 2, nonzero entries 1 to 3, ..., nonzero entries 1 to $d_i - 1$ of row i; we call these the *trailing sparsity patterns* of a row. For example, a row with nonzeros in columns $(2, 4, 7)$ would yield vertices labeled by $(2, 4, 7)$ and for trailing sparsity patterns $(4, 7)$ and (7). Each distinct pattern is represented by a single vertex, so that the number of vertices is bounded both by $2^d - 1$ (the number of nonempty subsets of $(1, 2, ..., d)$) and by $nnz \equiv \sum_{i=1}^{m} d_i$.

Next we add an edge to G from vertex a to vertex b if the subset corresponding to b is obtained by omitting the first (lowest numbered) member of subset a. For example, we would have an edge from $(2, 4, 7)$ to $(4, 7)$ and from $(4, 7)$ to (7).

The algorithm works as follows. We describe it in a streaming fashion, processing one row of A at a time, but other parallel versions are easy to imagine.

... **Algorithm 2:** QR factorization of a sparse matrix in $O(\sum_{i=1}^{m} d_i^2 + l.o.t.)$ operations
... input: matrix A (in a sparse row format)
Phase 1:
initialize the graph G described above to be empty
for each row $r = A(i, :)$ of A, insert(r)

proc insert(r)
    find the vertex of G that matches r's sparsity structure
    if the vertex is empty,
        store r and its row index in the vertex
    else
        do a Givens rotation with r and the row already stored in the vertex,
            yielding one row $r1$ with the same sparsity structure as r,
            and another row $r2$ whose first nonzero entry is zeroed out
        replace the row that was already in the vertex with $r1$
        if $r2$ has any remaining nonzero entries
            insert($r2$) ... just follow edge from vertex for $r1$ to vertex for $r2$
        endif
    endif

Phase 2:
perform conventional (sparse) QR on the rows in the data structure G produced by Phase 1

We note that the above implementation ignores accidental cancellation in $r1$ and $r2$, which could be checked for if desired, to insert $r1$ and/or $r2$ into the "sparsest" possible vertex. In the generic case, assuming no accidental cancellation, the cost of calling insert on row i of A is clearly $O(d_i^2)$. Thus the total cost of Phase 1 is $O(\sum_{i=1}^{m} d_i^2)$.

Now we bound the cost of Phase 2. The number of nonzero rows in G can be most easily, and perhaps very pessimistically, bounded by $\min(2^n - 1, m)$, since there are are most $2^n - 1$ nonempty subsets of $(1, ..., n)$, and we have inserted m rows. Ignoring the sparsity of these rows for a moment, the arithmetic cost of QR on these rows will be at most $O(\min(2^n, m)n^2) = O(\min(n^2 2^n, mn^2))$. We may consider this a "lower order term" since $n^2 2^n$ is independent of m, and $\sum_{i=1} d_i^2 = \Omega(m)$, assuming we ignore zero rows.

There are natural variations of the above algorithm that may tradeoff work between the two phases. For example, we could treat each row of A as n/b consecutive blocks of b entries, and treat each block as either zero or nonzero. Each node in the graph G would represent up to b upper-triangular rows, and inserting a row into a node may involve b Givens (or other orthogonal) transformations. This may increase the cost of Phase 1. But the number of nodes may be much smaller (the pessimistic upper bound drops from 2^n to $2^{n/b}$), possibly dropping the cost of Phase 2.

6 Related Work

Besides work mentioned earlier by Bürgisser, Clausen and Shokrollahi [10], as well as Klyuev and Kokovkin-Shcherbak [22, 23], work close in spirit to ours is due to Winograd [45, 46]. Theorem 1 in [45] gives a lower bound u on m/d's for $\Phi \cdot y$ (we have changed his notation slightly to match our own $B(x) \cdot y$ and so avoid confusion). Here Φ is an m-by-n matrix with entries in the infinite field F, which contains the infinite subfield $G \subset F$. For example, F could be rational functions of x over $\mathbb{R}$, and G could be $\mathbb{R}$. Winograd's lower bound u is related to the notion of column rank: u is the largest number of columns of Φ such that no nontrivial linear combination of them over G is in G^m. In our case, that would mean the largest number of columns of $B(x)$ such that no linear combination of them with coefficients from $\mathbb{R}$ lies in $\mathbb{R}^m$.

We see that Winograd's lower bound u is a related concept to the dimension of the manifold d defined by all the entries of Φ, but can be arbitrarily smaller, i.e. weaker than our bound. In particular $u \leq n$, whereas the dimension of the manifold can be as large as mn.

Winograd goes on in Corollary 2 to analyze $X \cdot y$, by rewriting it as $X \cdot y = \Phi \cdot z$, where z is mn-by-1, containing the rows of X stacked from top to bottom, and Φ is m-by-mn and sparse with copies of y^T on each row arranged so that $X \cdot y = \Phi \cdot z$. Assuming that the y_i are algebraically independent, Corollary 2 says the lower bound u from Theorem 1 is mn, the expected answer. But apparently left unstated is the assumption that the entries of X are algebraically independent (enough), since otherwise we know we can compute $X \cdot y$ much faster.

Pan [33] and Motzkin [31] both consider polynomial evaluation $\sum_{i=0}^{n} x_i t^i$, which can be thought of as the matrix-vector product $[x_0, ..., x_n] \cdot [1, t, ..., t^n]^T$. Pan's lower bound on m/d's is n (attained by Horner's rule), and Motzkin's is $\lceil (n+1)/2 \rceil$. Motzkin's bound is also attainable, because Motzkin does not count operations required for "preconditioning of coefficients" [45], i.e. operations involving only the x_i variables.

Our lower bound from Corollary 1 on m/d's is $\lceil n/2 \rceil$. Note that our proof also does not count operations that only combine expressions depending on x, i.e. "preconditioning". This also generalizes and strengthens Theorem 2 in [45], replacing the lower bound $\lceil u/2 \rceil$ by $\lceil d/2 \rceil$.

As another example of preconditioning, our lower bound is $n^3/2$ m/d's for dense n-by-n matrix multiplication using Model 3, half the standard algorithm. An algorithm using $n^3/2 + O(n^2)$ m/d's is given in [45, section 5].

There is of course a great deal of other related work on arithmetic lower bounds, for a variety of different problems (eg. matrix multiplication, DFTs, convolutions, multiplying polynomials, and dividing polynomials with remainder), with a variety of assumptions about the algorithms (eg. bilinear circuits, bounded coefficients) [37, 11, 3, 29, 30, 14, 13, 28, 26, 32, 35, 43, 8, 24, 9, 5, 4, 6, 40, 20, 38, 34, 39, 27]. For example, by assuming bounded coefficients in the algorithm, [37] shows a lower bound of $\Omega(n^2 \log n)$ for matrix multiplication, and [11] shows a lower bound of $\Omega(n \log n)$ for multiplying and dividing polynomials with remainder. In general, our computational model is less restrictive than this other work by making no assumptions about constants, more restrictive in the sense of excluding "Strassen-like" algorithms, and correspondingly provides larger lower bounds.

7 Conclusions and Open Problems

We have reasoned using the dimensions of manifolds to provide lower bounds on the number of arithmetic operations required to perform a variety of linear algebra operations, including matrix

multiplication, LU factorization and QR factorization. By assuming that the computation only uses the data on which the answer mathematically depends (an apparently reasonable assumption, but violated by fast algorithms like Strassen's method), we show that conventional dense algorithms are optimal, or within a constant of optimal.

Our bounds extend naturally to symmetric matrix factorizations such as Cholesky by considering the formulation that only accesses the upper triangular part of the matrix.

Our bounds extend to sparse and structured matrices (like Vandermonde matrices) where algorithms like Strassen's may not provide useful speedups. Our lower bounds are more complicated functions of the sparsity and structure than the general dense case, and provide metrics by which to evaluate existing algorithms. We illustrated this by some (extremely) sparse examples where the lower bounds were significantly lower than what a sparse solver would do (given a fixed, and poor, row and column ordering), and presented algorithms that did attain the lower bounds.

There are a number of open problems this work suggests:

- We should compare our lower bounds to the work done by conventional sparse solvers on other sparse matrices of interest, with better ordering of rows and columns; if the work performed greatly exceeds the lower bound, this provides motivation to look for a better algorithm.

- We illustrated our lower bounds for the simplest cases of structured matrices (Toeplitz, Vandermonde and Cauchy), and showed they were attainable (modulo polylogarithmic factors). There are many more kinds of structured matrices to which our bounds could be applied, and compare to available algorithms. In particular, we need to generalize our assumption about the algebraic independence of matrix columns (true for Vandermonde, but not Toeplitz) in order to extend our lower bounds to the factorization of more such structured matrices.

- Our examples of arrow matrices are special cases of hierarchically semiseparable (HSS) matrices [7], which have the property that off-diagonal blocks have low rank. There has been a great deal of recent work in developing fast solvers for such matrices. Again, these matrices do not satisfy our assumption that columns are algebraically independent, so it would again be of interest to modify this assumption to accommodate HSS matrices.

Finally, we note that it is possible to prove lower bounds on communication, i.e. data movement, which are proportional to the number of arithmetic operations performed [2]. Data movement is increasingly important because it is much more expensive than arithmetic. Thus our lower bounds on arithmetic, combined with results in [2], also provide lower bounds on communication for any algorithm satisfying the assumptions in this paper.

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.

[2] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Mat. Anal. Appl.*, 32(3):866–901, 2011.

[3] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical computer science*, 22(3):317–330, 1983.

[4] Markus Blaser. A 5/2n¡ sup¿ 2¡/sup¿-lower bound for the rank of n× n-matrix multiplication over arbitrary fields. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 45–50. IEEE, 1999.

[5] Markus Bläser. Lower bounds for the multiplicative complexity of matrix multiplication. *computational complexity*, 8(3):203–226, 1999.

[6] Markus Bläser. A 52n2–lower bound for the multiplicative complexity of n× n–matrix multiplication. In *STACS 2001: 18th Annual Symposium on Theoretical Aspects of Computer Science, Dresden, Germany, February 15-17, 2001. Proceedings*, volume 2010, page 99. Springer, 2001.

[7] S. Börm, L. Grasedyck, and W. Hackbusch. *Hierarchical Matrices*. Max Planck Institute for Mathematics in the Sciences, 2006. www.mis.mpg.de/preprints/ln/lecturenote-2103.abstr.html.

[8] Roger W Brockett and David Dobkin. On the optimal evaluation of a set of bilinear forms. *Linear Algebra and Its Applications*, 19(3):207–235, 1978.

[9] Nader H Bshouty. A lower bound for matrix multiplication. *SIAM Journal on Computing*, 18(4):759–765, 1989.

[10] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*. Springer Verlag, Berlin, 1997.

[11] Peter Bürgisser and Martin Lotz. Lower bounds on the bounded coefficient complexity of bilinear maps. *J. ACM*, 51(3):464–482, May 2004.

[12] S. Chandrasekaran, M. Gu, X. Sun, J. Xia, and J. Zhu. A superfast algorithm for Toeplitz systems of linear equations. *SIAM Journal on Matrix Analysis and Applications*, 29(4), 2007.

[13] Bernard Chazelle. Lower bounds on the complexity of polytope range searching. *Journal of the American Mathematical Society*, 2(4):637–666, 1989.

[14] Bernard Chazelle. A spectral approach to lower bounds with applications to geometric searching. *SIAM J. Comput.*, 27(2):545–556, April 1998.

[15] A. Edelman, T. Arias, and S. Smith. The geometry of algorithms with orthogonality constraints. *SIAM J. on Mat. Anal. Appl.*, 20(2):303–353, October 1998.

[16] I. Gohberg, T. Kailath, and V. Olshevsky. Fast Gaussian elimination with partial pivoting for matrices with displacement structure. *Math. Comp.*, 64(212):1557–1576, 1995.

[17] I. Gohberg and V. Olshevsky. Complexity of multiplication with vectors for structured matrices. *Lin. Alg. Appl.*, 202:163–192, 1994.

[18] I. Gohberg and V. Olshevsky. Fast algorithms with preprocessing for matrix-vector multiplication problems. *J. Complexity*, 10:411–427, 1994.

[19] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, June 2011.

[20] Maurice J Jansen and Kenneth W Regan. A nonlinear lower bound for constant depth arithmetical circuits via the discrete uncertainty principle. *Theoretical Computer Science*, 409(3):617–622, 2008.

[21] T. Kailath and A. H. Sayed. Displacement structure: Theory and applications. *SIAM Review*, 1995.

[22] V. V. Klyuev and N. I. Kokovkin-Shcherbak. Minimization of the number of arithmetic operations in the solution of linear algebra systems of equations. *USSR Computational Mathematics and Mathematical Physics*, 5(1):25–43, 1965. translated by H. F. Cleaves.

[23] N. I. Kokovkin-Shcherbak. Minimization of numerical algorithms for solving arbitrary systems of linear equations. *Ukrainskii Matematicheskii Zhurnal*, 22(4):494–502, 1970.

[24] Jean-Claude Lafon and Shmuel Winograd. A lower bound for the multiplicative complexity of the product of two matrices. *Centre de Calcul de L'Esplanade, UER de Mathematique, Univ. Louis Pasteur, Strasbourg, France*, 1978.

[25] X. Li. Direct solvers for sparse matrices. `http://crd-legacy.lbl.gov/~xiaoye/SuperLU/SparseDirectSurv` 2013.

[26] Satyanarayana V Lokam. Spectral methods for matrix rigidity with applications to size–depth trade-offs and communication complexity. *Journal of Computer and System Sciences*, 63(3):449–473, 2001.

[27] Satyanarayana V Lokam. *Complexity Lower Bounds Using Linear Algebra*, volume 4. Now Publishers Inc, 2009.

[28] Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10(1):157–182, 1993.

[29] Jacques Morgenstern. Note on a lower bound on the linear complexity of the fast fourier transform. *J. ACM*, 20(2):305–306, April 1973.

[30] Jacques Morgenstern. The linear complexity of computation. *J. ACM*, 22(2):184–194, April 1975.

[31] T. S. Motzkin. Evaluation of polynomials and evaluation of rational functions. *Bull. Amer. Math. Soc.*, 61:163, 1955.

[32] Noam Nisan and Avi Wigderson. On the complexity of bilinear forms: dedicated to the memory of jacques morgenstern. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 723–732. ACM, 1995.

[33] V. Pan. Methods of computing values of polynomials. *Russian Math Surveys*, 21:105–136, 1966.

[34] Alexey Pospelov. Faster polynomial multiplication via discrete fourier transforms. In *Computer Science–Theory and Applications*, pages 91–104. Springer, 2011.

[35] Pavel Pudlák. A note on the use of determinant for proving lower bounds on the size of linear circuits. *Information processing letters*, 74(5):197–201, 2000.

[36] R. Raz. On the complexity of matrix product. *SIAM J. Comput.*, 32(5):1356–1369 (electronic), 2003.

[37] Ran Raz. On the complexity of matrix product. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 144–151, New York, NY, USA, 2002. ACM.

[38] Ran Raz and Amir Shpilka. Lower bounds for matrix product in bounded depth circuits with arbitrary gates. *SIAM Journal on Computing*, 32(2):488–513, 2003.

[39] A. Schnhage and V. Strassen. Schnelle multiplikation groer zahlen. *Computing*, 7(3-4):281–292, 1971.

[40] Amir Shpilka. Lower bounds for matrix product. *SIAM Journal on Computing*, 32(5):1185–1200, 2003.

[41] M. Spivak. *A Comprehensive Introduction to Differential Geometry*. Publish or Perish, Houston, TX, 1979.

[42] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[43] Leslie Valiant. Graph-theoretic arguments in low-level complexity. *Mathematical Foundations of Computer Science 1977*, pages 162–176, 1977.

[44] V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. STOC, 2012.

[45] Shmuel Winograd. On the number of multiplications necessary to computer certain functions. *Comm. Pure and Applied Math.*, 23:165–179, 1970.

[46] Shmuel Winograd. *Arithmetic Complexity of Computations*. CBMS-NSF regional conference series in applied mathmatics. SIAM, 1987.