The (Non)Enumerability of the Determinant and the Rank

Alina Beygelzimer * Mitsunori Ogihara†

Abstract

We investigate the complexity of enumerative approximation of two fundamental problems in linear algebra, computing the rank and the determinant of a matrix. We show that both are as hard to approximate (in the enumerative sense) as to compute exactly. In particular, if there exists an enumerator that, given a matrix over some finite field, outputs a list of constantly many numbers, one of which is guaranteed to be the rank of the matrix, then it can be determined in AC^0 (with oracle access to the enumerator) which of these numbers is the rank. Thus, for example, if the enumerator is an FL function, then the problem of computing the rank is in FL. For the determinant function we establish the following two results: (1) If the determinant is poly-enumerable in logspace, then it can be computed exactly in FL. (2) For any prime p, if computing the determinant modulo p is $(p-1)$-enumerable in FL (i.e., if one could eliminate a single possible value for the determinant modulo p), then the determinant modulo p can be computed in FL. Because there is a close connection between these two functions and logspace counting classes, we hope that our results can give a better understanding of the power of counting in logspace, and the relationships among the complexity classes sandwiched between NL and uniform TC^1.

1 Introduction

Valiant [Val79b] proved that the permanent of integer matrices characterizes #P, the class of functions definable as the number of accepting computations of a nondeterministic polynomial-time Turing machine. A similar connection has been shown between the complexity of computing the determinant and #L, the logspace analog of #P [Tod91, Val92, Vin91, Dam91]. (Since the determinant of integer matrices can take on negative values, the determinant is in fact complete for GapL, the class of functions that can be expressed as the difference of two #L functions.) Toda's result [Tod89] showing the surprising power of counting in the context of

*Department of Computer Science, University of Rochester, Rochester, NY 14627. Email: beygel@cs.rochester.edu

†Department of Computer Science, University of Rochester, Rochester, NY 14627. Email: ogihara@cs.rochester.edu. Supported in part by grants NSF-DUE-9980943, NIH-RO1-AG18231, NIH-P30-AG18254, and an Alzheimer's Association Grant PIO-1999-1519.

nondeterministic polynomial-time, namely that the polynomial hierarchy is contained in $\mathrm{P}^{\#\mathrm{P}}$, naturally raises the question of whether #P functions are at least easy to approximate. In the standard sense of coming close to the correct value (i.e., within a multiplicative factor), approximating #P functions is in Δ_3^p. Moreover, any technique for showing that it has complexity lower than Δ_2^p, would have to be non-relativizable. (See [Sto85].) In search of a better answer, Cai and Hemachandra [CH89] proposed an alternative notion of approximation, which they called *enumerative counting*. Instead of restricting the range of a function value to an interval, they consider enumerating a short list of (not necessarily consecutive) values, guaranteed to contain the correct one. The number of values on the list depends only on the size of the input; in particular, it does *not* depend on the *value* of the function on that input. Thus, even if the range is exponential (as in the case of the determinant), the enumerator would still output an equally short list for every input of the same length, while the standard approximator would yield exponentially large intervals for some inputs. Which of these approximation tasks is to be preferred certainly depends on the function one is trying to approximate. Enumerative counting makes sense in a more general context when there is no natural ordering on the range of the function, thus restricting the value to an interval is not meaningful. Such an approximation, however, does not satisfy a natural "nesting" requirement: If we parameterize the enumerator by the size of the set it outputs, then a larger approximating set does not necessarily contain a smaller one for the same input, since the only requirement is the containment of the truth. We give a more formal definition of the enumerability below.

A function f is said to be $g(n)$-*enumerable* if there exists a function that, on input x, outputs a list of at most $g(|x|)$ values guaranteed to contain $f(x)$. Cai and Hemachandra [CH91] (see also Amir, Beigel, and Gasarch [ABG90]), showed that if the permanent function is poly-enumerable in polynomial time, then $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. (Their result is actually stated for #SAT, another classical #P-complete function.) Thus if we are given an $n \times n$ integer matrix A, and instead of $n!(2^n)^n$ values that $\mathrm{perm}(A)$ can possibly take, we get a restricted list of polynomially-many values, guaranteed to contain $\mathrm{perm}(A)$, we can easily (in polynomial time) determine which one of these values is correct. This is certainly a *non*-enumerability result, as it says that a very hard to compute function (hard for the whole polynomial hierarchy) can easily bootstrap its exact value if it is left with polynomially many candidates.

In this paper, we investigate the enumerability of functions complete for logspace counting classes. It is interesting whether enumerability implies a similar decrease in the complexity of the determinant; and if so, to what extent? Since #L functions have significantly less computational power (they are contained in TC^1, a subclass of NC^2), the enumerability properties of logspace counting analogs can be very different from those of #P-complete functions. This is all the more interesting, because, as far as we know, there are no results on approximating matrix determinant (or rank) in the standard sense. Another purpose of such an investigation is to get a better understanding of the relationships among the complexity classes sandwiched between NL and uniform TC^1.

We show that if the determinant of integer matrices is poly-enumerable in logspace, then it can be computed exactly in logspace. In other words, it is as hard to enumerate as to compute exactly. A natural question is whether this result can be extended to finite fields. We only establish a similar result for computing the determinant modulo any prime.

We also show that constant enumerability of the rank function over finite fields implies that the rank can be computed exactly in AC^0 (modulo the complexity of the enumerator). Thus, for example, if the enumerator runs in logspace, then the problem of computing the rank is also in logspace. The result actually holds for finite commutative rings with unity.

2 Preliminaries

We will be concerned with the complexity of computing the following functions (with input and output in $\{0,1\}^*$):

- DET_F: Given $A \in F^{n\times n}$, compute the determinant of A.
- RANK_F: Given $A \in F^{n\times n}$, compute the rank of A.
- $\mathrm{SINGULARITY}_F$: Given $A \in F^{n\times n}$, determine whether A is singular.
- $\mathrm{INDEPENDENCE}_F$: Given a set of vectors in F^n, determine whether they are linearly independent.

Here F is any commutative ring with unity. When $F = \mathbb{Z}$, the ring of integers, F is dropped. With some abuse of notation, we will consider these functions as sets by associating the function $f : \{0,1\}^* \to \{0,1\}^*$ with the set $\{(x,i) \mid \text{the } i\text{-th bit of } f(x) \text{ is } 1\}$. Notice that a function is in NC^k if and only if its associated set (its characteristic function) is in NC^k and is polynomially bounded; hence there will be no confusion in viewing circuit classes as functional, or in sets being identified with their characteristic functions.

Uniformity In order to make circuit classes comparable to traditional classes defined by time and space, we need to place uniformity restrictions on circuit families. For our purposes, it will be sufficient to use logspace uniformity, meaning that there exists a logspace machine that, on input 1^n, generates a description of the circuit for n inputs. For a detailed treatment of uniformity and a discussion of other uniformity conditions, see [Ruz81, BI97].

Reductions We use Wilson's model [Wil85] of oracle circuits to define the reductions. A function f is AC^0-*reducible* to a function g, if there is a uniform AC^0 family of circuits that computes f, where in addition to the usual gates, oracle gates for g are allowed. Similarly we define NC^1-*reducibility* except that now the circuits have a bounded fan-in, and thus an oracle gate with fan-in m has to count as depth $\log m$. Let $\mathcal{C}$ be a circuit class. We write $\mathcal{C}(f)$ to denote the class of functions $\mathcal{C}$-reducible to f. For a function class $\mathcal{F}$, $\mathcal{C}(\mathcal{F})$ denotes the class of functions $\mathcal{C}$-reducible to some function in $\mathcal{F}$. A $\mathcal{C}$-*truth-table reduction* of a function f to a function g (written as $f \leq_{\mathrm{tt}}^{\mathcal{C}} g$) is a $\mathcal{C}$-computable procedure for f which can make non-adaptive queries to g (i.e., no query can depend on the answers to previous queries). A bound on the number of queries can be made explicit in the notation as a prefix to tt. A *many-one* reduction is a reduction which makes only one query and outputs the answer to this query as its own answer. For more formal definitions see, for example, [HO02].

A usual way to represent an element of an algebraic structure F in a Boolean circuit is as an integer written in binary. Since we will be mostly interested in matrices over finite rings, or over rings whose size is polynomially related to the size of the matrix, the elements of F will be "short" (i.e., have logarithmic bit length). Thus arithmetic operations over F can be easily computed in AC^0.

Logspace counting classes and the complexity of problems in linear algebra

Many basic linear algebraic problems are known to be in NC^2. In order to classify and capture the exact complexity of these problems, Cook [Coo85] defined the class of problems NC^1-reducible to the determinant of integer matrices, and showed that most linear algebra problems with fast parallel algorithms are in this class. Many are in fact complete for this class; others were shown to be complete for the (potentially smaller) class of problems reducible to computing the rank [vzG93, ST98]. Santha and Tan [ST98] explored the computational difference between the functional and the verification versions of the problems under AC^0-Turing and AC^0-many-one reductions. Allender and Ogihara [AO96] observed that, even though for most natural problems the closures under AC^0 and NC^1-reductions coincide, this does not seem to be apparent for the determinant. This (among other things) motivated the definition of the following hierarchies, defined using the "Ruzzo-Simon-Tompa" oracle access model [RST84], which is standard for defining Turing reductions for space-bounded nondeterministic machines, see [AO96]:

- The exact counting logspace hierarchy $\mathrm{C_=L} \cup \mathrm{C_=L}^{\mathrm{C_=L}} \cup \mathrm{C_=L}^{\mathrm{C_=L}^{\mathrm{C_=L}}} \cdots = \mathrm{AC}^0(\mathrm{C_=L})$

  The class $\mathrm{C_=L}$ is defined as the class of languages, for which there exists a GapL function f such that for every x, x is in the language if and only if $f(x) = 0$. If follows immediately that the set of singular matrices is complete for $\mathrm{C_=L}$. Allender, Beals, and Ogihara [ABO99] showed that AC^0 and NC^1 reducibilities coincide on $\mathrm{C_=L}$; furthermore, the hierarchy collapses to $\mathrm{L}^{\mathrm{C_=L}}$. We show that $O(1)$-enumerability of Rank implies a slightly deeper collapse (to the closure of $\mathrm{C_=L}$ under appropriate reducibility).

- The PL hierarchy $\mathrm{PL} \cup \mathrm{PL}^{\mathrm{PL}} \cup \mathrm{PL}^{\mathrm{PL}^{\mathrm{PL}}} \cup \cdots = \mathrm{AC}^0(\mathrm{PL})$

  Ogihara [Ogi98] showed that the PL hierarchy collapses to PL under AC^0 (in fact TC^0) reductions, which was improved to NC^1 reducibility by Beigel and Fu [BF97]. A problem, easily seen to be complete for PL, is checking whether the determinant of integer matrices is positive.

- The #L hierarchy $\mathrm{L} \cup \mathrm{L}^{\#\mathrm{L}} \cup \mathrm{L}^{\#\mathrm{L}^{\#\mathrm{L}}} \cdots = \mathrm{AC}^0(\#\mathrm{L}) = \mathrm{AC}^0(\mathrm{Det})$

  It is not known whether the #L hierarchy collapses, or whether $\mathrm{AC}^0(\#\mathrm{L}) = \mathrm{NC}^1(\#\mathrm{L})$. The latter would imply the collapse [All97].

Allender et al. [ABO99] showed that the problems of computing the rank of integer matrices, determining whether the rank is odd, and determining the solvability of a system of linear equations, are all complete for $\mathrm{AC}^0(\mathrm{C}_=\mathrm{L})$. Clearly, the problems of computing and verifying the rank of a matrix are AC^0-equivalent (since there are just $n+1$ possibilities for the rank). However, Allender et al. classified the complexity of verifying the rank exactly, showing that it is complete for the second level of the Boolean Hierarchy above $\mathrm{C}_=\mathrm{L}$ (i.e. the class of sets expressible as an intersection of a $\mathrm{C}_=\mathrm{L}$ and a co-$\mathrm{C}_=\mathrm{L}$ set).

Our results We show that constant enumerability of the rank function over finite fields implies that the rank can be computed exactly in AC^0 (modulo the complexity of the enumerator). Thus, if there exists an enumerator that, given a matrix, outputs a list of constantly many numbers, including the rank of this matrix, it can be determined in AC^0 (with oracle gates for the enumerator) which of these numbers is the rank. For example, if the enumerator runs in logspace, then the problem of computing the rank is also in logspace. We also consider a related problem of counting the number of vectors in a given set that are spanned by the other vectors in the set, and show it to be $\leq^{\mathrm{TC}^0}_{O(n)\text{-tt}}$-equivalent to computing the rank. Both results actually hold for commutative rings with identity with the condition that the size of the ring that the input matrix is over is polynomially related to the dimension of the matrix. More formally, we consider a sequence of rings $R_1, R_2, \ldots, R_n, \ldots$, where the size of each R_n is polynomial in n, and our $n \times n$ matrices are over R_n.

The techniques used in the proof give, from the assumption of logspace constant-enumerability of the rank, $\leq^{\log}_{O(n)\text{-tt}}$-reductions from $\textsc{Rank}_F$ to $\textsc{Independence}_F$ and to $\textsc{Singularity}_F$. The reductions hold for arbitrary rings; in particular, for the ring of integers. Since for integer matrices, $\textsc{Independence}$ and $\textsc{Singularity}$ are easily seen to be complete for, respectively, co-$\mathrm{C}_=\mathrm{L}$ and $\mathrm{C}_=\mathrm{L}$, the existence of the enumerator implies that $\mathrm{AC}^0(\mathrm{C}_=\mathrm{L})$ (and thus the entire exact counting logspace hierarchy) coincides with the intersection of $\leq^{\log}_{O(n)\text{-tt}}$-closures of $\mathrm{C}_=\mathrm{L}$ and co-$\mathrm{C}_=\mathrm{L}$, a slight improvement over a $O(\mathrm{poly}(n))$-truth-table reduction that follows from [ABO99] (unconditionally). In fact, constantly many queries suffice for the reduction to $\textsc{Independence}_F$.

For the determinant function $\textsc{Det}$ (over the integers), we prove that if $\textsc{Det}$ is poly-enumerable in logspace, then $\textsc{Det}$ is in logspace. In other words, it is as hard to enumerate as to compute exactly. Similarly, for any prime p, if the determinant-modulo-p function is $(p-1)$-enumerable in logspace, then it can be computed exactly in logspace. Thus, if we could eliminate at least *one* of p possibilities for the determinant-modulo-p in logspace, we would be able to compute the determinant-modulo-p exactly in logspace. It is naturally interesting whether a similar result can be shown for finite fields.

Organization of the paper All the results pertaining to the rank and the determinant are collected in Sections 3 and 4, respectively. Section 5 concludes with a discussion.

3 Enumerability of the Rank

3.1 Computing the rank

Recall that a function f is logspace $g(n)$-enumerable if there exists a logspace computable function that, on input x, outputs a list of at most $g(|x|)$ values, one of which is $f(x)$. The following lemma shows how to combine several matrices into a single matrix such that the ranks of the original matrices can be read off the rank of the combined matrix.

Lemma 3.1 [Block diagonal construction] There exists a logspace computable function S that given an ordered list $\mathcal{Q} = \langle A_1, \ldots, A_q \rangle$ of $n \times n$ matrices, outputs a single matrix $S(\mathcal{Q})$ of dimension $O(n^q)$ such that a logspace procedure can uniquely decode the sequence of ranks $\langle \text{rank}(A_1), \ldots, \text{rank}(A_q) \rangle$ from the value of $\text{rank}(S(\mathcal{Q}))$. Moreover, both procedures can be implemented by uniform AC^0 circuit families.

Proof: Consider the following construction. On input $\mathcal{Q} = \langle A_1, \ldots, A_q \rangle$, S outputs a block diagonal matrix (i.e. a matrix of $n \times n$ blocks sitting on the main diagonal) with the following block structure. The first block of $S(\mathcal{Q})$ corresponds to A_1, the next $(n+1)$ blocks correspond to A_2, and so on, until we get to $n^{q-1} + n^{q-2} + \cdots + 1$ blocks of A_q. The multiplicity of A_i as a block is $\sum_{j=1}^{i} n^{j-1}$, thereby the dimension of $S(\mathcal{Q})$ is $\sum_{i=1}^{q} n^i(q-i+1) = O(n^q)$. The rank of $S(\mathcal{Q})$ is the sum of all block ranks, and since the rank of each block is at most n, the original sequence of ranks $\langle \text{rank}(A_1), \ldots, \text{rank}(A_q) \rangle$ can be read off from the value of $\text{rank}(S(\mathcal{Q}))$. It is easy to see that both the construction and the decoding can be done in uniform AC^0. ∎

The block diagonal construction above allows one to eliminate many candidate rank sequences. For example, if we were to feed each of q matrices to an r-enumerator separately, we would get r^q purported rank sequences, whereas combining the matrices into a single query reduces the number of candidates to r. In order for the dimension of $S(\mathcal{Q})$ to be polynomial in n, the number of matrices, q, has to be constant. A simple information-theoretic argument shows that this is the best possible. Indeed, the dimension of a matrix whose rank can encode $(n+1)^q$ possible rank sequences must be at least $(n+1)^q$. Combining matrices into a single query to an r-enumerator also allows us to *link* matrices in the sense that will be explained shortly. First we need the notion of *linked sequences.*

Definition 1 Two r-element sequences $\{p_1, \ldots, p_r\}$ and $\{q_1, \ldots, q_r\}$ are said to be *linked* if $p_i = p_j$ if and only if $q_i = q_j$, for all $1 \leq i < j \leq r$.

In other words, two sequences are linked if and only if there is a one-to-one correspondence between the set of values in one and the set of values in the other. For example, the sequences $\{1, 2, 2, 3\}$ and $\{2, 3, 3, 4\}$ are linked, while $\{1, 2, 2, 3\}$ and $\{2, 3, 4, 4\}$ are not. The following claim gives an upper bound on the number of sequences guaranteed to contain at least one linked pair.

Claim 3.1 There exists r_0 such that for any $r > r_0$, any set of $(\frac{r}{\ln r})^r$ r-element sequences, contains at least one linked pair.

Proof: The number of r-element sequences sufficient to guarantee the existence of a linked pair is precisely one more than the number of partitions of an r-element set into non-empty subsets. The latter is known as the rth Bell number, B_r. De Bruijn [dB70] gave the asymptotic formula

$$\frac{\ln B_r}{r} = \ln r - \ln\ln r - 1 + \frac{\ln\ln r}{\ln r} + \frac{1}{\ln r} + \frac{1}{2}\left(\frac{\ln\ln r}{\ln r}\right)^2 + O\left(\frac{\ln\ln r}{(\ln r)^2}\right),$$

immediately yielding the claim. Other (less explicit) asymptotic approximations for B_r are known (see, for example, [Lov93, Odl95]). ∎

Let $\kappa(r)$ be the minimum number of r-element sequences guaranteed to contain a linked pair. By Claim 3.1, $\kappa \stackrel{\text{def}}{=} \kappa(r) \leq (\frac{r}{\ln r})^r$. We will say that two matrices M_1 and M_2 are *linked*, relative to a fixed r-enumerator, if there exists a list of $\kappa - 2$ matrices $M_3, \ldots, M_\kappa$ satisfying the following condition: When $M_1, M_2, \ldots, M_\kappa$ are combined using the construction in Lemma 3.1, and the resulting matrix is fed to the enumerator, we get back a list of r values that (when decoded) correspond to κ sequences of r alleged ranks each, one sequence for each of $M_1, M_2, \ldots, M_\kappa$. The r-sequences corresponding to M_1 and M_2 must be linked in the sense of Definition 1. In other words, M_1 and M_2 are linked (as witnessed by $M_3, \ldots, M_\kappa$) if there is a direct correspondence between the values on their claimed sequences of ranks; hence knowing the true rank of one immediately gives the true rank of the other.

Given an $n \times n$ matrix A, whose rank we want to compute, let A_i (for $1 \leq i \leq n$) denote the $n \times n$ matrix with the first i rows of A and 0s elsewhere, so that $A_n = A$. An r-enumerator for the rank function defines the *equivalence graph* of A, a labeled graph on $\{1, \ldots, n\}$ with the set of nodes corresponding to $\mathcal{A} = \{A_1, \cdots, A_n\}$, and an edge between nodes i and j if and only if there is a set of $\kappa - 2$ matrices in $\mathcal{A} - \{A_i, A_j\}$ witnessing that A_i and A_j are linked. (We can assume without loss of generality that the enumerator is deterministic, and that the combining encoding of queries in Lemma 3.1 is symmetric, i.e., produces the same supermatrix for any permutation of the input matrices.) Each edge is labelled by the equivalence, i.e., direct correspondence between the r claimed values for $\text{rank}(A_i)$ and the r claimed values for $\text{rank}(A_j)$, given by the lexicographically smallest κ-tuple linking A_i and A_j.

Notice that by definition every subset of κ nodes in the equivalence graph induces at least one edge. Hence the number of connected components in the equivalence graph is at most $\kappa - 1$. The following proposition shows that every pair of nodes in the same component is connected by a short path (where the length of a path is the number of edges it contains).

Proposition 3.1 *Any pair of nodes in the same connected component of the equivalence graph is connected by a path of length at most $2\kappa - 3$.*

Proof: Suppose that there exist nodes i and j such that the shortest path between i and j is of length at least $2\kappa - 2$. Let $v_1 = i, v_2, \ldots, v_{2\kappa}$ be the first 2κ nodes on this path. Then either the nodes $\{v_{2i+1} \mid 0 \leq i < \kappa\}$ form an independent set of size κ contradicting the fact that every κ-tuple of nodes induces at least one edge, or there is an edge connecting a pair of nodes in this set, in which case the path from i to j can always be shortened, contradicting the assumption that it is the shortest path. ∎

We will use equivalence graphs in the proof of the theorem below. Throughout the section, F denotes a finite commutative ring with unity.

Theorem 1 If, for some integer r, there exists an r-enumerator for $\textsc{Rank}_F$, then $\textsc{Rank}_F$ is computable in AC^0 with oracle gates for the enumerator.

Proof: Given an enumerator and an input matrix A, the equivalence graph on the prefix forms of A is uniquely defined. Recall that the number of equivalence classes (i.e. the number of connected components in the graph) is at most $\kappa - 1$. Consider guessing a set of $\kappa - 1$ representative matrices, which we shall call a *kernel* of A. The fact that the actual number of components may be less than $\kappa - 1$ is not important, since we just need to find a constant number of matrices whose ranks uniquely determine the ranks of all other matrices. There are only $\binom{n}{\kappa-1}$ choices for the kernel, so we have no problem checking them all in parallel. We will follow a single kernel.

We can easily check whether every node in the graph is reachable from the kernel. Recall that we need only check paths of length at most $2\kappa - 3$. The existence of an edge (i, j) can be checked by an AC^0 sub-circuit with a single gate for the enumerator along any path; each path checks whether its subset of $\kappa - 2$ matrices in $\mathcal{A} - \{A_i, A_j\}$ witnesses the equivalence between A_i and A_j. If at least one of the matrices is unreachable, the kernel is incorrect, and can be weeded out. Assume without loss of generality that the kernel we follow is *good*, i.e., every node in the graph is reachable from at least one kernel node. Let $R_1, \ldots, R_\kappa \in \mathcal{A}$ be the kernel matrices, and $\tau_1, \ldots, \tau_\kappa$ be their corresponding ranks. Note that $\tau_1, \ldots, \tau_\kappa$ uniquely define the ranks of all matrices in $\mathcal{A}$. Of course, we do not know $\tau_1, \ldots, \tau_\kappa$. Instead, we will use the block diagonal construction in Lemma 3.1 to pack $R_1, \ldots, R_\kappa$ into a single super-matrix, which we can then feed to the enumerator to get a list of r sequences of ranks, one of which is $\langle \tau_1, \ldots, \tau_\kappa \rangle$. Denote the sequences by $\langle \tau_1^1, \ldots, \tau_\kappa^1 \rangle$, ..., $\langle \tau_1^r, \ldots, \tau_\kappa^r \rangle$.

Notice that, given a kernel, there may be a number of different paths of length at most $2\kappa - 3$ from any fixed $A_i \in \mathcal{A}$ to its closest kernel node R_j. (The closest R_j is itself not necessarily unique; the ties are broken arbitrarily.) Thus each of $\tau_j^1, \ldots, \tau_j^r$ may correspond, via a different chain of links, to a number of different values claiming to be the rank of A_i. Furthermore, not every $\tau_j^1, \ldots, \tau_j^r$ *will* have a chain of links leading to any value claimed for A_i. Since the correspondences of values marking the links on a path from R_j to A_i were given by different calls to the enumerator, not every $\tau_j^1, \ldots, \tau_j^r$ can be traversed back to A_i, as a value in any one of the correspondences is not necessarily present in the correspondence marking the next link. The *correct* sequence $\langle \tau_1, \ldots, \tau_\kappa \rangle$, however, will always result in the correct sequence $\langle \mathrm{rank}(A_1), \ldots, \mathrm{rank}(A_n) \rangle$, i.e., regardless of the path chosen for every A_i, the path from τ_j to $\mathrm{rank}(A_i)$ will not be broken. For consistency, we will use the following convention for each A_i and each claimed value for the rank of R_j: In the case of multiple paths, we will always follow the path from R_j to A_i that yields (via a sequence of correspondences along the path) the smallest claimed value for the rank of A_i. In the case of broken links, we will map each unmatched value to n. Using these conventions, each $\langle \tau_1^i, \ldots, \tau_\kappa^i \rangle$ gives a rank sequence $\langle \upsilon_1^i, \ldots, \upsilon_n^i \rangle$ claimed to be $\langle \mathrm{rank}(A_1), \ldots, \mathrm{rank}(A_n) \rangle$. Incorrect sequences will be weeded out during verification.

Let $U_i = \{1 \le j \le n \mid \upsilon_j^i = \upsilon_{j-1}^i + 1\}$, where we define $\upsilon_0^i = 0$ for all $1 \le i \le r$; thus each U_i claims to index a maximal set of linearly independent rows of A. Of course, we need to check whether the set indexed by U_i is indeed maximal, i.e. whether every remaining row of A is a linear combination of the rows indexed by U_i. As there are only constantly many U_is, testing them in parallel causes no problem. The rank of A is given by the size of the smallest U_i that passes the maximality test. (This number can be found as the corresponding v_n^i.)

Remark 1 Alternatively, we could arrive at sets $V_1, \ldots, V_r$ that supposedly index maximally independent sets of columns of A. The square submatrices indexed by $U_1 \times V_1, \cdots, U_r \times V_r$ all claim to be non-singular. (We can obviously discard all candidate sequences with $|U_i| \neq |V_i|$.) Now instead of verifying the maximality claim, we can check, in parallel, which submatrices are in fact non-singular, and then take the maximum over all that pass the test.

Notice that the discussion above is valid for arbitrary matrices. Thus, if there exists a logspace computable $O(1)$-enumerator for RANK_F, then RANK_F is $\leq^{\log}_{O(1)\text{-tt}}$-reducible to INDEPENDENCE_F (and $\leq^{\log}_{O(n)\text{-tt}}$-reducible to SINGULARITY_F). The reductions do not depend on the field that the matrices are over; in particular, they hold for integer matrices. We now show how to test the maximality of U_is for matrices whose entries are drawn from some finite field F; then we extend the test to finite commutative rings with unity. In fact, the results naturally generalize to the case when we have a sequence of rings $R_1, R_2, \ldots, R_n, \ldots$, and our $n \times n$ matrices are over R_n (i.e., the size of the ring that a matrix is over is polynomially related to the size of the matrix).

Testing maximality: Given row vectors $v_1, \ldots, v_q, w \in F^n$, verify that w is in the subspace spanned by $v_1, \ldots, v_q$.

Let $F = \{a_1, \cdots, a_m\}$ be the ground field. If $v_1, \ldots, v_q$ are linearly independent, w is spanned by $v_1, \ldots, v_q$, and $w \neq 0^n$ (where 0 is the null element of F), there must exist unique coefficients $c_1, \ldots, c_q \in F$ such that $c_1 v_1 + \cdots + c_q v_q + w = 0$. For each i and j, $1 \leq i \leq m$, $1 \leq j \leq q$, define the matrix

$$M_i^j = \begin{pmatrix} v_1 \\ \vdots \\ a_i v_j + w \\ \vdots \\ v_q \end{pmatrix}.$$

If the above conditions hold, then for each j, there is a unique i such that $\text{rank}(M_i^j) = q - 1$; namely, $\text{rank}(M_i^j) = q - 1$ iff $a_i = c_j$; otherwise $\text{rank}(M_i^j) = q$. We have at most nm such matrices, which is polynomial in n, provided that m is polynomial in n. We want to reduce the number of possibilities for the ranks of these matrices to a constant, and we already know how to force the enumerator to do this for us: Recall that there is a constant $\kappa = \kappa(r)$ such that combining any κ matrices into a single query witnesses at least one equivalence relation between a pair of claimed lists of ranks. Furthermore, there are at most $\kappa - 1$ equivalence classes, and thus only polynomially (in nm) many choices of how to partition M_i^js into the equivalence classes. All choices can be verified in parallel, each one giving only a constant number of possible values for the ranks of M_i^js. For each candidate sequence of ranks we collect the coefficients c_is, assuming that this sequence is correct (i.e., that for each j there is a unique

i such that $\mathrm{rank}(M_i^j) = q - 1$, and $\mathrm{rank}(M_l^j) = q$ for all $l \neq i$); then we verify (column-wise in parallel) that the equality $c_1 v_1 + \cdots + c_q v_q + w = 0$ holds. Notice that the maximality test can be run in parallel not only for all matrices $U_1, \cdots, U_r$, but also for *all* rows w claimed to be linear combinations of the rows of each matrix. The number of matrices that we are dealing with is less than rn^2m, so if m is polynomial in n, testing them in parallel causes no problem. It is easy to see that the entire computation can be done in Boolean AC^0 with oracle gates for the enumerator (with inputs and outputs in binary), since we are dealing with fields of polynomial size (and thus elements of logarithmic length).

Now, to extend this method to commutative rings, we just have to test whether, for each $d \in F$, there exist coefficients $c_1, \ldots, c_q \in F$ satisfying $c_1 v_1 + \cdots + c_q v_q + dw = 0$, which can be done in parallel for each possible d. For some i there can be more than one value of j for which $\mathrm{rank}(M_i^j) = q - 1$, but we have only to select the smallest such j. ∎

Corollary 3.1 If there exists a $O(1)$-enumerator for RANK_F that runs in logspace, then RANK_F is computable in logspace.

3.2 Computing the number of dependent rows

Consider the following related problem: Given a set of row vectors $v_1, \ldots, v_n \in F^n$, determine how many of them are in the span of the other vectors. We will call such vectors *dependent*. Define the problem DEP_F: Given a matrix $A \in F^{n \times n}$, compute the number of dependent rows of A, written as $\mathrm{dep}(A)$. As before, F is a finite commutative ring with unity.

Proposition 3.2 If, for some integer r, there exists an r-enumerator for DEP_F, then DEP_F is computable in TC^0 with oracle gates for the enumerator.

Proof: The argument is similar to the one used in the proof of Theorem 1. Namely, given a matrix $A \in F^{n \times n}$, we construct its prefix forms $A_0, \ldots, A_n$ and obtain a list of r candidates for the sequence $\langle \mathrm{dep}(A_1), \ldots, \mathrm{dep}(A_n) \rangle$, using an r-enumerator for DEP_F the same way we used the enumerator for the rank. Denote the candidate sequences by $\langle v_1^1, \ldots, v_n^1 \rangle$, …, $\langle v_1^r, \ldots, v_n^r \rangle$. For each j, $1 \leq j \leq n$, we have $\mathrm{dep}(A_j) = \mathrm{dep}(A_{j-1})$ if and only if row j is independent of rows $1, \ldots, j-1$. Let $R_i = \{1 \leq j \leq n \mid v_j^i = v_{j-1}^i\}$, where we define $v_0^i = 0$ for all $1 \leq i \leq r$; hence R_i indexes the lexicographically smallest basis of the row space of A, claimed by the ith sequence. Our task is to determine which v_n^i is correct. We test, as in the proof of Theorem 1, the maximality of all R_i in parallel, keeping, for each i, a boolean vector $u_i = (u_{ij})$ of length $|R_i|$ with 1s corresponding to the basis vectors whose dependency has been revealed by the test, i.e., for each i, u_i marks those vectors in R_i that were shown to have non-zero coefficients in linear decompositions of the remaining rows of A. We can ignore the bases that do not pass the maximality test. To simplify the notation, suppose that all of them do. Now we just need to test the consistency of each claimed v_n^i with the corresponding calculated dependency vector u_i. For each $1 \leq i \leq r$, let $c_i = v_n^i - (n - |R_i|)$; that is, c_i is the number of dependent vectors in R_i claimed by the ith candidate sequence $\langle v_1^i, \ldots, v_n^i \rangle$. We check the consistency by verifying, for all i in parallel, whether $c_i = \sum_{j=1}^{|R_i|} u_{ij}$ holds. The value of $\mathrm{dep}(A)$ is given by the v_n^i corresponding to the smallest R_i that passes the consistency test. (If such R_i is not unique, each one forms a valid basis of the row space of A; furthermore, since given a basis, every other row of A is uniquely represented as linear combination of the basis vectors, all bases that pass the consistency test must agree on their corresponding claimed value for $\operatorname{dep}(A)$.) To verify the consistency, we simply need to be able to subtract two numbers of value at most n, and to count the number of 1s in an n-bit binary string; both can clearly be done in TC^0. ∎

Proposition 3.3 DEP_F is $\leq^{\mathrm{TC}^0}_{O(n)\text{-tt}}$-equivalent to RANK_F, where F is a finite commutative ring with unity.

Proof: RANK_F is clearly n-tt-reducible to DEP_F in TC^0. Given a matrix $A \in F^{n\times n}$, we ask the DEP_F oracle for $\operatorname{dep}(A_0), \ldots, \operatorname{dep}(A_n)$ and simply count the number of i, $1 \leq i \leq n$, such that $\operatorname{dep}(A_i) = \operatorname{dep}(A_{i-1})$, which immediately gives the rank of A. To show that DEP_F is reducible to RANK_F, we essentially use the reduction in Proposition 3.2. Namely, we query the RANK_F oracle on $A_1, \ldots, A_n$ to obtain a valid basis of A, and then find the coefficients in the linear decompositions of the remaining rows in terms of the basis vectors. We need only count the number of the basis vectors that have a non-zero coefficient in some decomposition, which can be done in TC^0. Denote this number by s. Then the value of $\operatorname{dep}(A)$ is given by $n - \operatorname{rank}(A) + s$. ∎

4 Enumerability of the Determinant

Theorem 2 If, for some k, DET is n^k-enumerable in logspace, then $\mathrm{DET} \in \mathrm{FL}$.

Proof: Let G be the configuration graph of a nondeterministic logspace machine on some input x. Let $\#\mathrm{path}_G(s,t)$ denote the number of directed paths from node s to node t in G. Define $f(G)$ as the function, whose value (written in binary) consists of a sequence of n^2 blocks of length $s = 2n\lceil\log n\rceil$, the $(n(i-1)+j)$th segment corresponding to $\#\mathrm{path}_G(i,j)$, where $1 \leq i,j \leq n$; thus

$$f(G) = \sum_{i,j=1}^{n} 2^{(n(i-1)+j)s} \#\mathrm{path}_G(i,j).$$

It is easy to see that f is a #L function; this can be done by exhibiting an NL machine N whose number of accepting computation paths on G is $f(G)$. The machine N, on input G, nondeterministically guesses a number $p = n(i-1)+j$, $1 \leq p \leq n^2$, after which it guesses q, $1 \leq q \leq 2^{ps}$, followed by a guess of path from node i to node j in G. If the guess is correct, N accepts; otherwise, it rejects. It is easy to see that N is a nondeterministic logspace machine that has the required number of accepting paths. Hence $f \in \mathrm{GapL}$, and a function is in GapL if and only if it is logspace many-one reducible to the determinant. Thus there must exist a logspace function g such that for all G, $f(G) = \mathrm{DET}(g(G))$. We shall use g to transform G into a matrix $M = g(G)$, and then run the n^k-enumerator on M to obtain a list of n^k values, one of which is the determinant of M. Using the equality in the above reduction, we convert this list to a list of n^k candidates for $f(G)$ (each of which, if correct, certifies that the corresponding claimed value for $\det(M)$ is correct). Since we have only n^k candidates for $f(G)$, they can be checked in parallel. Given a purported $f(G)$, we can uniquely read off $\#\text{path}_G(i,j)$ for each pair (i,j), $1 \le i,j \le n$. These values can then be locally checked using the self-reducibility of $\#\text{path}_G$. ∎

A natural question is whether the same theorem holds for finite fields. We can only show a similar result for the determinant of integer matrices modulo some integer p. The problem of computing the determinant mod p is $\le_m^{\log}$-complete for Mod_pL, defined in [BDHM92]. Mod_pL is the class of sets A for which there exists $f \in \#\text{L}$ such that for all x, $x \in A$ iff $f(x) \not\equiv 0 \mod p$. We will also use the notion of membership comparability, due to Ogihara [Ogi95].

Definition 2 A set S is $g(n)$-*membership comparable*, written as $S \in \text{L-mc}(g(n))$, if there exists a function $f \in \text{FL}$ such that for any set of $g(n)$ inputs $x_1, \ldots, x_{g(n)}$, each of length at most n, f excludes one of $2^{g(n)}$ candidates for the characteristic sequence $\chi_S(x_1, \ldots, x_{g(n)})$.

We will also use the predicate version of the problem, namely $\text{Det-mod-}p = \{(A,i) \mid \det(A) \equiv i \text{ (modulo } p)\}$.

Theorem 3 Let p be any prime. If there exists a logspace computable $(p-1)$-enumerator for $\text{Det-mod-}p$, then $\text{Det-mod-}p$ is in logspace.

Proof: We will show that the existence of the enumerator above implies that $\text{Det-mod-}p$ is in $\text{L-mc}(p-1)$. The theorem will then follow from a result by Ogihara and Tantau [OT01].

Since $\text{Det-mod-}p \in \text{Mod}_p\text{L}$, there must exist a #L-function f certifying its membership. Requiring that p is a prime is just a matter of convenience, since in this case, according to Fermat's Theorem, we may assume that f is such that $x \in \text{Det-mod-}p$ iff $f(x) \equiv 1(\text{modulo } p)$, and $x \notin \text{Det-mod-}p$ iff $f(x) \equiv 0(\text{modulo } p)$. Let M_f be a nondeterministic logspace machine certifying that f is in #L.

Let $k = p-1$. Consider a k-tuple of inputs $(x_1, \ldots, x_k)$, each of length at most n. For $1 \le i \le k$, denote the configuration graph of M_f on input x_i by G_i; then $f(x_i)$ counts the number of paths in M_f from the initial configuration to the accepting configuration. (We may assume that there is a single accepting configuration.) Given a directed graph G, let $\#\text{path}_G(s,t)$ denote the number of directed paths from node s to node t in G; then $f(x_i) = \#\text{path}_{G_i}(s_i, t_i)$, where s_i and t_i are the initial and the accepting configurations of G_i, respectively. Consider a graph G consisting of all G_is plus two additional nodes s and t. Besides the edges internal to G_is, add, for each i, a new edge from s to s_i, and from t_i to t. We have $\#\text{path}_G(s,t) = \sum_{i=1}^{k} f(x_i)$.

For any integer p, given a topologically sorted directed graph G, nodes s and t, the problem of counting the number of paths from s to t in G modulo p, is in Mod_pL (in fact, it is $\le_m^{\log}$-complete for Mod_pL). Thus there must exist a logspace computable function that takes (G,s,t) as input and produces a matrix M such that $\det(M) \equiv \#\text{path}_G(s,t)(\text{modulo } p)$. We shall use this reduction to transform our G into such matrix M, and then run the enumerator on M. The enumerator gives $p-1$ candidates for $\text{Det-mod-}p$, thereby excluding one.

Thus we have a list of $p-1$ candidates for $\#\text{path}_G(s,t) \bmod p$. We will show that no subset of $p-1$ candidates can cover all 2^{p-1} possibilities for the characteristic function $\chi(x_1, \ldots, x_{p-1})$ (in $\text{Det-mod-}p$), implying that $\text{Det-mod-}p \in \text{L-mc}(p-1)$. Indeed, suppose that the enumeration list contains q. We have $(\sum_{i=1}^{p-1} f(x_i)) \bmod p = (\sum_{i=1}^{p-1} f(x_i) \bmod p) \bmod p$, and thus

$\#\text{path}_G(s,t) \equiv q(\text{modulo } p)$ iff exactly q out of $x_1, \ldots, x_{p-1}$ are in the set. (Since, for any x, $f(x)$ is congruent to either 0 or 1 modulo p, we have $\sum_{i=1}^{p-1} f(x_i) \equiv q(\text{modulo } p)$.) Therefore, each candidate q covers (i.e. is consistent with) at most $\binom{p-1}{q}$ characteristic sequences; hence any $p-1$ candidates can clearly cover at most $2^{p-1}-1$ sequences, putting DET-mod-$p \in$ L-mc$(p-1)$. To get the membership in logspace, we will use the following theorem by Ogihara [Ogi95].

(Theorem 5.5 in [Ogi95]) Let $p \geq 2$. If $\text{Mod}_p\text{P} \subseteq \text{P-mc}(c \log n)$ for some $c < 1$, then $\text{Mod}_p\text{P} \subseteq \text{P}$.

The theorem carries over to logspace, as observed by Ogihara and Tantau [OT01]. This finishes the proof, since for any constants p and c, L-mc$(p-1) \subseteq$ L-mc$(c \log n)$. ∎

5 Concluding Remarks

A natural question is whether the rank being, say, $O(\log n)$-enumerable implies that the rank is in logspace. As we mentioned, it does not seem possible to combine more than a constant number of queries into a single query to the enumerator. Another improvement would be to show that Proposition 3.2 holds for AC^0 in place of TC^0, the question being whether counting the number of dependent basis vectors can be avoided in this context.

A Discussion on Theorem 1: It is interesting to know whether the theorem can be generalized to arbitrary rings; in particular, to the ring of integers. It follows from Remark 1, that the problem reduces to finding the largest non-singular matrix among r matrices of dimension at most n. Let the matrices be $D_1, \ldots, D_r$ with the corresponding dimensions $\tau_1, \ldots, \tau_r$. Using a lemma by Allender et al. [ABO99] (Corollary 2.3 in their paper), we can in logspace extend every matrix to dimension n without changing its singularity, so that if it was non-singular, it remains so; otherwise its rank becomes one less than full. For notational simplicity, let $D_1, \ldots, D_r$ denote the extended matrices. We can run the enumerator on $D_1, \ldots, D_r$ (packed using the block diagonal construction) to generate r candidate sequences for $\langle \text{rank}(D_1), \ldots, \text{rank}(D_r) \rangle$. View the output of the enumerator as an $r \times r$ Boolean matrix $\mathcal{D} = (d_{ij})$ with $d_{ij} = 1$ iff $\text{rank}(D_j)$ claimed by the i-th sequence is n. We can safely remove all columns that do not contain ones; assume that all do. It is easy to see that our r-enumerator is of no further use. Since the number of matrices is at most the number of candidate sequences that the enumerator has to commit itself to, the enumerator can make each one of its r purported answers consistent with a different choice of a non-singular matrix among $D_1, \ldots, D_r$. Can we find the largest non-singular D_i without the use of the enumerator?

If the rows of $\mathcal{D}$ form a Sperner system (i.e. the rows are incomparable elements of the boolean r-dimentional cube), then $D_1, \ldots, D_r$ can be transformed into a set of r matrices such that exactly one matrix is non-singular; moreover, the non-singular one has the same index as the correct row of $\mathcal{D}$. To construct the jth matrix in the set, we simply take the product of all matrices that have ones in row j. The rank of the product is at most the minimum of the individual ranks; hence the product is not full-rank iff at least one matrix in the set is singular. The (only) non-singular product corresponds to the correct row of $\mathcal{D}$.

In the case when the row set of $\mathcal{D}$ contains a chain, the non-singular row product may not be unique. In this case we can perform the above process iteratively for all levels of the boolean r-dimensional lattice, starting from level r. That is, for every level i from r to 1, we take a subset of rows of $\mathcal{D}$ with exactly i ones, and get a set of less than r matrices with the property that *at most* one matrix in the subset is non-singular. The correct row of $\mathcal{D}$ corresponds to the first non-singular product.

Suppose we are given a set of matrices guaranteed to contain at most one non-singular matrix; and let $\mathcal{C}$ be a functional complexity class. If we were able to single out the only non-singular matrix (or to establish that none is full rank) in $\mathcal{C}$, then the problem of determining the singularity of a matrix (over the same domain) would trivially be in $\mathcal{C}$. Indeed, to determine whether a matrix A is singular, we would just run the above $\mathcal{C}$ procedure on A combined with some singular matrices. If A is non-singular, the procedure will find it; otherwise it will establish the singularity of all, certifying that A is singular. Thus it is unlikely that the scenario outlined in Remark 1 works for arbitrary rings. Since the interesting case is when the enumerator is bounded to run logspace, one could use randomness in constructing parallel queries from $D_1, \ldots, D_p$ (forcing the enumerator to reveal the singularity of some candidates due to its limited computational ability in guessing the random bits used in constructing the queries). However, approximate rank computations over arbitrary rings are complicated by the fact that the rank is not known to be random self-reducible, and the fact that infinite rings lack nice finite field properties simplifying the analysis (such as the existence of samplable distributions invariant under addition or multiplication by a fixed element of the ring).

Acknowledgements

The authors are grateful to anonymous referees for invaluable comments.

References

[ABG90] A. Amir, R. Beigel, and W. Gasarch. Some connections between bounded query classes and nonuniform complexity. In *5th Structure in Complexity Theory Conference*, pages 232–243, 1990.

[ABO99] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8(2):99–126, 1999.

[All97] E. Allender. A clarification concerning the #L hierarchy, October 1997. Available at ftp.cs.rutgers.edu/pub/allender/lh.pdf .

[AO96] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *Theoretical Informatics and Applications*, 30(1):1–21, 1996.

[BDHM92] G. Buntrock, C. Damm, U. Hertrampf, and C. Meinel. Structure and importance of Logspace-MOD class. *Mathematical Systems Theory*, 25(3):223–237, 1992.

[BF97] R. Beigel and B. Fu. Circuits over PP and PL. *Journal of Computer and System Sciences*, 60(2):422–441, 2000.

[BI97] D. Barrington and N. Immerman. Time, hardware, and uniformity. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Restrospective II*, pages 1–22. Springer-Verlag, 1997.

[CH89] J. Cai and L. Hemachandra. Enumerative counting is hard. *Information and Computation*, 82(1):34–44, 1989.

[CH91] J. Cai and L. Hemachandra. A note on enumerative counting. *Information Processing Letters*, 38(4):215–219, 1991.

[Coo85] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[Dam91] C. Damm. $\mathrm{DET} = \mathrm{L}^{\#\mathrm{L}}$? Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universität zu Berlin, 1991.

[dB70] N. G. de Bruijn. *Asymptotic methods in analysis.* North-Holland, Amsterdam, 1970.

[HO02] L. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion.* Springer-Verlag, 2002.

[Lov93] L. Lovász. *Combinatorial problems and exercises.* North-Holland, 2nd edition, 1993.

[Odl95] A. M. Odlyzko. Asymptotic enumeration methods. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics*, volume I, pages 1063–1229. MIT Press, 1995.

[Ogi95] M. Ogihara. Polynomial-time membership comparable sets. *SIAM Journal on Computing*, 24(5):1168–1181, 1995.

[Ogi98] M. Ogihara. The PL hierarchy collapses. *SIAM Journal on Computing*, 27(5):1430–1437, 1998.

[OT01] M. Ogihara and T. Tantau. On the reducibility of sets inside NP to sets with low information content. Manuscript, 2002.

[RST84] W. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28(2):216–230, 1984.

[Ruz81] W. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22(3):365–383, 1981.

[ST98] M. Santha and S. Tan. Verifying the determinant in parallel. *Computational Complexity*, 7(2):128–151, 1998.

[Sto85] L. Stockmeyer. On approximation algorithms for #P. *SIAM Journal on Computing*, 14(4):849–861, 1985.

[Tod89] S. Toda. On the computational power of PP and ⊕P. In *30th IEEE Symposium on Foundations of Computer Science*, pages 514–519, 1989.

[Tod89] S. Toda. PP is as hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991. pages 514–519, 1989.

[Tod91] S. Toda. Counting problems computationally equivalent to computing the determinant. Technical Report CSIM 91-07, Department of Computer Science, University of Electro-Communications, Tokyo, Japan, 1991.

[Val79a] L. Valiant. Completeness classes in algebra. In *11th ACM Symposium on Theory of Computing*, pages 249–261, 1979.

[Val79b] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val92] L. Valiant. Why is boolean complexity theory difficult. In M. Paterson, editor, *Boolean Function Complexity*, London Mathematical Society Lecture Notes Series 169, pages 84–94. Cambridge University Press, 1992.

[Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *6th IEEE Structure in Complexity Theory Conference*, pages 270–284, 1991.

[vzG93] J. von zur Gathen. Parallel linear algebra. In J. Reif, editor, *Synthesis of Parallel Algorithms*, pages 574–615. Morgan Kaufmann, 1993.

[Wil85] C. Wilson. Relatizived circuit complexity. *Journal of Computer and System Sciences*, 31(2):169–181, 1985.