Electronic Colloquium on Computational Complexity, Report No. 16 (2002)

On the Enumerability of the Determinant and the Rank

Alina Beygelzimer [*] Mitsunori Ogihara[†]

January 29, 2002

Abstract

We investigate the complexity of enumerative approximation of two elementary problems in linear algebra, computing the rank and the determinant of a matrix. In particular, we show that if there exists an enumerator that, given a matrix, 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. The result holds for matrices over any commutative ring whose size grows at most polynomially with the size of the matrix. The existence of such an enumerator also implies a slightly stronger collapse of the exact counting logspace hierarchy.

For the determinant function DET we establish the following two results:

1. If DET is poly-enumerable in logspace, then DET $\in$ FL.
2. For any prime p, if DET-mod-p is $(p-1)$-enumerable in $\mathrm{Mod}_p\mathrm{L}$, then DET-mod-$p \in$ FL.

These results give a perspective on the approximability of many elementary linear algebra problems equivalent to computing the rank or the determinant. Due to the close connection between the determinant function and #L, as well as between the rank function and $\mathrm{AC}^0(\mathrm{C}_=\mathrm{L})$, our results might yield a better understanding of the exact power of counting in logspace and the relationships among the complexity classes sandwiched between NL and uniform TC^1.

[*]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.

ISSN 1433-8092

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 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, 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. Which of these approximation tasks is more natural depends on the function one is trying to approximate: enumerative counting is more suitable in cases when there is no natural total ordering on the range of the function; or when the range is either exponentially large (as in the case of the determinant), or some intervals of the range are substantially denser than others, and it is preferred to obtain a fixed number of candidates for every input of the same length, instead of dealing with an interval that, though bounded, may contain vastly different numbers of possible values for different inputs (depending *not* on the input itself, but on the value of the function on this input).

More formally, 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 that is guaranteed to contain $f(x)$. Cai and Hemachandra [CH91], and 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 $\mathrm{perm}(A)$, then we can easily (in polynomial time) determine which one is the correct one. 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.

The present paper investigates the enumerability of functions complete for logspace counting classes. In particular, 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 the NC^2, or even TC^1), the enumerability properties of logspace counting analogs can be very different from those of #P-complete functions. This is even more interesting, because, as far as we know, there are no results on approximating the determinant (or the 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 there exists an enumerator that, given a matrix, 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 runs in logspace, then the problem of computing the rank is in logspace. The result holds for any commutative ring with identity whose size grows polynomially with the size of the matrix. The existence of such an enumerator implies a slightly stronger collapse of the exact counting logspace hierarchy; namely, it shows that the the hierarchy collapses to the closure of its base class $\mathrm{C}_{=}\mathrm{L}$ under $\leq^{\mathrm{AC}^0}_{O(1)\text{-tt}}$ reductions. We also consider a related problem of computing the number of dependent vectors in a given set, and prove it to be $\leq^{\mathrm{AC}^0}_{O(1)\text{-tt}}$-equivalent to computing the rank. For the determinant function we show that if the determinant is poly-enumerable in logspace, then it can be computed exactly in logspace. We also establish a similar result for computing the determinant modulo any prime.

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, sets being identified with their characteristic functions. For a function f, we will also consider its verification version $\mathrm{VER}.f = \{(x,y) \mid f(x) = y\}$.

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 function g, if there is a logspace 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 $\log m$. For a circuit class $\mathcal{C}$, 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 function f is AC^0 *many-one reducible* to a function g (written as $f \leq^{\mathrm{AC}^0}_m g$) is there exists an AC^0 family of circuits $\langle C_n \rangle$ such that for every x of length n, we have $f(x) = g(C_n(x))$.

Logspace counting classes and the complexity of problems in linear algebra

Many basic linear algebra 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] defined a more refined hierarchy of problems that reflects the computational difference between the functional and the verification versions of the problems under AC^0-Turing and AC^0-many-one reductions. Toda [Tod91] gave many examples of graph-theoretic problems that are (under appropriate reductions) equivalent to computing the determinant. (Although some of these problems still have natural matrix interpretations when graphs are identified with their adjacency matrices.)

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 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, if RANK is $O(1)$-enumerable in logspace, then it collapses to the $\leq^{\mathrm{AC}^0}_{O(1)\text{-tt}}$-closure of $\mathrm{C_=L}$.

- 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 $\mathrm{AC}^0(\mathrm{C_=L})$. 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 $\mathrm{C_=L}$ (i.e. the class of sets expressible as an intersection of a $\mathrm{C_=L}$ and a co-$\mathrm{C_=L}$ set).

Our results We show that if there exists a logspace computable $O(1)$-enumerator for RANK, then RANK is $\leq^{\mathrm{AC}^0}_{O(1)\text{-tt}}$-reducible to INDEPENDENCE, and thus to SINGULARITY. (The reduction holds for arbitrary rings.) SINGULARITY is complete for $\mathrm{C}_=\mathrm{L}$, and thus the existence of the enumerator implies that $\mathrm{AC}^0(\mathrm{C}_=\mathrm{L})$ coincides with the closure of $\mathrm{C}_=\mathrm{L}$ under $O(1)$-tt-reductions, a slight improvement over $O(\mathrm{poly}(n))$-tt that follows from [ABO99] (unconditionally). We also show that if RANK_F is $O(1)$-enumerable in logspace, then $\mathrm{RANK}_F \in \mathrm{FL}$, where F is any commutative ring with identity whose size grows at most polynomially with the size of the input matrix. Finally, we consider a related problem of computing the number of dependent vectors in a given set (i.e., vectors involved in some non-trivial linear dependencies with other vectors in the set), and show it to be $\leq^{\mathrm{AC}^0}_{O(1)\text{-tt}}$-equivalent to computing the rank.

For the determinant function, we establish the following two results:

1. If DET is poly-enumerable in logspace, then DET $\in$ FL.
2. For any integer p, if DET-mod-p is $(p-1)$-enumerable in $\mathrm{Mod}_p\mathrm{L}$, then DET-mod-$p \in$ FL.

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

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 $\mathcal{Q} = \langle A_1, \dots, A_q \rangle$ 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 $\langle \mathrm{rank}(A_1), \dots, \mathrm{rank}(A_q) \rangle$ from the value of $\mathrm{rank}(S(\mathcal{Q}))$. Moreover, both procedures can be implemented by uniform AC^0 circuit families.

Proof: Consider the following combining construction. On input $\mathcal{Q} = \langle A_1, \dots, A_q \rangle$, 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 $n^q + n^{q-1} + \cdots + 1$ 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 $\sum_{i=1}^{q} n^{i+1}(q-i+1) = O(n^q)$. 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 $\langle \mathrm{rank}(A_1), \dots, \mathrm{rank}(A_q) \rangle$ can be read off from the value of $\mathrm{rank}(S(\mathcal{Q}))$. It is easy to see that both the construction and the decoding can be done in uniform AC^0. ∎

The combining 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 $(n+1)^q$. Notice that combining matrices into a single query to an r-enumerator allows one to link matrices in the following sense.

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

In other words, two matrices are linked – relative to an enumerator – if there is a direct correspondence between the values on their claimed lists of ranks; hence knowing the rank of one immediately gives the rank of the other.

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 matrices that are guaranteed to contain a linked pair. By Claim 3.1, $\kappa \overset{\text{def}}{=} \kappa(r) \le (\frac{r}{\ln r})^r$. Given an $n \times n$ matrix A, let A_i (for $1 \le i \le n$) 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 $[n] \overset{\text{def}}{=} \{1, \dots, n\}$ 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 $\mathcal{A} - \{A_i, A_j\}$ certifying the equivalence between A_i and A_j (i.e. witnessing that A_i and A_j are linked). (We can assume without loss of generality that the enumerator is deterministic and the combining encoding of queries in Lemma 3.1 is symmetric.) The label of an edge is defined 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 in this case every pair of nodes 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 equivalence graph is connected by a path of length at most* $2\kappa - 3$.

Proof: Suppose there exist nodes i and j such that the shortest path between i and j is of length at least $2\kappa - 2$. Let $v_1, v_2, \dots, v_{2\kappa}$ be the first 2κ nodes on this path. Then either the nodes $\{v_{2i+1} \mid 0 \le i < \kappa\}$ 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.

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 calls to the enumerator. Here F is any commutative ring with identity whose size grows polynomially with the size of the input matrix.

Proof: Given an enumerator and a set of matrices, the equivalence graph 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 the number of equivalence classes, a representative matrix from each equivalence class, and, finally, the ranks of the representatives chosen. Since there are at most $(\kappa-1)\binom{n}{\kappa-1}r^{\kappa-1}$ possibilities total[1], which is polynomial in n when r is constant, we have no problem checking them all in parallel; thus we will concentrate on a single guess.

Once we have guessed the number of equivalence classes and their representatives, we can check whether every node is reachable from at least one representative, and whether the representatives are not reachable from each other. Recall that we only need to check all paths of length at most $2\kappa-3$ from every representative node. If at least one of these conditions is not satisfied, we reject; otherwise we proceed to checking the consistency of ranks, as we do next.

Let $R_1,\ldots,R_k \in \mathcal{A}$ be the representative matrices, and $\tau_1,\ldots,\tau_k$ be the corresponding ranks, where $1 \le k < \kappa$ is the number of equivalence classes. Note that $\tau_1,\ldots,\tau_k$ uniquely define the ranks of all matrices in $\mathcal{A}$. Of course, we do not know $\tau_1,\ldots,\tau_k$. Instead, we will use the block diagonal construction in Lemma 3.1 to pack $R_1,\ldots,R_k$ into a single matrix, which we can then feed to the enumerator to get a list of r sequences of ranks, one of which is $\langle\tau_1,\ldots,\tau_k\rangle$. Denote the sequences by $\langle\tau_1^1,\ldots,\tau_k^1\rangle$, $\ldots$, $\langle\tau_1^r,\ldots,\tau_k^r\rangle$. Each $\langle\tau_1^i,\ldots,\tau_k^i\rangle$ uniquely defines the rank sequence $\langle v_1^i,\ldots,v_n^i\rangle$ claimed to be $\langle\mathrm{rank}(A_1),\ldots,\mathrm{rank}(A_n)\rangle$. Let $U_i = \{1 \le j \le n \mid v_j^i = v_{j-1}^i + 1\}$, where we define $v_0^i = 0$ for all $1 \le i \le r$; thus each of the $U_1,\cdots,U_r$ claims to be a maximal set of linearly independent rows of A. Our goal is to test whether each U_i is indeed maximal, i.e. whether every remaining row of A is a linear combination of the rows in U_i. As there are only constantly many U_i's, 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 have obtained the (alleged) maximally independent sets of columns $V_1,\ldots,V_r$ (using the same procedure as for the rows). 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 test, in parallel, which submatrices are indeed non-singular, and then take the maximum over all that pass the test.

Notice that the discussion above is valid for arbitrary matrices. Now we show how to test the maximality of U_i's for matrices with entries from any commutative ring whose size does not grow more than polynomially with the size of the input matrix.

Testing maximality: Given row vectors $v_1,\ldots,v_q,w \in F^n$, verify that w is in subspace generated 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 dependent on $v_1,\ldots,v_q$, and $w \neq 0^n$ (where 0 is the null element of F), then there must exist unique coefficients

[1]Whenever necessary, we assume that n is sufficiently large.

$c_1, \dots, c_q \in F$ such that $c_1v_1 + \cdots + c_qv_q + w = 0$. For each i and j, $1 \le i \le m$, $1 \le j \le q$, define the matrix

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

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 $qm < nm$ matrices[2], which is polynomial in n, provided that m grows at most polynomially 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 gives 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. for each j there is a unique i such that $\text{rank}(M_i^j) = q-1$, and $\text{rank}(M_l^j) = q$ for all $l \ne i$); then we verify (column-wise in parallel) that the equality $c_1v_1 + \cdots + c_qv_q + w = 0$ holds. Notice that the maximality test can be run in parallel not only for all of $U_1, \cdots, U_r$, but also for *all* rows w claimed to be linear combinations of the rows in U_h (for each h, $1 \le h \le r$). The number of matrices that we are dealing with for each h is less than n^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 AC^0 with oracle gates for the enumerator, since we are dealing with fields of polynomial size (and thus elements of logarithmic length).

Now, to extend this method to commutative rings of polynomial size, we just have to test whether, for each $d \in F$, there exist coefficients $c_1, \dots, c_q \in F$ satisfying $c_1v_1 + \cdots + c_qv_q + dw = 0$, which can be done in parallel for each possible d. For some i there can be more than one value of j for which $\text{rank}(M_i^j) = q-1$, but since the underlying computation is AC^0 we have only to select the smallest such j. ∎

Corollary 3.1 Let F be any commutative ring with identity whose size grows polynomially with the size of the input matrix. If there exists a $O(1)$-enumerable for Rank_F that runs in logspace, then Rank_F is computable in logspace.

Computing the number of dependent rows

Consider the following related problem: Given a set of row vectors $v_1, \dots, v_n \in F^n$, determine how many of them are *dependent*, i.e., are involved in some non-trivial linear dependency with other vectors in the set. Define the problem Dep_F: Given a matrix $A \in F^{n \times n}$, compute the number of dependent rows of A, written as $\text{dep}(A)$.

[2]If $q = n$ (i.e. if the enumerator has claimed that A has full rank), we can verify whether each row is a linear combination of the remaining rows. Since there are just n rows, the verification can be done in parallel. The sequence passes the test if and only if it passes all of the n tests. Thus we may assume that $q < n$.

Proposition 3.2 If, for some integer r, there exists an r-enumerator for $\textsc{Dep}_F$, then $\textsc{Dep}_F$ is computable in TC^0 with oracle calls to the enumerator. Here F is any commutative ring of polynomial size.

Proof: The argument is very 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 $A_0, \dots, A_n$ and obtain a list of r candidates for the sequence $\langle \mathrm{dep}(A_1), \dots, \mathrm{dep}(A_n)\rangle$, using an r-enumerator for $\textsc{Dep}_F$ the same way we used the enumerator for the rank. Denote the candidate sequences by $\langle v_1^1, \dots, v_n^1\rangle, \dots, \langle v_1^r, \dots, v_n^r\rangle$. For each j, $1 \le j \le n$, we have $\mathrm{dep}(A_j) = \mathrm{dep}(A_{j-1})$ if and only if row j is independent of rows $1, \dots, j-1$. Let $R_i = \{1 \le j \le n \mid v_j^i = v_{j-1}^i\}$, where we define $v_0^i = 0$ for all $1 \le i \le r$; hence R_i is 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 $1 \le i \le r$, 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 $\langle v_1^i, \dots, v_n^i\rangle$. 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 $\mathrm{dep}(A)$.) To verify the consistency, we simply need to be able to subtract two n-bit strings and to count the number of 1s in an n-bit string; both can be done in TC^0. ∎

Proposition 3.3 For an arbitrary ring F, $\textsc{Dep}_F$ is $\le_{O(n)\text{-tt}}^{\mathrm{TC}^0}$-equivalent to $\textsc{Rank}_F$.

Proof: $\textsc{Rank}_F$ is clearly n-tt-reducible to $\textsc{Dep}_F$ in TC^0. Given a matrix $A \in F^{n\times n}$, we ask the $\textsc{Dep}_F$ oracle for $\mathrm{dep}(A_0), \dots, \mathrm{dep}(A_n)$ and simply count the number of i, $1 \le i \le n$, such that $\mathrm{dep}(A_i) = \mathrm{dep}(A_{i-1})$, which immediately gives the rank of A. To show that $\textsc{Dep}_F$ is reducible to $\textsc{Rank}_F$, we essentially use the reduction in Proposition 3.2. Namely, we query the $\textsc{Rank}_F$ oracle on $A_1, \dots, A_n$ 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 $\mathrm{dep}(A)$ is given by $n - \mathrm{rank}(A) + s$. ∎

4 Enumerability of the Determinant

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

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 $\#\text{path}_G(i,j)$, where $1 \leq i,j \leq n$; thus

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

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 $f \in$ 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, $f(G) = \text{Det}(g(G))$. 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), $1 \leq i,j \leq n$. 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 $\leq_m^{\log}$-complete for Mod$_p$L, defined in [BDHM92]. Mod$_p$L is the class of sets A for which there exists $f \in$ #L such that for all x, $x \in A$ iff $f(x) \not\equiv 0 \bmod p$. 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 $S \in$ L-mc$(g(n))$, if there exists a function $f \in$ 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 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 Det-mod-p, then Det-mod-p $\in$ FL.

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

Since Det-mod-p $\in$ Mod$_pL$, 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 $x \in$ Det-mod-p iff $f(x) \equiv 1$(modulo p), and $x \notin$ Det-mod-p iff $f(x) \equiv 0$(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 $1 \leq i \leq k$, 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 $\#\text{path}_G(s,t) = \sum_{i=1}^{k} f(s_i, t_i)$.

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 $\leq_m^{\log}$-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 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 DET-mod-p), implying that DET-mod-$p \in$ 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. (Recall that 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$(p-1)$. ∎

5 Concluding Remarks

A natural question is whether the rank being, say, $O(\log n)$-enumerable would imply 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 purported ranks $\tau_1, \ldots, \tau_r$. 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 ith sequence is τ_j. We can safely remove all columns that do not contain ones; for notational simplicity assume that all do. Extend the remaining matrices to dimension n, without changing the singularity. Using a lemma from [ABO99], we can, in logspace, transform every matrix so that if it was non-singular, it remains so; otherwise its rank becomes one less than full. It is easy to see that an r-enumerator is of no further use: queried on any set of matrices constructed from $D_1, \ldots, D_r$, the enumerator can make each one of its r purported rank sequences *consistent* with a different choice of a non-singular matrix among $D_1, \ldots, D_r$ (i.e. each sequence can consistently claim the non-singularity of a particular D_i). Since the number of matrices is at most the number of candidate sequences that the enumerator has to commit itself to, the enumerator cannot be forced to reveal the singularity of any single D_i. 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, we have a problem, since 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 filed properties simplifying the analysis (such as the existence of samplable distributions invariant under addition or multiplication by a fixed element of the ring).

More on Rank versus Independence The proof of Theorem 1 shows that we can use an r-enumerator to reduce the number of candidate rank sequences for $A_1, \ldots, A_n$ to r. Given a list of r purported sequences, consider the $r \times n$ Boolean matrix $H = (h_{ij})$ with $h_{ij} = \text{rank}_i(A_j) - \text{rank}_i(A_{j-1})$, where $\text{rank}_i(\cdot)$ denotes the rank of $\cdot$ claimed by the i-th candidate sequence, and $\text{rank}_i(A_0) = 0$ for all i, $1 \leq i \leq r$. (We can immediately reject all candidate rank sequences whose correponding rows in H contain elements other than 0 or 1.) Suppose you are given a black box for INDEPENDENCE; that is, you can ask whether any h_{ij} is indeed 1, which essentially reveals the correct value for $\text{rank}(A_j) - \text{rank}(A_{j-1})$. Consider the following question: how many (adaptive) oracle calls to INDEPENDENCE one needs to determine which purported sequence of ranks is correct? The number of queries clearly does not have to be more than r, since you can always eleminate at least one sequence with a single query. The question is whether it can it be much smaller.

This problem, viewed in a more general setting, seems to be fundamental. Namely, the rows of H can be treated as truth tables of r-variate boolean functions over $GF(2)$. (With no loss we can keep only unique columns of H; there are at most 2^r of them; we can include some duplicate columns if necessary, to make the number of columns precisely 2^r.) Given oracle access to some unknown function $f : GF(2)^r \to GF(2)$, we are asked to recognize it among the functions explicitely specified by H, using as few oracle calls to f as possible. In other words, the question is how many points one needs to (adaptively) examine in order to reconstruct an unknown function f, given a restricted list of candidates containing f. For example, imagine a situation when you pocess some cryptic program P_f, and you are told that it computes one of your favorite functions. On how many different inputs do you have to run P_f to determine which one of your favorite functions it computes?

The problem also has a natural interpretation in the context of distributed computing. Suppose that you are given a network with r nodes and you are told that exactly one node is faulty. The matrix H encodes all allowed routes: each route corresponds to a column of H, i.e. it goes through the nodes that have value 1 in the column; the ordering of nodes on the path can be arbitrary. You are allowed to send a probe along any routing path to determine whether the faulty node is on this path. The problem is then to choose (adaptively vs. nonadaptively) the smallest subset of tests sufficient to distinguish among all r nodes, thereby singling out the faulty one.

The lower bound on the number of queries for any *deterministic* procedure is $\Omega(\log r)$, given by a simple information-theoretic argument. Notice that if the number of unique columns of H is at least $2\sum_{i=\alpha r}^{r}\binom{r}{i}$ for some constant α, $\frac{1}{2} \leq \alpha < 1$, then a single query reduces the number of candidate sequences (i.e. rows of H) by at least $(1-\alpha)r$. Indeed, we just have to ask any h_{ij}, where j is the index of a column containing at least $(1-\alpha)r$ ones and at least $(1-\alpha)r$ zeros. We have $\sum_{i=\alpha r}^{r}\binom{r}{i} < \frac{\alpha 2^r}{r(2\alpha-1)^2}$, using standard approximation arguments. Denote the matrix resulting after query i by H_i (H_i contains the rows of H that agree with the first i queries), so $H_0 = H$. Let r_i denote the number of rows in H_i. The bound above shows that if the number of unique columns of H_0 is at least $\Omega(2^{r_0}/r_0)$, then a single query leaves us with at most αr_0 candidate rows. In this case, the number of unique columns in H_1 is at least $\frac{\alpha 2^{r+1}}{r(2\alpha-1)^2 2^{(1-\alpha)r}} = \Omega(2^{r_1}/r_1)$; hence the same argument inductively applies: a single query can eliminate at least $(1-\alpha)r_1$ rows, and so on. Therefore, if the condition on the number of unique columns of H is satisfied, $\log_{1/\alpha} r = O(\log r)$ calls to INDEPENDENCE suffice to solve RANK.

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*, 8:99–126, 1999.

[All97] E. Allender. A clarification concerning the #L hierarchy, October 1997. Available at http://www.cs.rutgers.edu/˜allender/.

[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. In *12st IEEE Conference on Computational Complexity*, pages 24–35, 1997.

[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*, 38:215–219, 1991.

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

[Dam91] C. Damm. 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.

[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*, 27:1430–1437, 1998.

[OT01] M. Ogihara and T. Tantau. On the reducibility of sets inside NP to sets with low information content. Preprint, November 2001.

[RST84] W. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28:216–230, 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*, 7:128–151, 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 $\oplus$P. In *30th IEEE Symposium on Foundations of Computer Science*, 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*, 31:169–181, 1985.