Isolation, Matching, and Counting: Uniform and Nonuniform Upper Bounds*

Eric Allender†
Department of Computer Science, Rutgers University
Piscataway, NJ 08855, USA
e-mail: allender@cs.rutgers.edu

Klaus Reinhardt‡
Wilhelm-Schickard Institut für Informatik
Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
e-mail: reinhard@informatik.uni-tuebingen.de

Shiyu Zhou§
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
e-mail: shiyu@cis.upenn.edu

Abstract

We show that the perfect matching problem is in the complexity class SPL (in the nonuniform setting). This provides a better upper bound on the complexity of the matching problem, as well as providing motivation for studying the complexity class SPL.

Using similar techniques, we show that counting *the number of accepting paths of a nondeterministic logspace machine can be done in NL/poly, if the number of paths is small. This clarifies the complexity of the class FewL (defined and studied in [BDHM91, BJLR91]). Using derandomization techniques, we then improve this to show that this counting problem is in NL.*

*The material in this paper appeared in preliminary form in papers in the Proceedings of the IEEE Conference on Computational Complexity, 1998, and in the Proceedings of the Workshop on Randomized Algorithms, Brno, 1998.

†Supported in part by NSF grants CCR-9509603 and CCR-9734918.

‡Supported in part by the DFG Project La 618/3-1 KOMET.

§The work was mainly done while the author was at Bell Laboratories.

Determining if our other theorems hold in the uniform setting remains an important open question, although we provide evidence that they do. More precisely, if there are problems in DSPACE(n) requiring exponential-size circuits, then all of our results hold in the uniform setting.

1 Introduction

In [RA97], two of the authors presented new results concerning NL, UL, and #L. The current paper builds on this earlier work, in an attempt to better understand these complexity classes, as well as some related classes. In the process, we present a new upper bound on some problems related to matchings in graphs.

The perfect matching problem is one of the best-studied graph problems in theoretical computer science. (For definitions, see Section 2.) It is known to have polynomial-time algorithms [Edm65], and it is known to be in RNC [KUW86, MVV87], although at present no deterministic NC algorithm is known. Our new upper bound for matching builds on the RNC algorithm. Before we can explain the nature of our bound, we need some definitions.

In [FFK94], Fenner, Fortnow, and Kurtz defined the complexity class SPP to be $\{A : \chi_A \in \mathrm{GapP}\}$. They also showed that this same class of languages can be defined equivalently as $\{A : \mathrm{GapP}^A = \mathrm{GapP}\}$.

The analogous class SPL (namely, the set: $\{A : \chi_A \in \mathrm{GapL}\}$) has not received very much attention. In this work, we show that SPL can be used to provide a better classification of the complexity of some important and natural problems, whose exact complexity remains unknown. In particular, we show that the following problems are in the non-uniform version of SPL:

- perfect matching (i.e., does a perfect matching exist).
- maximum matching (i.e., constructing a matching of maximum possible size)
- maximum flow with unary weights

All of these problems were previously known to be hard for NL, and were known to be (nonuniformly) reducible to the determinant [KUW86, MVV87].

It was observed in [BGW] that the perfect matching problem is in (nonuniform) $\mathrm{Mod}_m\mathrm{L}$ for every m, and as reported in [ABO], Vinay has pointed out that a similar argument shows that the matching problem is in (nonuniform) co-$\mathrm{C_=L}$. A different argument seems to be necessary to show that the matching problem is itself in (nonuniform) $\mathrm{C_=L}$. Since SPL is contained in $\mathrm{C_=L} \cap$co-$\mathrm{C_=L}$, this follows from our new bound on matching.

Under a natural hypothesis (that DSPACE(n) has problems of "hardness" $2^{\epsilon n}$), all of our results hold in the uniform setting, as well. (See Theorem 5.4.)

Most natural computational problems turn out to be complete for some natural complexity class. The perfect matching problem is one of the conspicuous examples of a natural problem that has, thus far, resisted classification by means of completeness. Our results place the matching problem between NL and SPL.

There are many complexity classes related to counting the number of accepting paths of an NL machine. As examples we mention $\mathrm{L}^{\#\mathrm{L}}$, PL, $\mathrm{C}_=\mathrm{L}$, $\mathrm{Mod}_m\mathrm{L}$, SPL, and NL. We think that existing techniques may suffice to find new relationships among these classes (at least in the nonuniform setting). As a start in this direction, we show that if an NL machine has only a polynomial number of accepting computations, then *counting* the number of accepting paths can be done in NL. First, we show that this holds in the nonuniform setting, and then we derandomize this construction to show that it holds also in the uniform setting.

2 Preliminaries

A *matching* in a graph is a set of edges, such that no two of these edges share a vertex. A matching is *perfect* if every vertex is adjacent to an edge in the matching.

#L (first studied by [AJ93]) is the class of functions of the form $\#acc_M(x) : \Sigma^* \to \mathrm{N}$ (counting the number of accepting computations of an NL machine M on input x). GapL consists of functions that are the difference of two #L functions. Alternatively, GapL is the class of all functions that are logspace many-one reducible to computing the determinant of integer matrices. (See, e.g. [AO96, MV97].)

By analogy with the class GapP [FFK94], one may define a number of language classes by means of GapL functions. We mention in particular the following three complexity classes, of which the first two have been studied previously.

- $\mathrm{PL} = \{A : \exists f \in \mathrm{GapL}, x \in A \Leftrightarrow f(x) > 0\}$ (See, e.g., [Gil77, RST84, BCP83, Ogi98, BF97].)
- $\mathrm{C}_=\mathrm{L} = \{A : \exists f \in \mathrm{GapL}, x \in A \Leftrightarrow f(x) = 0\}$ [AO96, ABO, ST98].
- $\mathrm{SPL} = \{A : \chi_A \in \mathrm{GapL}\}$.

It seems that this is the first time that SPL has been singled out for study. In the remainder of this section, we state some of the basic properties of SPL.

Proposition 2.1 *$\forall m$ $UL \subseteq SPL \subseteq \mathrm{Mod}_m\mathrm{L} \cap \mathrm{C}_=\mathrm{L} \cap \text{co-}\mathrm{C}_=\mathrm{L}$.*

(The second inclusion holds because SPL is easily seen to be closed under complement.)

Proposition 2.2 *$SPL = \{A : \mathrm{GapL}^A = \mathrm{GapL}\}$ (using the Ruzzo-Simon-Tompa notion of space-bounded Turing reducibility for nondeterministic machines [RST84]).*

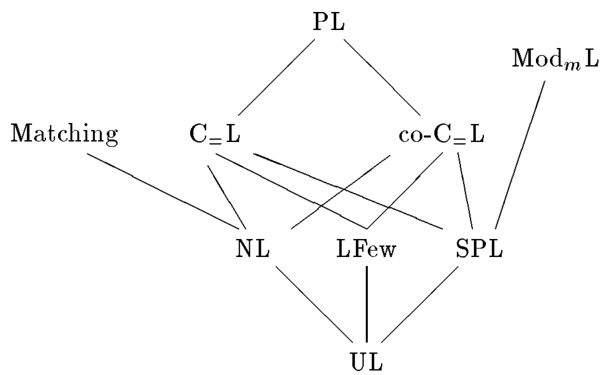

Figure 1: Previously-known inclusions among some logspace-counting problems and classes

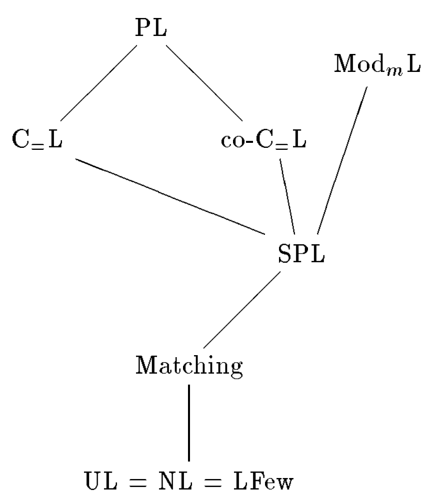

Figure 2: Inclusions established here assuming secure pseudorandom generators. (These inclusions also hold in the nonuniform setting.)

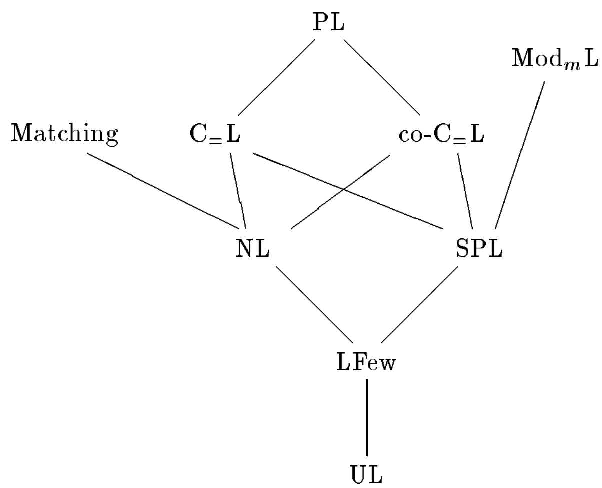

Figure 3: Uniform inclusions among these classes.

(This is proved very similarly to the analogous result in [FFK94]. In showing that $\mathrm{GapL}^A \subseteq \mathrm{GapL}$ if $A \in \mathrm{SPL}$, we need only to observe that in the simulation of an oracle Turing machine given in [FFK94], it is not necessary to guess all of the oracle queries and answers at the start of the computation, but that instead these can be guessed one-by-one as needed.)

Since UL/poly = NL/poly [RA97], it follows that, in the nonuniform setting, NL is contained in SPL. However, it needs to be noted at this point that it is not quite clear what the "nonuniform version of SPL" should be. Here are two natural candidates:

- SPL/poly $= \{A : \exists B \in \mathrm{SPL}\ \exists k \exists (\alpha_n) |\alpha_n| \leq n^k$ and $x \in A \Leftrightarrow (x, \alpha_{|x|}) \in B\}$.
- nonuniform SPL $= \{A : \chi_A \in \mathrm{GapL/poly}\}$.

It is easy to verify that SPL/poly is contained in nonuniform SPL. Containment in the other direction remains an open question. We will use the second class as the nonuniform version of SPL for the following reasons:

- The study of nonuniform complexity classes is motivated by questions of circuit complexity. GapL/poly has a natural definition in terms of skew arithmetic circuits. (See [All97] for a survey and discussion. Skew circuits were defined in [Ven91] and have received study in [Tod92].) Thus a natural definition of SPL is in terms of skew arithmetic circuits over the integers, which produce an output value in {0,1}. If the circuits are nonuniform, then this corresponds to the definition of nonuniform SPL given above.

- We are not able to show that the matching problem is in SPL/poly; we show only that it is in nonuniform SPL. (However, note that Theorem 5.4 shows that, under a plausible complexity-theoretic hypothesis, the matching problem is in uniform SPL.)

In addition to proving new results about the matching problem, we also prove new inclusions for the complexity class LFew, which was originally defined and studied in [BDHM91, BJLR91]. We defer the definition of this class until Section 5.1.1, but we note here that it is immediate from the definitions that LFew is closed under complement, and it was observed in [AO96] that LFew is contained in $C_=L$.

3 Matching

We will find it very helpful to make use of the GapL algorithm of [MV97] for computing the determinant of a matrix. (For our purposes, it is sufficient to consider only matrices with entries in $\{0, 1\}$.) The following definitions are from [MV97]:

> A **clow** (clow for clo-sed w-alk) is a walk $\langle w_1, \ldots, w_l \rangle$ starting from vertex w_1 and ending at the same vertex, where any $\langle w_i, w_{i+1} \rangle$ is an edge in the graph. w_1 is the least numbered vertex in the clow, and is called the **head** of the clow. We also require that the head occurs only once in the clow. This means that there is exactly one incoming edge ($\langle w_l, w_1 \rangle$) and one outgoing edge ($\langle w_1, w_2 \rangle$) at w_1 in the clow.
>
> A **clow sequence** is a sequence of clows $\langle C_1, \ldots, C_k \rangle$ with two properties.
> The sequence is ordered: $\text{head}(C_1) < \text{head}(C_2) < \ldots < \text{head}(C_k)$.
> The total number of edges (counted with multiplicity) adds to exactly n.

The main result of [MV97] is that the determinant of a matrix A is equal to the number of accepting computations of M minus the number of rejecting computations of M, where M is the nondeterministic logspace-bounded Turing machine that, when given a matrix A, tries to guess a clow sequence $C_1, \ldots, C_k$. (If M fails in this task, then M flips a coin and accepts/rejects with probability one-half. Otherwise, M does find a clow sequence $C_1, \ldots, C_k$.) If k is odd, M rejects, and otherwise M accepts.

The crucial insight that makes the construction of [MV97] work correctly is this: If $C_1, \ldots, C_k$ is not a cycle cover (that is, a collection of disjoint cycles covering all of the vertices of M), then there is a corresponding distinct "twin" clow sequence $D_1, \ldots, D_{k\pm 1}$ using exactly the same multiset of edges as that of $C_1, \ldots, C_k$. Note that the parity of the number of clows in this "twin" clow sequence is the opposite of that of $C_1, \ldots, C_k$, and thus their contributions to the count of the number of accepting computations cancel each other. The only clow sequences that survive this cancellation are the cycle covers. Since cycle covers correspond to permutations, this yields exactly the determinant of A.

Here is an algorithm showing that the perfect matching problem is in SPL (nonuniformly). For simplicity, we consider only the bipartite case here. The general case follows as in [MVV87].

First, note that there is a sequence

$$(w_1, w_2, \ldots, w_r)$$

having length $n^{O(1)}$ with the property that, for every bipartite graph G on $2n$ vertices, either G has no perfect matching, or there is some i and some $j \leq n^6$ such that, under weight function w_i, the minimum-weight matching in G is unique and has weight j. (To see this, note that [MVV87] shows that if a weight function is chosen at random, giving each edge a weight in the range $[1, 4n^2]$, then with probability at least $\frac{3}{4}$ there is at most one minimum-weight matching. Now pick a sequence of n^2 such weight functions independently at random. The probability that $(w_1, w_2, \ldots, w_{n^2})$ is "bad" for all G is $\leq (\frac{1}{4})^{n^2} \cdot 2^{n^2} < 1$. Thus some sequence does satisfy the required property.)

Thus there is a function f in GapL/poly with the following properties:

- If G has a perfect matching, then for some i, j, $|f(G, i, j)| = 1$.
- If G has no perfect matching, then for all i, j, $f(G, i, j) = 0$.

To see this, consider the machine that, on input G, i, j, attempts to find a clow sequence in G having weight j under weight function w_i. (The weight function w_i is given as "advice" to the machine.) If there is no perfect matching then for all i, j, the only clow sequences that the machine finds will be cancelled by their "twins", and the value of $f(G, i, j)$ will be zero. If there is a unique perfect matching having weight j, then only one computation path will remain uncanceled (and thus $f(G, i, j)$ will be either 1 or $\Leftrightarrow 1$).

Now consider the function $g(G) = \prod_{i,j}(1 \Leftrightarrow (f(G, i, j))^2)$. This function is in GapL/poly (See, e.g. [AO96]). If G has a perfect matching, then $g(G) = 0$. Otherwise, $g(G) = 1$. This completes the proof of the following theorem.

Theorem 3.1 *The perfect matching problem is in nonuniform SPL.*

3.1 Construction algorithm

So far we have described the decision algorithm for the existence of a perfect matching. As shown in [KUW86], there is a function that *finds* a perfect matching (if it exists) in Random-NC. We will now show that this can be done in SPL. However, first, we must define what it means for a function to be in SPL.

One natural way to define a class of functions computable in SPL is to first consider $\mathrm{FL}^{\mathrm{SPL}}$, which is the set of functions calculated by a logspace machine with an SPL oracle. This class of functions can be defined equivalently as the set of all functions where $|f(x)| = |x|^{O(1)}$ and the language $\{(x, i, b)$: the ith bit of $f(x)$ is $b\}$ is in $\mathrm{L}^{\mathrm{SPL}}$. However, by Proposition 2.2, $\mathrm{L}^{\mathrm{SPL}} = \mathrm{SPL}$, so there is no need to consider logspace-reductions at all (although this turns out to be a convenient way to present the algorithms). An equivalent definition can be formulated in terms of arithmetic circuits, or using NC^1 reductions to SPL. Since all of these definitions are equivalent, we feel justified in denoting this class of functions by FSPL.

In order to build a perfect matching, we will construct an oracle machine that finds an (i,j) such that $|f(G,i,j)| = 1$ (which means that there is a unique matching with minimum-weight j under the weight function w_i). If we can find such an (i,j), then the machine can output all edges e with $|f(G^{-e},i,j)| = 0$, where G^{-e} is the result of deleting e from G. (We know that $|f(G^{-e},i,j)| = 1$ if e does not belong to the perfect matching.) The obvious approach would be to ask the oracle the value of $f(G,i,j)$ for each value of i and j – but the problem is that, for some "bad" values of i and j, the value of f would *not be zero-one valued* and thus the oracle would not be in SPL. The proof consists of avoiding this problem.

Theorem 3.2 *Constructing a perfect matching is in nonuniform FSPL.*

Proof: By analogy to the proof of the previous theorem, note that there is a sequence

$$(w'_1, w'_2, \ldots, w'_{r'})$$

having length $n^{O(1)}$ with the property that, for every $i \leq r$ and $j \leq n^6$ and every bipartite graph G on $2n$ vertices, either G has no perfect matching with weight j under the weight function w_i, or there is some $i' \leq r'$ and some $j' \leq n^6$ such that, among those matchings having weight j under the weight function w_i, under weight function $w'_{i'}$, the minimum-weight matching in G is unique and has weight j'.

(Randomly choose each weight between 0 and $4n^2$ for each of the weight functions $w'_{i'}$. For fixed G,i,j, the probability $p(G,i,j)$ that, among those matchings having weight j under the weight function w_i, under weight function $w'_{i'}$, there is more than one minimum-weight matching in G is upper bounded by the sum over all edges e of the probability of the event $Bad(e)$ that e occurs in one minimum-weight matching but not in another. As shown in [MVV87], given any weight assignment w'_{-e} to the edges in G other than e, there is at most one value for the weight of e that can cause the event $Bad(e)$ occur. Thus the probability $p(G,i,j)$ is at most $\sum_e \sum_{w'_{-e}} \mathrm{Prob}(Bad(e)|w'_{-e})\mathrm{Prob}(w'_{-e}) \leq \sum_e \sum_{w'_{-e}} 1/(4n^2)\mathrm{Prob}(w'_{-e}) = \sum_e 1/(4n^2) \leq 1/4$. For fixed G,i,j, the probability that all $w'_{i'}$ are "bad" is $\leq (1/4)^{r'} = 2^{-2r'}$. The probability that $(w'_1, w'_2, \ldots, w'_{r'})$ is "bad" for all G,i,j is $\leq 2^{-2r'} \cdot 2^{n^2} \cdot r \cdot n^6 < 1$ for $r' = n^2 + \log r + 6\log n$.)

By using a machine that, on input G,i,j,i',j', looks for a clow sequence having weight j under w_i *and simultaneously* having weight j' under $w'_{i'}$, we obtain a function in GapL/poly with the following properties:

- If G has a perfect matching with weight j under the weight function w_i, then for some (i', j'), $|f(G, i, j, i', j')| = 1$.
- If G has no perfect matching with weight j under the weight function w_i, then for all (i', j'), $f(G, i, j, i', j') = 0$.

Here again if there is no perfect matching with weight j under the weight function w_i, then the only clow sequences that the machine finds will be cancelled by their "twins", and the value of $f(G, i, j, i', j')$ will be zero. If there is a unique perfect matching having weight j under w_i and simultaneously j' under $w'_{i'}$, then only one computation path will remain uncanceled (and thus $f(G, i, j, i', j')$ will be either 1 or $\Leftrightarrow 1$).

If G has a perfect matching with weight j under the weight function w_i, then $g(G, i, j) = \prod_{i',j'}(1 \Leftrightarrow (f(G, i, j, i', j'))^2) = 0$. Otherwise, $g(G, i, j) = 1$.

If $g(G, i, j) = 0$, this does not necessarily mean that there is a unique matching with minimum weight j, and thus we still need to check that the set $\{e : g(G^{-e}, i, j) = 1\}$ really is a perfect matching (meaning that each vertex is adjacent to exactly one edge). However, the logspace oracle machine can easily check this condition until a good pair (i, j) is found.

To ensure keeping to the same advice string (consisting of $r(|G|) + r'(|G|)$ weight functions and weights) for all calculations of the oracle answers, the encoding of the oracle question is chosen in a way such that the length of an oracle question stays always the same for a given graph G. ■

By adding an increasing number of vertices having edges to every vertex until a perfect matching is found (and eliminating these vertices afterwards), we get:

Corollary 3.3 *Constructing a maximum matching is in nonuniform FSPL.*

Since by [KUW86], constructing a maximum flow in a graph with unary weights can be reduced to constructing a maximum matching, we get:

Corollary 3.4 *Constructing a maximum flow in a graph with unary weights is in nonuniform FSPL.*

Corollary 3.5 *Deciding the existence of flow $\geq k$ in a graph with unary weights is in nonuniform SPL.*

As Steven Rudich has pointed out (personal communication), a standard reduction shows that this latter problem is in fact equivalent to testing for the existence of a matching of size $\geq k$ in a bipartite graph, under AC^0 many-one reducibility. More precisely, given a bipartite graph $G = (V_1, V_2, E)$ (with $E \subseteq V_1 \times V_2$) one can build a new graph G' by adding two new vertices s (connected to all vertices of V_1) and t (connected to all vertices of V_2); note that G has a matching of size k if and only if G' has a flow of size k. Conversely, given a directed graph $G = (V, E)$ with unary weights on the edges, and with distinguished vertices s and t, build a bipartite graph $G' \subseteq V_1 \times V_2$, where for $i \in \{1, 2\}$, if edge e of G has weight j, then V_i contains vertices $(e, 1, i), \ldots, (e, j, i)$.

Let m_s be the sum of the weights of all edges adjacent to s in G, and let m_t be the sum of the weights of all edges adjacent to t in G. Let m be the maximum of m_s and m_t. V_1 contains vertices $(s,1)\ldots(s,m)$, and V_2 contains vertices $(t,1)\ldots(t,m)$. The vertices $(e,j,1)$ and $(e,j,2)$ are adjacent (for all e and j), and also the vertices $(e,j,2)$ and $(e',j,1)$ are adjacent if e,e' is a path of length two in G. Similarly, there is an edge between (s,j) and $(e,j,2)$ if e is an edge starting at s in G, and there is an edge between (t,j) and $(e,j,1)$ if e is an edge ending at s in G. It is straightforward to verify that G has a flow of size k if and only if G' has a matching of size $k+|E|$. (Similar observations are made in [CSV84].)

4 Machines with Few Accepting Computations

The main result of this section can be stated as follows:

Theorem 4.1 *Let f be in #L. Then the language $\{(x,0^i) : f(x) = i\}$ is in NL/poly.*

In particular, if f is a #L function such that $f(x)$ is bounded by a polynomial in $|x|$, then in the nonuniform setting, computing f is no harder than NL.

Proof: First we use the Isolation Lemma of [MVV87] to show that, if we choose a weight function $w : (V \times V) \to [1..4p(n)^2n^2]$ at random, then with probability $\geq \frac{3}{4}$, any graph with at most $p(n)$ accepting paths will have no two accepting paths with the same weight. To see this, note that this property fails to hold if and only if there exist some i,j and (v,w) such that the i-th accepting path (in lexicographic order) has the same weight as the j-th accepting path, and (v,w) is on the i-th path and not on the j-th path. Call this event $\textsc{Bad}(i,j,v,w)$. Thus it suffices to bound

$$\sum_i \sum_j \sum_v \sum_w \mathrm{Prob}(\textsc{Bad}(i,j,v,w)).$$

Now just as in [MVV87] (or as in our application of the Isolation Lemma in [RA97]), $\mathrm{Prob}(\textsc{Bad}(i,j,v,w))$ is at most $1/(4p(n)^2n^2)$, which completes the proof.

Thus, just as in [RA97], there must exist some sequence $(w_1, w_2, \ldots, w_{n^2})$ of weight functions such that, for all graphs G on n vertices, if G has at most $p(n)$ accepting paths, then there is some i such that, when w_i is used as the weight function, then G will not have two accepting paths with the same weight.

Now it is easy to see that the language $\{(x,0^j) : f(x) \geq j\}$ is in NL/poly. On input x, for each i, for each $t \leq 4p(n)^2n^3$, try to guess an accepting path having weight t using weight function w_i, and remember the number of t's for which such a path can be found. If there is some i for which this number is at least j, then halt and accept.

The theorem now follows by closure of NL/poly under complement [Imm88, Sze88]. ∎

This is also an appropriate place to present two results that improve on a lemma of [BDHM91] in the nonuniform setting. Lemma 12 of [BDHM91] states that, if M is a "weakly unambiguous" logspace machine with $f(x) = \#acc_M(x)$, and g is computable in logspace, then the function $\binom{f(x)}{g(x)}$ is in #L.

(Although we will not need the definition of a "weakly unambiguous machine" here, we note that as a consequence, $f(x)$ is bounded by a polynomial in $|x|$.) Below, we remove the restriction that M be weakly unambiguous, and we relax the restriction on g, allowing g to be any function in #L – but we obtain only a nonuniform result.

Theorem 4.2 *Let f and g be in #L, where $f(x)$ is bounded by a polynomial in $|x|$. Then $\binom{f(x)}{g(x)}$ is in #L/poly.*

Proof: Use Theorem 4.1 to find the number $i = |x|^{O(1)}$ such that $f(x) = i$. If, for all $j \leq i, g(x) \neq j$, then output zero. Otherwise, let $j = g(x)$. It is clear that determining the correct values of i and j can be done in NL/poly. Using the fact that NL/poly = UL/poly [RA97], we may assume that there is a unique path that determines the correct values of i and j. Our #L/poly machine will reject on all the other paths and continue on this unique path to produce $\binom{i}{j}$ accepting paths as follows.

As in the proof of Theorem 4.1, we may assume that our nonuniform advice consists of a sequence of weight functions, and our algorithm can find one of these weight functions such that each of the i paths of the machine realizing $f(x)$ have distinct weights. Our #L/poly machine will pick j of these weights $t_1, \ldots, t_j$ and attempt to guess j paths of $f(x)$ having these weights. This gives a total of $\binom{i}{j}$ accepting paths. ■

The preceding can be improved even to FNL/poly.

Theorem 4.3 *Let f and g be in #L, where $f(x)$ is bounded by a polynomial in $|x|$. Then $\binom{f(x)}{g(x)}$ is in FNL/poly.*

Proof: Compute $i = f(x)$ and $j = g(x)$ as in the preceding proof. Now note that $\binom{i}{j}$ can be computed using a polynomial number of multiplications and one division, and thus has P-uniform NC^1 circuits [BCH86]. The resulting algorithm is NC^1 reducible to NL, and thus is in FNL/poly. ■

(Note that, in contrast to Theorem 4.2, Theorem 4.3 cannot be derandomized using Theorem 5.4, since the construction in [BCH86] does not use a probabilistic argument.)

5 Derandomizing the Constructions

It is natural to wonder if our constructions hold also in the uniform setting. In this section, we show that Theorem 4.1 does hold in the uniform setting, and we present reasons to believe that our other results probably do, too.

5.1 An unconditional derandomization

Theorem 5.1 *Let f be in #L. Then the language $\{(x, 0^i) : f(x) = i\}$ is in NL.*

Proof: First, we show that the language $\{(x, 0^i) : f(x) \geq i\}$ is in NL. In fact, since counting paths in directed acyclic graphs is complete for #L, we will consider only the problem of taking as input $(G, 0^i)$, where G is a directed acyclic graph with distinguished vertices s and t, and determining if there are at least i paths from s to t in G.

On input $(G, 0^i)$, for all prime numbers p in the range $i \leq p \leq n^4$, see if there are at least i numbers $q \leq p$ with the property that there is a path from s to t that is equivalent to q mod p. That is, for each prime p in this range, guess a sequence of numbers $q_1, q_2, \ldots, q_i$, and for each j attempt to find a path in the graph (where a path may be viewed as a sequence of bits) such that this path (again, viewed as a sequence of bits encoding a binary number) is equivalent to q_j mod p.

It is easy to see that the above computation can be done by an NL machine, since logarithmic space is sufficient to compute the residue class mod p of the path. By [FKS82][Lemma 2] (see also [Meh82][Theorem B], if there are at least i distinct paths from s to t, then there is some prime p in this range such that none of the first i paths are equivalent mod p. Thus the nondeterministic logspace algorithm sketched above will accept if and only if there are at least i paths.

Now, since NL is closed under complementation, it follows that an NL machine can determine if there are *exactly* i paths from s to t, which completes the proof. ■

We remark that a more complicated proof of this theorem, using ϵ-biased probability spaces, was presented in an earlier version of this work [AZ98].

5.1.1 The classes LogFew and LogFewNL

Theorem 5.1 has the following consequences. In [BDHM91], the complexity classes LogFewNL and LogFew were defined. In a companion paper at about the same time [BJLR91], the class LogFewNL was called FewUL, and we will follow this latter naming scheme here.

Before we can present the definitions of these classes, we need one more definition from [BDHM91]. An NL machine is *weakly unambiguous* if, for any two accepting computation paths, the halting configurations are distinct.

Definition 1 *[BDHM91, BJLR91]* FewUL *consists of languages accepted by weakly unambiguous logspace-bounded machines.*

LogFew *is the class of languages A for which there exists (a) a logspace-bounded weakly-unambiguous machine M, and (b) a logspace-computable predicate R, such that x is in A if and only if $R(x, \#\mathrm{acc}_M(x))$ is true.*

FewL *consists of languages accepted by NL machines having the property that the number of accepting computations is bounded by a polynomial.*

The definitions in [BDHM91, BJLR91] were made in analogy with the complexity classes FewP and Few ([AR88, CH90]). However, in [BDHM91] the authors considered only the classes FewUL and LogFew (defined in terms of weakly-unambiguous machines), whereas in [BJLR91], the authors defined classes without the restriction to weakly-unambiguous machines, but did not consider LogFew or its analog, which here we will call LFew.

Definition 2 LFew *is the class of languages A for which there exists (a) an NL machine M such that $\#\mathrm{acc}_M(x)$ is bounded by a polynomial, and (b) a logspace-computable predicate R such that x is in A if and only if $R(x, \#\mathrm{acc}_M(x))$ is true.*

It is obvious that UL $\subseteq$ FewUL $\subseteq$ FewL $\cap$ LogFew $\subseteq$ FewL $\subseteq$ NL. FewL and LogFew are obviously both contained in LFew. Thus it is immediate from [RA97] that in the nonuniform setting FewUL and FewL coincide with UL. We conjecture that these classes all coincide in the uniform setting as well, but this remains open. It was shown in [BDHM91] that LogFew is contained in $\mathrm{Mod}_m\mathrm{L}$ for every m. Although [BDHM91] leaves open the relationship between LogFew and NL, Buntrock [Bun98] has pointed out that there is a simple direct argument showing that LogFew is in NL.

It remained open whether LFew is contained in NL. An affirmative answer follows from Theorem 5.1.

Theorem 5.2 *LFew $\subseteq$ NL $\cap$ SPL.*

Proof: Let N be an NL machine accepting a language A in LFew, and let B be the logspace-computable predicate such that $x \in A$ iff $(x, \#acc_N(x)) \in B$. By Theorem 5.1, the language $\{(x, i) : \#acc_N(x) \geq i\}$ is in NL. Thus an NL machine can determine the value of $i = \#acc_N(x)$ exactly, and then check if $(x, i) \in B$. This shows that LFew is in NL.

Let $g(x, i)$ be the #L function that counts the number of accepting computations of the NL machine that, on input x, tries to find at least i paths in the graph G. Note that if G really has exactly i accepting paths, then $g(x, i) = 1$ (since there is exactly one sequence of guesses that will cause the NL machine to find the i paths). Also, if i is larger than the number of paths in G, then $g(x, i) = 0$.

Now consider the function $h(x, i)$ that is defined to be

$$g(x,i) \prod_{i < i' \leq |x|^{O(1)}} (1 \Leftrightarrow g(x, i')).$$

It follows from the standard closure properties of GapL that h is in GapL. (See, e.g. [AO96].)

For the correct value of i, $h(x, i)$ is equal to 1. For all other values of i, $h(x, i)$ is equal to 0.

It now follows easily that any LFew language is in $\mathrm{L}^{\mathrm{SPL}}$, which is equal to SPL. ∎

It is perhaps worth noting that Theorem 5.2 is in some sense the logspace-analog of the inclusion Few $\subseteq$ SPP, which was proved in [KSTT92]. Their proof relies on the fact that, for any #P function f and any polynomial-time function g that is bounded by a polynomial in n, the function $\binom{f(x)}{g(x)}$ is in #P. Note that, in contrast, this closure property is not known to hold for #L or GapL functions (but compare this with Theorem 4.3).

In contrast, we still do not know how to "derandomize" Theorem 4.3.

5.2 A conditional derandomization

Nisan and Wigderson [NW94] defined a notion of "hardness" of languages. A language A has hardness $h(n)$ if there is no circuit family $\{C_n\}$ of size less than $h(n)$ with the property that, for all input lengths n, $C_n(x)$ agrees with $\chi_A(x)$ on more than $(\frac{1}{2} + \frac{1}{2h(n)})2^n$ strings.

The techniques and results of Nisan and Wigderson [NW94], together with some technical material from [IW97][Lemma18], can be used to show that if there is a set K in DSPACE(n) having hardness $2^{\epsilon n}$, then there is a pseudorandom generator g computable in space $\log n$ with the property that no statistical test of size n can distinguish pseudorandom inputs from truly random strings. In this section, we describe how this can be done.

More precisely, we will show that, given the language K as above, then for some constant k (depending on ϵ), there is a function $g : \{0,1\}^{k \log N} \rightarrow \{0,1\}^N$ computable in space $O(\log N)$ with the property that, for all circuits C of size N, the following two probabilities differ by at most $1/N$:

Prob($C(x)$ accepts), where x is a random input of size N.
Prob($C(x)$ accepts), where $x = g(y)$ for a random y of size $k \log N$.

The desired function g is defined as follows. We need that there is a function h computable in space $\log N$ with the following property: $h(N)$ is a binary matrix with N rows and $l = k \log N$ columns, where each row has $m = k' \log N$ 1's (we'll be more specific about the exact values of l and m later on, but clearly $k' < k$), and any two distinct rows (viewed as subsets of $\{1, \ldots, l\}$) intersect in at most $\log N$ points. The construction of a function h meeting these parameters and *computable in logspace* is not explicit in [NW94], but we will use a construction communicated to us by Avi Wigderson [Wig97]. (See also [IW97].) Assume for now that we have the function h.

Here is how to compute g: On input y of length $l = k \log N$, produce a sequence of N output bits, where the ith bit is produced as follows. Let A be the subset of $\{1, \ldots, l\}$ given by the ith row of the matrix $h(N)$. Let z be the string of length m corresponding to the bits of y in the positions in A. Output $K(z)$ as the ith bit of $g(y)$ (where K is the language in DSPACE(n) with hardness $2^{\epsilon n}$).

The argument given in [NW94] shows that, given the constant ϵ in the hardness condition for K, then for all large enough constants k' (where in particular k' must be greater than $1/\epsilon$), then for any k with the property that the desired function h exists, g has the desired pseudorandomness property.

For completeness, we need to specify how to compute the function h. In particular, given any $k' \geq 3$, we show that if we choose $k = 1 + 8k'^2$, then h can be computed in logspace. We shall first present a probabilistic logspace algorithm, and then derandomize it to obtain our deterministic algorithm. We will need a function $\texttt{SET}$: $\mathrm{GF}(2^l) \rightarrow \{A \subseteq \{1, \ldots, l\} : |A| = m\}$ so that each m-set A has a preimage of approximately the same size. A simple way to do this is the following: Fix some standard enumeration of the elements in $\mathrm{GF}(2^l)$. For each $a \in \mathrm{GF}(2^l)$, $\texttt{SET}(a)$ is defined to be the ath item found in cycling through all possible m-sets in some standard order. That is, if we let $\{A_0, ..., A_{q-1}\} = \{A \subseteq \{1, \ldots, l\} : |A| = m\}$, then $\texttt{SET}(a) := A_{(a \bmod q)}$. Thus the preimage of A_i has size $\lceil \frac{2^l}{q} \rceil$ or $\lfloor \frac{2^l}{q} \rfloor$ for every $0 \leq i \leq q \Leftrightarrow 1$. To simplify the subsequent analysis, we shall assume that the preimage of each set A has *exactly* the same size. It is straightforward to modify the proof to handle the necessary approximations.

Here is the probabilistic algorithm. Pick elements a and b from $\mathrm{GF}(2^l)$ uniformly at random. Let $i_1, i_2, \ldots, i_N$ be the N first elements of $\mathrm{GF}(2^l)$ in some standard enumeration. Let S be the set $\{a + i_j * b : 1 \leq j \leq N\}$, and let $S' = \{\texttt{SET}(c) : c \in S\}$. Output the N-by-l matrix whose rows encode the sets in S'.

Claim 5.3 Prob*(no two sets in S' intersect in more than $\log N$ positions)* > 0.

Assume for the moment that the claim is true. Here is our deterministic algorithm to compute h.

By the claim, there must be some choice of the random values a and b for which the probabilistic algorithm produces a good matrix. Thus we can simply cycle through all of the choices for a and b, and check whether for each $1 \leq j_1 < j_2 \leq N$ the sets $\texttt{SET}(a + i_{j_1} * b)$ and $\texttt{SET}(a + i_{j_2} * b)$ intersect in at most $\log N$ places, until a good pair (a, b) is found, and then simulate the probabilistic algorithm using the pair (a, b). Clearly all of this computation can be done in logspace.

It suffices now to provide the proof of our claim.

Proof: For $1 \leq j \leq n$, let r_j be the random variable with value $\texttt{SET}(a + i_j * b)$. As in [CG88][Section 3] (see also [Wig95]), for each pair of m-sets A and B, the events $r_i = A$ and $r_j = B$ are independent (so $\mathrm{Prob}(r_i = A \wedge r_j = B) = \mathrm{Prob}(r_i = A)\mathrm{Prob}(r_j = B)$). Thus

$$\begin{aligned}
& \mathrm{Prob}(|r_i \cap r_j| > \log N) \\
= \; & \textstyle\sum_{A,B} \mathrm{Prob}(|r_i \cap r_j| > \log N \mid r_i = A \wedge r_j = B) \\
& \qquad\qquad\qquad\qquad \mathrm{Prob}(r_i = A \wedge r_j = B) \\
= \; & \textstyle\sum_{A,B} \mathrm{Prob}(|r_i \cap r_j| > \log N \mid r_i = A \wedge r_j = B) \\
& \qquad\qquad\qquad\qquad \mathrm{Prob}(r_i = A)\mathrm{Prob}(r_j = B) \\
= \; & \textstyle\sum_{A,B} \mathrm{Prob}(|A \cap B| > \log N)\mathrm{Prob}(r_i = A)^2 \\
= \; & \mathrm{Prob}_{A,B}(|A \cap B| > \log N)
\end{aligned}$$

where the third equality holds since the events $(r_i = B)$ have uniform distribution, and the fourth equality holds since, for fixed A and B, $\mathrm{Prob}(|A \cap B| >$

$\log N$) is either zero or one.

Assume for the moment that $\text{Prob}_{A,B}(|A \cap B| > \log N) < 1/N^2$. (We show below that this is the case.)

Let C be a random variable counting the number of pairs (i, j) (with $i \neq j$) such that $|r_i \cap r_j| > \log N$. Thus C is the sum of the random variables $C_{i,j}$ taking value 1 if (i, j) is such a pair, and 0 otherwise. Since we are assuming that $\text{Prob}_{A,B}(|A \cap B| > \log N) < 1/N^2$, the expected value of each $C_{i,j}$ is less than $1/N^2$. Thus the expected value of C, which is the sum of the expected values of the variables $C_{i,j}$, is less than 1. (In fact, by choosing appropriate constants k and k', this value can be made much less than 1.)

It suffices now to prove that $\text{Prob}_{A,B}(|A \cap B| > \log N) < 1/N^2$, where A and B are randomly-chosen sets of size m. Since this is equal to

$$\sum_B \text{Prob}_A(|A \cap B| > \log N | B)\text{Prob}(B),$$

it suffices to show that, for each given m-set $B \subseteq \{1, \ldots, l\}$, if we let D denote the probability that a random m-set $A \subseteq \{1, \ldots, l\}$ intersects B in more than $\log N$ positions, then $D < 1/N^2$.

In order to apply the Chernoff bounds, let us consider a different way of picking the set A. For each $i \in \{1, \ldots, l\}$, let i be in A independently with probability m/l. Note that $D = \text{Prob}_A(|A \cap B| > \log N | m = |A|)$, where A is chosen according to this experiment. Thus

$$\begin{array}{rcl} D/(l+1) & \leq & D \cdot \text{Prob}(|A| = m) \\ & < & \sum_i \text{Prob}_A(|A \cap B| > \log N | i = |A|)\text{Prob}(|A| = i) \\ & = & \text{Prob}_A(|A \cap B| > \log N) \end{array}$$

where the first inequality holds because the most likely size (out of $l + 1$ possibilities) for A is m (e.g., see [Fel50][Section VI.3]).

The Chernoff bound can be used to bound this probability, by noting that the expected size of $A \cap B$ is $\mu = m^2/l = (k'^2/k) \log N$. Since we have picked $k = 1 + 8k'^2$, it follows that $\mu < (\log N)/8$. Thus

$$\begin{array}{rcl} D & \leq & (l+1) \cdot \text{Prob}_A(|A \cap B| > 8\mu) \\ & < & \frac{2(l+1)}{e^{(-\ln(e^8/9^9))\mu}} \\ & = & \frac{2(l+1)}{N^{(\log_2 e)(-\ln(e^8/9^9))k'^2/(1+8k'^2)}} \\ & = & O(\frac{\log N}{N^{2.03}}) \end{array}$$

where the second inequality follows by [AS92][Corollary A.14], and the final equality holds since $k' \geq 3$. Thus, for all large N, $D < 1/N^2$, as desired. ■

Theorem 5.4 *If there is a set in DSPACE(n) with hardness $2^{\epsilon n}$ for some $\epsilon > 0$, then the nonuniform constructions in this paper (and in [RA97]) hold also in the uniform setting.*

Proof: We illustrate with Theorem 3.1. The other constructions can be derandomized in a similar manner.

The argument in Theorem 3.1 uses a sequence of weight functions $w_1, \ldots, w_r$ with the property that, for each graph G, (G has a perfect matching) implies (there is some $i \leq r$ and some $j \leq n^6$ such that $|f(G, i, j)| = 1$), where f is the GapL algorithm that uses weight function i, and looks for clow sequences of weight j.

Under the hardness assumption about DSPACE(n), we may use the Nisan-Wigderson pseudorandom generator (as described above), to produce $N = n^{13}$ bits, and interpret these bits as n^{10} weight functions (where each weight function can easily be described using n^3 bits).

Assume, for the sake of a contradiction, that these pseudorandom bits do not produce a correct algorithm. Thus there are infinitely many values n for which there is a graph G_n on n vertices for which the algorithm gives an incorrect answer. This will give us the following statistical test of size N distinguishing pseudorandom input from random input, in contradiction to [NW94]:

Given an input of length $N = n^{13}$, check if at least one of the first n^3 weight functions works correctly for graph G_n. That is, check if there is some $i \leq n^3$ and $j \leq n^6$ such that $|f(G_n, i, j)| = 1$. The computation of each $f(G_n, i, j)$ can be done by doing a determinant calculation, and hence can be done in size $< n^3$. The total number of such tests is n^9. Thus the total size of the circuit is easily bounded by $n^{13} = N$.

By hypothesis, this statistical test will reject all of the pseudorandom strings. However, the analysis of Theorem 3.1 easily can be used to show that truly random strings are accepted with probability greater than 3/4 (and indeed, with probability almost 1). ∎

Recently, it was shown by Klivans and van Melkebeek [KvM99] that the techniques of [IW97] allow for an even weaker assumption than is used in Theorem 5.4.

Theorem 5.5 *[KvM99] If there is a set in DSPACE(n) and an $\delta > 0$ with the property that, for all large n, no circuit of size less than $2^{\delta n}$ accepts exactly the strings of length n in A, then the nonuniform constructions in this paper (and in [RA97]) hold also in the uniform setting.*

Although Klivans and van Melkebeek use the techniques of [IW97], an alternate proof is possible using the framework developed by Sudan, Trevisan, and Vadhan [STV99].

6 Open Problems

Our results sandwich the matching problem between two classes that are closed under complement (NL and SPL). Is the perfect matching problem reducible to its complement?

Is the matching problem in NL? Is it complete for SPL? (Does SPL even have any complete problems?) Is the matching problem complete for some "natural" class between NL and SPL?

As in [MVV87], our techniques apply equally well to both the perfect matching problem and to the bipartite perfect matching problem. What is the true relationship between these two problems? Is the perfect matching problem reducible to the bipartite perfect matching problem?

Can more inclusions be shown among other logspace-counting classes (at least in the nonuniform setting)? Is $C_=L$ contained in $\oplus$L? Is LogCFL contained in $L^{\#L}$?

Is SPL/poly equal to nonuniform SPL? Note that in an analogous way one can define both UL/poly and "nonuniform UL" (where "nonuniform UL is equal to the class of all languages A such that χ_A is in #L/poly). However, since UL/poly $\subseteq$ nonuniform UL $\subseteq$ NL/poly = UL/poly [RA97], it follows that these classes all coincide. No similar argument for SPL is known.

Can some of the other probabilistic inclusions relating to NL and UL be derandomized? Can one show that FewL = UL, or that LFew = UL? Can one show that UL = coUL? It seems that some of these questions should be in reach of current methods.

Acknowledgments

We thank Sunny Daniels, Samir Datta, Russell Impagliazzo, Marcos Kiwi, Dieter van Melkebeek, Michael Saks, D. Sivakumar, Luca Trevisan, and Avi Wigderson for helpful conversations.

References

[ABO] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity.* To appear. A preliminary version appeared in STOC 96.

[AJ93] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[All97] E. Allender. Making computation count: Arithmetic circuits in the nineties. *SIGACT NEWS*, 28(4):2–15, December 1997.

[AO96] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO - Theoretical Information and Application*, 30:1–21, 1996.

[AR88] E. Allender and R. Rubinstein. P-printable sets. *SIAM J. Comput.*, 17:1193–1202, 1988.

[AS92] N. Alon and J. Spencer. *The Probabilistic Method.* John Wiley and Sons, 1992.

[AZ98] E. Allender and S. Zhou. Uniform inclusions in nondeterministic logspace. In R. Freivalds, editor, *Randomized Algorithms*, pages 35–41, 1998. MFCS Satellite Workshop, Brno, Czech Republic.

[BCH86] P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15:994–1003, 1986.

[BCP83] A. Borodin, S. Cook, and N. Pippenger. Parallel Computation for Well-Endowed Rings and Space-Bounded Probabilistic Machines. *Inf. Control*, 58(1–3):113–136, 1983.

[BDHM91] Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf, and Christoph Meinel. Structure and importance of logspace-MOD class. *Math. Systems Theory*, 25:223–237, 1991.

[BF97] R. Beigel and B. Fu. Circuits over PP and PL. In *IEEE Conference on Computational Complexity*, pages 24–35, 1997.

[BGW] L. Babai, A. Gál, and A. Wigderson. Superpolynomial lower bounds for monotone span programs. DIMACS Technical Report 96-37. To appear in *Combinatorica*.

[BJLR91] G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguity and fewness for logarithmic space. In *Proc. 8th International Conference on Fundamentals of Computation Theory (FCT '91)*, volume 529 of *Lecture Notes in Computer Science*, pages 168–179. Springer-Verlag, 1991.

[Bun98] G. Buntrock. Personal communication. 1998.

[CG88] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, April 1988.

[CH90] Jin-Yi Cai and Lane A. Hemachandra. On the power of parity polynomial time. *Mathematical Systems Theory*, 23:95–106, 1990.

[CSV84] A. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM J. Comput.*, 13:423–439, 1984.

[Edm65] J. Edmonds. Matching and a polyhedron with O-1 vertices. *J. Res. Natl. Bur. Stand.*, 69:125–130, 1965.

[Fel50] W. Feller. *An Introduction to Probability Theory and Its Applications.* John Wiley and Sons, 3rd edition, 1950.

[FFK94] Stephen A. Fenner, Lance J. Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

[FKS82] M. Fredman, J. Kómlós, and Endre Szemerédi. Storing a sparse table with $o(1)$ worst case access time. In *23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 165–169, 1982.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM J. Comput.*, 6(4):675–695, December 1977.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

[IW97] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *ACM Symposium on Theory of Computing (STOC)*, pages 220–229, 1997.

[KSTT92] J. Köbler, U. Schöning, S. Toda, and J. Torán. Turing machines with few accepting computations and low sets for PP. *Journal of Computer and System Sciences*, 44:272–286, 1992.

[KUW86] R.M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

[KvM99] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *ACM Symposium on Theory of Computing (STOC)*, pages 659–667, 1999.

[Meh82] K. Mehlhorn. On the program size of perfect and universal hash functions. In *23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 170–175, 1982.

[MV97] M. Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science*, (5), 1997.

[MVV87] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[Ogi98] M. Ogihara. The PL hierarchy collapses. *SIAM J. Comput.*, 27:1430–1437, 1998.

[RA97] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. In *38 th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 244–253, 1997. to appear in SIAM J. Comput.

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

[ST98] M. Santha and S. Tan. Verifying the Determinant. *Computational Complexity*, 7:128–151, 1998.

[STV99] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. In *ACM Symposium on Theory of Computing (STOC)*, pages 537–546, 1999.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Tod92] S. Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Trans. Inf. and Syst.*, E75-D:116–124, 1992.

[Ven91] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43:380–404, 1991.

[Wig95] Avi Wigderson. Lectures on the fusion method and derandomization. Technical Report SOCS-95.2,School of Computer Science, McGill University, 1995.

[Wig97] Avi Wigderson. Personal communication. 1997.