Electronic Colloquium on Computational Complexity, Report No. 19 (1998)

ECCC

Isolation, Matching, and Counting*

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

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 the complexity class LogFew (defined and studied in [BDHM92] coincides with NL in the nonuniform setting. Finally, we also provide evidence that our results also hold in the uniform setting.

1 Introduction

In [RA97], 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.

*A revised version of this work will appear in the Proc. IEEE Conference on Computational Complexity.

†Supported in part by NSF grant CCR-9509603.
‡Supported in part by the DFG Project La 618/3-1 KOMET.

ISSN 1433-8092

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 [ABO97], Vinay has pointed out that a similar argument shows that the matching problem is in (nonuniform) co-$\mathrm{C}_=\mathrm{L}$. A different argument seems to be necessary to show that the matching problem is itself in (nonuniform) $\mathrm{C}_=\mathrm{L}$. Since SPL is contained in $\mathrm{C}_=\mathrm{L} \cap$ co-$\mathrm{C}_=\mathrm{L}$, 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 5.1.)

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 present some some new results concerning #L functions in Section 4, and we use these results to show that the complexity classes LogFewNL and LogFew (introduced in [BDHM92]) coincide with NL in the nonuniform 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 $\#acc_M(x) : \Sigma^* \to \mathbf{N}$ (counting the number of accepting computations of an NL machine

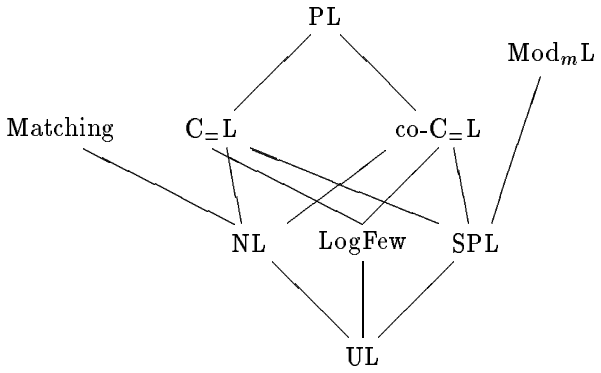

Figure 1: Previously-known inclusions among some logspace-counting problems and classes

M). 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.

- PL $= \{A : \exists f \in \text{GapL}, x \in A \Leftrightarrow f(x) > 0\}$ (See, e.g., [Gil77, RST84, BCP83, Ogi96, BF97].)
- C$_=$L $= \{A : \exists f \in \text{GapL}, x \in A \Leftrightarrow f(x) = 0\}$ [AO96, ABO97, ST94].
- SPL $= \{A : \chi_A \in \text{GapL}\}$.

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 *UL* $\subseteq$ *SPL* $\subseteq$ C$_=$L $\cap$ co-C$_=$L.

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

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

(This is proved very similarly to the analogous result in [FFK94]. In showing that GapL$^A \subseteq$ GapL if $A \in$ SPP, 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.)

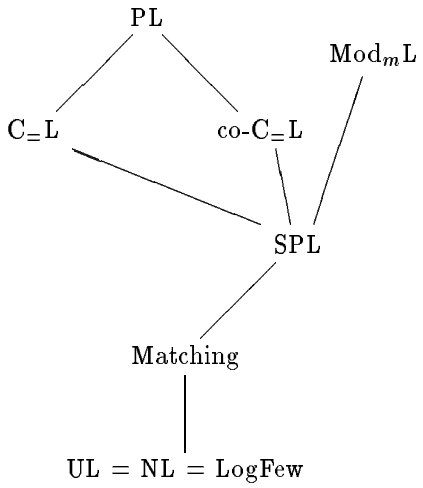

Figure 2: Inclusions established here assuming secure pseudorandom generators. (These inclusions also hold in the nonuniform setting.)

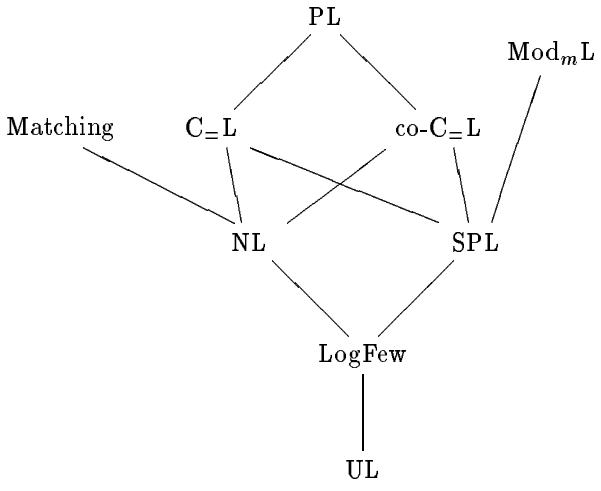

Figure 3: Uniform inclusions among these classes established by derandomizing some of our results [AZ98].

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

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 5.1 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 LogFew, which was originally defined and studied in [BDHM92]. We defer the definition of this class until Section 4, but we note here that it is immediate from the definitions that LogFew is closed under complement, and it was observed in [AO96] that LogFew 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 "twin" clow sequence $D_1, \ldots, D_{k \pm 1}$ using exactly the same multiset of edges. (Note that the "sign" of this "twin" clow sequence is the opposite of that of $C_1, \ldots, C_k$, and thus their contributions 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 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 -1).

Now consider the function $g(G) = \prod_{i,j}(1-(f(G,i,j))^2)$. This function is in GapL/poly. 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 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' 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 fixed G, i, j, the probability of 2 minimal matchings (one of them using e) is $\leq \sum_e 1/(4n^2) \leq 1/4$. 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 -1).

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

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.

Theorem 4.1 has the following consequences. In [BDHM92], the complexity classes LogFewNL and LogFew were defined. (In [BDHM92], these classes were defined using "weakly unambiguous machines"; for our results we do not need this additional complication. Our results hold even in the stronger setting using the definitions as we present them here.) LogFewNL consists of languages accepted by NL machines having the property that the number of accepting computations is bounded by a polynomial. LogFew is also defined in terms of NL machines M such that $\#acc_M(x)$ is bounded by a polynomial; but now there is also a logspace-computable predicate R such that x is in A if and only if $R(x, \#acc_M(x))$ is true. From the definitions, it is immediate that UL $\subseteq$ LogFewNL $\subseteq$ NL, and LogFewNL $\subseteq$ LogFew. Thus it is immediate from [RA97] that in the nonuniform setting LogFewNL and NL coincide with UL. It remained open whether LogFew is contained in NL in the nonuniform setting. An affirmative answer follows from Theorem 4.1.

Corollary 4.2 *LogFew/poly = UL/poly*

Proof: It suffices to show that LogFew is contained in NL/poly. Let $A \in$ LogFew. Let $f \in$ #L, $R \in$ L, and polynomial p be such that $x \in A$ if and only if $R(x, f(x))$, where for all $x, f(x) \leq p(|x|)$. The NL/poly algorithm merely needs to check, for all $i \leq p(|x|)$, if $f(x) = i \wedge R(x, i)$. That this is in NL/poly follows from Theorem 4.1. ■

We remark that it has recently been shown that LogFew is contained in NL $\cap$ SPL also in the *uniform* setting [AZ98]. In contrast, we still do not know how to "derandomize" any of the other constructions in this section.

Proof: (of Theorem 4.1)

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

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

Now just as in [MVV87] (or as in our application of the Isolation Lemma in [RA97]), Prob(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 [BDHM92] in the nonuniform setting. Lemma 12 of [BDHM92] 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 4.3 *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 #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 $j = g(x)$. 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.

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. ■

The preceding can be improved even to FNL/poly.

Theorem 4.4 *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 4.3, Theorem 4.4 cannot be derandomized using Theorem 5.1, 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 present reasons to believe that they probably do.

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 results of Nisan and Wigderson can be used to show that if there is a set A 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 input from truly random strings. That is, there is a function $g : \{0,1\}^{k \log N} \to \{0,1\}^N$ 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 $1/N$:

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

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 is not explicit in [NW94], but we will use a construction communicated to us by Avi Wigderson [Wig97]. 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 $A(z)$ as the ith bit.

For completeness, we need to specify how to compute the function h. We shall present a probabilistic logspace algorithm, and then derandomize it. We will need a function SET: $\text{GF}(2^l) \to \{A \subseteq \{1, \ldots, l\} : |A| = m\}$ so that each m-set A has a preimage of approximately the same size. A simple way to do this is to view a string a of length l as a number, and merely cycle through all possible m-sets in some standard order, until the ath item in this sequence is found. 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 $GF(2^l)$ in some standard enumeration. Let S be the set $\{a + i_j * b : 1 \leq j \leq n\}$, and let $S' = \{\textsc{set}(c) : c \in S\}$. Output the N-by-l matrix whose rows encode the sets in S'.

Claim 1 Prob*(no two sets in S' intersect in more than* $\log N$ *positions)* > 0.

Proof: For $1 \leq j \leq n$, let r_j be the random variable with value $\textsc{set}(a + i_j * b)$. 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

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

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. Since, for randomly chosen m-sets A and B, the expected size of the intersection is $m^2/l = (k'^2/k)\log N$, by use of the Chernoff bounds (see, e.g. [AS92][Corollary A.14]) the probability that this intersection is greater than $\log N$ can be made less than $1/n^2$ by choosing k to be about $4k'^2$.

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. The expected value of C is the sum of the expected values of the variables $C_{i,j}$, and the preceding paragraph shows that the expected value of each $C_{i,j}$ is at most $1/n^2$. Thus the expected value of C is less than 1 (and in fact by appropriate choice of constants can be made much less than 1). This suffices to prove the claim. ■

It remains only to make the probabilistic algorithm deterministic. 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 $\textsc{set}(a + i_{j_1} * b)$ and $\textsc{set}(a + i_{j_2} * b)$ intersect in more than $\log N$ 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.

Theorem 5.1 *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). ■

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 $\mathrm{C_{=}L}$ contained in $\oplus$L? Is LogCFL contained in $\mathrm{L}^{\#\mathrm{L}}$? Can the equality LogFewNL = UL be shown also to hold in the uniform setting?

Acknowledgments

We thank Samir Datta, Marcos Kiwi, Avi Wigderson, and Shiyu Zhou for helpful conversations.

References

[ABO97] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. DIMACS technical report 97-40, submitted for publication, a preliminary version appeared in STOC 96, 1997.

[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.

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

[AZ98] E. Allender and Shiyu Zhou. Uniform inclusions in nondeterministic logspace. Manuscript, 1998.

[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.

[BDHM92] Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf, and Christoph Meinel. Structure and importance of logspace-MOD class. *Math. Systems Theory*, 25:223–237, 1992.

[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.

[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.

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

[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.

[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.

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

[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.

[Ogi96] M. Ogihara. The PL hierarchy collapses. In *ACM Symposium on Theory of Computing (STOC)*, pages 84–88, 1996. to appear in SIAM J. Comput.

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

[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.

[ST94] M. Santha and S. Tan. Verifying the Determinant. In *Proceedings of the 5th Symposium on Algorithms and Computation*, LNCS 834, pages 65–73. Springer-Verlag, 1994.

[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.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'