Theoretical Computer Science 107 (1993) 3–30
Elsevier

A very hard log-space counting class

Carme Àlvarez*
Departament L.S.I., Universitat Politècnica de Catalunya, Pau Gargallo 5, 08028 Barcelona, Spain

Birgit Jenner**
Institut für Informatik, Technische Universität München, Arcisstraße 21, 8000 München 2, Germany

Abstract

Àlvarez, C. and B. Jenner, A very hard log-space counting class, Theoretical Computer Science 107 (1993) 3–30.

We consider the logarithmic-space counting and optimization classes #L, span-L, and opt-L, which are defined analogously to their polynomial-time counterparts. We obtain complete functions for these three classes in terms of graphs and finite automata. We show that #L and opt-L are both included in NC^2, but that, surprisingly, span-L seems to be a much harder class than #L and opt-L. We demonstrate that span-L functions can be computed in polynomial time if and only if P (#P) and all the classes of the polynomial-time hierarchy are included in P. This result follows from the fact that span-L and #P are very similar: span-L$\subseteq$#P, and any function in #P can be represented as the difference of a function in FL and a function in span-L. Nevertheless, the inclusion #P$\subseteq$span-L would imply NL$=$P$=$NP. We, furthermore, investigate restrictions of the classes opt-L and span-L.

1. Introduction

During the past several years the topic of "counting" has appeared in many different settings in complexity theory. In the case of logarithmic space, for example, powerful counting techniques revealed the *intrinsic* computational power of various machine models to count, most notably NL and LOGCFL [4, 14, 27]. On the other hand, counting and optimization was used to *increase* the computational power of polynomial-time machines by defining functional variants of NP, like the function classes opt-P [20], #P [30, 31], and span-P [18, 19, 26]. These classes have been shown to contain interesting functional counting variants of NP-complete problems.

*Work partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

**Research done while visiting the L.S.I. Department of UPC in Barcelona supported by Universität Hamburg and Deutsche Forschungsgemeinschaft (DFG-Je 154/1-1). Work was also partially supported by DFG-SFB 342, subproject A4 "KLARA".

Recently, Toda pointed out the enormous power that such functions can have: the whole polynomial-time hierarchy is included in P(#P) [28].

This raises the question of how general the phenomenon that counting and optimization increases computational power might be. For instance, does it appear for log-space classes as well, and if so, to what extent? Stated alternatively, this question concerns the complexity of functional variants of NL-complete problems. For example, the nonemptiness problem for finite-state automata is NL-complete with respect to log-space many-one reductions, and this holds for both deterministic and nondeterministic automata. Consider the following variants of this problem: (i) computing the number of words accepted by a given deterministic finite-state automaton that are smaller than a given word, i.e. the "ranking function" for the automaton; (ii) computing the ranking function for a nondeterministic finite-state automaton; and (iii) computing the maximal accepted word that is smaller than or equal to a given word, i.e. the "maximal word function" for the automaton. How difficult are these functions to compute? In particular, are they log-space Turing reducible to problems in NL?

We show that these functions are, respectively, many-one complete for the three log-space counting and optimization classes #L, span-L, and opt-L that we define analogously to their polynomial-time counterparts: Functions in #L count the number of accepting computations of a nondeterministic log-space-bounded Turing machine, functions in span-L count the number of different output values of such a machine with additional output tape, and functions in opt-L compute the maximum of all output values. Although it seems at first sight that these three classes have similar computational power, we will show that this is unlikely. span-L turns out to be a very much harder class than both #L and opt-L.

The paper is organized as follows. Section 2 contains all the necessary preliminaries and the definitions of the classes #L, span-L, and opt-L.

In Section 3, we show various functional counting variants of automata and graph problems to be complete for these classes.

Section 4 reveals the difference in the computational power of #L and opt-L on the one hand, and span-L on the other. First, we show that both #L and opt-L are included in NC^2 and, hence, in FP. The inclusion opt-L $\subseteq$ #L, however, would imply that all languages in NL could be accepted unambiguously, with a unique accepting computation. In contrast, span-L seems to be a very hard log-space counting class. In the remaining part of Section 4 we demonstrate that span-L is included in #P, and that any function in #P is metric reducible to a function in span-L. The latter result is obtained by showing that span-L is powerful enough to compute the number of non-satisfying assignments of a Boolean formula. Thus, although the equality span-L = #P is unlikely, since it would imply that NP $\subseteq$ NL, the classes span-L and #P are nevertheless very similar. In particular, they share the ranking function for nondeterministic finite-state automata as a complete function with respect to metric reducibility. The similarity of span-L and #P, furthermore, provides us with some information about the computational power of span-L. Our results yield a new characterization of the class P(#P) as P(span-L), and Toda [28] has shown that P(#P) includes the whole polynomial-time hierarchy (PH). Consequently, the "ranking function" of nondeterministic finite-state automata can be computed in polynomial time if and only if P = NP = PH = P(#P). Similar results about the difficulty of computing the ranking function of languages in various other "small" complexity classes have been published before (see [3, 7, 10,13]). Our result extends this list with an even simpler case and, furthermore, with completeness results.

In Section 5 we investigate restrictions of the classes opt-L and span-L. We show that span-L functions that have values of size logarithmic in the length of their inputs can be computed in NC^2. Such functions are witnessed by NL-transducer whose number of different valid outputs for all inputs is polynomially bounded. Thus, such a restriction significantly reduces the power of span-L functions, which are hard for the polynomial hierarchy as shown in Section 4. We, furthermore, consider opt-L functions with logarithmic value size and compare their computational power with the power of span-L and opt-L functions witnessed by NL-transducer that produce their output deterministically. We show that all of these restrictions yield new characterizations of function classes defined by deterministic log-space Turing machines with oracle in NL and logarithmically bounded number of oracle queries. As one consequence of these characterizations, if functions in opt-L could be computed by NL-machines that write their output deterministically, then L = NL would follow.

2. The log space counting classes #L, span-L, and opt-L

In this section, we define function classes by considering certain counting and optimization operators defined over the computation tree of a nondeterministic logarithmic-space Turing machine with output (NL-transducer) or without output (NL-machine). The computation graph of the machine should not contain cycles, which might lead to infinitely many accepting computations on a single input. This is achieved by imposing additionally a polynomial time bound on the NL-machines by attaching a clock. It is well-known that as far as the corresponding language class NL is concerned the attachment of a polynomial time clock leads to the same class of accepted languages. This is also true for the one-way restriction of this class, 1NL, defined by machines that read their input by moving the input head just from left to right (see e.g. [9]). In the following, we, thus, assume that all log-space-bounded Turing machines (with or without output) are polynomially time-bounded.

We take a configuration of an NL-machine or NL-transducer to consist of the head positions on the input and work tapes, the contents of the work tapes, the value of the clock, the current state, and, in the case of a transducer, a symbol denoting which output symbol, if any, is produced by the current state. To write down a complete configuration on an input of length n, $O(\log n)$ space is clearly sufficient. An NL-transducer has both accepting and rejecting final states, and the output of a computation is only considered to be "valid" if the machine stops in an accepting state. We may assume that initial and final states produce no output. Note that although the number of reachable configurations is bounded by a polynomial, the numbers of accepting computation paths and of valid outputs are not so bounded in general: those numbers can become exponential.

All the sets and functions we consider are defined over the alphabet $\{0,1\}$. The cardinality of a set A is represented by $\|A\|$. The set of natural numbers is denoted by $\mathbf{N}$, and the length of a word x by $|x|$. For words x, y, the notation $x \leqslant y$ means that x is smaller than or equal to y with respect to *lexicographical order*. (Recall that this means that words are ordered according to length, and for a given length, according to alphabetical order.) The ranking function $rank_A : \{0,1\}^* \to \mathbf{N}$ of a set $A \subseteq \{0,1\}^*$ is defined by $rank_A(x) = \|\{w \in A \mid w \leqslant x\}\|$. The census function $cens_A$ of a set A is the function $cens_A : 1^* \to \mathbf{N}$ such that $cens_A(1^n) = \|\{w \in A \mid |w| \leqslant n\}\|$, i.e. $cens_A$ is the restriction of $rank_A$ to 1^*. For a set A, its complement is denoted by A^c; for a language class A (always in roman), the class of complements of all languages in A by co-A. We use the prefix "F" to denote the class of functions (FL, FP) as opposed to the language class (L, P).

In some of our proofs we use the fact that nondeterministic log-space is closed under complementation:

$$\mathrm{NL} = \text{co-NL} \quad [14, 27].$$

Note that the one-way class 1NL is not closed under complementation, i.e. $1\mathrm{NL} \neq \text{co-}1\mathrm{NL}$ [11].

By counting the number of different accepting computations of an NL-machine, we obtain the class #L, the log-space analog of the class #P introduced by Valiant [30, 31].

Definition 2.1. For a machine M, let acc_M denote the function from $\{0,1\}^*$ to $\mathbf{N}$ such that $acc_M(x)$ is the number of accepting computations of M on x. Define

$$\#\mathrm{L} := \{f \mid f = acc_M \text{ for some NL-machine } M\}.$$

The class opt-L is defined in terms of the maximum of all possible valid output values of an NL-transducer. This class is the log-space analog of the function class opt-P introduced by Krentel [20]. (Note that opt-L contains only maximization functions. Most of the results obtained for opt-L hold for the case of minimization functions as well.)

Definition 2.2. For a transducer M, let opt_M denote the function from $\{0,1\}^*$ to $\mathbf{N}$ such that $opt_M(x)$ is the maximum valid output value of M on input x with respect to lexicographical order, or if there is no valid output, then $opt_M(x)$ equals $\perp$. Define

$$\text{opt-L} := \{f \mid f = opt_M \text{ for some NL-transducer } M\}.$$

For consistency with earlier publications [19, 20] the function opt_M is defined as a function to $\mathbf{N}$. We consider the output of our functions to be encoded as a binary string in any natural way.

The class span-L is defined in terms of the number of different valid ouputs that occur in a computation tree of an NL-transducer (the "span" of the tree). The name "span-L" is chosen here to indicate the closeness to the analogous class span-P introduced in [19] (see also [18, 26, 29]; in [29] span-P is named "# · NP").

Definition 2.3. For a transducer M, let $span_M$ denote the function from $\{0,1\}^*$ to $\mathbf{N}$ such that $span_M(x)$ is the number of different valid outputs that occur in the nondeterministic computation tree induced by M on input x, and $span_M(x)=0$ if there are no valid outputs. Define

$$\text{span-L} := \{f \mid f = span_M \text{ for some NL-transducer } M\}.$$

In [19] it was shown that opt-P$\cup \#P \subseteq$span-P. The corresponding statement holds for log-space classes, although here the outputs of the transducers can be exponentially longer than the size of the work tapes.

Proposition 2.4. opt-L$\cup \#$L$\subseteq$span-L.

Proof. For the inclusion $\#$L$\subseteq$span-L, let M be an NL-machine witnessing a function f in $\#$L. We can construct an NL-transducer M' that simulates M and outputs the computation path. Then, clearly, $f = acc_M = span_{M'}$.

For the inclusion opt-L$\subseteq$span-L, let $f = opt_M$ for some NL-transducer M. We can construct a new transducer M' that on input x simulates M and for every output y of M guesses a value $z \leqslant y$, and outputs z. Since neither y nor z can be stored on the work tapes, M' already starts guessing the output symbols of z before the complete output y of the computation has been produced. Therefore, M' first guesses $|y|$; this value can be stored on the worktapes. M' then outputs all values z of length $|y|-1$, and, according to the symbols of y that are produced by M, M' outputs all values z of length $|y|$ that are smaller or equal to y. M' accepts if M accepts and the guessed length of y was correct. M' then will have as many different valid outputs as the maximum of the output values of M. □

Note that in the polynomial-time case, opt-P$\subseteq$span-P is known to hold for maximization functions, but is unknown for minimization functions (see [18]). Although NL is known to be closed under complementation, the same situation is given for logarithmic space: the inclusion opt-L$\subseteq$span-L might not hold for minimization functions.

With the inclusion span-L$\subseteq \#$P, shown in Section 4 (Theorem 4.5), we get the inclusion diagram given in Fig. 1.

$$\begin{tikzcd}
 & \text{span-P} \arrow[dl, dash] \arrow[d, dash] \\
\text{opt-P} & \#\text{P} \arrow[d, dash] \\
 & \text{span-L} \arrow[dl, dash] \arrow[d, dash] \\
\text{opt-L} & \#\text{L}
\end{tikzcd}$$

Fig. 1. Inclusion relations among log-space and polynomial-time counting and optimization classes.

It is easily verified that all of the three classes $\#$L, span-L, and opt-L contain the class FL of functions computable with deterministic logarithmic space, and that they are closed under *log-space functional many-one reducibility*, defined as follows. Let $f, g\colon \{0,1\}^* \rightarrow \mathbb{N}$. A log-space functional many-one reduction from f to g is a function $h\colon \{0,1\}^* \rightarrow \{0,1\}^*$ with $h \in \mathrm{FL}$, such that for all $x \in \{0,1\}^*$ we have $f(x) = g(h(x))$. The class of functions that are log-space many-one reducible to functions in F is denoted by $\mathrm{FL_m}(\mathrm{F})$.

In Sections 4 and 5 we also consider a more powerful reducibility between functions, which we call *log-space metric reducibility* following Krentel [20]. Such reductions are computed by deterministic log-space Turing transducers (L-transducers) that may ask *at most one* query to an oracle function. The L-transducer have an additional unbounded oracle tape on which the query can be written one-way, and, after querying, the query answer can be read two-way. The class of functions that are log-space metric reducible to functions in F is denoted by $\mathrm{FL_1}(\mathrm{F})$. Note that $f \in \mathrm{FL_1}(g)$ for a function g if and only if there exists a function $h \in \mathrm{FL}$ such that for all x: $f(x) = h(x, g(x))$.

Without any bound on the number of queries for the transducer and the requirement that each query answer is erased before a new query is being constructed, we obtain (*nonadaptive*) *log-space functional Turing reducibility* $\mathrm{FL}(\cdot)$. This concept was studied in [1].

The polynomial-time counterparts $\mathrm{FP_m}(\cdot)$, $\mathrm{FP_1}(\cdot)$, and $\mathrm{FP}(\cdot)$ are obtained by considering polynomially time-bounded machines instead of log-space-bounded machines. The $\mathrm{FP_1}(\cdot)$-reducibility is equivalent to the metric reducibility defined in [20].

We denote by $\mathrm{P}(\cdot)$ the closure under deterministic polynomial-time Turing reducibility, and by $\mathrm{L}(\cdot)$ the closure under logarithmic-space Turing reducibility. $\mathrm{FP}(\cdot)$ and $\mathrm{FL}(\cdot)$ denote the corresponding function classes. If the number of queries is bounded by $O(\log n)$ for any input of length n, we obtain $\mathrm{P_{log}}(\cdot)$, $\mathrm{FP_{log}}(\cdot)$, and $\mathrm{L_{log}}(\cdot)$, $\mathrm{FL_{log}}(\cdot)$.

3. Complete functions

In this section, we present complete functions for the classes $\#$L, span-L, and opt-L that are "counting and optimization variants" of NL-complete automata and graph

Table 1
Complete functions for log-space counting and optimization classes

Function	DFA	NFA
Ranking function	#L-complete	span-L-complete
Census function	#L-complete	span-L-complete
Maximal relative word function	opt-L-complete	opt-L-complete
Maximal word function	$\in$FL(NL)	opt-L-complete

problems. It is known that the nonemptiness problem $L(M)\overset{?}{=}\emptyset$, where M is a deterministic (DFA) or nondeterministic (NFA) finite-state automaton, is NL-complete with respect to log-space many-one reductions [16, 17]. Consider the following functional versions of this problem:

- the *census function*: "Given an automaton M and 1^n, how many words of length up to n are accepted by M?",

or its generalization

- the *ranking function*: "Given an automaton M and x, how many words lexicographically smaller than or equal to x are accepted by M?",

and

- the *maximal word function*: "Given an automaton M and x, what is the lexicographically greatest word smaller than or equal to x accepted by M?",

or its generalization

- the *maximal relative word function*: "Given an automaton M with $L(M)\subseteq\Sigma^*$, a subalphabet $\Sigma'\subseteq\Sigma$, and $x\in\Sigma'^*$, what is the lexicographically greatest word $h(w)\leqslant x$ with $w\in L(M)$, where $h:\Sigma\to\Sigma'\cup\{\lambda\}$ is the homomorphism $h(a):=a$, if $a\in\Sigma'$, and $h(a)=\lambda$, otherwise?".

To make the meaning of the maximal relative word function clearer, we state an example: Let $L(M)\subseteq\{0,1,b,c\}^*$ be the finite set $\{c1b0c11bbc0, b11c1, 0001, 0cc1b0c1\}$. Let $\Sigma'=\{0,1\}$, and let h be defined as above. Then the (lexicographically) greatest word $h(w)$ less than or equal to $x=0111$ is 0101.

In the rest of this section we will develop the completeness results shown in Table 1 and give some examples of graph problems complete for #L, span-L, and opt-L.

The first problem we consider is the ranking function for DFA, $f_{\#\text{DFA}}$, which is complete for #L. As can be seen from the proof of Theorem 3.1, already the special case of the census function is hard for #L, and even the ranking function for unambiguous NFA is contained in that class. Unambiguous NFA are NFA that accept an input with a unique accepting computation.

$f_{\#\text{DFA}}$:

Input: An encoding of a DFA M and a string $x\in\{0,1\}^*$.

Output: Number of words lexicographically smaller than or equal to x accepted by M.

Theorem 3.1. *$f_{\#\text{DFA}}$ is log-space many-one complete for #L.*

Proof. $f_{\#\mathrm{DFA}} \in \#\mathrm{L}$ can be seen as follows. Construct an NL-machine N, which on input a DFA M and $x \in \{0,1\}^*$ guesses a word $y \leqslant x$ bit-by-bit and records for every new guessed bit the corresponding state of M. N accepts if a final state is ever reached. Since the automaton M is unambiguous, there is exactly one computation path of N for any word smaller than or equal to x. Furthermore, the number of *valid* computation paths of N corresponds to the number of such words accepted by M.

For the hardness property $\#\mathrm{L} \subseteq \mathrm{FL_m}(f_{\#\mathrm{DFA}})$, let $f \in \#\mathrm{L}$ be given such that $f(x)$ equals the number of accepting computations of an NL-machine N for any input $x \in \{0,1\}^*$. Let $p(|x|)$ be the polynomial that bounds the running time of N. We have to show that there exists a function $h \in \mathrm{FL}$ such that $f(x) = f_{\#\mathrm{DFA}}(h(x))$. This function will be $h(x) := (\langle N_x \rangle, 1^{p(|x|)})$, where $\langle N_x \rangle$ denotes the encoding of a state transition graph of a DFA constructed as follows. Let $C_{(N,x)}$ denote the set of all configurations of N on input x, let $c_{(start,x)}$ be the start configuration, and $C_{(acc,x)}$ the set of all accepting configurations. Furthermore, let *sink* denote an element not contained in $C_{(N,x)}$. The 5-tuple $N_x := (C_{(N,x)} \cup \{sink\}, C_{(N,x)} \cup \{sink\}, \delta, c_{(start,x)}, C_{(acc,x)})$ denotes the set of states, the alphabet, the transition function, the initial state, and the set of final states, respectively, where for all $c_i, c_j \in C_{(N,x)} \cup \{sink\}$:

$$\delta(c_i, c_j) := \begin{cases} c_j & \text{if } c_i \text{ reaches } c_j \text{ in one step in a computation of } N \text{ on } x, \\ sink & \text{otherwise.} \end{cases}$$

The automaton N_x is deterministic and can obviously be constructed easily with logarithmic space. Furthermore, it is clear that the construction ensures that the number of different computation paths of N on input x equals the number of different words accepted by N_x of length at most $p(|x|)$. With a little more care, the construction can be made such that the alphabet of N_x is $\{0,1\}$. □

Let $f_{\#\mathrm{co-DFA}}$ be the function that on input of an encoding of a DFA M and a string 1^n computes the number of words of length up to n *not* accepted by M. Since the automaton N_x constructed in the previous proof is complete, switching of accepting and nonaccepting states yields the following corollary.

Corollary 3.2. *$f_{\#\mathrm{co-DFA}}$ is log-space many-one complete for $\#\mathrm{L}$.*

The ranking function for NFA, $f_{\#\mathrm{NFA}}$ is complete for span-L. Here again, the census function is already complete.

$f_{\#\mathrm{NFA}}$:

Input: An encoding of an NFA M and a string $x \in \{0,1\}^*$.

Output: Number of words lexicographically smaller than or equal to x accepted by M.

Theorem 3.3. *$f_{\#\mathrm{NFA}}$ is log-space many-one complete for* span-L.

Proof. To see $f_{\#\mathrm{NFA}} \in$ span-L, construct an NL-transducer N, which on input an NFA M and $x \in \{0,1\}^*$ does the following. Starting with the initial state of M, N guesses and outputs a word $y \leqslant x$ bit-by-bit, guessing and recording a new state of M consistent with each guessed bit and the transition table of M. N accepts iff a final state is ever reached. Thus, only in this case is the output valid. If the machine is ambiguous, there may be more than one valid computation with the same output, but $span_N(\langle M, x\rangle)$, the number of different valid outputs of N, corresponds to the number of words accepted by M.

To see that span-L $\subseteq \mathrm{FL_m}(f_{\#\mathrm{NFA}})$, consider an arbitrary NL-transducer N that witnesses $f(x)$ via its span. We have to show that there exists a function h computable with log-space such that $f(x) = f_{\#\mathrm{NFA}}(h(x))$. h will be the function $h(x) := (\langle N_x\rangle, 1^{p(|x|)})$, where p is the polynomial that bounds the running time of N, and $\langle N_x\rangle$ denotes the encoding of a state transition graph of the NFA $N_x := (C_{(N,x)}, \{0,1\}, \delta, c_{(start,x)}, c_{(acc,x)})$. Here $C_{(N,x)}$ denotes the set of all configurations of N on input x, $c_{(start,x)}$ the start configuration, $c_{(acc,x)}$ the unique accepting configuration (in which no output occurs), and for all $c_i, c_j \in C_{(N,x)}$ such that c_i reaches c_j in one step in a computation of N on x, and $b \in \{0,1,\lambda\}$ we define:

$$\delta(c_i, b) = c_j \quad \text{if the output in } c_j \text{ is } b \in \{0,1\},$$

$$\delta(c_i, \lambda) = c_j \quad \text{if in } c_j \text{ no output occurs.}$$

Note that since M is polynomially clocked this graph contains no cycles. The number of different words up to length $p(|x|)$ that are accepted by N_x is exactly the number of different valid outputs the transducer N can produce on input x. Furthermore, it is obvious that the construction of N_x can be done with $\mathrm{O}(\log|x|)$ space. □

The maximal relative word function for both DFA and NFA is complete for opt-L.

$f_{maxrel\,\mathrm{DFA}}, f_{maxrel\,\mathrm{NFA}}$:

Input: An encoding of a DFA (NFA) M with $L(M) \subseteq \Sigma^*$, a finite set $\Sigma' \subseteq \Sigma$, and a string $x \in \Sigma'^*$.

Output: Lexicographically greatest word $h(w) \leqslant x$ such that $w \in L(M)$, where $h: \Sigma \to \Sigma' \cup \{\lambda\}$ is the homomorphism

$$h(a) := \begin{cases} a & \text{if } a \in \Sigma', \\ \lambda & \text{otherwise.} \end{cases}$$

(If no such word exists, the output is $\perp$.)

Consider also the maximal word function. This is the special case of $f_{maxrel\,\mathrm{DFA}}$ and $f_{maxrel\,\mathrm{NFA}}$ in which $\Sigma' = \Sigma = \{0,1\}$.

$f_{max\,\mathrm{DFA}}, f_{max\,\mathrm{NFA}}$:

Input: An encoding of a DFA (NFA) M and a string $x \in \{0,1\}^*$.

Output: Lexicographically greatest word $y \in L(M)$ with $y \leqslant x$. (If no such word exists, the output is $\perp$.)

The function $f_{max\,\mathrm{NFA}}$ is already complete for opt-L. On the other hand, $f_{max\,\mathrm{DFA}}$ can be shown to be contained in FL(NL), a subclass of opt-L (see Theorem 4.2).

Theorem 3.4. (i) $f_{maxrel\,\mathrm{DFA}}$ and $f_{maxrel\,\mathrm{NFA}}$ *are log-space many-one complete for* opt-L.

(ii) $f_{max\,\mathrm{NFA}}$ *is log-space many-one complete for* opt-L.

(iii) $f_{max\,\mathrm{DFA}} \in \mathrm{FL(NL)}$.

Proof. (i): It suffices to show that $f_{maxrel\,\mathrm{NFA}} \in$ opt-L, and that $f_{maxrel\,\mathrm{DFA}}$ is hard for opt-L.

For $f_{maxrel\,\mathrm{NFA}} \in$ opt-L, construct an NL-transducer N, which on input an encoding of a nondeterministic automata M with $L(M) \subseteq \Sigma^*$, a finite set $\Sigma' \subseteq \Sigma$, and a string $x \in \Sigma'^*$ guesses and outputs symbol-by-symbol a string $y = y_1 y_2 \ldots y_n \in \Sigma'^*$ while recording the state p of M reached so far. For each guessed symbol $y_i \in \Sigma'$, N guesses two states q, q' of M, and verifies that there is a transition from q to q' on which the symbol is read, and that there is a path from p to q on which only symbols in $\Sigma - \Sigma'$ are read. q' then becomes the new state reached so far. N accepts when q' is a final state of M. Since N has guessed an arbitrary word smaller than or equal to x, the maximal valid output of N, $opt_N(\langle M \rangle, \Sigma', x)$, equals the (lexicographically) greatest word $h(w) \leqslant x$ such that $w \in L(M)$, where h is the homomorphism that deletes all symbols in $\Sigma - \Sigma'$.

For the hardness property opt-L $\subseteq \mathrm{FL_m}(f_{maxrel\,\mathrm{DFA}})$, let f be an arbitrary function in opt-L. Let N be an NL-transducer such that $opt_N(x) = f(x)$ for all inputs x. We will construct a function h computable with log-space such that $f(x) = f_{maxrel\,\mathrm{DFA}}(h(x))$. h is defined by

$$h(x) := (\langle N_x \rangle, \{0,1\}, 1^{p(|x|)}),$$

where p denotes the polynomial that bounds the running time of N, $\{0,1\}$ denotes the specified subalphabet, and $\langle N_x \rangle$ denotes the encoding of a state transition graph of the DFA $N_x := (C \cup C' \cup \{sink\}, \{0,1\} \cup C, \delta, c_{start}, c_{acc})$. Here C is the set of all configurations of N on input x, C' a (distinct) set of all configurations in which an output 0 or 1 occurs, c_{start} the start configuration, and c_{acc} the unique accepting configuration. We assume that in c_{acc} no output occurs. The transition function δ from $(C \cup C' \cup \{sink\}) \times (C \cup \{0,1\})$ to $C \cup C' \cup \{sink\}$ is defined for all configurations c_i, c_j, such that c_i reaches c_j in one step in a computation of N on x by:

$$\delta(c_i, c_j) = c_j \quad \text{if no output occurs in } c_j,$$

$$\delta(c_i, c_j) = c'_j \quad \text{if some output occurs in } c_j,$$

and

$$\delta(c'_j, b) = c_j \quad \text{if the output } b \in \{0,1\} \text{ occurs in } c_j.$$

To make the automaton complete lead all the remaining labels into the state *sink*.

It is easy to see that the automaton is deterministic. Furthermore, for any path of N on input x, there is a word in $L(N_x)$ consisting of the sequence of configurations of the path, each of which is followed by the possible output (0 or 1) of the configuration (for example, $c_1 0 c_2 c_3 1 c_4 \dots c_{p(|x|)}$). But then, clearly, $f(x)$, the maximal output produced by N on input x, equals the maximal word $h(w) \leqslant 1^{p(|x|)}$ such that $w \in L(N_x)$, where h is the homomorphism that deletes all symbols in C. Thus, $f_{maxrel\mathrm{DFA}}$ is hard for opt-L, too.

(ii): We have $f_{max\mathrm{NFA}} \in$ opt-L with (i), since $f_{max\mathrm{NFA}}$ is a special case of $f_{maxrel\mathrm{NFA}}$. For opt-L $\subseteq \mathrm{FL_m}(f_{max\mathrm{NFA}})$, the proof for the hardness in Theorem 3.3 carries over. Let $f \in$ opt-L be arbitrarily chosen and let N be the NL-transducer with time bound p such that $f = opt_N$. For a given N and $x \in \{0,1\}^*$, construct N_x exactly as for the proof of Theorem 3.3. It is easily verified that $f(x) = f_{max\mathrm{NFA}}(h(x))$ with h as defined there.

(iii): Given an encoding of a DFA M and a string $x \in \{0,1\}^*$, a deterministic log-space transducer N can compute $f_{max\mathrm{DFA}}(\langle M \rangle, x)$ with the help of the following two oracles $A, B \in \mathrm{NL}$:

$$A := \{\langle M \rangle \$ x \$ 0^m \mid m \geqslant 1, \exists w \in \{0,1\}^m \text{ such that } w \leqslant x \text{ and } w \in L(M)\},$$

$$B := \{\langle M \rangle \$ q \$ 0^n \mid n \geqslant 1, q \text{ is a state of } M \text{ such that there exists a path of length } n \text{ from } q \text{ to a final state of } M\}.$$

With at most $|x|$ queries to A, N can find the length l_{max} of the maximal word smaller than or equal to x accepted by M. Since the given automaton is deterministic, with at most $2l_{max}$ queries to B, N can find longer and longer prefixes of this word, outputting each bit found. □

Note that it will not be easy to improve the upper bound of FL(NL) for $f_{max\mathrm{DFA}}$, since $f_{max\mathrm{DFA}} \in \mathrm{FL_{log}(NL)}$ would imply L = NL (see Section 5, the remark after Proposition 5.6). Hence, $f_{max\mathrm{DFA}}$ is clearly an example of "hardest" function in FL(NL).

Other complete functions for the classes #L, span-L, and opt-L can be obtained by defining "counting" versions of the graph accessibility problem, *GAP*, which is well-known to be NL-complete [16]. Consider the functional graph problems:

$f_{\#\,path}$:

Input: A directed graph $G = (V, E)$ with vertex set $V = \{1, 2, \dots, n\}$.

Output: Number of different paths of length at most n from vertex 1 to vertex n.

$f_{\#\,special\,path}$:

Input: A directed labelled graph $G = (V, E)$ with vertex set $V = \{1, 2, \dots, n\}$ and edges labelled over L, and $L' \subseteq L$.

Output: Number of lexicographically different paths from vertex 1 to vertex n of length at most n, with the labels in L' deleted.

$f_{maxpath}$:

Input: A directed labelled graph $G = (V, E)$ with vertex set $V = \{1, 2, \dots, n\}$.

Output: Lexicographically maximal path from vertex 1 to vertex n of length at most n. (If no such path exists, the output is $\perp$.)

It is not hard to show that these three functions are contained in #L, span-L, and opt-L, respectively. By (many-one) reducing the complete problems of Theorems 3.1, 3.3, and 3.4(ii) to the corresponding graph functions it can, furthermore, be shown that these functions are also complete for these classes.

Corollary 3.5. (i) $f_{\#path}$ *is log-space many-one complete for* #L;

(ii) $f_{\#special\,path}$ *is log-space many-one complete for* span-L;

(iii) $f_{maxpath}$ *is log-space many-one complete for* opt-L.

4. #L and opt-L are easy, but span-L is hard (for the polynomial hierarchy)

Since there are only polynomially many different configurations possible for an NL-machine or an NL-transducer, one expects that the definition of counting and optimization classes with the help of log-space-bounded Turing machines leads to classes included in FP. This is indeed the case for the classes #L and opt-L. With the following two theorems we show that both classes are even included in NC^2.

Recall that $\mathrm{FL} \subseteq \mathrm{NC}^2 \subseteq \mathrm{FP}$, where NC^2 denotes the class of Boolean functions computable by (log-space) uniform polynomial size and $(\log n)^2$ depth circuits with bounded fan-in gates, where n is the length of the input. (For a formal definition of NC^2 see [5] or [24].)

Theorem 4.1. $\#\mathrm{L} \subseteq \mathrm{NC}^2$.

Proof. Let f be in #L. Then there exists an NL-machine M such that $f(x) = acc_M(x)$. We assume that M has a unique accepting configuration, and that for any input x of length n all computation paths of M on x have length exactly $p(n)$ for a polynomial p. Let q be the polynomial that bounds the number of configurations of M on input x.

Let x be an input of length n. We compute $acc_M(x)$ by computing the $p(n)$th power of the $q(n) \times q(n)$ integer matrix $A = (a_{ij})$ that represents the adjacency matrix of the computation graph of M on x:

$$a_{ij} := \begin{cases} 1 & \text{if configuration } j \text{ is accessible from configuration } i \\ & \text{in one step in a computation of } M \text{ on input } x, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, the element $a_{ij}^{p(n)}$ of the matrix $A^{p(n)}$, the product of A with itself $p(n)$ times, contains the number of paths from vertex i to vertex j with exactly length $p(n)$. It was shown by Cook in [5] that the nth power of a given $n \times n$ matrix can be computed by an NC^2 circuit. Furthermore, we can construct an NC^1 circuit that computes on input x the matrix A, and the initial and accepting configuration s and t. Thus, we conclude that $\#\mathrm{L} \subseteq \mathrm{NC}^2$.

Note that in fact $\#\mathrm{L} \subseteq \mathrm{DET}^* \subseteq \mathrm{NC}^2$, where DET^* is the class of functions NC^1-Turing-reducible to computing integer matrix determinants [5]. In [5] it is proved that integer matrix powering is complete for DET^*. □

opt-L can be shown to be included in FP, since due to the polynomial bound on the number of configurations of an NL-transducer, the optimal output value can be constructed bit-by-bit by a breadth-first search method. But with a little more effort, we can even show that opt-L is included in NC^2.

Theorem 4.2. $\mathrm{FL(NL)} \subseteq \text{opt-L} \subseteq \mathrm{NC}^2$.

Proof. For the inclusion $\mathrm{FL(NL)} \subseteq$ opt-L, let f be in FL(NL). Since NL is closed under complementation, an L-transducer M with oracle $A \in \mathrm{NL}$ computing f can be simulated by an NL-transducer N, which uses subroutines for A and A^c to solve the oracle queries. For any oracle query w, the answer is guessed and correspondingly either the subroutine for A or A^c is started. Since $|w|$ can be up to a polynomial in the length of x, for this, N saves the configuration c_w in which M starts to write w on its oracle tape, and produces each symbol of w again, if necessary, by starting M in c_w. By construction of N, all accepting paths on input x have the same output value $f(x)$. Hence, $f(x) = opt_N(x)$.

For the inclusion opt-$\mathrm{L} \subseteq \mathrm{NC}^2$, let f be in opt-L. Then there exists an NL-transducer M such that $f(x) = opt_M(x)$. We will reduce the problem of computing $opt_M(x)$, the maximum output of M on input x, to computing a special power of a matrix, whose entries are words from the set $\{0,1\}^* \cup \{\perp\}$.

First, define the special concatenation $*$ for words $w, v \in \{0,1\}^* \cup \{\perp\}$ as follows:

$$w * v := \begin{cases} wv & \text{if } w, v \in \{0,1\}^*, \\ \perp & \text{otherwise.} \end{cases}$$

Let $\perp$ be lexicographically smaller than any other word in $\{0,1\}^* \cup \{\perp\}$. For a tuple $(x_1, \ldots, x_n)$ of words in $\{0,1\}^* \cup \{\perp\}$ define $MAX(x_1, \ldots, x_n)$ to be the maximum word with respect to lexicographic order. Then the special "matrix product" $A * B = ((a * b)_{ij})$ of two word matrices $A = (a_{ij})$ and $B = (b_{ij})$ of dimension n is defined by

$$(a * b)_{ij} := \operatorname*{MAX}_{k=1}^{n} (a_{ik} * b_{kj}) \quad \text{for all } 1 \leqslant i, j \leqslant n.$$

It is not hard to see that for two matrices with entries from $\{0,1\}^*$ the "special matrix product" can be computed in NC^1.

Let p and q be the polynomials that bound the time and the number of configurations of M. Let s, t with $s \neq t$ denote the initial and (unique) final configuration of M, respectively. We assume that neither in s nor in t occurs an output. For any input x of length n define the following $q(n) \times q(n)$ matrix $A = (a_{ij})$ by

$$a_{ij} := \begin{cases} b & \text{if in a computation of } M \text{ on input } x \\ & \text{the configuration } j \text{ is accessible from configuration } i \text{ in one step,} \\ & \text{and in } j \text{ the output bit is } b \in \{0,1\}, \\ \perp & \text{otherwise.} \end{cases}$$

Let $A^{p(n)}=(a_{ij}^{p(n)})$ be the $p(n)$th power of A with respect to the special matrix product $*$. It can be proved by induction that the element $a_{ij}^{p(n)}$ contains the maximal word that M can produce on a path of length $p(n)$ from configuration i to configuration j. Here the smallest word $\perp$ stands for "no output". Hence, $opt_M(x)$ equals $a_{st}^{p(n)}$, if $a_{st}^{p(n)}$ contains a word in $\{0,1\}^+$, and equals 0, otherwise.

Since for any input x of length n the matrix A, s, t, and $p(n)$ can be obtained from M and x with an NC^1 circuit, and the special "matrix product" is computable in NC^1, $opt_M(x)$ can be obtained by computing $O(\log n)$ times suitable matrix products in parallel. Altogether, this yields an NC^2 algorithm. □

Inclusion relations between #L and opt-L seem to be difficult to determine. The inclusion of opt-L in #L would imply that nondeterministic computation can be made unambiguous in the case of logarithmic space. Let UL denote the class of languages acceptable by NL-machines that in the case of acceptance have a unique accepting computation. Then we have the following proposition.

Proposition 4.3. FL(NL)$\subseteq$ #L *if and only if* NL$=$UL.

Proof. Clearly, for the characteristic function c_A of a language $A\in$NL, $c_A\in$FL(NL). But $c_A\in$ #L implies that there exists an NL-transducer that on input x has exactly one computation path if $x\in A$ and none if not. Thus, $A\in$UL, and NL$\subseteq$UL. UL$\subseteq$NL holds by definition.

Conversely, assume that NL$\subseteq$UL. Let $f\in$FL(NL), M, and N be as in the proof of Theorem 4.2 for the inclusion FL(NL)$\subseteq$opt-L. Now, by assumption, the subroutines A and A^c compute their solution with a unique accepting path. Hence, on input x, N will have exactly one valid computation path with output $f(x)$. Construct an NL-machine N' that on input x guesses a value z and checks $z\leqslant f(x)$ by simulating N. (Again, guessing and checking will have to interleave as in the proof for opt-L$\subseteq$ span-L of Proposition 2.4.) Then, $acc_{N'}(x)=f(x)$, and FL(NL)$\subseteq$ #L follows. □

With Theorem 4.2 and Proposition 4.3 it follows Corollary 4.4.

Corollary 4.4. *If* opt-L$\subseteq$ #L, *then* UL$=$NL.

The corresponding result for polynomial-time function classes (opt-P$\subseteq$ #P if and only if UP$=$NP) was shown in [19]. Note that the other direction in Corollary 4.4 can be obtained if one-way classes are considered (see Theorem 4.16).

Although FL(NL)$\subseteq$opt-L, we cannot show that FL(NL) is included in #L. But we have

FL(NL)$\subseteq$FL(PL)$\subseteq$FL(#L),

where PL denotes the log-space counterpart to PP, and DET* is the set of functions NC^1-Turing reducible to computing integer matrix determinants [5]. This is parallel to the case of polynomial time, where even P(NP)$\subseteq$P(PP)$=$P(#P) [2].

We have seen that functions in $\#$L and opt-L can be computed in polynomial time. It is surprising that span-L turns out to be a hard log-space counting class. In the following we will show that if span-L is included in FP, then the polynomial-time hierarchy collapses to P. This follows from the fact that the complexity of span-L is closely tied to the complexity of $\#$P. The two classes are log-space metric reducible to each other, that is, $\mathrm{FL}_1(\#\mathrm{P})=\mathrm{FL}_1(\text{span-L})$, as shown in the remaining part of this section.

Theorem 4.5. span-L$\subseteq\#$P.

Proof. Let f_M be a function in span-L which on input x computes the number of different valid output values of an arbitrarily chosen NL-transducer M. Let p be the polynomial that bounds the time of M. Then any output value y of M on input x satisfies $|y|\leqslant p(|x|)$. Consider the set

$$A:=\{x\$y\mid x,y\in\{0,1\}^*,\ y \text{ is a valid output of } M \text{ on input } x\}.$$

It is easy to see that $A\in$NL. Since NL$\subseteq$P, there exists a P-algorithm for deciding A. Construct an NP-machine N which on input x guesses y of length less than or equal to $p(|x|)$ and then verifies that $x\$y\in A$ by executing the P-algorithm for A on $x\$y$. Then N has exactly one accepting computation for each y that is an output value of M on input x. Thus, N has exactly $f_M(x)$ accepting computations, and $f_M\in\#$P. □

To show that, conversely, $\#$P is log-space metric reducible to span-L, we first give an example of a function log-space many-one complete for $\#$P and prove a lemma.

3SAT, the set of satisfiable Boolean formulas in 3-conjunctive normal form (conjunctive normal form with at most 3 literals per clause) is well-known to be complete for NP with respect to log-space many-one reductions [12]. Consider the function

$f_{\#3SAT}$:

Input: A Boolean formula F in 3-conjunctive normal form.

Output: Number of satisfying assignments of F.

Valiant showed in [31] that $f_{\#3SAT}$ is complete for $\#$P with respect to polynomial-time Turing reductions. His proof (the standard many-one reduction from *SAT* to *3SAT*) even yields the following theorem.

Theorem 4.6 (Valiant [31]). *$f_{\#3SAT}$ is complete for $\#$P with respect to log-space many-one reductions.*

For the proof of $\#\mathrm{P}\subseteq\mathrm{FL}_1(\text{span-L})$ (Theorem 4.8), we will use the fact that the "complement" of this function, the function that counts the number of *nonsatisfying* assignments of F, is contained in span-L:

$f_{\#UN3SAT}$:

Input: A Boolean formula F in 3-conjunctive normal form.

Output: Number of *nonsatisfying* assignments of F.

First, consider the set of pairs of Boolean formulas and assignments:

$EVAL3SAT := \{F\$b_1 b_2 \ldots b_n \mid b_i \in \{0,1\}$, F Boolean formula in 3-conjunctive normal form, and $b_1 \ldots b_n$ is a satisfying assignment for $F\}$.

Lemma 4.7. $EVAL3SAT \in$ co-1NL.

Proof. A co-1NL-machine can be understood as a one-way nondeterministic log-space machine with universal states. On input of

$$(\tilde{x}_{11}, \tilde{x}_{12}, \tilde{x}_{13}), \ldots, (\tilde{x}_{m1}, \tilde{x}_{m2}, \tilde{x}_{m3})\$b_1 b_2 \ldots b_n,$$

where each $\tilde{x}_{ij}$ is a literal, a clause X_i can be guessed universally, the head of the input tape can be moved to that clause and its three literals $\tilde{x}_{i1}, \tilde{x}_{i2}$, and $\tilde{x}_{i3}$ can be stored on the working tape. By moving the head to the assignment, the three relevant bits can be picked up and it can be checked whether the clause is satisfied. That the input is well-formed can be checked by a different universal tree from the start configuration.

The reader who prefers to guess existentially can easily verify conversely, that the complement of $EVAL3SAT$ is contained in 1NL: a clause can be guessed and verified that it is not satisfied by the assignment or it can be guessed and verified that the input is not well-formed. Clearly, in both cases the input has to be read only once from left to right for the verification. In the former case, first the (at most 3) variables of the guessed clause will have to be picked up and stored and then by moving the input head further to the right, the corresponding three variable assignments must be made out to accept in the case that all three make its literal have value 0. □

It is known that L$\neq$co-1NL, since 1NL is not closed under complementation [11]. But it is easy to see that $EVAL3SAT$ is also contained in L. As a consequence, NP = NL(L) [25], where NL($\cdot$) denotes the closure under *nondeterministic log-space Turing reducibility*. $A \in$ NL(B) if there exist a set B and a polynomial-time bounded NL-machine M with (unbounded) oracle tape such that $A = L(M, B)$ (see [22]). In fact, it even holds NP = NLOG(L), where NLOG($\cdot$) denotes the closure under *nondeterministic log-space many-one reducibility* [23]. For a set A, $A \in$ NLOG(B), if there exist a set B and a polynomial-time bounded NL-transducer N such that for all x:

$$x \in A \Leftrightarrow \text{one of the computations of } N \text{ on } x \text{ produces a word } y \in B.$$

Lemma 4.7 implies that NP = NLOG(co-1NL). It is easy to see that the set $3SAT$ can be nondeterministically log-space many-one reduced to $EVAL3SAT$. An NLOG-transducer simply copies its input to the output tape, writes the separation symbol \$, counts the number of different variables, and guesses and outputs an assignment of appropriate length. (This proof is very similar to the proof in [15] to show that simultaneously polynomial-time- and log-space-bounded auxiliary push-down automata with one alternation accept $3SAT$ by first *existentially* guessing an assignment onto the pushdown store and subsequent *universally* checking that it was correct.) Because co-1NL$\subseteq$P, the other inclusion is trivial. It even holds NP$=$NLOG(NL).

Note that, on the other hand, NL$=$NLOG(1NL). To see NLOG(1NL)$\subseteq$NL, let $A\in$NLOG(B) via an NLOG-transducer N, and let $B=L(M)$ for a 1NL-machine M. Construct another NL-machine that takes turns in simulating N and M. First, N is simulated until the first output occurs; then M is simulated on this output until it asks for its next input bit (M reads the input one-way), which is the following output bit of N and can be obtained by simulating N further, and so on. For taking turns in the simulation, only the current configuration of N or M must be stored. The inclusion in the other direction is trivial.

The idea for the proof of NP$=$NLOG(co-1NL) can be put to use to show that span-L contains the function $f_{\#UN3SAT}$, and $f_{\#UN3SAT}$ can then be computed as the difference of a function in FL and a function in span-L as shown in the proof of the following theorem.

Theorem 4.8. $\#\text{P}\subseteq\text{FL}_1(\text{span-L})$.

Proof. Since $f_{\#3SAT}$ is log-space many-one complete for $\#$P (Theorem 4.6), it suffices to show that $f_{\#3SAT}\in\text{FL}_1(\text{span-L})$.

We will first show that $f_{\#UN3SAT}\in$span-L. For this, let N be a 1NL-machine for the complement of *EVAL3SAT*. By Lemma 4.6 such a machine exists. We construct an NL-machine M such that on input of a well-formed formula F in 3-conjunctive normal form, $span_M(F)$ equals the number of *nonsatisfying* assignments of F. On input x, M first checks whether x is of the required form; if this is not the case, M rejects, and in this case $span_M(x)=0$. If the input x is a well-formed formula F, M counts the number of different variables in F, and simulates the machine N up to the point, where N reads the separation symbol \$. For the further simulation of N, M guesses – symbol-by-symbol – an assignment of appropriate length. Every symbol (0 or 1) guessed will be written by M on the output tape. N accepts if and only if the guessed assignment is nonsatisfying for F. Thus, $span_M(F)$, the number of different valid outputs of M equals the number of different nonsatisfying assignments for F. We can conclude that $f_{\#UN3SAT}\in$span-L.

We will now show that $f_{\#3SAT}$ can be computed by a deterministic L-transducer M' that asks one query to $f_{\#UN3SAT}$. This yields $f_{\#3SAT}\in\text{FL}_1$(span-L). On input x, M' first checks that x is a formula in 3-conjunctive normal form. If this is not the case, M' outputs 0. If x is a formula F of the required form, M' copies F to the oracle tape, queries its oracle for $f_{\#UN3SAT}(F)$ and then has two-way read access to $f_{\#UN3SAT}(F)$ on its oracle tape. M' now computes the difference $2^n-f_{\#UN3SAT}(F)$, where n denotes the number of different variables in F, and outputs this value. Clearly, the output of M' is correct, since for a formula F with n variables, $f_{\#3SAT}(F)=2^n-f_{\#UN3SAT}(F)$.

It is not hard to show that this difference can be computed with O$(\log n)$ space. (Note that 2^n can be "stored" by storing n). Since the value $f_{\#UN3SAT}(F)$ on M's oracle tape can be read two-way, M' can produce its output either least significant bits first or most significant bits first. □

Corollary 4.9. $\mathrm{FL}_1(\text{span-L}) = \mathrm{FL}_1(\#P)$, *i.e. every function in* #P *is logarithmic-space metric-reducible to a function in* span-L *and vice versa.*

Thus, span-L and #P are classes very similar in computation power. Nevertheless, they do not seem to be the same function class. Since $f_{\#3SAT}$ is complete for #P, the inclusion $\#\mathrm{P} \subseteq \text{span-L}$ would imply that there exists an NL-transducer M such that $span_M(F)$ equals the number of different *satisfying* assignments of a Boolean formula F. By simulating M and checking that M has a valid output on input F (this is an NL-predicate), an NL-machine could accept $3SAT$. Thus, we have Proposition 4.10.

Proposition 4.10. *If* span-L = #P, *then* NL = P = NP.

Since $\mathrm{P(NP)} \subseteq \mathrm{P}(\#\mathrm{P})$, Corollary 4.9 furthermore implies that span-L is Turing hard for $\Delta_2^p = \mathrm{P(NP)}$, the second deterministic level of the polynomial-time hierarchy and, thus, $\text{span-L} \subseteq \mathrm{FP}$ implies P = NP. But the implications of this inclusion are even stronger. As recently shown by Toda, the class PP is hard for the polynomial-time hierarchy (PH) with respect to Turing reducibility [28]. Since $\mathrm{P}(\#\mathrm{P}) = \mathrm{P(PP)}$ [2], this class can now be also characterized using span-L. Furthermore, results obtained by Toda and Watanabe [29] imply $\mathrm{P}(\#\mathrm{P}) = \mathrm{P}(\text{span-P})$. Thus, we have the following Corollaries.

Corollary 4.11. $\mathrm{PH} \subseteq \mathrm{P}(\text{span-L}) = \mathrm{P}(\#\mathrm{P}) = \mathrm{P}(\text{span-P}) = \mathrm{P(PP)}$.

Corollary 4.12. $\text{span-L} \subseteq \mathrm{FP}$ *if and only if* $\mathrm{P} = \mathrm{NP} = \mathrm{PH} = \mathrm{P}(\#\mathrm{P})$.

By Corollary 4.9, span-L and #P share the same complete languages with respect to metric reducibility. With Theorem 3.3 and Corollary 3.5(ii) we can, thus, add the two functions $f_{\#\mathrm{NFA}}$ and $f_{\#specialpath}$ to the list of functions complete for #P. These functions are not merely counting versions of NP-complete problems (as are $f_{\#3SAT}$ and most of the #P-complete functions in [31]), but are rather counting versions of NL-complete problems.

Corollary 4.13. $f_{\#\mathrm{NFA}}$ *and* $f_{\#specialpath}$ *are complete for* #P *with respect to log-space metric reducibility.* □

Consequently, the census or ranking functions of NFA and of 1NL-machines can be computed in polynomial time if and only if FP = #P. Similar results about the difficulty of computing the ranking function of languages in various other "small" complexity classes have been published before (see [3, 7, 10, 13]). Our result extends this list with an extremely simple case and, furthermore, with completeness results.

We can say even more about the relationship between span-L and #P in terms of automata problems. As shown in Section 3, computing the ranking function or the census function for an NFA is log-space *many-one* complete for span-L. The "complement" of the census function is

$f_{\#\text{co-NFA}}$
Input: An encoding of a NFA M and 1^n.
Output: Number of words of length n that are not accepted by M.

Theorem 4.14. *$f_{\#\text{co-NFA}}$ is log-space many-one complete for* #P.

Proof. Since the word problem for NFA is decidable in NL, it is easy to see that $f_{\#\text{co-NFA}}$ is contained in #P.

To see that $f_{\#\text{co-NFA}}$ is hard for #P, let $f \in \#\text{P}$, and let M be an NP-machine such that $f = acc_M$. We assume that all computations of M on inputs of size n have length $q(n)$ for a polynomial q. In [8] it was shown that the nonuniversality of regular expressions is hard for PSPACE with respect to log-space reductions. For the proof of this result the nonaccepting computations of a NPSPACE-machine on input x were described by a regular expression R_x. For the construction of R_x, $O(\log|x|)$ space is sufficient. It is not hard to see that regular expressions of size n can be transformed into an NFA in $O(\log n)$ space. Let N_x be the finite-state automaton that results from applying the construction of [8] to M and the subsequent transformation. Then $L(N_x)$ describes the nonaccepting computations of M on input x, and any word $w \notin L(N_x)$ with $|w| \leq q(|x|)$ codes an accepting computation of N on input x. Consequently, f can be log-space many-one reduced to $f_{\#\text{co-NFA}}$. □

The coding variant of $f_{\#\text{co-NFA}}$, where n is given in binary, rather than in unary, is (many-one) hard for #PSPACE = FPSPACE [21]. Note that the classes #P and #PSPACE can trivially be separated, since functions in the latter class are not in general polynomially bounded. (Ladner studies in [21] also restrictions of #PSPACE.)

An overview of the complete automata functions for #L, span-L, and #P is given in Table 2.

Table 2
Overview of the complexity of some automata problems (Completeness w.r.t. (functional) log-space many-one reducibility)

Input $\langle\langle M\rangle, 1^n\rangle$, compute	DFA	NFA
Nonemptiness $L(M) \neq \emptyset$?	NL-complete [16]	NL-complete [16]
Nonuniversality $L(M) \neq \Sigma^*$?	NL-complete [16]	PSPACE-complete [8]
Number of members $\|L(M)^{\leq n}\|$	#L-complete [Theorem 3.1]	span-L-complete [Theorem 3.3]
Number of nonmembers $\|(L(M)^{\leq n})^c\|$	#L-complete [Corollary 3.2]	#P-complete [Theorem 4.14]

The last two lines of the table in Table 2 reflect a second time that any function in #P can be computed as the difference of a function in FL and a function in span-L (compare the proof of Theorem 4.8). This parallels the relationship between #NP and span-P: any function in #NP can be computed as the difference of a function in FP and a function in span-P [19]. Here #NP is the class of functions that witness the number of accepting computations of NP(NP)-machines (NP-machines with oracles in NP).

In fact, we can just substitute "L" for "P" in this statement, and say: any function in #NL can be computed as the difference of a function in FL and a function in span-L. Let #NL denotes the class of functions that witness the number of accepting computations of NL(NL)-machines. Since NP = NL(NL), it is not hard to show with the considerations after Lemma 4.7 that #NL = #P.

Corollary 4.12 together with Theorems 4.1 and 4.2. show that the inclusion of span-L in either of the subpolynomial counting classes #L or opt-L is unlikely.

Corollary 4.15. *If either* span-L $\subseteq$ #L *or* span-L $\subseteq$ opt-L, *then* P = NP = P(#P) = P(span-L).

We obtain further time–space downward separations by considering the relationships between the one-way classes 1NL, 1UL, #1L, opt-1L, and span-1L. These classes are defined by restricting the underlying NL-machines or NL-transducers to read their input one-way. It holds that opt-1L $\cup$ #1L $\subseteq$ span-1L, since the proof of Proposition 2.4 carries over.

Since the difference between #- and span-classes exactly corresponds to the difference between unambiguous and ambiguous computation, the proof that NP $\subseteq$ UP $\Leftrightarrow$ #P = span-P given in [19] can be translated to log-space counting classes when one-way machines are considered. In the case of two-way classes the proof technique cannot be used to show the implication from right to left.

Theorem 4.16. *The following propositions are equivalent*:

(i) 1UL = 1NL;

(ii) span-1L = #1L;

(iii) opt-1L $\subseteq$ #1L.

Proof. (i)$\Rightarrow$(ii): Assume 1NL $\subseteq$ 1UL and let $f = span_M$, where M is a 1NL-machine. Consider the set

$$L_M = \{x_1 \# y_1 \$ \ldots \$ x_n \# y_n \$ \mid x_i \in \{0,1\},\ y_j \in \{0,1\}^*, \text{ and on input } x_1 x_2 \ldots x_n \text{ there is a computation path on which } M \text{ outputs } y_1 \ldots y_n \text{ such that output } y_i \text{ occurs after reading bit } x_i \text{ and before reading input bit } x_{i+1}\}.$$

As M is a one-way machine, it can be shown that $L_M \in$ 1NL and, hence, $L_M \in$ 1UL by assumption. Let M' be a 1UL-machine for L_M. M' has a unique accepting computation, when it accepts. Construct a 1NL-machine M'' that on input x does the following: M'' guesses $x_1 \# y_1 \$ x_2 \# y_2 \$ \ldots x_n \# y_n \$$ bit-by-bit, simulates M' on this word, and checks that $x = x_1 \ldots x_n$. Then $f(x) = acc_{M''}(x)$, and $f \in \#1\text{L}$.

(ii)$\Rightarrow$(iii) follows with opt-1L$\subseteq$span-1L, for which the proof of Proposition 2.4 carries over.

(iii)$\Rightarrow$(i): Since 1UL$\subseteq$1NL it suffices to show that (ii) implies 1NL$\subseteq$1UL. Assume that opt-1L$\subseteq \#$1L. Let $A \in$1NL, and let M be a 1NL-machine M that accepts A. Construct a 1NL-transducer M' that on input x simulates M and outputs "1" if M accepts. Then $opt_{M'}$ is the characteristic function c_A of A. By assumption, $c_A \in \#1\text{L}$. Thus, there exists a 1NL-machine that on input x has one accepting computation, if $x \in A$, and none, if $x \notin A$. Hence, $A \in$1UL, and 1NL$\subseteq$1UL. □

Corollary 4.17. *The following statements imply* P = NP = P($\#$P) = P(span-L):

(i) 1UL = 1NL;

(ii) span-1L = $\#$1L;

(iii) opt-1L $\subseteq \#$1L.

Proof. Because of Theorem 4.16 and Corollary 4.12, it is sufficient to show that span-1L = $\#$1L $\Rightarrow$ span-L$\subseteq$FP. It is easy to see that the span-L-complete function $f_{\#\text{NFA}}$ is already contained in span-1L. Assume now that span-1L = $\#$1L. Then, $f_{\#\text{NFA}} \in \#1\text{L}$. And since $\#1\text{L} \subseteq \text{NC}^2$ by Theorem 4.1, and $\text{NC}^2 \subseteq \text{FP}$, $f_{\#\text{NFA}} \in \text{FP}$. This implies span-L$\subseteq$FP, because $f_{\#\text{NFA}}$ is complete for span-L (Theorem 3.3). □

5. Some restrictions of opt-L and span-L

In this section, we study some restrictions of the classes opt-L and span-L. The first type of restriction results from bounding the output size of the functions by a logarithm in the length of their input.

Definition 5.1. For a class of functions F, define

$$\text{F}[\log n] := \{ f \in \text{F} \mid \exists \text{ constant } c \, \forall x \, |f(x)| \leqslant c \log |x| \}.$$

Such a restriction has been considered in [20] for opt-P and in [19] for span-P. In [20] it was shown that opt-P[log n] contains functions that are complete for $\text{FP}_{\log}(\text{NP})$ with respect to metric reducibility:

$$(*) \qquad \text{FP}_{\log}(\text{NP}) = \text{FP}_1(\text{opt-P}[\log n]).$$

In [19] it was shown that although the inclusion span-P$\subseteq$opt-P in the unrestricted case is unlikely (because it implies NP = co-NP) the two classes coincide in the restricted case:

$$(**) \qquad \text{span-P}[\log n] = \text{opt-P}[\log n].$$

$(*)$ and $(**)$ together show that span-P[log n] functions have considerably less power than span-P functions, which are hard for the polynomial-time hierarchy.

In the following, we will be interested in whether similar properties as $(*)$ and $(**)$ hold for the classes span-L$[\log n]$ and opt-L$[\log n]$. In the preceding section, we have shown that span-L, like span-P, is hard for the polynomial-time hierarchy. We will first show that span-L$[\log n]$ functions are computable in NC^2. This means that a polynomial bound on the number of different output values of an NL-transducer on a single input is a severe restriction.

In the rest of this section, we consider NL-transducers that produce their output deterministically. We will show that opt-L is exactly the class of functions in span-L that are witnessed by such transducers. We then show that $(*)$ translates to log-space, i.e. it holds that $\mathrm{FL}_{\log}(\mathrm{NL}) = \mathrm{FL}_1(\text{opt-L}[\log n])$. The class $\mathrm{FL}_{\log}(\mathrm{NL})$, furthermore, can be characterized in terms of opt-L functions that are witnessed by NL-transducers that write their output deterministically.

By Proposition 2.4, opt-L$[\log n] \subseteq$ span-L$[\log n]$. Combining the proof technique for opt-L$\subseteq \mathrm{NC}^2$ (Theorem 4.2) with some precomputation yields a parallel upper bound for span-L$[\log n]$.

Theorem 5.2. span-L$[\log n] \subseteq \mathrm{NC}^2$.

Proof. Let M be an NL-transducer witnessing a function in span-L$[\log n]$; i.e. M is such that for all inputs x the number of different output values is polynomially bounded in $|x|$. We assume that M has a unique final accepting state in which no output occurs, and that each computation path of M on input x has length $p(|x|)$ for a polynomial p.

We can use the technique of computing the output values via iterated matrix multiplication by defining a special matrix product, if we do some precomputation. Clearly, the output values cannot be constructed nonadaptively, starting with the full adjacency matrix of the configurations as in the proofs of Theorems 4.1 or 4.2, because there may be exponentially many output values between two arbitrary configurations c and c' of M. But a simple observation shows that this occurs if and only if either c or c' does not reach the final configuration or is not reachable from the start configuration. On the other hand, any configuration on a computation path on which a valid output is computed, is clearly reachable from the start configuration and reaches a final configuration. The precomputation will check these facts.

We start with the following matrix $A = (a_{ij})$ of configurations of M on input x, where t is the (unique) final accepting configuration, and s is the start configuration:

$$a_{ij} = \begin{cases} \{b\} & \text{if } s \xrightarrow{*} i \xrightarrow{1} j \xrightarrow{*} t, \\ & \text{and in configuration } j,\ M \text{ outputs } b \in \{0,1\}, \\ \{\lambda\} & \text{if } i = j \text{ or } s \xrightarrow{*} i \xrightarrow{1} j \xrightarrow{*} t, \\ & \text{and } M \text{ makes no output in configuration } j, \\ \perp & \text{otherwise,} \end{cases}$$

where $i \xrightarrow{1} j$ $(i \xrightarrow{*} j)$ means that the configuration j is accessible from configuration i

in one step (in an arbitrary number of steps) of M in a computation on x. The matrix A can be obtained with an NC^2-algorithm.

After computing the following "special matrix product" of $A, p(|x|)$ times with itself, the entry $a_{st}^{p(|x|)}$ of the resulting matrix will contain all the output values ($\neq \lambda$) of M on input x. To compute $span_M(x)$ these values have only to be summed up.

Denote the complex product of two sets S and S' by $S \cdot S'$. Define

$$S * S' := \begin{cases} S \cdot S' & \text{if } S, S' \subseteq \{0,1\}^*, \\ \{\perp\} & \text{otherwise.} \end{cases}$$

Extend this definition to the special complex product $B * C$ of $n \times n$ matrices $B = (B_{ij})$ and $C = (C_{ij})$ whose entries are subsets of $\{0, 1, \perp\}^*$ as follows:

$$(B * C)_{ij} := \bigcup_{k=1}^{n} B_{ik} * C_{kj}.$$

It is not hard to show that this "special matrix complex product" can be computed in NC^1, and that $A^{p(|x|)}$ can be computed in NC^2. □

A result like span-L[log n] $\subseteq$ opt-L[log n], however, corresponding to the result (**) for the polynomial-time classes, seems unlikely. The proof of (**) makes use of the fact that all the output values of an NP-transducer witnessing a function in span-P[log n] can be guessed by an NP-machine. Precisely this cannot be done by an NL-machine; it cannot check for more than a constant number of guessed values that they are different.

We will show that opt-L nevertheless can be characterized by span-L functions, if the output values of the NL-transducers are "compressible". This is the case, if the transducer in a Ruzzo–Simon–Tompa fashion has to write *deterministically* on its output tape (Ruzzo et al. considered such a restriction for the use of the oracle tape in [25]). This means that there is no further nondeterministic choice possible after the first output occurs on a path.

Definition 5.3. For a transducer M, let opt_M and $span_M$ be as in Definitions 2.2 and 2.3. Define

$$\text{opt-L}\{\textit{determ}\} := \{f \mid f = opt_M \text{ for some NL-transducer } M \text{ that writes its output deterministically}\};$$

$$\text{span-L}\{\textit{determ}\} := \{f \mid f = span_M \text{ for some NL-transducer } M \text{ that writes its output deterministically}\}.$$

Printing deterministically means that what is printed along one path is completely determined by the first configuration on the path in which an output occurs. Since there are only polynomially many different configurations, the number of different valid outputs is, therefore, bounded by a polynomial as well. Thus, span-L{*determ*} $\subseteq$ Span-L[log n].

Precisely this restriction of span-L[log n] characterizes opt-L[log n].

Proposition 5.4. *The following are descriptions of the same class*:

(i) opt-L[log n];

(ii) span-L{*determ*};

(iii) FL(NL)[log n];

(iv) $\mathrm{FL}_{\log}$(NL)[log n].

Proof. (i)$\Rightarrow$(ii): For opt-L[log n]$\subseteq$span-L{*determ*}, let $f\in$opt-L[log n]. Then $f = opt_M$ such that M is an NL-transducer all of whose valid outputs y on input x satisfy $|y| \leqslant c \log|x|$ for a constant c. We will construct an NL-transducer M' that tests for all strings v with $|v| \leqslant c\log|x|$, whether (x, v) is contained in

$$L_M = \{x\$v \mid |v| \leqslant c \log|x| \text{ and } v \text{ is a valid output of } M \text{ on input } x\}.$$

Since $L_M \in$NL, and NL is closed under complementation, this cycling can be done by M'. Clearly, M' will find $opt_M(x)$. Then, M' simply guesses a word v such that $v \leqslant opt_M(x)$, and outputs v. Since $opt_M(x) \leqslant c\log|x|$, M' can keep v on its work tape and can copy it deterministically onto its output tape. We have $span_{M'}(x) = opt_M(x)$, and $span_{M'}$ witnesses a function in span-L{*determ*}.

(ii)$\Rightarrow$(iii): For span-L{*determ*}$\subseteq$FL(NL)[log n], let $f\in$span-L{*determ*} and let M be an NL transducer that witnesses f. There are at most polynomially many different configurations in which M starts to write (deterministically) its output. Several of these configurations might correspond to the same valid output value. Define

$$\begin{aligned} L_M := \{x\$c \mid c \ & \text{is a configuration in which } M \text{ starts to write a valid output } w_c, \\ & \text{and for all configurations } c' \text{ with } c' > c \text{ in which } M \text{ starts to write} \\ & \text{a valid output } w_{c'}, w_{c'} \neq w_c\}. \end{aligned}$$

Since $span_M(x)$, the number of different output values, equals the number of different c which satisfy $x\$c \in L_M$, an L-transducer with oracle L_M can compute this value on input x by cycling through all the possible configurations c of M in lexicographical order. $L_M \in$NL holds, since NL is closed under complementation: $w_{c'} \neq w_c$ is an L-property, and the conditions specified for L_M yield one alternation (with logarithmic space).

(iii)$\Rightarrow$(iv): To prove the inclusion FL(NL)[log n]$\subseteq\mathrm{FL}_{\log}$(NL)[log n], let $f\in$FL(NL)[log n]. Then there is an L-transducer M with oracle $A\in$NL which outputs on input $x, f(x) = y$ with $|y| \leqslant c\log|x|$ for a constant c. Define

$$L_{(M,A)} := \{x\$i \mid \text{the } i\text{th output bit of } M \text{ with oracle } A \text{ on input } x \text{ is } 1\}.$$

Since NL is closed under complementation, it is easily seen that $L_{(M,A)}$ is contained in NL. By using $L_{(M,A)}$ as oracle, each of the $c\log|x|$ output bits of M can be found by using only a logarithmic number of questions.

(iv)$\Rightarrow$(i) follows from FL(NL)$\subseteq$opt-L (Theorem 4.2). □

The restriction $\{determ\}$ is slightly weaker than restricting the size of all the values of an NL-transducer to a logarithm in the length of the input. It is not hard to show that such a restriction leads to further characterizations of opt-L$[\log n]$ in terms of both the corresponding opt-L or span-L functions.

As shown in the following proposition, opt-L$\{determ\}$ and opt-L$[\log n]$ behave very similar. opt-L$\{determ\}$ coincides with the class of functions that are computable by an L-transducer with oracle in NL and a logarithmic bound on the number of queries.

Proposition 5.5. $\mathrm{FL}_{\log}(\mathrm{NL}) = \mathrm{FL}_1(\text{opt-L}[\log n]) = \text{opt-L}\{determ\}$.

Proof. For the inclusion $\mathrm{FL}_{\log}(\mathrm{NL}) \subseteq \mathrm{FL}_1(\text{opt-L}[\log n])$, it suffices with Proposition 5.4 to show that $\mathrm{FL}_{\log}(\mathrm{NL}) \subseteq \mathrm{FL}_1(\mathrm{FL}_{\log}(\mathrm{NL})[\log n])$. Let $f \in \mathrm{FL}_{\log}(\mathrm{NL})$, and let M be an L-transducer that computes $f(x)$ with at most $c\log|x|$ many queries to an oracle $A \in \mathrm{NL}$. Define $g(x) := b_1 b_2 \dots b_{m_x}$, where for all $1 \leqslant i \leqslant m_x \leqslant c\log|x|$ $b_i \in \{0, 1\}$ denotes the answer to the ith query of M to A on input x. g can be computed by an L-transducer M' with oracle A that simulates M and outputs the answers to the queries of M instead of M's output. f can be computed by an L-transducer with oracle g that on input x queries g only once for $g(x)$. $g(x)$ contains all the necessary information for finding the correct path in the "query tree" of M without asking the oracle A.

For the inclusion $\mathrm{FL}_1(\text{opt-L}[\log n]) \subseteq \text{opt-L}\{determ\}$, let $f \in \mathrm{FL}_1(\text{opt-L}[\log n])$ be computed by an L-transducer M that on input x queries $g \in \text{opt-L}[\log n]$ exactly once for $g(y)$ (y depends on x). We may assume that M produces no output before querying (this can always be achieved by recomputing the output that occurred before the query). Let N be an NL-transducer such that $g = opt_N$ and for all y it holds $opt_N(y) \leqslant c\log|y|$ for a constant c.

We first claim that the set $L_{opt_N} := \{(y, z) \mid z = opt_N(y)\}$ is contained in NL. With log-space and one alternation it can be verified that z is an output value of N on input y and that no output value z' with $z \leqslant z' \leqslant c\log|y|$ is produced by N' on input y. The claim follows since NL is closed under complementation.

Now we construct an NL-transducer M' that does the following on input x. M' first computes $|y| \leqslant |x|^k$, where $|x|^k$ denotes the polynomial time bound of M. Then M' guesses a value z such that $z \leqslant ck\log|x|$ and verifies that $z = g(y) = opt_N(y)$ by guessing the answer of $(y, z) \in L_{opt_N}$ and simulating correspondingly an NL-machine for L_{opt_N} or an NL-machine for the complement. For this simulation each bit of the query y has to be recomputed again by simulating M. With the correct value $g(y)$, M' finally produces the output $f(x)$ by simulating M. Since M is an L-transducer, the output by M' is produced deterministically. Note that M' only has *one* valid output $f(x) = opt_M(x)$.

For the inclusion $\text{opt-L}\{determ\} \subseteq \mathrm{FL}_{\log}(\mathrm{NL})$, let $f \in \text{opt-L}\{determ\}$ and let M be an NL-transducer such that $f(x) = \mathrm{opt}_M(x)$. On input x there are at most polynomially many different configurations of M in which M starts to write (deterministically) a valid output. All configurations have length $c\log|x|$ for a constant c. Thus, f can be computed by a deterministic transducer M', which first constructs on its work tape a configuration of M in which M starts to write, deterministically (!), its maximal valid output, and then simulates M from this configuration. M' can find such a configuration with $O(\log|x|)$ questions to the following oracle:

$$A := \{x\$z \mid z \text{ is a prefix of a configuration of } M \text{ of length } c\log|x|, \text{ in which } M \text{ starts to write the maximal valid output on input } x.\}.$$

We claim that $A \in \mathrm{NL}$. Construct an NL-machine N as follows: Given a word $x\$z$, N guesses a string v of length at most $c\log|x|$ such that zv codes a configuration of M, in which M starts to write the valid output w_{zv}. N then verifies that $w_c \leqslant w_{zv}$ for all configurations c of M in which M starts to write a valid output w_c. N verifies that c is a configuration in which M starts to write a valid output by cycling through all the possible configurations c of M in lexicographical order, and simulating M from the start. Furthermore, for comparing the output M produces starting in c with the output starting in zv N simulates M simultaneously starting in c and starting in zv. M accepts, if for all c the output produced in c is smaller than or equal to the output produced in zv. Since NL is closed under complementation, N is an NL-machine, and $L(N) = A \in \mathrm{NL}$. □

The proof also shows that single-valued NL-transducer that produce their output deterministically characterize the class $\mathrm{FL}_{\log}(\mathrm{NL})$.

The unrestricted class opt-L seems to be more powerful than opt-L{*determ*}. With Theorem 4.2 and Proposition 5.4 we have

$$\text{opt-L}\{\textit{determ}\} = \mathrm{FL}_{\log}(\mathrm{NL}) \subseteq \mathrm{FL}(\mathrm{NL}) \subseteq \text{opt-L} \subseteq \mathrm{NC}^2.$$

We will show that $\mathrm{FL}(\mathrm{NL}) \subseteq \mathrm{FL}_{\log}(\mathrm{NL})$ would have strong implications: Krentel's result, $\mathrm{FP}_{\log}(\mathrm{NP}) = \mathrm{FP}(\mathrm{NP})$ if and only if $\mathrm{P} = \mathrm{NP}$[20], can be translated to log-space classes.

Proposition 5.6. $\mathrm{FL}_{\log}(\mathrm{NL}) = \mathrm{FL}(\mathrm{NL})$ *if and only if* $\mathrm{L} = \mathrm{NL}$.

Proof. The implication from right to left is obvious. For the implication from left to right, recall that the graph accessibility problem, *GAP*, is complete for NL [16, 17]. Clearly, a deterministic transducer M with oracle *GAP* can be constructed that produces, given as input an instance of *GAP* (a graph and two specified nodes s and t), a path from s to t in the graph, if a path exists, and 0 otherwise. Let f be the function computed by M. By assumption, we have $f \in \mathrm{FL}_{\log}(\mathrm{NL})$. Let M' be the transducer that computes, on input x, $f(x)$ with the sequence of oracle answers s_x with $|s_x| \leqslant c\log|x|$ for a constant c. Construct an L-machine N that recognizes *GAP* as follows: N cycles through all binary strings $v \leqslant c\log|x|$. For each string v, N simulates M', interpreting v as the sequence of answers of the oracle M', and checks whether the output of M' corresponds to a path from s to t in the graph. Clearly, if N finds a correct path, then $x \in GAP$, and if there exists such a path, N will find it with the help of the correct oracle answer sequence s_x of M'. □

With a similar demonstration it can be shown that $f_{max\,\mathrm{DFA}} \in \mathrm{FL}_{\log}(\mathrm{NL}) \Leftrightarrow \mathrm{L} = \mathrm{NL}$. Hence, $f_{max\,\mathrm{DFA}}$ is an example of a "hardest" function in FL(NL) (see also Section 3).

Corollary 5.7. *If* opt-L $\subseteq$ opt-$L\{determ\}$, *then* L = NL.

Since it is not at all clear whether the result opt-L $\subseteq \mathrm{NC}^2$ can be improved to opt-L $\subseteq$ FL(NL), the implication from right to left remains open.

6. Conclusions

We have studied counting and optimization versions of nondeterministic log-space and obtained interesting complete problems for the corresponding classes #L, span-L, and opt-L. We showed that there exist span-L functions that are complete for #P with respect to metric reducibility. This result not only reveals the computational power of span-L, showing this class to be hard for the polynomial hierarchy, but also ties span-L to another already well-studied class. We, furthermore, showed that #L and opt-L functions have significantly less computational power; they are contained in the relatively low parallel classes DET* and NC^2, respectively. These are first upper bounds which do not tie these classes to other classes in terms of completeness. Recently, and independently of each other, Damm and Vinay could show that #L is in fact tied to DET* in this sense (the class DET* was introduced and studied by Cook in [5]). They showed that integer matrix powering can be computed by a difference of two #L functions [6, 32]. Since integer matrix powering is complete for DET* with respect to NC^1 Turing reducibility [5], their results imply DET* = NC^1(#L). It would be interesting to see whether the complexity of opt-L functions can be fixed in a similar way.

Acknowledgment

The authors thank José Balcázar, Fred Green, Jacobo Torán, and especially Celia Wrathall for many helpful comments on earlier versions of this article. Remarks of an anonymous referee were helpful to improve the presentation of the results.

Note added in proof

Recently we showed that opt-L is included in AC^1. E. Allender also obtained this result independently.

References

[1] C. Àlvarez, J.L. Balcázar and B. Jenner, Functional oracle queries as a measure of parallel time, in: *Proc. 8th STACS*, Lecture Notes in Computer Science, Vol. 480 (Springer, Berlin, 1991) 422–433.

[2] J.L. Balcázar, R.V. Book and U. Schöning, The polynomial-time hierarchy and sparse oracles, *J. ACM* **33** (1986) 603–617.

[3] A. Bertoni, M. Goldwurm and N. Sabadini, Computing the counting function of context-free languages, in: *Proc. 4th STACS*, Lecture Notes in Computer Science, Vol. 247 (Springer, Berlin, 1987) 169–179.

[4] A. Borodin, S.A. Cook, P.W. Dymond, W.L. Ruzzo and M. Tompa, Two applications of inductive counting for complementation problems, *SIAM J. Comput.* **18** (1989) 559–578.

[5] S.A. Cook, A taxonomy of problems with fast parallel algorithms, *Inform. and Comput.* **64** (1985) 2–22.

[6] C. Damm, manuscript, 1991.

[7] A.V. Goldberg and M. Sipser, Compression and ranking, in: *Proc. 17th ACM STOC* (1985) 59–68.

[8] J. Hartmanis and H.B. Hunt, The LBA problem and its importance in the theory of computing, in: R.M. Karp, ed., *Complexity of Computation*, SIAM-AMS Proceedings, Vol. 7 (American Mathematical Society, Providence, RI, 1974) 1–26.

[9] J. Hartmanis and S. Mahaney, Languages simultaneously complete for one-way and two-way automata, *SIAM J. Comput.* **10** (1981) 383–390.

[10] L. Hemachandra and S. Rudich, On the complexity of ranking, *J. Comput. System Sci.* **41** (1990) 251–271.

[11] J. Hopcroft and J. Ullman, Some results on tape-bounded Turing machines, *J. ACM* **16** (1969) 168–179.

[12] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).

[13] D.T. Huynh, The complexity of ranking simple languages, *Math. Systems Theory* **23** (1990) 1–19.

[14] N. Immerman, Nondeterministic space is closed under complement, *SIAM J. Comput.* **17** (1988) 935–938.

[15] B. Jenner and B. Kirsig, Characterizing the polynomial hierarchy by alternating auxiliary pushdown automata (extended version), *RAIRO Inform. Théor. Appl.* **23** (1989) 87–99.

[16] N.D. Jones, Space-bounded reducibility among combinatorial problems, *J. Comput. System Sci.* **11** (1975) 68–85.

[17] N.D. Jones, E. Lien and W.T. Laaser, New problems complete for nondeterministic log space, *Math. Systems Theory* **10** (1976) 1–17.

[18] J. Köbler, Strukturelle Komplexität von Anzahlproblemen, Doctoral dissertation (in German), Universität Stuttgart, 1989.

[19] J. Köbler, U. Schöning and J. Torán, On counting and approximation, *Acta Inform.* **26** (1989) 363–379.

[20] M.W. Krentel, The complexity of optimization problems, *J. Comput. System Sci.* **36** (1988) 490–509.

[21] R. Ladner, Polynomial space counting problems, *SIAM J. Comput.* **18** (1989) 1087–1097.

[22] R. Ladner and N. Lynch, Relativization of questions about log space computability, *Math. Systems Theory* **10** (1976) 19–32.

[23] K.J. Lange, Nondeterministic log-space reductions, in: *Proc. 11th MFCS*, Lecture Notes in Computer Science, Vol. 176 (Springer, Berlin, 1984) 378–388.

[24] W. Ruzzo, On uniform circuit complexity, *J. Comput. System Sci.* **22** (1981) 365–383.

[25] W. Ruzzo, J. Simon and M. Tompa, Space-bounded hierarchies and probabilistic computation, *J. Comput. System Sci.* **28** (1984) 216–230.

[26] U. Schöning, The power of counting, in: *Proc. 3rd Structure in Complexity Theory* (IEEE, 1988) 2–9.

[27] R. Szelepcsényi, The method of forced enumeration for nondeterministic automata, *Acta Inform.* **26** (1988) 279–284.

[28] S. Toda, On the computational power of PP and $\oplus$P, in: *Proc. 30th IEEE FOCS* (1989) 514–519.

[29] S. Toda and O. Watanabe, On the power of # P functions, manuscript, 1989 *Theoret. Comput. Sci.*, to appear.

[30] L.G. Valiant, The complexity of computing the permanent, *Theoret. Comput. Sci.* **8** (1979) 189–201.

[31] L.G. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Comput.* **8** (1979) 410–421.

[32] V. Vinay, Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits, in: *Proc. 6th Structure in Complexity Theory* (IEEE, 1991) 270–284.