Streaming algorithms for language recognition problems

Ajesh Babu[a], Nutan Limaye[b], Jaikumar Radhakrishnan[c], Girish Varma[c,]

[a] *Yahoo! Labs*
[b] *Indian Institute of Technology, Mumbai, India*
[c] *Tata Institute of Fundamental Research, Mumbai, India*

Abstract

We study the complexity of the following problems in the streaming model.

Membership testing for DLIN. We show that every language in DLIN can be recognised by a randomized one-pass $O(\log n)$ space algorithm with inverse polynomial one-sided error, and by a deterministic p-pass $O(n/p)$ space algorithm. We show that these algorithms are optimal.

Membership testing for LL(k). For languages generated by LL(k) grammars with a bound of r on the number of nonterminals at any stage in the left-most derivation, we show that membership can be tested by a randomized one-pass $O(r \log n)$ space algorithm with inverse polynomial (in n) one-sided error.

Membership testing for DCFL. We show that randomized algorithms as efficient as the ones described above for DLIN and LL(k) (which are subclasses of DCFL) cannot exist for all of DCFL: there is a language in VPL (a subclass of DCFL) for which any randomized p-pass algorithm with error bounded by $\epsilon < 1/2$ must use $\Omega(n/p)$ space.

Degree sequence problem. We study the problem of determining, given a sequence $d_1, d_2, \ldots, d_n$ and a graph G, whether the degree sequence of G is precisely $d_1, d_2, \ldots, d_n$. We give a randomized one-pass $O(\log n)$ space algorithm with inverse polynomial one-sided error probability. We show that our algorithms are optimal.

Our randomized algorithms are based on the recent work of Magniez et al. [1]; our lower bounds are obtained by considering related communication complexity problems.

Keywords: streaming algorithms, randomized algorithms, communication complexity, context free language

Email addresses: ajesh@yahoo-inc.com (Ajesh Babu), nutan@cse.iitb.ac.in (Nutan Limaye), jaikumar@tifr.res.in (Jaikumar Radhakrishnan), girish@tcs.tifr.res.in (Girish Varma)

1. Introduction

Modeling computational problems as language recognition is well-established in theoretical computer science. By studying the complexity of recognising languages, one seeks to understand the power and limitations of various computational models, and also classify problems according to their hardness. In this paper, we study language recognition problems in the data stream model.

The data stream model was invented to understand issues that arise in computations involving large amounts of data, when the processors have limited memory and are allowed limited access to the input (typically, restricted to a small number of passes over it). Such a situation arises when the input is in secondary storage and it is infeasible to load it all in the main memory. In recent years, this model has gained popularity for modeling the actions of routers and other agents on the internet that need to keep aggregate information about the packets that they handle; the number of packets is large, and the routers themselves are allowed only a small amount of memory. In this case, the final decision needs to be based on just one pass over the input.

In the data stream model, the two main parameters of interest are the memory available for processing and the number of passes allowed. An algorithm is considered efficient if the space it uses is significantly smaller than the input length (ideally, only polylogarithmic), and the number passes on the input is small (ideally, just one). Given these constraints, most interesting problems become intractable in this model if the algorithm is required to be *deterministic*. Randomness, however, is remarkably effective, and many interesting randomized algorithms have been proposed (starting with Alon et al. [2] and see the survey by Muthukrishnan [3]).

When the number of passes over the input is not restricted, or when random access to the input is available, the data stream model corresponds closely to the model of space bounded Turing machines. Often, techniques developed for such unrestricted space bounded computations, carry over to the data stream model with limited access to inputs (e.g. Nisan's pseudorandom generator [4] designed for derandomizing space bounded randomized computations, has been effectively employed in many data stream algorithms, starting with Indyk [5]). In this paper, we consider streaming algorithms for several language recognition problems that can be solved in $polylog(n)$ space on a Turing machine.

We will assume that the reader is familiar with basic formal language theory, in particular, the class of *context free languages* (CFL). Our results concern some subclasses of CFLs, namely DLIN, LL(k) and DCFL (we recall their definitions in Sections 2, 3 and 4). Slightly differing definitions for DLIN were first given by Ibarra et al. [6] and Nasu et al. [7]; the definition we use is due to Higuera et al. [8], where the several similar definitions are compared and a more general class is defined. It was shown by Holzer et al. [9] that membership in these languages can be tested in space $O(\log n)$. LL(k) languages were defined by Lewis et al. and Knuth [10, 11], and they play an important role in parsing theory. Informally, they are the languages for which the left-most derivation

can be obtained deterministically by making a single pass on the input from left to right with k-lookaheads. Apart from some technicalities arising from ϵ-rules in the grammar, the class $\mathsf{LL}(k)$ includes DLIN. It was shown by [12] that all deterministic context-free languages can be recognised in space $O(\log^2 n)$. In this paper, we examine if languages in DLIN and $\mathsf{LL}(k)$ admit similar efficient membership testing in the streaming model.

Our work is motivated by a recent membership testing algorithm of Magniez et al. [1] for the language Dyck_2, which is the language of balanced parentheses on two types of parentheses. The algorithm uses $O(\sqrt{n}\log n)$ space. We apply their fingerprinting based method to the subclass DLIN and also give a deterministic p-pass, $O(n/p)$ space algorithm.

Theorem 1. *For every $L \in \mathsf{DLIN}$,*

1. *there is a randomized one-pass $O(\log n)$ space streaming algorithm such that for $x \in \{0,1\}^n$*
   (a) *if $x \in L$ then the algorithm accepts with probability 1;*
   (b) *if $x \notin L$ then the algorithm rejects with probability at least $1 - \frac{1}{n}$.*
2. *there is a deterministic one-pass $O(n/p)$ space streaming algorithm for testing membership in L.*

(Note that our result does not generalize the result of [1] for Dyck_2, because Dyck_2 does not belong to DLIN.) However, Theorem 1 cannot be improved.

Theorem 2. *Let*

$$1\text{-}turn\text{-}\mathsf{Dyck}_2 = \{w\bar{w}^R : w \in \{(,[\}^n,\ n \geq 1\},$$

where $\bar{w}$ is the string obtained from w by replacing each opening parenthesis by its corresponding closing parenthesis; $\bar{w}^R$ is the reverse of $\bar{w}$.

1. *Any p-pass randomized streaming algorithm that determines membership in 1-turn-Dyck_2 with probability of error bounded by $\epsilon < \frac{1}{2}$, must use $\Omega((\log n)/p)$ space.*
2. *Any p-pass deterministic streaming algorithm that determines membership in 1-turn-Dyck_2 must use $\Omega(n/p)$ space.*

This result is obtained by deriving, from the streaming algorithm, a two-party communication protocol for determining if two strings are equal, and then appealing to known lower bounds for the communication problem.

We next investigate if efficient membership testing is possible for languages in classes larger than DLIN. Similar, fingerprinting based algorithms apply to the class $\mathsf{LL}(k)$, but their efficiency depends on a certain parameter based the underlying grammar G. In order to state our result precisely, we now define this parameter.

Let L be a language generated by an $\mathsf{LL}(k)$ grammar G. For a string $w \in L$, let $rank_G(w)$ denote the maximum number of nonterminals in any sentential form arising in the (unique) leftmost derivation generating w. Let the rank of the grammar, $rank_G : \mathbb{N} \to \mathbb{N}$, be defined as $rank_G(n) = \max_{w \in \{0,1\}^n \cap L(G)} rank_G(w)$. We will assume that $rank_G(n)$ is a well-behaved function, say it is log-space computable.

Theorem 3. *Let G be an $\mathsf{LL}(k)$ grammar. There is a randomized one-pass streaming algorithm that given an input $w \in \{0,1\}^n$ and a positive integer b, using space $O(b \log n)$(the dependence on k varies with the grammar),*

1. *accepts with probability 1 if $w \in L(G)$ and $rank_G(w) \leq b$;*
2. *rejects with probability at least $1 - \frac{1}{n}$ if $w \notin L(G)$ or $rank_G(w) > b$.*

Corollary 4. *Let L be a language generated by an $\mathsf{LL}(k)$ grammar G. There is a randomized one-pass streaming algorithm that given an input $w \in \{0,1\}^n$, using space $O(rank_G(n) \log n)$,*

1. *accepts with probability 1 if $x \in L$;*
2. *rejects with probability at least $1 - \frac{1}{n}$ if $x \notin L$.*

Note that the above result does not give efficient streaming algorithms unconditionally, for the space required depends on $rank_G(x)$, which in general may grow as $\Omega(n)$. Note, however, that results based on such properties of the derivation have been considered in the literature before. In fact, the class of left derivation bounded languages defined by Walljasper [13], consists precisely of languages for which $rank_G(n)$ is a constant independent of n; this class was also shown to be closed under AFL operations in [13]. Many well-studied classes of languages are subclasses of left derivation bounded languages. The nonterminal bounded languages generated by nonterminal bounded grammars (which have a bounded number of nonterminals in the sentential forms in *any* derivation) were studied by Workman [14], who also proved that they contain all ultralinear languages, which are languages accepted by finite turn pushdown automata (defined by Ginsburg et al. [15]). However, nonterminal bounded grammars need not be $\mathsf{LL}(k)$.

Despite the dependence on $rank_G(n)$, the above corollary is applicable to classes of languages such as rest-VPL defined in [16] (this considered the restriction of VPLs which have $\mathsf{LL}(1)$ grammars) and DLINs restricted to grammars without derivations of the form $A \to \epsilon$. For these classes $rank_G(n)$ is bounded by a constant independent of n.

We now turn to show that classes provably do not admit solutions in the streaming model with polylogarithmic space. Lower bounds for membership testing of context-free languages in the streaming model were studied by Magniez et al. [1]. They proved that any one-pass randomized algorithm requires $\Omega(\sqrt{n \log n})$ space for testing membership in Dyck_2. More recently, Jain et al. [17] proved that if the passes on the input are made only from left to right, then in spite of making p passes on the input, the membership testing for Dyck_2 requires $\Omega(\sqrt{n}/p)$ space. Here we prove that in general for languages in DCFL, no savings in space over the trivial algorithm of simulating the PDA can be expected.

Theorem 5. *There exists a language $L \in \mathsf{VPL} \subseteq \mathsf{DCFL}$ such that any randomized p-pass streaming algorithm requires $\Omega(n/p)$ space for testing membership in L with probability of error at most $\epsilon < \frac{1}{2}$.*

The language L in the above result is a slight modification of Dyck_2. This result is proved by reducing the membership problem in the streaming model to the two-party communication problem of checking whether two subsets of an an n-element universe are disjoint.

The upper bounds above show that the method of fingerprinting can be fruitfully applied to many problems to check equality of elements located far away in the input string. We provide one more illustration of the amazing power of this technique.

Degree-Sequence, Deg-Seq*:.* The degree sequence problem is the following.

Input: A positive integer n and sequence of directed edges

$$(u_1, v_1), (u_2, v_2), \ldots, (u_m, v_m) \text{ where } u_i, v_i \in \{1, 2, \ldots, n\}$$

on vertex set $\{1, 2, \ldots, n\}$.

Task: Determine if vertices $1, 2, \ldots, n$ have out-degrees $d_1, d_2, \ldots, d_n$, respectively?

This problem is known to be in log-space (in fact in TC^0 (see for example [18])). It has been observed [19, 20] that the complexity of graph problems changes drastically depending on the order in which the input is presented to the streaming algorithm. If the input to Deg-Seq is such that the degree of a vertex along with all the edges out of that vertex are listed one after the other, then checking whether the graph has the given degree sequence is trivial. If the degrees sequence is listed first, followed by the adjacency list of the graph then we observe that a one-pass deterministic algorithm needs $\Omega(n)$ space to compute Deg-Seq. For a more general ordering of the input where the degree sequence is followed by a list of edges in an arbitrary order, we prove the following theorem:

Theorem 6. *If the input is a degree sequence followed by a list of edges in an arbitrary order, then* Deg-Seq *can be solved*

1. *by a one-pass, $O(\log n)$ space randomized streaming algorithm such that if vertices $1, 2, \ldots, n$ have out-degrees $d_1, d_2, \ldots, d_n$, respectively, then the algorithm accepts with probability 1 and rejects with probability $1 - \frac{1}{n}$, otherwise.*
2. *by a p-passes, $O((n \log n)/p)$-space deterministic streaming algorithm.*

We also show that the above result is optimal up to a $\log n$ factor.

Theorem 7.

1. *Any p-pass randomized streaming algorithm for* Deg-Seq *with probability of error bounded by $\epsilon < \frac{1}{2}$, must use $\Omega((\log n)/p)$ space.*
2. *Any p-pass deterministic streaming algorithm for* Deg-Seq *must use $\Omega(n/p)$ space.*

2. Membership testing of DLIN

In this section, we study the complexity of membership testing for a subclass of context free languages called DLIN, in the streaming model. Informally it is the class of languages accepted by 1-turn PDA(i.e. PDA which do not make a push move after having made a pop move), with restrictions similar to LL(1).

We start with some definitions. See [21] for the basic definitions regarding context-free grammars(CFG) and pushdown automata(PDA).

Definition 1 (Higuera et al.[8])**.** ***Deterministic linear CFG*** *or* DL-CFG*, is a* CFG (Σ, N, P, S) *for which, every production is of the form* $A \to a\omega$ *or* $A \to \epsilon$*, where* $a \in \Sigma$ *and* $\omega \in (N \cup \{\epsilon\})\Sigma^*$ *and for any two productions,* $A \to a\omega$ *and* $B \to b\omega'$*, if* $A = B$ *then* $a \neq b$*, where* $a, b \in \Sigma$ *and* $\omega, \omega' \in (N \cup \{\epsilon\})\Sigma^*$*.*

Definition 2. ***Deterministic linear CFL****,* DLIN*, is the class of languages for which there exists a* DL-CFG *generating it.*

DLIN is a well studied class in language theory. Higuera et al. [8] gives algorithms for learning such grammars. Many variations of the above definition have been considered in earlier works. The above definition is more general than the ones given in [6, 9] as was proved in [8]. Note that the set of languages accepted by deterministic 1-turn PDA is a strict super-set of DLIN. For example $L = \{a^n b^n \text{ or } a^n c^n \mid n > 0\} \notin$ DLIN but is accepted by a deterministic 1-turn PDA.

Definition 3. ***Canonical Pushdown Automaton*** *or* CPDA *for a language* L *generated by a* CFG $G = (\Sigma, N, P, S)$ *is a* PDA $M_L = (Q = \{q\}, \Sigma, \Gamma = N \cup \Sigma, \delta, q_0 = q, S)$ *, where the transition function* δ *is defined as follows:*

1. *for each production of the form* $A \to a\omega$ *where* $\omega \in (N \cup \Sigma)^*$*,* $\delta(q, a, A) = (q, \omega)$[1]*.*
2. *for every production* $A \to \omega$ *that is not considered above,* $\delta(q, \epsilon, A) = (q, \omega)$*.*
3. *for all* $a \in \Sigma$*,* $\delta(q, a, a) = (q, \epsilon)$*.*

M_L *starts with only the start symbol* S *on the stack and it accepts by empty stack. The language accepted by* M_L *is* L*.*

If the rules of the form $A \to \epsilon$ are removed from a DL-CFG, then the corresponding CPDA is deterministic. However if the length of the string is known before hand, then we can infer when such a rule is to be applied. It is precisely when the sum of the length of the string seen so far and the number of nonterminals in the stack add up to the total length. So the CPDA can be simulated deterministically by making only a single pass over the input, but the stack can take up $\Omega(n)$ space. The algorithm for membership testing of DLIN is

[1] $\delta(q, a, A) = (q, \omega)$ implies that when the PDA is at state q, has a as the next input symbol and A on top of stack, will remain in state q, replacing A by ω at the top of the stack.

obtained by simulating the CPDA with a compressed stack. The stack is compressed by using a hash function which is a random evaluation of a polynomial constructed from the stack. This method commonly known as fingerprinting (see [22], Chapter 7) was used by Magniez et al. [1], for giving a streaming algorithm for membership testing of Dyck_2. Here we apply the technique to the class of DLIN. Note that Dyck_2 is not contained in DLIN.

2.1. Compressing the stack

First we make an observation about the stack of a CPDA for a language L, generated by a DL-CFG.

Observation 4. *For any string $w \in L$ and at any step $i \in [|w|]$, the stack of the* CPDA *contains at most one nonterminal.*

Consider the run of the CPDA on $w \in \Sigma^*$ in which any transition of the form $\delta(q, \epsilon, A) = (q, \epsilon)$ is applied only at the step i when the sum of $i-1$ and the number of terminals in the stack adds up to $|w|$. For $w \in \Sigma^*$, $i \in [|w|]$, let $\texttt{Stack}(w, i) \in \Sigma^*$ be the sequence of terminals in the stack of the CPDA, when it encounters the ith symbol of input w. We consider it from bottom to the first nonterminal or the top if there is no nonterminal. If the CPDA rejects before reaching i, then $\texttt{Stack}(w, i)$ is not defined. Similarly, let $\texttt{NonTerm}(w, i)$ be the unique nonterminal on top of the stack of the CPDA, when it has reached position i on input w. If there is no nonterminal in the stack, then it is ϵ.

We will assume a fixed bijective map from $\Sigma = \{a_1, a_2, \ldots, a_m\}$ to $[m] = \{1, 2, \ldots, m\}$. Furthermore we will use $\mathbf{a_i}$ to denote the value of the map on a_i. For any string $v \in \Sigma^*$, a prime p, formal variable x, let

$$\texttt{FP}(v, h, x, p) = \sum_{j=1}^{|v|} \mathbf{v[j]} x^{h+j-1} \mod p$$

be a polynomial over $\mathbb{F}_p$. Then

$$\texttt{CompStack}(w, i, x) = \texttt{FP}(\texttt{Stack}(w, i), 0, x, p)$$

can be considered as an encoding of $\texttt{Stack}(w, i)$.

Observation 5. *$\texttt{CompStack}(w, n, x)$ has degree at most n and is the zero polynomial if and only if $w \in L$.*

It is therefore sufficient to check whether $\texttt{CompStack}(w, n, x)$ is the zero polynomial for testing membership in L. To explicitly store this polynomial, $\Omega(n)$ space may be required. But a random evaluation of a non-zero degree d polynomial over $\mathbb{F}_p$ is zero with probability at most d/p (due to Schwartz Zippel Lemma). Hence it suffices to keep a random evaluation of $\texttt{CompStack}(w, i, x)$, which can be stored using just $\lceil \log p \rceil$ bits, for checking if it is zero. If $p = O(n)$ then the space needed is considerably reduced to $O(\log n)$.

Algorithm 1 Randomized one pass algorithm

Input : $w \in \Sigma^*$. Let $|w| = n$.
Pick α uniformly at random from $\mathbb{F}_p$.
$\texttt{comp\_stack} \leftarrow 0$; $\texttt{non\_term} \leftarrow S$; $h \leftarrow 0$
for $i = 1$ to n **do**
  if $\texttt{non\_term} \neq \epsilon$ **then**
    if $h + i - 1 = n$ **then**
      if a rule of the form $\texttt{non\_term} \rightarrow \epsilon$ does not exist **then** reject
      else $\texttt{non\_term} \leftarrow \epsilon$
    else
      Find the unique rule of the form below. Otherwise reject

$$\texttt{non\_term} \rightarrow w[i]Bv,\ v \in \Sigma^*, B \in N \cup \{\epsilon\}$$

      $\texttt{comp\_stack} \leftarrow \texttt{comp\_stack} + \texttt{FP}(v^R, h, \alpha, p) \mod p$
      {where v^R is v reversed}
      $\texttt{non\_term} \leftarrow B$; $h \leftarrow h + |v|$
    end if
  else
    $\texttt{comp\_stack} \leftarrow \texttt{comp\_stack} - \mathbf{w[i]}\alpha^{h-1} \mod p$
    $h \leftarrow h - 1$
  end if
end for
if $\texttt{comp\_stack} = 0$ and $h = 0$ **then** accept
else reject

2.2. Algorithm

The algorithm (Algorithm 1) is obtained by observing that the CPDA can be simulated using a compressed stack.

Algorithm 1 uses $\lceil \log p \rceil$ bits to store α, $\lceil \log p \rceil$ for `comp_stack`, $2\lceil \log n \rceil$ for i, h and some constant space that depends on the grammar for `non_term`. Hence the space complexity is $2\lceil \log p \rceil + 2\lceil \log n \rceil + c$. It also uses $\lceil \log p \rceil$ random bits.

2.3. Proof of Correctness

Lemma 8. *If the input is not rejected on or before the i^{th} iteration of for loop on line 4 of algorithm 1 then*

- $h = |\mathit{Stack}(w, i)|$
- $\mathit{comp\_stack} = \mathit{CompStack}(w, i, \alpha)$
- $\mathit{non\_term} = \mathit{NonTerm}(w, i)$.

Proof. The lemma is proved using induction on i. At $i = 1$, $h = 0$, $\texttt{comp\_stack} = \texttt{CompStack}(w, 1, \alpha) = 0$ and $\texttt{non\_term} = \texttt{NonTerm}(w, 1) = S$. Assuming above is true for the ith iteration of the loop. After the updates in line 11 (or 15), we have that $\texttt{comp\_stack} = \texttt{CompStack}(w, i, \alpha) + \sum_{j=1}^{|v|} \mathbf{v^R}[\mathbf{j}]\alpha^{h+j-1} \mod p = \texttt{CompStack}(w, i{+}1, \alpha)$ (or $\texttt{comp\_stack} = \texttt{CompStack}(w, i, \alpha) - w[i]\alpha^{h-1} \mod p = \texttt{CompStack}(w, i{+}1, \alpha)$, respectively). Similarly h and `non_term` are updated correctly in lines 8, 12 and 16. □

Applying Lemma 8 for $i = n$, we get the following corollary.

Corollary 9. *If $w \in L$ then Algorithm 1 accepts with probability 1.*

Lemma 10. *If $w \notin L$ then* $\Pr[\textit{Algorithm 1 accepts}] \leq n/p$.

Proof. If $w \notin L$ then CPDA rejects, say at step j. There are three cases.

1. $\texttt{NonTerm}(w, j)$ was defined and it rejected as a matching rule of the form $\texttt{NonTerm}(w, j) \to w[j]\omega,\ \omega \in \Sigma^*(N \cup \{\epsilon\})\Sigma^*$ could not be found.
2. $\texttt{NonTerm}(w, j)$ was not defined and it rejected as the last character of $\texttt{Stack}(w, j)$ was not $w[j]$.
3. the stack was not empty at the end of the string.

In the case 1, Algorithm 1 rejects with probability 1. For case 2, the monomial subtracted by the algorithm is $\mathbf{w}[\mathbf{j}]\alpha^{h-1}$. The only other monomial in the sum with the degree $h-1$ is $\mathbf{a}\alpha^{h-1}$ where a is the last character of $\texttt{Stack}(w, j)$. Also after the jth step no monomial of degree h is subtracted. So the polynomial for which `comp_stack` is an evaluation is not the zero polynomial. This is also true in case 3, as the stack is not empty. The lemma follows, by an application of the Schwartz Zippel Lemma. □

Theorem 1 is obtained by finding a prime p between n^2 and $2n^2$ by brute force search and then using Algorithm 1.

2.4. A deterministic multi-pass algorithm

In this section we give a deterministic multi-pass algorithm for the membership testing of any language in DLIN. This is done by first reducing the membership testing problem for any $L \in \mathsf{DLIN}$ to membership testing of a particular language $\mathsf{Dyck}_k \in \mathsf{DLIN}$. Recall that Dyck_k is the language generated by the grammar

$$S \to SS \mid ({}_1S)_1 \mid ({}_2S)_2 \mid \cdots \mid ({}_kS)_k \mid \epsilon$$

and 1-turn-Dyck_2 is generated by

$$S \to (S) \mid [S] \mid \epsilon.$$

We will be using the following definition of streaming reduction:

Definition 6 (Streaming Reduction)**.** *Fix two alphabets Σ_1 and Σ_2. A problem P_1 is $f(n)$-streaming reducible to a problem P_2 in space $s(n)$ if for every input $x \in \Sigma_1^n$, there exists $y_1 y_2 \dots y_n$ with*

$$y_i \in \cup_{i=1}^{f(n)} \Sigma_2^i \cup \{\epsilon\}$$

such that:

- *y_i can be computed from x_i using space $s(n)$.*
- *From a solution of P_2 on input y, a solution on P_1 on input x can be computed in space $s(n)$.*

Note that our definition is a slight modification of the definition from [1][2]. In [1], it was observed that the membership testing of Dyck_k $O(\log k)$-streaming reduces in $O(\log k)$ space to membership testing of Dyck_2. We show that the membership testing for any language in DLIN $O(1)$-streaming reduces in $O(\log n)$ space to membership testing in 1-turn-Dyck_k, where k is the alphabet size of the language. It is easy to see that in the the reduction of Magniez et al. [1], the output of the reduction is in 1-turn-Dyck_2 if and only if the input is in 1-turn-Dyck_k. Hence we have the following theorem:

Theorem 11. *The membership testing for any language in* DLIN *$O(\log |\Sigma|)$-streaming reduces in $O(\log n)$ space to membership testing in 1-turn-Dyck_2, where Σ is the alphabet of the language.*

Say L is a fixed DLIN, with $\Sigma = \{a_1, a_2, \dots, a_k\}$. Given an input w, the streaming reduction outputs a string $w' \in \Sigma \cup \overline{\Sigma}$ so that w' is in 1-turn-Dyck_k if and only if w belongs to L. Here $\overline{\Sigma} = \{\overline{a_1}, \overline{a_2}, \dots, \overline{a_k}\}$ and for each $i \in [k]$ $(a_i, \overline{a_i})$ is a matching pair. The streaming reduction is obtained by making a change to the steps 11 and 15 of Algorithm 1 and is given as Algorithm 2.

[2] In [1], y_i s are assumed to be of fixed length, i.e. from $\Sigma_2^{f(n)}$

Algorithm 2 Streaming reduction from $L \in \mathsf{DLIN}$ to Dyck_k

Input : $w \in \Sigma^*$. Let $|w| = n$.
Output : $w' \in \Sigma \cup \overline{\Sigma}$
$\texttt{non\_term} \leftarrow S$; $w' \leftarrow \epsilon$
$i \leftarrow 1$
while $i \leq n$ **do**
  if $\texttt{non\_term} \neq \epsilon$ **then**
    if $|w'| + i - 1 = n$ **then**
      if a rule of the form $\texttt{non\_term} \rightarrow \epsilon$ does not exist **then** reject
      else $\texttt{non\_term} \leftarrow \epsilon$
    else
      Find the unique rule of the form below. Otherwise reject

$$\texttt{non\_term} \rightarrow w[i]Bv,\ v \in \Sigma^*, B \in N \cup \{\epsilon\}$$

      $w' \leftarrow w' \cdot v^R$
      $\texttt{non\_term} \leftarrow B$; $i \leftarrow i + 1$
    end if
  else
    $w' = w' \cdot \overline{w[i]}$; $i \leftarrow i + 1$
  end if
end while

From Theorem 11, we know that any language in DLIN $O(\log |\Sigma|)$-streaming reduces to 1-turn-Dyck_2. Thus it suffices to give a p-passes, $O(n/p)$-space deterministic algorithm for membership testing of 1-turn-Dyck_2.

The algorithm divides the string into blocks of length $n/2p$. Let the blocks be called $B_0, B_1, \ldots, B_{2p-1}$ from left to right. (i.e. $B_i = w[i(n/2p)+1]\, w[i(n/2p)+2] \ldots w[(i+1)n/2p]$.) The algorithm considers a pair of blocks $(B_j, B_{2p-(j+1)})$ during the jth pass. Using the stack explicitly, the algorithm checks whether the string formed by the concatenation of B_j and $B_{2p-(j+1)}$ is balanced. If it is balanced, it proceeds to the next pair of blocks. The number of passes required is p. Each pass uses $O(n/p)$ space and the algorithm is deterministic. Later in Section 4 we show that this algorithm is optimal.

3. Membership Testing of LL(k) languages

In this section we give a randomized streaming algorithm for testing membership in LL(k) languages. Let $G = (N, \Sigma, P, S)$ be a fixed grammar. For a string $w \in \Sigma^*$, let

$$\text{pref}_k(w) = \begin{cases} \text{if } |w| > k \text{ then the first } k \text{ characters of } w \\ \text{else } w \end{cases} .$$

The *select set* of a production $A \to \alpha$, where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$ is

$$\texttt{SELECT}(A \to \alpha) = \{u \mid \exists v, w \in \Sigma^*, \alpha v \text{ derives } w \text{ and } \text{pref}_k(w) = u\}.$$

Definition 7 (Lewis et al. [10])**.** *A grammar $G = (N, \Sigma, P, S)$ is* LL(k) *if for any two distinct productions of the form $A \to \alpha,\ A \to \beta$, the select sets are disjoint.* LL(k) *languages are the class of languages generated by* LL(k) *grammars.*

From now on, we describe an algorithm for LL(1) languages. It is easy to observe that it generalises for LL(k) languages. Let L be a language generated by an LL(1) grammar G. It is known that for any two distinct rules $R \neq R'$ in the production set of G with the same left side, SELECT(R) and SELECT(R') are disjoint. We call this the LL(1) property. Note that DL-CFGs with no epsilon rules have this property. Therefore, languages generated by DL-CFGs with no epsilon rules, are a subclass of LL(1). As noted by Kurki-Suonio [23], they are in fact a proper subclass of languages generated by LL(1) grammars with no epsilon rules. As the part of the preprocessing, for every rule R of the grammar we compute the set SELECT(R). This requires only $O(1)$ space as the grammar is fixed.

Our membership testing algorithm for DLIN uses the LL(1) property non trivially. Algorithm 1 can be thought of as working in two main steps. The first step involves reading a terminal from the input and deciding the next rule to be applied. The second step consists of updating the stack appropriately. The LL(1) property enables the CPDA to deterministically decide the next rule to be applied having seen the next input terminal. Therefore, the first step will remain unchanged even in the case of membership testing of LL(1) languages. In what follows we describe the second step.

Let $\Gamma_k\gamma_k \dots \Gamma_0\gamma_0,\ \gamma_i \in \Sigma^*$ and $\Gamma_i \in N$ be any sentential form arising in the derivation of $w \in L$. Then the corresponding CPDA will store this in the stack(in the above order from top to bottom). It is easy to see that the CPDA is generating the left most derivation of w. The space efficient algorithm that we give below compresses the strings γ_is as before and stores Γ_i, compression of γ_i and $|\gamma_i|$ as a tuple on the stack. For a string $w \in L$, the algorithm runs in space $O(rank(w)(\log p + \log n))$, where p is the size of the field over which the polynomial is evaluated.

3.1. Streaming algorithm for testing membership in LL(1) *languages*

Given below is the randomized streaming algorithm for testing membership in LL(1) languages.

Algorithm 3 Randomized one pass algorithm

Input : $w \in \Sigma^*$. Let $|w| = n$.
Pick α uniformly at random from $\mathbb{F}_p$.
$\texttt{comp\_part} \leftarrow 0$; $\texttt{non\_term} \leftarrow S$; $h \leftarrow 0$
$\texttt{comp\_stack}.\text{push}(\texttt{comp\_part}, \texttt{non\_term}, h)$
$i \leftarrow 1$
while $i \leq n$ and $\texttt{comp\_stack}$ not empty **do**
  $(\texttt{comp\_part}, \texttt{non\_term}, h) \leftarrow \texttt{comp\_stack}.\text{pop}()$
  if $\texttt{non\_term} \neq \epsilon$ **then**
    Find the unique rule R of the form below such that $w[i] \in \texttt{SELECT}(R)$. Otherwise reject.

    $\texttt{non\_term} \longrightarrow B_t\beta_t B_{t-1}\beta_{t-1} \ldots B_0\beta_0$ where all $\beta_i \in \Sigma^*$, and $B_i \in N \cup \{\epsilon\}$

    $\texttt{comp\_stack}.\text{push}(\texttt{comp\_part} + \texttt{FP}(\beta_0^R, h, \alpha, p), B_0, h + |\beta_0|)$
    for $k \leftarrow 1$ to t **do**
      $\texttt{comp\_stack}.\text{push}(\texttt{FP}(\beta_k^R, 0, \alpha, p), B_k, |\beta_k|)$ \\ $\beta_i^R = reverse(\beta_i)$
    end for
  else
    if $h \neq 0$ **then**
      $\texttt{comp\_part} \leftarrow \texttt{comp\_part} - \mathbf{w}[\mathbf{i}]\alpha^{h-1} \mod p$; $h \leftarrow h - 1$
      $\texttt{comp\_stack}.\text{push}(\texttt{comp\_part}, \texttt{non\_term}, h)$
    else if $\texttt{comp\_part} \neq 0$ **then**
      reject
    end if
    $i \leftarrow i + 1$
  end if
end while
if $\texttt{comp\_stack}$ is not empty **then** reject **else** accept

The algorithm uses $\lceil \log p \rceil$ space to store α, $\lceil \log p \rceil + O(1) + \log n$ space to store a tuple on the stack. On input w, the space used by the algorithm is at most $rank_G(w)(\log n + \lceil \log p \rceil + O(1))$. Therefore, for a language generated by grammar G, the space used by the algorithm for checking $w \in L$ is at most $O(rank_G(n)(\log n + \log p))$. For proving Theorem 3, Algorithm 3 can be modified to to take an additional parameter b as an input and reject when $w \notin L$ or the number of items in the stack exceeds b. Also p can be set to a prime between n^2 and $2n^2$ which can be found by brute force search, so that error probability $n/p \leq 1/n$.

3.2. Correctness of the algorithm

In this section we prove the correctness of the algorithm. Note that, given an LL(1) grammar, the simulating CPDA performs a top-down parsing of the grammar. On reading a symbol from the input, and the top of the stack, it deterministically picks a rule to be applied next. If no such rule exists, it halts and rejects. If such a rule is found, it pushes the right hand side into the stack. As long as the stack-top is a nonterminal it repeats this process. If the stack-top is a terminal, it pops the top terminal from the stack, provided it matches with the next input letter. Suppose there is a mismatch, it halts and rejects. If after processing the whole string the stack is empty, it accepts.

We now prove that the working of the algorithm has a close correspondence with the working of the CPDA.

Lemma 12. *Let* $\mathit{Stack}(t) = \Gamma_k\gamma_k \ldots \Gamma_0\gamma_0,\ \Gamma_i \in N, \gamma_i \in \Sigma^*$ *be the contents of the stack of* CPDA *before the* t^{th} *step(counted in terms of application of the transition function) and*

$$\mathit{stack}(t) = [(\mathit{comp\_part}_j, \mathit{non\_term}_j, h_j), \ldots, (\mathit{comp\_part}_0, \mathit{non\_term}_0, h_0)]$$

be the contents of the stack of Algorithm 3 before the t^{th} *iteration of the while loop in line* 6. *If the* CPDA *has not rejected on or before step* t *then* $j = k$ *and* $\forall i \in \{0, \cdots k\}$,

- $\mathit{comp\_part}_i = \mathit{FP}(\gamma_i, 0, \alpha, p)$ [3]
- $\mathit{non\_term}_i = \Gamma_i$
- $h_i = |\gamma_i|$

Proof. The lemma can be proved by induction on t. At $t = 1$, $\mathtt{Stack}(1) = S$, $\mathtt{stack}(1) = [(0, S, 0)]$(due to the initialisation steps $3, 4$) and the lemma is true. Suppose it is true at step t, we will prove that the lemma holds at $t + 1$st step. We consider various cases. Assume that $\Gamma_k \neq \epsilon$ in $\mathtt{Stack}(t)$. Therefore, by inductive hypothesis the stack-top maintained by the algorithm has Γ_k in its second component. Then steps $9, 10, 12, 16$, makes sure that updates are made

[3] FP was defined in Section 2.1

correctly. Suppose $\Gamma_k = \epsilon$ and $\gamma_k = av,\ a \in \Sigma, v \in \Sigma^*$ then by inductive hypothesis, the top most item of $\texttt{stack}(t)$ is $(\texttt{FP}(\gamma_k, 0, \alpha, p), \epsilon, |\gamma_k|)$. By definition $\texttt{FP}(\gamma_k, 0, \alpha, p) = a\alpha^{|\gamma_k|-1} + \texttt{FP}(v, 0, \alpha, p)$. If $|v| > 0$ then after the execution of step 17, this will become $(\texttt{FP}(v, 0, \alpha, p), \epsilon, |v|)$ which is same as the top most item of $\texttt{Stack}(t+1)$. On the other hand if $v = \epsilon$, this item not push back in to the stack. □

Lemma 13. *If $w \in L$ then the algorithm accepts with probability 1. If $w \notin L$ then the probability that the algorithm accepts is bounded by n/p.*

Proof. If $w \in L$ then by Lemma 12 we have that the algorithm always accepts. Suppose the CPDA rejects at a certain step t, when symbol at the t'th position of the input was accessed. Let the There are three cases:

1. CPDA had a non-terminal on the stack-top and it rejected as a matching rule to be applied could not be found.
2. CPDA had a terminal on the stack-top, say a and it rejected because $a \neq w[t']$.
3. the stack was not empty at the end of the string.

In Case 1, the algorithm rejects with probability 1. For Case 2, let the top most item in the stack of the algorithm at step t be $(\texttt{comp\_part}, \texttt{non\_term}, h)$. Then the algorithm subtracts $\mathbf{w}[\mathbf{j}]\alpha^{h-1}$ from the stack and decreases the height by 1. The only other monomial in comp_part with degree $h-1$ is $\mathbf{a}\alpha^{h-1}$. Hence comp_part is a random evaluation of a nonzero polynomial of degree at most n. From Lemma 12, $\texttt{non\_term} = \epsilon$ and hence no other monomial of degree h is added or subtracted from comp_part. Now either the stack item $(\texttt{comp\_part}, \texttt{non\_term}, h)$ is never popped, or at the time of popping comp_part is checked to be zero. In the former case, the algorithm rejects with probability 1 and in the latter with probability at least $1 - n/p$. In Case 3 the algorithm rejects with probability 1. □

Now Theorem 3 follows from the above lemma by appropriately selecting the value of p to be a prime between n^{c+1} and $2n^{c+1}$. Such a prime can be obtained in time polynomial in n by exhaustive search.

4. Lower bounds for membership testing

In this section, we prove that the algorithms given in Section 2 are optimal.

Proof of Theorem 2. We reduce the two-party communication problem of testing equality ($\forall x, y \in \{0,1\}^n$, $\mathsf{EQUALITY}(x,y) = 1 \leftrightarrow x = y$) of strings to membership testing for 1-turn-Dyck_2. In this communication problem, the first party, Alice, is given a string x and the other party, Bob, is given the string y, and they need to communicate to determine if x and y are equal.

Suppose there is a p-pass streaming algorithm for 1-turn-Dyck_2 using space s. We will show that such an algorithm leads to protocol for the communication problem, where the total communication is $(2p-1)s$. First, Alice and Bob transform their inputs as follows. Let x' be the string obtained from x by replacing every 0 by a [and every 1 by a (; let y' be the string obtained from y by first reversing it and then replacing $0, 1$ by $],)$ respectively. Note that the string $z = x'y' \in \{(,[,],)\}^{2n} \in$ 1-turn-Dyck_2 iff $x = y$. Alice and Bob will simulate the streaming algorithm on z in the following natural way: Alice runs the streaming algorithm on x' and on reaching the end of the her input, passes on the contents of the memory to Bob who continues the simulation on y' and passes the contents of the memory back to Alice at the end. If there algorithm makes p (left to right) passes, then during the simulation the contents of the memory change hands $2p-1$ times. If the algorithm is deterministic, the protocol is deterministic. If the algorithm is randomized, the protocol is randomized and has the same error probability.

Since any deterministic protocol for $\mathsf{EQUALITY}(x,y)$ requires n bits of communication and any randomized protocol requires $\Omega(\log n)$ of communication for (error bounded by a constant strictly less than $\frac{1}{2}$) (see for example [24]), both our claims follow immediately. □

We now establish our lower bound for DCFLs.

Proof of Theorem 5. Consider the language L generated by the CFG with rules

$$S \to [S] \mid [S) \mid (S] \mid \epsilon.$$

Note that L is in DCFL; in fact, it is a VPL.

It is easy to verify that two strings $x, y \in \{0,1\}^n$ represent characteristic vectors of disjoint subsets of $\{1, 2, \ldots, n\}$ iff the string $x'y' \in L$, where x' is obtained from x and y' from y exactly as in the proof of Theorem 5. Thus, a p-pass space s streaming algorithm for membership testing in L can be used to derive a protocol for the set disjointness problem using communication $(2p-1)s$. Since the bounded error randomized communication complexity of the set disjointness problem is $\Omega(n)$ (see [24]), our claim follows immediately. □

5. Streaming algorithms for checking degree sequence of graphs

In this section, we study the complexity of solving the problem Deg-Seq defined in Section 1. We present the proof of the first part of Theorem 6.

Proof of part 1 *of Theorem 6.* We come up with a uni-variate polynomial from the given degree sequence and the set of edges such that the polynomial is identically zero if and only if the graph has the given degree sequence.

We do not store the polynomial explicitly. Instead, we evaluate this polynomial at a random point chosen from a large enough field and only maintain the evaluation of the polynomial. The Schwartz-Zippel lemma [22] gives us that with high probability the evaluation will be non-zero if the polynomial is non-zero. (If the polynomial is identically zero, its evaluation will also be zero.)

Let the vertex set of the graph be $\{1, \ldots, n\}$. The uni-variate polynomial that we construct is:

$$q(x) = \sum_i d_i x^i - \sum_{i=1}^{m} x^{u_i}$$

The algorithm can be now described as:

Algorithm 4 Randomized streaming algorithm for Deg-Seq

Pick $\alpha \in_R \mathbb{F}_p$($p$ will be fixed later).
$Sum \leftarrow 0$
for $i = 1$ to n **do**
  $Sum \leftarrow Sum + d_i \alpha^i$
end for
for $i = 1$ to m (where m number of edges) **do**
  $Sum \leftarrow Sum - \alpha^{u_i}$
end for
if $Sum = 0$ **then**
  accept
else
  reject
end if

It is easy to note that the algorithm requires only log-space as long as p is $O(poly(n))$. The input is being read only once from left to right. For the correctness, note that if the given degree sequence corresponds to that of the given graph, then $q(x)$ is identically zero and the value of Sum is also zero for any randomly picked α. We know that $q(x)$ is non-zero when the given degree sequence does not correspond to that of the given graph. However, the evaluation may still be zero. Note that degree of $q(x)$ is n. If the field size is chosen to be $n^{1+c} \leq p \leq n^{2+c}$ then due to Schwartz-Zippel lemma [22] the probability that Sum is zero given that $q(x)$ is non-zero is at most n/p which is at most n^{-c}. □

Now we give a p-pass, $O((n \log n)/p)$-space deterministic algorithm for Deg-Seq and hence prove part 2 of Theorem 6. The algorithm simply stores the degrees of n/p vertices during a pass and checks whether those vertices have exactly the degree sequence as stored. If the degree sequence is correct, then proceed to the next set of n/p vertices. The algorithm needs to store $O((n \log n)/p)$ bits during any pass. The algorithm makes p-passes.

Finally we show that both the algorithms presented for Deg-Seq is optimal up to a $\log n$ factor, by proving Theorem 7.

Proof of Theorem 7. We reduce the two party communication problem of testing equality to that of Deg-Seq. Given strings $x, y \in \{0,1\}^n$ we obtain a degree sequence $d = (d_1, d_2, \cdots, d_n)$ and a list of edges $e_1 e_2 \cdots e_m$. Take $d_i = x_i$ and for each i such that $y_i = 1$, add an edge (i, i). Clearly EQUALITY$(x, y) = 1$ if and only if d is the degree sequence of the graph with edges $e_1 e_2 \cdots e_m$. Again, as in proof of Theorem 2, the theorem follows because of the known communication complexity lower bounds for EQUALITY. □

References

[1] F. Magniez, C. Mathieu, A. Nayak, Recognizing well-parenthesized expressions in the streaming model, in: STOC.

[2] N. Alon, Y. Matias, M. Szegedy, The space complexity of approximating the frequency moments, in: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, STOC '96, ACM, New York, NY, USA, 1996, pp. 20–29.

[3] S. Muthukrishnan, Data streams: algorithms and applications, in: SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 413–413.

[4] N. Nisan, Pseudorandom generators for space-bounded computations, in: Proceedings of the twenty-second annual ACM symposium on Theory of computing, STOC '90, ACM, New York, NY, USA, 1990, pp. 204–212.

[5] P. Indyk, Stable distributions, pseudorandom generators, embeddings, and data stream computation, J. ACM 53 (2006) 307–323.

[6] O. Ibarra, T. Jiang, B. Ravikumar, Some subclasses of context-free languages in NC^1., Information Processing Letters 29 (1988) 111–117.

[7] M. Nasu, N. Honda, Mappings induced by pgsm-mappings and some recursively unsolvable problems of finite probabilistic automata, Information and Control 15 (1969) 250 – 273.

[8] C. D. L. Higuera, J. Oncina, Learning deterministic linear languages, in: In: Computational Learning Theory, COLT 02. Number 2375 in Lecture Notes in Artificial Intelligence, Springer Verlag, 2002, pp. 185–200.

[9] M. Holzer, K.-J. Lange, On the complexities of linear LL(1) and LR(1) grammars, in: FCT '93: Proceedings of the 9th International Symposium on Fundamentals of Computation Theory, Springer, London, UK, 1993, pp. 299–308.

[10] P. M. Lewis, II, R. E. Stearns, Syntax-directed transduction, J. ACM 15 (1968) 465–488.

[11] D. E. Knuth, Top-down syntax analysis, Acta Informatica 1 (1971) 79–110. 10.1007/BF00289517.

[12] S. A. Cook, Deterministic CFL's are accepted simultaneously in polynomial time and log squared space, in: Proceedings of the eleventh annual ACM symposium on Theory of computing, STOC '79, pp. 338–345.

[13] S. Walljasper, Left-derivation bounded languages, Journal of Computer and System Sciences 8 (1974) 1 – 7.

[14] D. Workman, Turn-bounded grammars and their relation to ultralinear languages, Information and Control 32 (1976) 188 – 200.

[15] S. Ginsburg, E. H. Spanier, Finite-turn pushdown automata, SIAM Journal on Control 4 (1966) 429–453.

[16] A. Babu, N. Limaye, G. Varma, Streaming algorithms for some problems in log-space, in: J. Kratochvl, A. Li, J. Fiala, P. Kolman (Eds.), Theory and Applications of Models of Computation, volume 6108 of *Lecture Notes in Computer Science*, Springer Berlin, Heidelberg, 2010, pp. 94–104. 10.1007/978-3-642-13562-0-10.

[17] R. Jain, A. Nayak, The Space Complexity of Recognizing Well-Parenthesized Expressions, Technical Report TR10-071, Electronic Colloquium on Computational Complexity, `http://eccc.hpi-web.de/`, 2010. Revised July 5, 2010.

[18] H. Vollmer, Introduction to Circuit Complexity: A Uniform Approach, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[19] D. N. Atish Das Sarma, Richard J. Lipton, Best-order streaming model, in: The 6th Annual Conference on Theory and Applications of Models of Computation (TAMC), pp. 178–191.

[20] J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan, Testing and spot-checking of data streams, Algorithmica 34 (2002) 67–80.

[21] J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation (3rd Edition), Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[22] R. Motwani, P. Raghavan, Randomized algorithms, Cambridge University Press, New York, NY, USA, 1995.

[23] R. Kurki-Suonio, Notes on top-down languages, BIT Numerical Mathematics 9 (1969) 225–238. 10.1007/BF01946814.

[24] E. Kushilevitz, N. Nisan, Communication Complexity, Cambridge University Press, New York, NY, USA, 2006.