# Sliding Window Property Testing for Regular Languages

**Moses Ganardi**
Universität Siegen, Germany
ganardi@eti.uni-siegen.de

**Danny Hucke**
Universität Siegen, Germany
hucke@eti.uni-siegen.de

**Markus Lohrey**
Universität Siegen, Germany
lohrey@eti.uni-siegen.de

**Tatiana Starikovskaya**
DI/ENS, PSL Research University, Paris, France
tat.starikovskaya@gmail.com

**Abstract**

We study the problem of recognizing regular languages in a variant of the streaming model of computation, called the sliding window model. In this model, we are given a size of the sliding window $n$ and a stream of symbols. At each time instant, we must decide whether the suffix of length $n$ of the current stream (“the active window”) belongs to a given regular language.

Recent works [14, 15] showed that the space complexity of an optimal deterministic sliding window algorithm for this problem is either constant, logarithmic or linear in the window size $n$ and provided natural language theoretic characterizations of the space complexity classes. Subsequently, [16] extended this result to randomized algorithms to show that any such algorithm admits either constant, double logarithmic, logarithmic or linear space complexity.

In this work, we make an important step forward and combine the sliding window model with the property testing setting, which results in ultra-efficient algorithms for all regular languages. Informally, a sliding window property tester must accept the active window if it belongs to the language and reject it if it is far from the language. We show that for every regular language, there is a deterministic sliding window property tester that uses logarithmic space and a randomized sliding window property tester with two-sided error that uses constant space.

**2012 ACM Subject Classification** Theory of computation → Streaming, sublinear and near linear time algorithms

**Keywords and phrases** Streaming algorithms, approximation algorithms, regular languages

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2019.6

**Related Version** A full version of this paper is available at `https://arxiv.org/abs/1909.10261`.

**Funding** *Moses Ganardi*: Funded by DFG project LO 748/13-1.
*Markus Lohrey*: Funded by DFG project LO 748/13-1.

## 1 Introduction

Regular expression search constitutes an important part of many search engines for biological data or code, such as, for example, Elasticsearch Service[1]. In this paper, we consider the following formalization of this problem. We assume to be given an integer $n$, a regular

[1] `https://www.elastic.co`

language $L$, and a stream of symbols that we receive one symbol at a time. At each time instant, we have direct access only to the last arrived symbol, and must decide whether the suffix of length $n$ of the current stream ("the active window") belongs to $L$.

The model described above is a variant of the streaming model and was introduced by Datar et al. [10], where the authors proved that the number of 1's in a 0/1-sliding window of size $n$ can be maintained in space $\mathcal{O}(\frac{1}{\epsilon} \cdot \log^2 n)$ if one allows a multiplicative error of $1 \pm \epsilon$. The motivation for this model of computation is that in many streaming applications, data items are outdated after a certain time, and the sliding window setting is a simple way to model this. In general, we aim to avoid storing the window content explicitly, and, instead, to work in considerably smaller space, e.g. polylogarithmic space with respect to the window length. For more details on the sliding window model see [1, Chapter 8].

The study of recognizing regular languages in the sliding window model was commenced in [14, 15]. In [15], Ganardi et al. showed that for every regular language $L$ the optimal space bound for a deterministic sliding window algorithm is either constant, logarithmic or linear in the window size $n$. In [14], Ganardi et al. gave characterizations for these space classes. More formally, they showed that a regular language has a deterministic sliding window algorithm with space $\mathcal{O}(\log n)$ (resp., $\mathcal{O}(1)$) if and only if it is a Boolean combination of so-called regular left-ideals and regular length languages (resp., suffix-testable languages and regular length languages). A subsequent work [16] studied the space complexity of randomized sliding window algorithms for regular languages. It was shown that for every regular language $L$ the optimal space bound of randomized sliding window algorithm is $\mathcal{O}(1)$, $\mathcal{O}(\log \log n)$, $\mathcal{O}(\log n)$, or $\mathcal{O}(n)$. Moreover, complete characterizations of these space classes were provided.

## 1.1 Our results

Previous study implies that even simple languages require linear space in the sliding window model, which gives the motivation to seek for novel approaches in order to achieve efficient algorithms for all regular languages. We take our inspiration from the property testing model introduced by Goldreich et. al [22]. In this model, the task is to decide whether the input has a particular property $P$, or is "far" from any input satisfying it. For a function $\gamma : \mathbb{N} \to \mathbb{R}_{\geq 0}$, we say that a word $w$ of length $n$ is $\gamma$-far from satisfying $P$, if the Hamming distance between $w$ and any word $w'$ satisfying $P$ is at least $\gamma(n)$. We will call the function $\gamma(n)$ the Hamming gap of the tester. We must make the decision by inspecting as few symbols of the input as possible, and the time complexity of the algorithm is defined to be equal to the number of inspected symbols. The motivation is that when working with large-scale data, accessing a data item is a very time-expensive operation. The membership problem for a regular language in the property testing model was studied by Alon et al. [2] who showed that for every regular language $L$ and every constant $\epsilon > 0$, there is a property tester with Hamming gap $\gamma(n) = \epsilon n$ for deciding membership in $L$ that can make the decision by inspecting a random constant-size sample of symbols of the input word.

In this work, we introduce a class of algorithms called *sliding window property testers*. Informally, at each time moment, a sliding window property tester must accept if the active window has the property $P$ and reject if it is far from satisfying $P$. The space complexity of a sliding window property tester is defined to be all the space used, including the space we need to store information about the input. We consider deterministic sliding window property testers and randomized sliding window property testers with one-sided and two-sided errors (for a formal definition, see Section 2). A similar but simpler model of streaming property testers, where the whole stream is considered, was introduced by Feigenbaum et al. [11].

François et al. [12] continued the study of this model in the context of language membership problems and came up with a streaming property tester for visibly pushdown languages that uses polylogarithmic space. Note that deciding membership in a regular languages becomes trivial in this model (where the active window is the whole stream): one can simply simulate a deterministic finite automaton on the stream. What makes the sliding window model more difficult is the fact that the oldest symbol in the active window expires in the next step.

While at first sight the only connection between property testers and sliding window property testers is that we must accept the input if it satisfies $P$ and reject if it is far from satisfying $P$, there is, in fact, a deeper link. In particular, the above mentioned result of Alon et al. [2] combined with an optimal sampling algorithm for sliding windows [4], immediately yields a $\mathcal{O}(\log n)$-space, two-sided error sliding window property tester with Hamming gap $\gamma(n) = \epsilon n$ for every regular language. We will improve on this observation. Our main contribution are tight complexity bounds for each of the following classes of sliding window property testers for regular languages: deterministic sliding window property testers and randomized sliding window property testers with one-sided and two-sided error.

**Deterministic sliding window property testers.** We call a language $L$ *trivial*, if for some constant $c > 0$ the following holds: For every word $w \in \Sigma^*$ such that $L$ contains a word of length $|w|$, the Hamming distance from $w$ to $L$ is at most $c$. Every trivial regular language has a constant-space deterministic sliding window property tester with constant Hamming gap (Theorem 4). For generic regular languages, we show a deterministic sliding window property tester with constant Hamming gap that uses $\mathcal{O}(\log n)$ space. This is particularly surprising, because for Hamming gap zero (i.e., the exact case) [16] showed a space lower bound of $\Omega(n)$ for generic regular languages. In other words, a constant Hamming gap allows an exponential space improvement. We also show that for *non-trivial* regular languages, $\mathcal{O}(\log n)$ space is the best one can hope to achieve, even for Hamming gap $\gamma(n) = \epsilon n$ (Theorem 6).

**Randomized sliding window property testers with two-sided error.** Next, we show that for every regular language, there is a randomized sliding window property tester with Hamming gap $\gamma(n) = \epsilon n$ and two-sided error that uses constant space (Theorem 7). This is an optimal bound and a considerable improvement compared to the tester that can be obtained by combining the property tester of Alon et al. [2] and an optimal sampling algorithm for sliding windows [4]. Our constant space tester makes use of a probabilistic counter from [16].

**Randomized sliding window property testers with one-sided error.** While our randomized sliding window property tester with two-sided error is optimal, we believe that a two-sided error is a very strong relaxation that has to be avoided in some applications. To this end, we study the one-sided error randomized setting. The general landscape for this setting is the most complex: In Theorems 8 and 9, we show that for every regular language $L$, the space complexity of an optimal randomized sliding window property tester with one-sided error is either $\mathcal{O}(1)$, $\mathcal{O}(\log \log n)$, or $\mathcal{O}(\log n)$, and we provide language theoretic characterizations of these space classes.

In order to show our upper bound results, we demonstrate novel combinatorial properties of automata and regular languages and develop new streaming techniques, such as probabilistic counters, which can be of interest on their own. To show the lower bound results, we introduce a new methodology, which could potentially simplify further establishments of lower bounds in string processing tasks in the streaming setting: namely, we view the testers as nondeterministic automata, and study their behaviour.

### 1.2 Related work

The results above assume that the regular language admits a constant-space description and we will follow the same assumption in this work. Currently, there are few studies on the dependency of the complexity of sliding window algorithms on the size of the language description. On the negative side, Ganardi et al. [14] showed that there are regular languages such that any sliding window algorithm that achieves logarithmic space (in the window size) depends exponentially on the automata size. On the positive side, there is an extensive study of the pattern matching problem and its variants that gives sub-exponential upper bounds for a class of (very simple) regular languages. In this problem, we are given a pattern and a streaming text $T$, and at each moment we must decide if the active window is equal to the pattern. This problem and its generalisations have been studied in [5, 6, 7, 8, 9, 19, 20, 21, 28, 30].

Similar to regular languages, we can ask whether the current active window belongs to a given context-free language. This question was studied in [3, 24, 25, 26] for the model where the active window is the complete stream and in [13, 18] for the sliding-window model.

## 2 Sliding window property tester

We fix a finite alphabet $\Sigma$ for the rest of the paper. We denote by $\Sigma^*$ the set of all words over $\Sigma$ and by $\Sigma^n$ the set of words over $\Sigma$ of length $n$. The empty word is denoted by $\lambda$. Let $w$ be a word. We say that $v$ is a *prefix (suffix)* of $w$ if $w = xv$ ($w = vx$) for some word $x$. We say that $v$ is a *factor* of $w$ if $w = xvy$ for some words $x, y$. The *Hamming distance* between two words $u = a_1 \cdots a_n$ and $v = b_1 \cdots b_n$ of equal length is the number of positions where $u$ and $v$ differ, i.e. $\mathrm{dist}(u, v) = |\{i : a_i \neq b_i\}|$. The distance of a word $u$ to a language $L$ is defined as $\mathrm{dist}(u, L) = \inf\{\mathrm{dist}(u, v) : v \in L\} \in \mathbb{N} \cup \{\infty\}$.

A *deterministic finite automaton* (DFA) is a tuple $A = (Q, \Sigma, q_0, \delta, F)$ where $Q$ is a finite set of states, $\Sigma$ is the input alphabet, $q_0$ is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition mapping and $F \subseteq Q$ is the set of final states. We extend $\delta$ to a mapping $\delta : Q \times \Sigma^* \to Q$ inductively in the usual way: $\delta(q, \lambda) = q$ and $\delta(q, aw) = \delta(\delta(q, a), w)$. The language accepted by $A$ is $L(A) = \{w \in \Sigma^* : \delta(q_0, w) \in F\}$. A language is *regular* if it is accepted by a DFA. For more background in automata theory see [23].

A *stream* is a word $a_1 a_2 \cdots a_m$ over $\Sigma$. A *sliding window algorithm* is a family $\mathcal{A} = (A_n)_{n \geq 0}$ of streaming algorithms. Given a window size $n \in \mathbb{N}$ and an input stream $a_1 a_2 \cdots a_m \in \Sigma^*$ the algorithm $A_n$ reads the stream symbol by symbol from left to right and thereby updates its memory content. After reading a prefix $a_1 \cdots a_t$ ($0 \leq t \leq m$) the algorithm is required to compute an output value that depends on the *active window* $\mathrm{last}_n(a_1 \cdots a_t) = a_{t-n+1} \cdots a_t$ at time $t$. For convenience, for $i < 0$ we define $a_i = \Box$ where $\Box \in \Sigma$ is an arbitrary fixed symbol. In other words, we assume an initial window $\Box^n$ that is active at time $t = 0$. We consider *deterministic sliding window algorithms* (where every $A_n$ can be viewed as a DFA) and *randomized sliding window algorithms* (where every $A_n$ can be viewed as a probabilistic finite automaton in the sense of Rabin [29]). In the latter case, $A_n$ updates in each step its memory content according to a probability distribution that depends on the current memory content and the current input symbol. Let $\gamma : \mathbb{N} \to \mathbb{R}_{\geq 0}$ be a function such that $\gamma(n) \leq n$ for all $n \in \mathbb{N}$, let $\alpha, \beta$ be probabilities, and let $L \subseteq \Sigma^*$ be a language.

▶ **Definition 1.** *A* deterministic sliding window (property) tester *for $L$ with Hamming gap $\gamma(n)$ is a deterministic sliding window algorithm $\mathcal{A} = (A_n)_{n \geq 0}$ such that for every input stream $w \in \Sigma^*$ and every window size $n$ the following properties hold:*

- *if* $\mathrm{last}_n(w) \in L$, *then $A_n$ accepts;*
- *if* $\mathrm{dist}(\mathrm{last}_n(w), L) > \gamma(n)$, *then $A_n$ rejects.*

▶ **Definition 2.** *A* randomized sliding window (property) tester *for $L$ with Hamming gap $\gamma(n)$ and error $(\alpha, \beta)$ is a randomized sliding window algorithm $\mathcal{A} = (A_n)_{n \geq 0}$ such that for every input stream $w \in \Sigma^*$ and every window size $n$ the following properties hold:*

- *if* $\mathrm{last}_n(w) \in L$, *then $A_n$ accepts with probability at least $1 - \alpha$;*
- *if* $\mathrm{dist}(\mathrm{last}_n(w), L) > \gamma(n)$, *then $A_n$ rejects with probability at least $1 - \beta$.*

*We say that $\mathcal{A}$ has* one-sided error *if $\mathcal{A}$ has error $(0, 1/2)$ and* two-sided error *if $\mathcal{A}$ has error $(1/3, 1/3)$.*

Notice that our definition is non-uniform since we allow an arbitrary algorithm $A_n$ for each window size $n$. If the window size is not specified, then it is implicitly universally quantified. The space consumption of $\mathcal{A}$ is the mapping $s(n)$, where $s(n)$ is the space consumption of $A_n$, i.e., the maximal number of bits stored by $A_n$ while reading any input stream. We can assume that $s(n) \in \mathcal{O}(n)$ since $A_n$ can store the active window in $\mathcal{O}(n)$ bits. The goal is to devise algorithms which only use $o(n)$ space. Using probability amplification (similar to [16]) one can replace the error probability $1/3$ in the two-sided error setting (resp. $1/2$ in the one-sided error setting) by any probability $p < 1/2$ (resp. $p < 1$). This influences the space complexity only by a constant factor. The case of Hamming gap $\gamma(n) = 0$ corresponds to exact membership testing to $L$ which was studied in [14, 15, 16]. In this paper, we focus on the two cases $\gamma(n) = c$ for some constant $c > 0$ and $\gamma(n) = \epsilon n$ for some $\epsilon > 0$.

Before we come to the main results of the paper we state two simple facts about the sliding window testers.

▶ **Lemma 3.** *Assume that $L = \bigcup_{i=1}^{k} L_i$ and that for every $1 \leq i \leq k$ there exists a randomized sliding window tester for $L_i$ with Hamming gap $\gamma(n)$ and error $(\alpha, \beta)$ that uses space $s_i(n)$. Then there exists a sliding window tester for $L$ with Hamming gap $\gamma(n)$ and error $(\alpha, \beta)$ that uses space $\mathcal{O}(\sum_{i=1}^{k} s_i(n))$.*

The second fact concerns so-called trivial languages. Let $\gamma : \mathbb{N} \to \mathbb{R}_{\geq 0}$ be a mapping with $\gamma(n) \leq n$ for all $n \geq 0$. A language is $L \subseteq \Sigma^*$ is *$\gamma$-trivial* if there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ with $L \cap \Sigma^n \neq \emptyset$ and all $w \in \Sigma^n$ we have $\mathrm{dist}(w, L) \leq \gamma(n)$. If $\gamma(n) \in \mathcal{O}(1)$, we say that $L$ is *trivial*. Note that Alon et al. [2] call a language $L$ trivial if $L$ is $(\epsilon n)$-trivial for all $\epsilon > 0$ according to our definition. In the long version [17] we show that both definitions coincide for regular languages, but we will not make use of this fact.

▶ **Theorem 4.** *For every trivial (but not necessarily regular) language there is a deterministic sliding window tester with constant Hamming gap that uses constant space. The converse is also true: If for a language $L$ there is a deterministic constant-space sliding window tester with Hamming gap $\gamma(n)$, then there exists a constant $c$ such that $L$ is $(\gamma + c)$-trivial.*

## 3 Main results

Our first main contribution is a deterministic logspace sliding window tester for every regular language, together with a matching lower bound for so-called *nontrivial* regular languages (defined above).

▶ **Theorem 5.** *For every regular language $L$, there exists a deterministic sliding window tester for $L$ with constant Hamming gap which uses $\mathcal{O}(\log n)$ space.*

▶ **Theorem 6.** *For every non-trivial regular language $L$, there exist $\epsilon > 0$ and infinitely many window sizes $n \in \mathbb{N}$ on which every deterministic sliding window tester for $L$ with Hamming gap $\epsilon n$ uses space $\Omega(\log n)$.*

Our second main contribution is a constant-space randomized sliding window property tester with two-sided error for any regular language:

▶ **Theorem 7.** *For every regular language $L$ and every $\epsilon > 0$, there exists a randomized sliding window tester for $L$ with two-sided error and Hamming gap $\gamma(n) = \epsilon n$ that uses space $\mathcal{O}(1/\epsilon)$.*

While the randomized setting with two-sided error allows ultra-efficient testers, we find that allowing a two-sided error is a very strong relaxation. To this end, we study the randomized setting with one-sided error. In this setting, only a small class of regular languages admits sliding window testers working in space $o(\log n)$. A language $L \subseteq \Sigma^*$ is *suffix-free* if $xy \in L$ and $x \neq \lambda$ imply $y \notin L$.

▶ **Theorem 8.** *If $L$ is a finite union of trivial regular languages and suffix-free regular languages, then there exists a randomized sliding window tester for $L$ with one-sided error and constant Hamming gap which uses $\mathcal{O}(\log \log n)$ space.*

▶ **Theorem 9.** *Let $L$ be a regular language.*

- *If $L$ is not a finite union of trivial regular languages and suffix-free regular languages, there exist $\epsilon > 0$ and infinitely many window sizes $n$ on which every randomized sliding window tester for $L$ with one-sided error and Hamming gap $\epsilon n$ uses space $\Omega(\log n)$.*
- *If $L$ is non-trivial, then there exist $\epsilon > 0$ and infinitely many window sizes $n$ on which every sliding window tester for $L$ with one-sided error and Hamming gap $\epsilon n$ uses space $\Omega(\log \log n)$.*

We sketch the proofs of Theorem 5, 7, and 8 in Sections 4.1, 4.2, and 4.3, respectively. The proofs of the lower bounds (Theorems 6 and 9) can be found in the long version [17]. We would like to emphasize that the lower bounds shown in [17] are stronger than those stated in Theorems 6 and 9. More precisely, we show space lower bounds for nondeterministic and co-nondeterministic sliding window testers; see [17] for definitions.

## 4 Proofs of the upper bounds

In this section we sketch proofs of Theorems 5, 7, and 8 that give upper bounds for deterministic and (one-sided and two-sided error) randomized sliding window testers. All algorithms in this section satisfy the stronger property that words with large prefix distance are rejected by the algorithm with high probability (probability one in the deterministic setting). The *prefix distance* between words $u = a_1 \cdots a_n$ and $v = b_1 \cdots b_n$ is $\mathrm{pdist}(u, v) = \min\{i \in \{0, \dots, n\} : a_{i+1} \cdots a_n = b_{i+1} \cdots b_n\}$. Clearly, we have $\mathrm{dist}(u, v) \leq \mathrm{pdist}(u, v)$. We extend the definition to languages: for a language $L$, let $\mathrm{pdist}(u, L) = \min\{\mathrm{pdist}(u, v) : v \in L\}$. The prefix distance between two runs $\pi = (q_0, a_1, \dots, q_{n-1}, , a_n, q_n)$ and $\rho = (p_0, b_1, \dots, p_{n-1}, b_n, p_n)$ is defined as $\mathrm{pdist}(\pi, \rho) = \min\{i \in \{0, \dots, n\} : (q_i, a_{i+1}, \dots, q_{n-1}, a_n, q_n) = (p_i, b_{i+1}, \dots, p_{n-1}, b_n, p_n)\}$.

For our upper bound proofs it is convenient to work with DFAs which read the input word from right to left. A *right-deterministic finite automaton (rDFA)* is a tuple $B = (Q, \Sigma, F, \delta, q_0)$, where $Q$, $\Sigma$, $q_0$ and $F$ are as in a DFA, and $\delta \colon \Sigma \times Q \to Q$ is the transition function. We extend $\delta$ to a mapping $\delta : Q \times \Sigma^* \to Q$ analogously to DFAs: $\delta(q, \lambda) = q$ and $\delta(q, wa) = \delta(\delta(q, a), w)$. The regular language recognized by the rDFA $B$ is $L(B) = \{w \in \Sigma^* : \delta(w, q_0) \in F\}$. A run from $p_0 \in Q$ to $p_n \in Q$ on a word $x = a_n \cdots a_2 a_1 \in \Sigma^*$ is a sequence $\pi = (p_n, a_n, p_{n-1}, \dots, p_2, a_2, p_1, a_1, p_0)$ such that $p_i = \delta(a_i, p_{i-1})$ for all $1 \leq i \leq n$. The *length* of $\pi$ is $|\pi| = n$. We visualize $\pi$ in the form

$$\pi \colon p_n \xleftarrow{a_n} p_{n-1} \xleftarrow{a_{n-1}} \cdots \xleftarrow{a_2} p_1 \xleftarrow{a_1} p_0.$$

If $p_n \in F$, then $\pi$ is an *accepting run*. A run of length 1 is a *transition*. If $\pi$ is a run from $p$ to $q$ on a word $v$, and $\rho$ is a run from $q$ to $r$ on a word $u$, then $\rho\pi$ denotes the unique run from $p$ to $r$ on $uv$. We denote by $\pi_{w,q}$ the unique run on $w$ from $q$.

**Strongly connected graphs.** With a DFA $A = (Q, \Sigma, q_0, \delta, F)$ we associate the directed graph $(Q, E)$ with edge set $E = \{(p, \delta(p, a)) \mid p \in Q, a \in \Sigma\}$. Similarly, with an rDFA $A = (Q, \Sigma, F, \delta, q_0)$ we associate the directed graph $(Q, E)$ with edge set $E = \{(p, \delta(a, p)) \mid p \in Q, a \in \Sigma\}$. Let $A$ be a DFA or an rDFA. Two states $p, q$ in $A$ are *strongly connected* if there exists a path in $(Q, E)$ from $p$ to $q$, and vice versa. The *strongly connected components (SCCs)* of $A$ with state set $Q$ are the maximal subsets $C \subseteq Q$ in which all states $p, q \in C$ are strongly connected. A state $q \in Q$ is *transient* if there exists no nonempty path from $q$ to $q$. An SCC $C$ is *transient* if it only contains a single transient state. There is a natural partial order on the SCCs, called the *SCC-ordering*, where the SCC $C_1$ is smaller than the SCC $C_2$ if there exists a path in $(Q, E)$ from a state in $C_1$ to a state in $C_2$.

The following combinatorial result from [2] will be used in this paper. Consider a directed graph $G = (V, E)$. The period of $G$ is the greatest common divisor of all cycle lengths in $G$. If $G$ is acyclic we define the period to be $\infty$.

▶ **Lemma 10** (cf. [2]). *Let $G = (V, E)$ be a strongly connected directed graph with $E \neq \emptyset$ and finite period $g$. Then there exist a partition $V = \bigcup_{i=0}^{g-1} V_i$ and a constant $m(G) \leq 3|V|^2$ with the following properties:*

- *For every $0 \leq i, j \leq g-1$ and for every $u \in V_i$, $v \in V_j$ the length of every directed path from $u$ to $v$ in $G$ is congruent to $j - i$ modulo $g$.*
- *For every $0 \leq i, j \leq g-1$, for every $u \in V_i$, $v \in V_j$ and every integer $r \geq m(G)$, if $r$ is congruent to $j - i$ modulo $g$, then there exists a directed path from $u$ to $v$ in $G$ of length $r$.*

If $G = (V, E)$ is strongly connected with $E \neq \emptyset$ and finite period $g$, and $V_0, \ldots, V_{g-1}$ satisfy the properties from Lemma 10, then we define the *shift* from $u \in V_i$ to $v \in V_j$ by

$$\mathrm{shift}(u, v) = j - i \pmod{g} \in \{0, \ldots, g-1\}. \tag{1}$$

Notice that this definition is independent of the partition $\bigcup_{i=0}^{g-1} V_i$ since any path from $u$ to $v$ has length $\ell \equiv \mathrm{shift}(u, v) \pmod{g}$ by Lemma 10. Also note that $\mathrm{shift}(u, v) + \mathrm{shift}(v, u) \equiv 0 \pmod{g}$. In the following let $g(C)$ denote the period of the SCC $C$.

▶ **Lemma 11.** *For every regular language $L$ there exists an rDFA $A$ for $L$ and a number $g$ such that every non-transient SCC $C$ in $A$ has period $g(C) = g$.*

**Path summaries.** We start by recalling the notion of a path summary from [14], where it was used in order to prove a logspace upper bound for regular left-ideals (in the exact setting where the Hamming gap is zero). For the rest of Section 4 we fix a regular language $L \subseteq \Sigma^*$ and an rDFA $B = (Q, \Sigma, F, \delta, q_0)$ which recognizes $L$. By Lemma 11, we can assume that every non-transient SCC $C$ of $B$ has period $g(C) = g$. Consider a run $\pi = (p_n, a_n, \ldots, a_1, p_0)$ on $x = a_n \cdots a_1$. If all states $p_n, \ldots, p_0$ are contained in a single SCC we call $\pi$ *internal*. We can decompose $\pi = \pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_1 \pi_1$, where each $\pi_i$ is a possibly empty internal run and each $\tau_i$ is a single transition connecting two distinct SCCs. We call this unique factorization the *SCC-factorization* of $\pi$, which is illustrated in Figure 1. The *path summary* of $\pi$ is

$$\mathrm{ps}(\pi) = (|\pi_m|, q_m)(|\tau_{m-1}\pi_{m-1}|, q_{m-1}) \cdots (|\tau_2\pi_2|, q_2)(|\tau_1\pi_1|, q_1),$$

where $q_i$ is the first state in $\pi_i$ $(1 \leq i \leq m)$. Note that $m$ is bounded by the constant number of states of $B$. Hence, a path summary can be stored with $\mathcal{O}(\log |\pi|)$ bits.

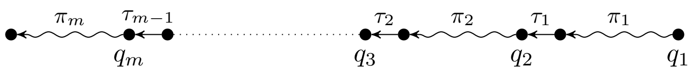

**Figure 1** The SCC-factorization of a run.

**Periodic acceptance sets.** For $a \in \mathbb{N}$ and $X \subseteq \mathbb{N}$ we use the standard notation $X + a = \{a + x : x \in X\}$. For a state $q \in Q$ we define $\mathrm{Acc}(q) = \{n \in \mathbb{N} : \exists w \in \Sigma^n : \delta(w, q) \in F\}$. A set $X \subseteq \mathbb{N}$ is *eventually $d$-periodic*, where $d \geq 1$ is an integer, if there exists a *threshold* $t \in \mathbb{N}$ such that for all $x \geq t$ we have $x \in X$ if and only if $x + d \in X$. If $X$ is eventually $d$-periodic for some $d \geq 1$, then $X$ is *eventually periodic*.

▶ **Lemma 12.** *For every $q \in Q$ the set $\mathrm{Acc}(q)$ is eventually $g$-periodic.*

Two sets $X, Y \subseteq \mathbb{N}$ are *equal up to a threshold* $t \in \mathbb{N}$, in symbol $X =_t Y$, if for all $x \geq t$: $x \in X$ iff $x \in Y$. Sets $X, Y \subseteq \mathbb{N}$ are *almost equal* if $X =_t Y$ for some threshold $t \in \mathbb{N}$.

▶ **Lemma 13.** *Let $C$ be a non-transient SCC in $B$, $p, q \in C$ and $s = \mathrm{shift}(p, q)$. Then $\mathrm{Acc}(p)$ and $\mathrm{Acc}(q) + s$ are almost equal.*

▶ **Corollary 14.** *There exists a threshold $t \in \mathbb{N}$ such that*
1. $\mathrm{Acc}(q) =_t \mathrm{Acc}(q) + g$ *for all $q \in Q$, and*
2. $\mathrm{Acc}(p) =_t \mathrm{Acc}(q) + \mathrm{shift}(p, q)$ *for all non-transient SCCs $C$ and all $p, q \in C$.*

We fix the threshold $t$ from Corollary 14 for the rest of Section 4. The following lemma is the main tool to prove the correctness of our sliding window testers. It states that if a word of length $n$ is accepted from $p$ and $\rho$ is any internal run from $p$ of length at most $n$, then, up to a bounded length prefix, $\rho$ can be extended to an accepting run of length $n$. Formally, a run $\pi$ *$k$-simulates* a run $\rho$ if one can factorize $\rho = \rho_1\rho_2$ and $\pi = \pi'\rho_2$ where $|\rho_1| \leq k$.

▶ **Lemma 15.** *If $\rho$ is an internal run starting from $p$ of length at most $n$ and $n \in \mathrm{Acc}(p)$, then there exists an accepting run $\pi$ from $p$ of length $n$ which $t$-simulates $\rho$.*

## 4.1 Deterministic logspace tester

**Proof of Theorem 5.** Let $n \in \mathbb{N}$ such that $n \geq |Q|$ (for $n < |Q|$ we use a trivial streaming algorithm which stores the window explicitly). The algorithm maintains the set $\{\mathrm{ps}(\pi_{w,q}) \mid q \in Q\}$ where $w \in \Sigma^n$ is the active window. Initially this set is $\{\mathrm{ps}(\pi_{w,q}) \mid q \in Q\}$ for $w = \square^n$. Now suppose $w = av$ for some $a \in \Sigma$ and the next symbol of the stream is $b \in \Sigma$, i.e. the new active window is $vb$. For each transition $q \xleftarrow{b} p$ in $B$ we can compute $\mathrm{ps}(\pi_{vb,p})$ from $\mathrm{ps}(\pi_{av,q})$ as follows. Suppose that $\mathrm{ps}(\pi_{av,q}) = (\ell_m, q_m) \cdots (\ell_1, q_1)$ where $q = q_1$.
- If $p$ and $q$ belong to the same SCC, then we increment $\ell_1$ by one, else we append a new pair $(1, p)$.
- If $\ell_m > 0$ we decrement $\ell_m$ by one. If $\ell_m = 0$ we remove the pair $(\ell_m, q_m)$ and we decrement $\ell_{m-1}$ by one (in this case we must have $m > 1$ and $\ell_{m-1} > 0$).

The obtained path summary is $\mathrm{ps}(\pi_{vb,p})$. This data structure can be stored with $\mathcal{O}(\log n)$ bits since it contains $|Q|$ path summaries, each of which can be stored in $\mathcal{O}(\log n)$ bits.

It remains to define a proper acceptance condition. Consider the run $\pi = \pi_{w,q_0}$, its SCC-factorization $\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_1\pi_1$ and its path summary $(\ell_m, q_m)\cdots(\ell_1, q_1)$. The algorithm accepts if and only if $\ell_m = |\pi_m| \in \mathrm{Acc}(q_m)$. If $w \in L$, then clearly $|\pi_m| \in \mathrm{Acc}(q_m)$. If $|\pi_m| \in \mathrm{Acc}(q_m)$, then the internal run $\pi_m$ can be $t$-simulated by an accepting run $\pi'_m$ of equal length by Lemma 15. The run $\pi'_m\tau_{m-1}\pi_{m-1}\cdots\tau_1\pi_1$ is accepting and witnesses that $\mathrm{pdist}(w, L) \leq t$. ◀

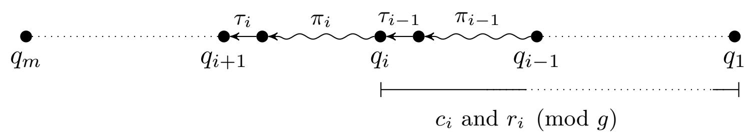

**Figure 2** A compact summary of a run $\pi$.

## 4.2 Randomized constant-space tester with two-sided error

Let us first define a probabilistic counter, similar to the approximate counter by Morris [27], which uses $O(\log\log n)$ bits. For our purposes it suffices to distinguish high and low counters states. Consider a probabilistic data structure $Z$ representing a counter. Its operations are incrementing the counter (using random coins) and querying whether the state of the counter is *low* or *high*. Initially $Z$ is in a low state. The random state reached after $k$ increments is denoted by $Z(k)$. Given numbers $0 \le \ell < h$ (they will depend on our window size $n$) we say that $Z$ is an *$(h,\ell)$-counter with error probability $\delta < \frac{1}{2}$* if for all $k \in \mathbb{N}$ we have:

- If $k \le \ell$, then Prob[$Z(k)$ is high] $\le \delta$.
- If $k \ge h$, then Prob[$Z(k)$ is low] $\le \delta$.

▶ **Lemma 16.** *For all $h, \ell, \xi > 0$ with $\ell \le (1-\epsilon)h + \mathcal{O}(1)$ there exists an $(h,\ell)$-counter $Z$ with error probability $1/3|Q|$ which internally stores $\mathcal{O}(\log(1/\epsilon))$ bits.*

Fix a parameter $0 < \epsilon < 1$ and a window length $n \in \mathbb{N}$. Based on the previous concepts, we are now able to describe a randomized sliding window tester for a regular language $L$ with Hamming gap $\epsilon n$ that uses $\mathcal{O}(\log(1/\epsilon))$ bits. Let $Z$ be the $(h,\ell)$-counter with error probability $1/(3|Q|)$ from Lemma 16 where $h = n - t$ and $\ell = (1-\epsilon)n + t + 1$. The counter is used to define so-called compact summaries of runs.

▶ **Definition 17.** *A* compact summary $cs = (q_m, r_m, c_m) \cdots (q_2, r_2, c_2)(q_1, r_1, c_1)$ *is a sequence of triples, where each triple $(q_i, r_i, c_i)$ consists of a state $q_i \in Q$, a remainder $0 \le r_i \le g-1$, and a state $c_i$ of the $(h,\ell)$-counter $Z$. The state $c_1$ must be low and $r_1 = 0$.*

*A compact summary $(q_m, r_m, c_m) \cdots (q_1, r_1, c_1)$* represents *a run $\pi$ if the SCC-factorization of $\pi$ has the form $\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_1 \pi_1$, and the following properties hold:*

1. *for all $1 \le i \le m$, $\pi_i$ starts in $q_i$;*
2. *for all $2 \le i \le m$, if $|\tau_{i-1}\pi_{i-1}\cdots\tau_1\pi_1| \le (1-\epsilon)n + t + 1$, then $c_i$ is the low state; and if $|\tau_{i-1}\pi_{i-1}\cdots\tau_1\pi_1| \ge n - t$, then $c_i$ is the high state;*
3. *for all $2 \le i \le m$, $r_i = |\tau_{i-1}\pi_{i-1}\cdots\tau_1\pi_1| \pmod g$.*

The idea of a compact summary is visualized in Figure 2. If $m > |Q|$ then the above compact summary cannot represent a run. Therefore, we can assume that $m \le |Q|$. For every triple $(q_i, r_i, c_i)$, the entries $q_i$ and $r_i$ only depend on the rDFA $B$, and hence can be stored with $\mathcal{O}(1)$ bits. Every state $c_i$ of the probabilistic counter needs $\mathcal{O}(\log(1/\epsilon))$ bits. Hence, a compact summary can be stored in $\mathcal{O}(\log(1/\epsilon))$ bits. In contrast to Theorem 5, we maintain a set of compact summaries which represent all runs of $B$ on the *complete* stream read so far (not only on the active window) with high probability.

▶ **Lemma 18.** *For a given input stream $w \in \Sigma^*$, we can maintain a set of compact summaries $S$ containing for each $q \in Q$ a compact summary $cs_q \in S$ starting in $q$ such that $cs_q$ represents the unique run $\pi_{w,q}$ with probability at least $2/3$.*

It remains to define an acceptance condition on compact summaries. For every $q \in Q$ we define $\mathrm{Acc}_{mod}(q) = \{\ell \pmod g : \ell \in \mathrm{Acc}(q) \text{ and } \ell \geq t\}$, which is intuitively speaking the set of accepting remainders. Let $\mathrm{cs} = (q_m, r_m, c_m) \cdots (q_1, r_1, c_1)$ be a compact summary. Since $c_1$ is the low initial state of the probabilistic counter, there exists a maximal index $i \in \{1, \dots, m\}$ such that $c_i$ is low. We say that cs is *accepting* if $n - r_i \pmod g \in \mathrm{Acc}_{mod}(q_i)$.

▶ **Proposition 19.** *Assume that $\epsilon n \geq t$. Let $w \in \Sigma^*$ with $|w| \geq n$ and let cs be a compact summary which represents $\pi_{w,q_0}$.*

1. *If* $\mathrm{last}_n(w) \in L$, *then cs is accepting.*
2. *If cs is accepting, then* $\mathrm{pdist}(\mathrm{last}_n(w), L) \leq \epsilon n$.

**Proof of Theorem 7.** Assume that $\epsilon n \geq t$, otherwise we use a trivial streaming algorithm that stores the window explicitly with $\mathcal{O}(1/\epsilon)$ bits. We use the algorithm from Proposition 18 for each incoming symbol from the stream. To initialize, we run the algorithm on $\square^n$. The algorithm accepts if the computed compact summary starting in $q_0$ is accepting. From Proposition 18 and 19 we get:

- If $\mathrm{pdist}(\mathrm{last}_n(w), L) > \epsilon n$, then the algorithm rejects with probability at least 2/3.
- If $\mathrm{last}_n(w) \in L$, then the algorithm accepts with probability at least 2/3.

This concludes the proof of the theorem. ◀

Comparing Theorems 5 and 7 leads to the question whether one can replace the Hamming gap $\gamma(n) = \epsilon n$ in Theorem 7 by $\gamma(n) = o(n)$ while retaining constant space at the same time. We show that this is not the case:

▶ **Lemma 20.** *Every randomized sliding window tester with two-sided error for $a^* \subseteq \{a, b\}^*$ with Hamming gap $\gamma(n)$ needs space $\Omega(\log n - \log \gamma(n))$ for infinitely many $n$.*

## 4.3 Randomized loglogspace tester with one-sided error

Let $L$ be a finite union of trivial regular languages and suffix-free regular languages. In this section, we present a randomized sliding window tester for $L$ with one-sided error and Hamming gap $\gamma(n) = \epsilon n$ that uses space $\mathcal{O}(\log \log n)$. By Lemma 3 and Theorem 4, it suffices to consider the case when $L$ is a suffix-free regular language. As in Section 4 we fix an rDFA $B = (Q, \Sigma, F, \delta, q_0)$ for $L$ such that $g(C) = g$ for all SCCs of $A$. Since $L$ is suffix-free, $B$ has the property that no final state can be reached from a final state by a non-empty run. We decompose $B$ into a finite union of *partial automata*, similar to [14].

▶ **Definition 21.** *A sequence $(q_k, a_k, p_{k-1}), C_{k-1}, \dots, (q_2, a_2, p_1), C_1, (q_1, a_1, p_0), C_0, q_0$ is a* path description *if $C_{k-1}, \dots, C_0$ is a chain (read from right to left) in the SCC-ordering of $B$, $p_i, q_i \in C_i$, $q_{i+1} \xleftarrow{a_{i+1}} p_i$ is a transition in $B$ for all $0 \leq i \leq k-1$, and $q_k \in F$.*

Each path description defines a *partial rDFA* $B_P = (Q_P, \Sigma, \{q_k\}, \delta_P, q_0)$ by restricting $B$ to the state set $Q_P = \bigcup_{i=0}^{k-1} C_i \cup \{q_k\}$, restricting the transitions of $B$ to internal transitions from the SCCs $C_i$ and the transitions $q_{i+1} \xleftarrow{a_{i+1}} p_i$, and declaring $q_k$ to be the only final state. The rDFA is partial since for every state $p_i$ and every symbol $a \in \Sigma$ there exists at most one transition $q \xleftarrow{a} p_i$. Since the number of path descriptions $P$ is finite and $L(B) = \bigcup_P L(B_P)$, it suffices to provide a sliding window tester for $L(B_P)$ (we again use Lemma 3 here).

From now on, we fix a path description $P$ from Definition 21 and the partial automaton $B_P = (Q_P, \Sigma, \{q_k\}, \delta_P, q_0)$ corresponding to it. The acceptance sets $\mathrm{Acc}(q)$ are defined with respect to $B_P$. If all $C_i$ are transient, then $L(B_P)$ is a singleton and we can use a trivial sliding window tester with space complexity $\mathcal{O}(1)$. Now assume the contrary and let $0 \leq e \leq k-1$ be maximal such that $C_e$ is nontransient.

▶ **Lemma 22.** *There exist $r_0, \dots, r_{k-1}, s_0, \dots, s_e \in \mathbb{N}$ such that the following holds:*
1. *For all $e + 1 \le i \le k$, the set $\mathrm{Acc}(q_i)$ is a singleton.*
2. *Every run from $q_i$ to $q_{i+1}$ has length $r_i \pmod g$.*
3. *For all $0 \le i \le e$, $\mathrm{Acc}(q_i) =_{s_i} \sum_{j=i}^{k-1} r_j + g\mathbb{N}$.*

Let $s = \max\{k, \sum_{j=0}^{k-1} r_j, s_0, \dots, s_e\}$ and for a word $w \in \Sigma^*$ define the function $\ell_w \colon Q \to \mathbb{N} \cup \{\infty\}$ where $\ell_w(q) = \inf\{\ell \in \mathbb{N} \mid \delta_P(\mathrm{last}_\ell(w), q) = q_k\}$ (we set $\inf \emptyset = \infty$).

Let $p$ be a random prime with $\Theta(\log \log n)$ bits. We now define an acceptance condition on $\ell_w(q)$. If $n \notin \mathrm{Acc}(q_0)$, we always reject. Otherwise, we accept $w$ iff $\ell_w(q_0) \equiv n$ modulo our randomly chosen prime $p$.

▶ **Lemma 23.** *Let $n \in \mathrm{Acc}(q_0)$ be a window size with $n \ge s + |Q_P|$ and $w \in \Sigma^*$ with $|w| \ge n$. There exists a constant $c > 0$ such that:*
1. *if $\mathrm{last}_n(w) \in L(B_P)$, then $w$ is accepted with probability 1;*
2. *if $\mathrm{pdist}(\mathrm{last}_n(w), L(B_P)) > c$, then $w$ is rejected with probability at least 2/3.*

**Proof of Theorem 8.** Let $n \in \mathbb{N}$ be the window size. From the discussion above, it suffices to show a tester for a fixed partial automaton $B_P$. Assume $n \ge s + |Q|$, otherwise a trivial tester can be used. If $n \notin \mathrm{Acc}(q_0)$, the tester always rejects. Otherwise, the tester picks a random prime $p$ with $\Theta(\log \log n)$ bits and maintains $\ell_w(q) \pmod p$ for all $q \in Q_P$, where $w$ is the stream read so far, which requires $\mathcal{O}(\log \log n)$ bits. When a symbol $a \in \Sigma$ is read, we can update $\ell_{wa}$ using $\ell_w$: If $q = q_k$, then $\ell_{wa}(q) = 0$, otherwise $\ell_{wa}(q) = 1 + \ell_w(\delta_P(a, q)) \pmod p$ where $1 + \infty = \infty$. The tester accepts if $\ell_w(q_0) \equiv n \pmod p$. Lemma 23 guarantees correctness of the tester in the one-sided error setting. ◀

## 5 Further research

We gave a complete characterization of the space complexity of sliding window testers for regular languages. A natural open research problem is, whether similar results can be shown for context-free languages:
- Does every context-free language $L$ have a deterministic sliding window tester with Hamming gap $\epsilon n$ (or even $\mathcal{O}(1)$) that uses space $\mathcal{O}(\log n)$ (or at least space $o(n)$)?
- Does every context-free language $L$ have a randomized sliding window tester with Hamming gap $\epsilon n$ (or even $\mathcal{O}(1)$) that uses space $\mathcal{O}(1)$ (or at least space $o(n)$)?

If the answers to these questions turn out be negative, then one might look at deterministic context-free languages or visibly pushdown languages.

## References

1 Charu C. Aggarwal. *Data Streams – Models and Algorithms*. Springer, 2007.

2 Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular Languages are Testable with a Constant Number of Queries. *SIAM Journal on Computing*, 30(6):1842–1862, 2000. doi:10.1137/S0097539700366528.

3 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013.

4 Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *Journal of Computer and System Sciences*, 78(1):260–272, 2012.

5 Dany Breslauer and Zvi Galil. Real-Time Streaming String-Matching. *ACM Transactions on Algorithms*, 10(4):22:1–22:12, 2014. doi:10.1145/2635814.

6 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. Dictionary Matching in a Stream. In *Proceedings of ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2015. doi:10.1007/978-3-662-48350-3_31.

7 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The $k$-mismatch problem revisited. In *Proceedings of SODA 2016*, pages 2039–2052. SIAM, 2016. doi:10.1137/1.9781611974331.ch142.

8 Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. In *Proceedings of SODA 2019*, pages 1106–1125. SIAM, 2019. doi:10.1137/1.9781611975482.68.

9 Raphaël Clifford and Tatiana Starikovskaya. Approximate Hamming Distance in a Stream. In *Proceedings of ICALP 2016*, volume 55 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.20.

10 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

11 Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. Testing and Spot-Checking of Data Streams. *Algorithmica*, 34(1):67–80, 2002. doi:10.1007/s00453-002-0959-4.

12 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming Property Testing of Visibly Pushdown Languages. In *Proceedings of ESA 2016*, volume 57 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

13 Moses Ganardi. Visibly Pushdown Languages over Sliding Windows. In *Proceedings of STACS 2019*, volume 126 of *LIPIcs*, pages 29:1–29:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.29.

14 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *Proceedings of STACS 2018*, volume 96 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

15 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying Regular Languages over Sliding Windows. In *Proceedings of FSTTCS 2016*, volume 65 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

16 Moses Ganardi, Danny Hucke, and Markus Lohrey. Randomized Sliding Window Algorithms for Regular Languages. In *Proceedings of ICALP 2018*, volume 107 of *LIPIcs*, pages 127:1–127:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

17 Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. Technical report, arXiv.org, 2020. arXiv:1909.10261.

18 Moses Ganardi, Artur Jeż, and Markus Lohrey. Sliding Windows over Context-Free Languages. In *Proceedings of MFCS 2018*, volume 117 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

19 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Streaming Pattern Matching with d Wildcards. In *Proceedings of ESA 2016*, volume 57 of *LIPIcs*, pages 44:1–44:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.44.

20 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards Optimal Approximate Streaming Pattern Matching by Matching Multiple Patterns in Multiple Streams. In *Proceedings of ICALP 2018*, volume 107 of *LIPIcs*, pages 65:1–65:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.65.

21 Shay Golan and Ely Porat. Real-Time Streaming Multi-Pattern Search for Constant Alphabet. In *Proceedings of ESA 2017*, volume 87 of *LIPIcs*, pages 41:1–41:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.41.

22 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property Testing and its Connection to Learning and Approximation. *Journal of the ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

23 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison–Wesley, Reading, MA, 1979.

24 Rahul Jain and Ashwin Nayak. The Space Complexity of Recognizing Well-Parenthesized Expressions in the Streaming Model: The Index Function Revisited. *IEEE Transactions on Information Theory*, 60(10):6646–6668, October 2014. doi:10.1109/TIT.2014.2339859.

25 Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming Algorithms for Recognizing Nearly Well-Parenthesized Expressions. In *Proceedings of MFCS 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 412–423. Springer, 2011.

26 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing Well-Parenthesized Expressions in the Streaming Model. *SIAM Journal on Computing*, 43(6):1880–1905, 2014.

27 Robert H. Morris. Counting Large Numbers of Events in Small Registers. *Communications of the ACM*, 21(10):840–842, 1978. doi:10.1145/359619.359627.

28 Benny Porat and Ely Porat. Exact and Approximate Pattern Matching in the Streaming Model. In *Proceedings of FOCS 2009*, pages 315–323. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.11.

29 Michael O. Rabin. Probabilistic Automata. *Information and Control*, 6(3):230–245, 1963.

30 Tatiana Starikovskaya. Communication and Streaming Complexity of Approximate Pattern Matching. In *Proceedings of CPM 2017*, volume 78 of *LIPIcs*, pages 13:1–13:11. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.13.