Property Testing of Regular Languages with Applications to Streaming Property Testing of Visibly Pushdown Languages

Gabriel Bathie
École normale supérieure de Lyon, France

Tatiana Starikovskaya
DIENS, École normale supérieure de Paris, PSL Research University, France

Abstract

In this work, we revisit the problem of testing membership in regular languages, first studied by Alon et al. [1]. We develop a one-sided error property tester for regular languages under weighted edit distance that makes $\mathcal{O}(\varepsilon^{-1}\log(1/\varepsilon))$ non-adaptive queries, assuming that the language is described by an automaton of constant size. Moreover, we show a matching lower bound, essentially closing the problem for the edit distance. As an application, we improve the space bound of the current best streaming property testing algorithm for visibly pushdown languages from $\mathcal{O}(\varepsilon^{-4}\log^6 n)$ to $\mathcal{O}(\varepsilon^{-3}\log^5 n\log\log n)$, where n is the size of the input. Finally, we provide a $\bar{\Omega}(\max(\varepsilon^{-1}, \log n))$ lower bound on the memory necessary to test visibly pushdown languages in the streaming model, significantly narrowing the gap between the known bounds.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases property testing, regular languages, streaming algorithms, visibly pushdown languages

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.119

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding This work was partially funded by the grants ANR-19-CE48-0016 and ANR-20-CE48-0001 from the French National Research Agency (ANR).

1 Introduction

A one-sided error ε-property tester for a language L is a randomised algorithm that accepts an input u of length n if $u \in L$ with probability one, and rejects if the distance from u to L is at least εn with probability $\geq 2/3$. In the property testing setting, we do not have access to the whole input u, but instead must take the decision by querying as few symbols of the input as possible. The number of queried symbols is called the query complexity of the tester.

The study of property testing of formal languages was initiated in the seminal paper of Alon et al. [1], who showed a property tester for regular languages as well as lower bounds for context-free languages. Property testing of regular languages has been also studied in [6, 17, 19, 20]. Apart from regular languages, there is a series of work that studied the question of testing membership in Dyck(s), the language of well-parenthesized expressions with s types of parentheses [1, 4, 21]. When the distance allows sufficient modifications of the input, such as moves of arbitrarily large factors, it was shown that any context-free language is testable with a constant number of queries [3].

In this work, we revisit the problem of property testing of regular languages. Recall that the Hamming distance is the number of mismatches between two equal-length strings, and the edit distance between two strings is the smallest number of insertions, deletions, and substitutions required to convert one string into another. The weighted edit distance is a

48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).
Editors: Nikhil Bansal, Emanuela Merelli, and James Worrell; Article No. 119; pp. 119:1–119:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

generalisation of the edit distance for weighted words (see Section 2 for a definition). Fix a regular language specified by an automaton with m states and k connected components. The first property tester for regular languages was given by Alon et al. [1]. The tester of Alon et al. is for the Hamming distance, it queries $\mathcal{O}(k^2 m \cdot \varepsilon^{-1} \log^3(m/\varepsilon))$ symbols of the input, and its runtime is exponential in the size of the automaton that defines the regular language. Alon et al. [1] also showed a $\bar{\Omega}(1/\varepsilon)$ lower bound for such testers[1]. Magniez and De Rougemont [17] built upon [1] to show a property tester for regular languages under the edit distance with moves. The query complexity of their algorithm is $\mathcal{O}(m^3 \cdot \varepsilon^{-1} \log^2(m/\varepsilon))$ and its running time exponential. Later, Fischer et al. [3] improved the query complexity to remove the dependency on m. Ndione et al. [19, 20] continued this line of work with a goal of devising property testers that run in polynomial time, both for the Hamming distance and the edit distance. Their tester for the Hamming distance has query complexity $\mathcal{O}(k^2 m^4 \cdot \varepsilon^{-1} \log^3(km^4/\varepsilon))$ and runtime $\mathcal{O}(k^2 m^{10} \cdot \varepsilon^{-1} \log^3(km^4/\varepsilon))$. Unfortunately, their tester for the edit distance, which is of more interest to us, contains a fatal error in one of the key lemmas, which is why we do not give its complexities here[2]. Finally, François et al. [5, 6] gave a tester for regular languages under the weighted edit distance with query complexity $\mathcal{O}(m^3 \cdot \varepsilon^{-2})$ and $\mathcal{O}(km^5 \cdot \varepsilon^{-2})$ running time.

Table 1 Summary of property testing algorithms for regular languages, assuming that a regular language is described by an automaton on a constant-size alphabet Σ with m states and k strongly connected components.

	Queries	**Time**	**Distance**
Alon et al. [1]	$\mathcal{O}(k^2 m \cdot \varepsilon^{-1} \log^3(m/\varepsilon))$	$\mathcal{O}(2^{m^2} + \varepsilon^{-k})$	Hamming
Magniez et De Rougemont [17]	$\mathcal{O}(m^3 \cdot \varepsilon^{-1} \log^2(m/\varepsilon))$	$\mathcal{O}(2^m m^5 \cdot \varepsilon^{-1} \log^2(m/\varepsilon))$	edit w. moves
Fischer et al. [3]	$\mathcal{O}(\frac{\Sigma^{2/\varepsilon} \log\lvert\Sigma\rvert}{\varepsilon^4})$	$m^{\lvert\Sigma\rvert^{\mathcal{O}(1/\varepsilon)}}$	edit w. moves
Ndione et al. [19, 20]	$\mathcal{O}(k^2 m^4 \cdot \varepsilon^{-1} \log^3(km^4/\varepsilon))$	$\mathcal{O}(k^2 m^{10} \cdot \varepsilon^{-1} \log^3(km^4/\varepsilon))$	Hamming
François et al. [6]	$\mathcal{O}(m^3 \cdot \varepsilon^{-2})$	$\mathcal{O}(km^5 \cdot \varepsilon^{-2})$	weight. edit
This work	$\mathcal{O}(km \cdot \varepsilon^{-1} \log(m/\varepsilon))$	$\mathcal{O}(km^3 \cdot \varepsilon^{-1} \log(m/\varepsilon))$	weight. edit

Our main contribution is a tight bound on query complexity of property testing of regular languages under the weighted edit distance. First, we show a new property tester with query complexity $\mathcal{O}(km \cdot \varepsilon^{-1} \log(m/\varepsilon))$ and time complexity $\mathcal{O}(km^3 \cdot \varepsilon^{-1} \log(m/\varepsilon))$(Theorem 5). Essentially, our tester is very simple: it samples a set of short factors of the input string u, and checks whether the factors can be complemented to a word that belongs to the given regular language L. The analysis is much more involved. Our inspiration originates from the ideas from [5, 19, 20], which we extend in a non-trivial way to show a better (and correct) bound. In Theorem 15, we complement the upper bound with a matching lower bound. As the Hamming distance is always larger than the edit distance, the bound also holds for the Hamming distance, improving the lower bound by Alon et al. [1]. See Table 1 for a summary of previously known results and comparison with our work.

[1] We use the following asymptotic notation: For functions $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$, $f(n) = \bar{\Omega}(g(n))$ holds if $f(n) \geq c \cdot g(n)$ for some $c > 0$ and infinitely many $n \in \mathbb{N}$.

[2] The error is in [19, Lemma 12], namely, the value l' chosen in the last two sentences of the proof does not necessarily exist (confirmed via personal communication with the authors).

As an almost straightforward application of our result, in Section 4 we plug our property tester into the algorithm of François et al. [5] to show an improved space bound for streaming property testing for the class of visibly pushdown languages (VPL) (that, in particular, contains regular languages and $\textsc{Dyck}(s)$). Informally, a streaming property tester for a language L is an algorithm that receives an input word u of length n as a stream, one symbol at a time, and must accept if $u \in L$ with probability one and reject if the distance from u to L is at least εn with probability at least $2/3$. Assuming that the automaton that specifies a VPL L is of constant size, the streaming property tester of François et al. [6] requires $\mathcal{O}(\varepsilon^{-4} \log^6 n)$ bits of space. Our result improves this bound to $\mathcal{O}(\varepsilon^{-3} \log^5 n \log\log n)$ (Corollary 22). François et al. [5] showed that for $\varepsilon = 0$, a streaming property tester for visibly pushdown languages must use $\Omega(n)$ bits, even when randomisation is allowed. As our final contribution, we show a space lower bound of $\bar{\Omega}(\max(\varepsilon^{-1}, \log n))$ bits (Theorem 23), thus narrowing the gap between the best existing bounds by $\log^2 n/\log\log n$.

Apart from VPL, the problem of testing membership in formal languages in the streaming setting has been studied for $\textsc{Dyck}(s)$ [14, 16, 18], and for DLIN and $\mathsf{LL}(k)$ [2]. A variant of the streaming setting, called the sliding window model, where one must decide the membership for the n-length suffix of the stream after each symbol arrival, has been considered for regular languages [9, 10, 11, 12], VPL [8], and context-free languages [13].

2 Preliminaries

An alphabet Σ is a finite set the elements of which are called symbols. The length of a word u, denoted $|u|$, is the number of symbols in u. For $1 \leq i \leq j \leq |u|$, we let $u[i]$ denote the i-th symbol in u, and $v = u[i, j]$ the word $u[i] \ldots u[j]$, which we call a *factor* of u. A factor of length l is called an *l-factor*. If $i = 1$, then v is called a *prefix* of u, and if $j = n$, a *suffix*. We let Σ^n denote the set of all n-length words over Σ and $\Sigma^* = \bigcup_{n\in\mathbb{N}} \Sigma^n \cup \epsilon$, where ϵ is the empty word. Any subset of Σ^* is called a *language*.

The *edit distance* between two words u and v, $\mathsf{ed}(u, v)$, is defined as the minimum number of deletions, insertions, and substitutions of symbols required to transform u into v. The *indel distance* between two words u, v, $\delta(u, v)$, is defined as the smallest number of insertions and deletions needed to convert u into v.

We say that a word $u \in \Sigma^*$ is *weighted* if each position i of u has a non-zero integer weight that we denote by $\mathsf{weight}(u[i])$. We define the weight of u, $\mathsf{weight}(u)$, as the sum of the weights of its positions. The *weighted edit distance* between a weighted word u and a (non-weighted) word v, $\mathsf{wed}(u, v)$, is defined as the minimum cost of deletions and insertions of symbols that we must apply to u to obtain v. The cost of deletion or insertion of a symbol is equal to its weight. (Substitutions are not allowed.) For example, if $u = abb$ and the weights of the positions are $3, 2, 2$, and $v = abc$ then $\mathsf{wed}(u, v) = 3$: we delete $u[2] = b$ with weight 2, and insert c with weight 1.

▶ **Observation 1.** *Consider two words u, v, and assume that the weight of any position in u equals 1. We then have $\mathsf{ed}(u, v) \leq \mathsf{wed}(u, v) \leq 2\mathsf{ed}(u, v)$.*

▶ **Definition 2.** *The* weight distribution *over a word $u \in \Sigma^n$ is the probability distribution over $\{1, \ldots n\}$ where the probability to sample a position i is equal to $\mathsf{weight}(u[i])/\mathsf{weight}(u)$.*

▶ **Definition 3.** *A* non-deterministic finite automaton (NFA) *is defined as a tuple $A = (\Sigma, Q, Q_{in}, Q_f, \Delta)$, where:*

- *Σ is a finite input alphabet,*
- *Q is a finite set of states $Q_{in} \subseteq Q$ of initial states, and $Q_f \subseteq Q$ of final states,*
- *$\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation. For $(p, a, q) \in \Delta$, we write $p \xrightarrow{a}_{\Delta} q$.*

For a word $u = u[1] \dots u[n]$ and states $p, q \in Q$, we write $p \xrightarrow{u}_{\Delta} q$ if there exists a sequence of states $p = q_0, \dots, q_n = q$ such that $\forall i = 1, \dots, n \;\; q_{i-1} \xrightarrow{u[i]}_{\Delta} q_i$. The sequence $q_0, \dots, q_n$ is called a *run* labelled by u. A word u is recognized by A if there exists $(q_{in}, q_f) \in Q_{in} \times Q_f$ such that $q_{in} \xrightarrow{u}_{\Delta} q_f$. We write $L(A)$ for the set of all words recognized by A. A language $L \subseteq \Sigma^*$ is *regular* if there exists an NFA A such that $L = L(A)$.

Property testing model

For a given distance function d on Σ^*, we define the distance from a word $u \in \Sigma^*$ to L, $d(u, L) = \min_{w \in L} d(u, w)$. If a word u is weighted, we say that it is ε-*far* from L if $d(u, L) \geq \varepsilon \cdot \mathsf{weight}(u)$. In the unweighted case, if $d(u, L) \geq \varepsilon \cdot |u|$.

In the property testing model, we assume that we can query any symbol of the input according to the weight distribution over it in constant time. In the unweighted case, this is equivalent to constant-time random access.

▶ **Definition 4** (Property tester). *An ε-*property tester *for a language L for a distance d is a randomised algorithm that:*

- *accepts if $u \in L$ with probability 1,*
- *rejects with probability at least $2/3$ if u is ε-far from L under d,*
- *accepts or rejects otherwise.*

In addition to standard complexity measures, we are interested in so-called *query complexity* which is defined to be the number of symbols of u that a tester queries. The time complexity is defined as usual, and the space complexity of a tester is defined as space used beyond the space required to store the input.

Property testers can be *non-adaptive* (the symbols to query are selected offline) and *adaptive* (the position of the i-th queried symbol depends on the first $i - 1$ queried symbols). In this work, we focus on non-adaptive testers.

3 Property testing of regular languages

In this section, we show an improved upper bound and a matching lower bound on the query complexity of property testing of regular languages under weighted edit distance.

3.1 Upper bound

We start by showing the following theorem:

▶ **Theorem 5.** *Let $A = (\Sigma, Q, Q_{in}, Q_f, \Delta)$ be an NFA, $m = |Q|$, and k be the number of strongly connected components in the underlying graph. There exists an ε-property tester for membership of a weighted word u in the regular language $L = L(A)$ under the weighted edit distance. The query complexity of the tester is $\mathcal{O}(km \cdot \varepsilon^{-1} \log(m/\varepsilon))$ and the time complexity is $\mathcal{O}(km^3 \cdot \varepsilon^{-1} \log(m/\varepsilon))$, assuming constant-size alphabet.*

3.1.1 Combinatorics of blocking factors and fragments

▶ **Definition 6** (Blocking factor). *Consider a strongly connected component C of A. We say that a factor v of a word u is a C-*blocking *factor if for any two states p, q of C we have $p \overset{v}{\not\to}_{\Delta} q$.*

We say that a sequence of components $P = (C_1, \dots, C_j)$ is a *component path* of A if for any $1 \le i \le j-1$ there is $p \in C_i, q \in C_{i+1}$ such that $p \xrightarrow{a}_\Delta q$ for some symbol $a \in \Sigma$. Note that $j \le k \le m$.

▶ **Definition 7** (P-partition, β-saturation)**.** *Let $u \in \Sigma^*, \beta > 0$ and $P = (C_1, \dots, C_j)$ be a component path. We build the P-*partition *of u recursively. We start with $i = 1$. Let $u = vxu'$, where $v \in \Sigma^*$, $x \in \Sigma$, and vx is the shortest C_i-blocking prefix of u. If $\mathsf{weight}(x) > \beta \cdot \mathsf{weight}(u)$, we say that x is* heavy*, and otherwise we call it* light*. Consider three cases:*

- *If x is not C_i-blocking, add vx to the set $B_i(P)$;*
- *If x is C_i-blocking and heavy, add it to the set $H_i(P)$;*
- *If x is C_i-blocking and light, add it to the set $L_i(P)$.*

Recurse for $u = u'$ and the value of i computed as follows:

$$i = \begin{cases} \min_{i'}(\{i' \mid i' > i, \exists p \in C_{i'-1} \cup C_{i'}, q \in C_{i'}, p \xrightarrow{x}_\Delta q\} \cup \{j+1\}), \text{ if } x \text{ is heavy;} \\ i+1, \text{ if } x \text{ is light and } \big(|B_i(P)| \ge \beta \cdot \mathsf{weight}(u) \text{ or } \mathsf{weight}(L_i(P)) \ge \beta \cdot \mathsf{weight}(u)\big); \\ i, \text{ otherwise.} \end{cases}$$

*Stop if $i = j+1$, if u is empty, or if there is no C_i-blocking factor in u. If we reach $i = j+1$, we say that u β-*saturates *P. The sets $B_i(P), H_i(P), L_i(P), i \le j$ form the P-partition of u.*

We show that if a word is ε-far from L, then it β-saturates P for $\beta = \varepsilon/(6m)$. The proof is by contradiction: we show that if a word does not β-saturate P, then we can obtain a word in L by deleting the factors in $\cup_i L_i(P)$ and then inserting a small number of factors of length and weight at most m.

▶ **Lemma 8.** *Let $u \in \Sigma^*$ be such that $\mathsf{weight}(u) \ge 6km/\varepsilon$ and $P = (C_1, \dots, C_j)$ be a component path of A. Let $\beta = \varepsilon/(6m)$. If u is ε-far from L, then u β-saturates P.*

Proof. By contrapositive: assume that u does not β-saturate P, in other words, the algorithm that built the P-partition of u reached the end of u but $i \ne j+1$. We will show that there is a word $u' \in L$ such that the weighted edit distance between u and u' is at most $\varepsilon \cdot \mathsf{weight}(u)$. We first delete all the elements in $L_i(P)$, for all i. Note that $\mathsf{weight}(L_i(P)) \le 2\beta \cdot \mathsf{weight}(u)$ for any i, so the total weight of the deleted factors is at most $2k\beta \cdot \mathsf{weight}(u) = \varepsilon \cdot \mathsf{weight}(u)k/(3m)$. Next, we edit the elements of $B_i(P)$ as follows: let $v \in B_i(P)$, $v = v'x$ where $v' \in \Sigma^*, x \in \Sigma$. Since v is a minimal blocking prefix, v' is not C_i-blocking, and there exist $p, q \in C_i$ such that $p \xrightarrow{v'}_\Delta q$. Similarly, as $v \in B_i(P)$, x is not C_i-blocking and labels a run within C_i, from p' to q'. Therefore, we can edit v into $v'' = v'wx$, where $q \xrightarrow{w}_\Delta p'$ and $|w| \le |C_i|$. Since C_i is strongly connected, we can similarly insert factors of length at most $|C_i|$ between the elements of $B_i(P)$ to obtain a word w_i that labels a run within C_i. Note that we can choose the weights of the inserted symbols arbitrarily, and we put them equal to one. Hence, the cost of this step is at most $|B_i(P)| \cdot |C_i|$ for each i. Finally, we insert factors of length at most m in between the words w_i and the heavy blocking 1-factors in $H_i(P)$, as well as before and after the last factor, so that the resulting word u' belongs to L. Again, we put the weights of the symbols of the inserted factors equal to one, and hence the cost of this step is at most $2km$. We have the following bound on the weighted edit distance of u to L:

$$\begin{aligned} d(u, L) \le d(u, u') &\le \varepsilon \cdot \mathsf{weight}(u)k/(3m) + 2km + \sum_{i=1}^{j} 2 \cdot |B_i(P)| \cdot |C_i| \\ &\le \varepsilon(\mathsf{weight}(u)/3 + \mathsf{weight}(u)k/(3m)) + 2km \le \varepsilon \cdot \mathsf{weight}(u) \end{aligned}$$

A contradiction. ◀

Suppose that u β-saturates a component path $P = (C_1, \dots C_j)$, in other words, the algorithm that built the P-partition reached $i = j + 1$. For every $1 \leq i \leq j$, let $S_i(P) = B_i(P) \cup L_i(P) \cup H_i(P)$. We define indices $0 = a_0 \leq \cdots \leq a_j$ so that for each i either $S_i(P)$ is empty (because C_i was skipped by the algorithm of Definition 7 due to a heavy factor), or all the factors in it are the factors of $u[a_{i-1} + 1, a_i]$. For $i = 0$, we let $a_0 = 0$, and then for all i, $a_i = a_{i-1}$ if $S_i(P) = \emptyset$ and the largest index of a symbol in $S_i(P)$ otherwise. If $S_i(P) \neq \emptyset$, then by Definition 7, one of the following holds:

- The total weight of C_i-blocking 1-factors in $H_i(P) \cup L_i(P)$ is at least $\beta \cdot \mathsf{weight}(u)$;
- $|B_i(P)| \geq \beta \cdot \mathsf{weight}(u)$.

Blocking factors are witnesses of the fact that $u \notin L$. If there are many short blocking factors, as in the first case, we can sample them with a few queries. However, in the second case, the factors can be arbitrarily long. To develop an efficient tester, we must give a more accurate bound.

▶ **Corollary 9.** *Let $\gamma = \lceil 2/\beta \rceil = \lceil 12m/\varepsilon \rceil$. If $|B_i(P)| \geq \beta \cdot \mathsf{weight}(u)$, then $B_i(P)$ contains at least $\mathsf{weight}(u)/\gamma$ disjoint C_i-blocking factors of length at most γ.*

Proof. Since the factors in $B_i(P)$ are disjoint and their total weight is bounded by $\mathsf{weight}(u)$, at most $\mathsf{weight}(u)/\gamma$ of them can have weight (and length) larger than γ. Consequently, $B_i(P)$ contains at least $\beta \cdot \mathsf{weight}(u) - \mathsf{weight}(u)/\gamma \geq \mathsf{weight}(u)/\gamma$ factors of length $\leq \gamma$. ◀

Let us now introduce the notion of a fragment that will allow us to test the sampled factors efficiently.

▶ **Definition 10** (Fragment). *Given a set of factors $u[i_r, j_r]$, $1 \leq r \leq t$, consider the decomposition of the set $S = \bigcup_{1 \leq r \leq t} [i_r, j_r]$ into maximal disjoint intervals, that is, $S = \sqcup_{1 \leq r \leq t'} [i'_r, j'_r]$, where $i'_1 \leq j'_1 < i'_2 \leq j'_2 < \dots i'_{t'} \leq j'_{t'}$. The* fragment *$F$ formed by the factors is the word $F = *\ u[i'_1, j'_1] * u[i'_2, j'_2] * \cdots * u[i'_{t'}, j'_{t'}]\ *$, where "$*$" is a special symbol not in Σ. A word v* contains F as a fragment *if there exist words $w_1, \dots w_{t'+1} \in \Sigma^*$ such that by replacing the i-th symbol "$*$" with w_i in F, we obtain v.*

For example, for $u = abbabbab$ the fragment formed by factors $u[1, 2]$, $u[2, 3]$, $u[4, 4]$, and $u[6, 8]$, is $F = *\ u[1, 4] * u[6, 8]\ * = *\ abba * bab\ *$. The word $cabbadebabcde$ contains F.

▶ **Definition 11** (Blocking fragment). *A fragment F is A*-blocking *if none of the words that contain F as a fragment belongs to $L(A)$.*

The following lemma relates C_i-blocking factors and A-blocking fragments and is crucial for correctness of our tester. To show it, we prove by induction that any run that starts in C_1 and is labelled by the prefix of the fragment containing the factors in $\cup_{1 \leq i \leq t} H_i(P)$ and at least one C_i-blocking factor from each $S_i \neq \emptyset$, $1 \leq i \leq t$, must end at a state in $\cup_{i \geq t+1} C_i$.

▶ **Lemma 12.** *If for any component path $P = (C_1, \dots, C_j)$ and i, the fragment F contains all factors in $\bigcup_i H_i(P)$, and a C_i-blocking factor from $u[a_{i-1} + 1, a_i]$ for each i such that $H_i(P) = \emptyset$ and $L_i(P) \cup B_i(P) \neq \emptyset$, then F is A-blocking.*

Proof. By contradiction, suppose that there is a word $w \in L(A)$ that contains F as a fragment. Since $w \in A$, there is a run in A labelled by w that goes through the connected components $C_1, \dots, C_j$. Consider the path $P = (C_1, \dots, C_j)$ and let $0 \leq a_0 \leq a_1 \cdots \leq a_j$ be the indices such that for each i either $S_i(P)$ is empty (because C_i was skipped by the algorithm of Definition 7 due to a heavy factor), or all the factors in it are the factors of $u[a_{i-1} + 1, a_i]$. Let w_t be the shortest prefix of w that contains the blocking factors from each of the non-empty intervals $u[a_{i-1}+1, a_i]$, $1 \leq i \leq t$, as well as all factors in $\bigcup_{1\leq i \leq t} H_i(P)$. We show by induction on t that every run that starts at a state in C_1 and is labelled by w_t ends after C_t, i.e. in a state of $\bigcup_{t'>t} C_{t'}$.

As w_1 contains a C_1-blocking factor, any run labelled by w_1 that starts in C_1 exits C_1 and therefore ends after C_1. Suppose that the induction hypothesis holds for some $t' < j$. We show that there is no run labelled by $w_{t'+1}$ from C_1 to $C_{t'+1}$. There are two possibilities. First, $C_{t'+1}$ was skipped because of a heavy C_t-blocking factor, for some $t \leq t'$, which is therefore included in $w_{t'+1}$, and ends after $C_{t'+1}$. The other possibility is that $w_{t'+1}$ is $w_{t'}$ with an additional $C_{t'+1}$-blocking factor. If any run labelled by $w_{t'}$ ends after $C_{t'+1}$, so does the run labelled by $w_{t'+1}$, and we are done. If some of these runs end in $C_{t'+1}$, the $C_{t'+1}$-blocking factor that appears in $w_{t'+1}$ but not in $w_{t'}$ ensures that the run labelled by $w_{t'+1}$ ends after $C_{t'+1}$.

Adding symbols to w_j to obtain w, we get that there is no run labelled by w that starts in C_1 and ends in C_j, a contradiction. ◀

3.1.2 Tester

Our tester is as stated in Algorithm 1. We define the *l-factor sampling* over u as the distribution over factors v of u that have length at most l, where a position i is selected according to the weight distribution over u, and $v = u[i, \min(i+l-1, |u|)]$.

Algorithm 1 ε-property tester for regular languages.

```
$\beta \leftarrow \varepsilon/(6m)$, $\gamma \leftarrow 2/\beta$
if weight$(u) \leq 6km/\varepsilon$ then
    Query all symbols of $u$ and run the automaton $A$ on it
    Reject if $A$ rejects, else accept
else
    Query $\tau = \lceil 2\ln(9k \cdot 2^k)/\beta \rceil$ 1-factors of $u$
    for $t = 0$ to $T = \lceil \log(2\gamma) \rceil$ do
        $\ell_t \leftarrow 2^t, r_t \leftarrow \lceil 2\ln(9k \cdot 2^k)\gamma/\ell_t \rceil$
        Query $u[1, 2\ell_t]$
        Query $r_t$ factors of $u$ according to the $2\ell_t$-factor distribution
    Reject if the fragment formed by the sampled factors is $A$-blocking, else accept
```

If weight$(u) \leq 6km/\varepsilon$, the length of u is at most $6km/\varepsilon$ as well, and hence Algorithm 1 queries $\mathcal{O}(6km/\varepsilon)$ symbols. Otherwise, Algorithm 1 first makes $\mathcal{O}(2\ln(9k \cdot 2^k)/\beta) = \mathcal{O}(km \cdot \varepsilon^{-1})$ queries to sample 1-factors, and then for each $t = 0, \ldots, T$, another $2^{t+1} \cdot (r_t + 1) = \mathcal{O}(4\ln(9k2^k)\gamma) = \mathcal{O}(km \cdot \varepsilon^{-1})$ queries. Hence, it has query complexity $O(km \cdot \varepsilon^{-1} \log(m/\varepsilon))$. To estimate the time complexity, we must explain how we check if the sampled fragment F is A-blocking. Given a fragment F and $S \subseteq Q$, let reach(F, S) denote the set of states that can be reached from a state of S when following a run labelled by some word v that contains F as a fragment, i.e.

$$\text{reach}(F, S) = \{q \in Q \mid \exists p \in S, v \in \Sigma^* \ : \ p \xrightarrow{v}_{\Delta} q \text{ and } v \text{ contains the fragment } F\}$$

By definition, F is A-blocking if and only if reach$(F, Q_{in}) \cap Q_f = \emptyset$.

▶ **Lemma 13.** *For any F, S, reach(F, S) can be computed in time $\mathcal{O}(|F| \cdot m^2)$.*

Proof. Recall that a constant-size alphabet is assumed. If $|F| = 1$, $\mathsf{reach}(F, S)$ can be computed in time and space $\mathcal{O}(m^2)$. If $F = a \in \Sigma$, $\mathsf{reach}(a, S) = \{q \in Q \mid \exists p \in S, p \xrightarrow{a}_{\Delta} q\}$. In this case, we can compute $\mathsf{reach}(a, S)$ by following every transition that is labelled by a and starts at a state $p \in S$. If $F = *$, $\mathsf{reach}(*, S)$ is the set of states q such that there exists a path from a state $p \in S$ to q. It can therefore be computed using a breadth-first traversal of the graph induced by the automaton, initialized at every $p \in S$. For any F, $|F| > 1$, we have $\mathsf{reach}(F, S) = \mathsf{reach}(F[2, |F|], \mathsf{reach}(F[1], S))$. Therefore, we can compute $\mathsf{reach}(F, S)$ in a recursive manner: let $S_0 = S$, and for every $1 \le i \le |F|$, $S_i = \mathsf{reach}(F[i], S_{i-1})$. By induction, we have $S_{|F|} = \mathsf{reach}(F, S)$. The algorithm makes $|F|$ calls to the reach function on a single symbol, which requires $\mathcal{O}(|F| \cdot m^2)$ time. ◀

The fragment F constructed in Algorithm 1 has size $\mathcal{O}(km \cdot \varepsilon^{-1} \log(m/\varepsilon))$, and therefore the time complexity of the tester is $\mathcal{O}(km^3 \cdot \varepsilon^{-1} \log(m/\varepsilon))$.

3.1.3 Correctness of the tester

In what follows, let $W := \mathsf{weight}(u)$. If $W \le 6km/\varepsilon$, we query all symbols of u and run the automaton on u, the answer is correct with probability 1. Below we assume that $W > 6km/\varepsilon$.

If $u \in L$, there is no A-blocking fragment in u, and therefore the tester accepts with probability 1. We must now show that if u is ε-far from L, then the tester accepts with probability at most 1/3, or in other words, the probability that F is not blocking is at most 1/3. By Lemma 12, the probability that F is not blocking is smaller than the probability that there exist a component path $P = (C_1, \ldots, C_j)$ and an index $i, 1 \le i \le j$ such that F contains neither the factor from $H_i(P)$ (if $H_i(P) \neq \emptyset$), nor a factor from $L_i(P) \cup B_i(P)$ (if $H_i(P) = \emptyset$). Fix a path P and an index i. If $\mathsf{weight}(H_i(P)) \ge \beta \cdot W$ or $\mathsf{weight}(L_i(P)) \ge \beta \cdot W$, then by sampling independently $\ln(9k \cdot 2^k)/\beta$ factors of length 1 w.r.t. the 1-factor distribution, we miss such a factor with probability at most $1/(9k \cdot 2^k)$. Otherwise, if $S_i \neq \emptyset$, we have $|B_i(P)| \ge \beta \cdot W$.

▶ **Lemma 14.** *Assume $W > 6km/\varepsilon$ and $|B_i(P)| \ge \beta \cdot W$. Algorithm 1 fails to sample a C_i-blocking factor with probability at most $1/(9k2^k)$.*

Proof. Consider a fixed $t \in [0, T]$. If $u[1, 2\ell_t]$ contains a factor from $B_i(P)$, $u[1, 2\ell_t]$ is C_i-blocking and we are done. Assume that this is not the case.

We estimate the number of $2\ell_t$-factors that contain a factor from $B_i(P)$. For brevity, let $B' = \{v \in B_i(P) : |v| \le \gamma\}$. Let $v_1, v_2, \ldots, v_{f_t}$ be the factors in B' of length at most ℓ_t, in the order of appearance in u. For all $j > 1$, the number of $2\ell_t$-factors such that v_j is the first factor appearing in them is equal to $\min(2\ell_t - |v_j| + 1, \mathsf{dist}(v_{j-1}, v_j))$, where $\mathsf{dist}(v_{j-1}, v_j)$ is the difference between the starting positions of v_{j-1} and v_j. Since $|v_j| \le \ell_t$, we have $2\ell_t - |v_j| \ge \ell_t$. We also have $\mathsf{dist}(v_{j-1}, v_j) \ge |v_{j-1}|$, since the factors v_j and v_{j-1} are disjoint. Therefore, for all $j > 1$, $\min(2\ell_t - |v_j| + 1, \mathsf{dist}(v_{j-1}, v_j)) \ge \min(\ell_t, |v_{j-1}|) = |v_{j-1}|$. The first term corresponds to the case where there is no interval of length $2\ell_t$ that contains both v_{j-1} and v_j: if v_j starts at a position p in u, it ends at position $p + |v_j| - 1$, and the interval can start at any position p' such that $p' \le p$ and $p' + 2\ell_t - 1 \ge p + |v_j| - 1$, i.e. $p - (2\ell_t - |v_j|) \le p'$. The second term is equal to the number of positions between the start of v_{j-1} and the start of v_j in u: if an interval that contains v_j also contains v_{j-1}, then it does not contain v_j as its first blocking factor, and therefore it is associated with v_{j-1}. By summing over all j, we obtain that the number of $2\ell_t$-factors containing a factor from B' is at least

$$2\ell_t - |v_1| + \sum_{j=1}^{f_t} |v_{j-1}| \geq \ell_t + \sum_{j=0}^{f_t-1} |v_j| \geq l_{v_{f_t}} + \sum_{j=0}^{f_t-1} |v_j| \geq \sum_{j=0}^{f_t} |v_j|.$$

Let p_t be the probability that a factor of length $2\ell_t = 2^{t+1}$ sampled according to the $2\ell_t$-factor distribution is C_i-blocking. As any factor containing a factor from B' is C_i-blocking, from above we obtain that $p_t \geq \frac{1}{W}\sum_{j=0}^{f_t} |v_j| \geq \frac{1}{W}\sum_{v\in B':|v|\leq \ell_t} |v|$.

Consequently, for a fixed t, the probability that none of the r_t factors is C_i-blocking is at most $(1-p_t)^{r_t} \leq e^{-p_t r_t}$. By independence, the probability p that the algorithm failed to sample a C_i-blocking factor for every t satisfies:

$$p \leq \prod_{t=0}^{T} \exp(-p_t r_t) \leq \exp\left(\sum_{t=0}^{T} -p_t r_t\right) \leq \exp\left(-\sum_{t=0}^{T} \frac{2\ln(9k\cdot 2^k)\gamma 2^{-t}}{W} \sum_{v\in B':|v|\leq \ell_t} |v|\right).$$

We now show that $\sum_{t=0}^{T} 2^{-t}\sum_{v\in B':|v|\leq \ell_t} |v| \geq W/(2\gamma)$, which implies that $p \leq e^{-\ln(9k\cdot 2^k)} = 1/(9k2^k)$. We have:

$$\begin{aligned}\sum_{t=0}^{T} 2^{-t} \sum_{v\in B':|v|\leq \ell_t} |v| &= \sum_{v\in B'} |v| \sum_{t=\lceil \log(|v|)\rceil}^{T} 2^{-t} = \sum_{v\in B'} |v| \frac{1}{2^{\lceil \log |v|\rceil}} \frac{1-2^{-(T-\lceil \log |v|\rceil+1)}}{1-1/2} \\ &\geq \sum_{v\in B'} \left(1-2^{-(T-\lceil \log |v|\rceil+1)}\right) \geq \sum_{v\in B'} 1-\frac{|v|}{2\gamma} \geq \sum_{v\in B'} 1/2 \geq W/(2\gamma),\end{aligned}$$

where the last inequality holds because of Corollary 9. ◀

Hence, for fixed P and i, the probability that the fragment F built by Algorithm 1 contains neither the factor from $H_i(P)$ (if $H_i(P) \neq \emptyset$) nor a factor from $L_i(P)\cup B_i(P)$ (if $H_i(P) = \emptyset$) is bounded from above by $\left(\frac{1}{9k2^k}+\frac{1}{9k2^k}+\frac{1}{9k2^k}\right) \leq \frac{1}{3k2^k}$. By the union bound over all P and all k, and since there are at most 2^k component paths in A, we obtain that $\Pr[F \text{ is not blocking}] \leq \frac{1}{3}$. This concludes the proof of Theorem 5.

3.2 Lower bound

In this section we show that the query complexity of Theorem 5 is tight. Note that the indel distance is the weighted edit distance when all weights are equal to one, and hence it suffices to show the lower bound for the former.

▶ **Theorem 15.** *There exists a regular language L and constants $\varepsilon_0, C > 0$ such that on an input of length n and for any $1/n^{1/3} < \varepsilon < \varepsilon_0$, a non-adaptive ε-property tester for L under the indel distance has query complexity $\geq C\varepsilon^{-1}\log(1/\varepsilon)$.*

Note that by running a property tester twice, the error probability can be reduced from 1/3 to 1/9, while increasing the query complexity by a factor of two. From that and by Yao's minimax principle [23], it suffices to find a distribution $\mathcal{D}$ on $\{a,b,c,d\}^*$ and a constant $C > 0$ such that any *deterministic* algorithm A that makes at most $C\varepsilon^{-1}\log(1/\varepsilon)$ queries and accepts all inputs in L, errors with probability $> 1/9$.

Let n be the length of the input. Below, we fix arbitrarily a precision parameter $1/n^{1/3} \leq \varepsilon < 1/2^{-48}$. Consider regular languages $L_0 = \{u \in \{a,c\}^* : u \text{ contains an even number of } c\}$ and $L = (a \mid bL_0d)^*$. Define the distribution $\mathcal{D}$ as follows. Let r be a fair coin. For $\ell = \lceil 5/\varepsilon\rceil$, divide $[1,n]$ into $z = \lfloor n/\ell \rfloor \geq 1$ intervals of size ℓ each, except for the last one that can be longer. Let $[a_j, b_j]$ be the j-th interval. We associate with it a random variable τ_j, distributed as follows:

$$\tau_j = \begin{cases} t, & \text{with probability } p_t = 12\varepsilon 2^t / \log(1/\varepsilon) \text{ for } t = 1, 2, \ldots, \lceil \log(1/\varepsilon) \rceil; \\ 0, & \text{with probability } p_0 = 1 - \sum_{t=1}^{\lceil \log(1/\varepsilon) \rceil} p_t. \end{cases}$$

For any $\varepsilon < 2^{-48}$, we have $p_0 = 1 - \sum_{t=1}^{\lceil \log(1/\varepsilon) \rceil} p_t > 0$, and hence the distribution is well-defined. The variable τ_j characterizes the length of the instances of L_0 that we will put in the j-th interval. If $\tau_j = 0$, we put no instances at all: set $u[a_j, b_j] = aa \ldots a$. If $\tau_j = t > 0$, set $u[a_j, b_j] = (bw_jd)^{\lceil 2^{-t}/\varepsilon \rceil} aa \ldots a$, where w_j is a word of length 2^t chosen uniformly at random from L_0 if $r = 0$ and from $\{a, c\}^* \setminus L_0$ otherwise.

Notice that $\mathcal{D}$ produces a positive instance with probability at least $1/2$, since whenever $r = 0$, $u \in L$. We prove in the following lemma that $\mathcal{D}$ also produces a word ε-far from L with constant probability.

▶ **Lemma 16.** *Let u be a word of length n sampled w.r.t. $\mathcal{D}$. With probability at least $1/4$, $\delta(u, L) \geq \varepsilon n$.*

Proof. Assume $r = 1$ and let $\xi_j = \lceil 2^{-\tau_j}/\varepsilon \rceil$ if $\tau_j > 0$, and 0 otherwise (ξ_j is the number of instances of L_0 in the j-th interval). The variables ξ_j are independent, and for every j, we have $1 \leq \xi_j \leq 1/\varepsilon$. Let $\xi = \left(\sum_{j=1}^{z} \xi_j\right)/z$. We have:

$$\mathbb{E}[\xi] = \mathbb{E}[\xi_j] = \sum_{t>0} p_t \cdot \lceil 2^{-t}/\varepsilon \rceil = \sum_{t>0} \frac{12\varepsilon 2^t}{\log(1/\varepsilon)} \cdot \lceil 2^{-t}/\varepsilon \rceil \geq \sum_{t>0} \frac{12}{\log(1/\varepsilon)} \geq 12.$$

By Hoeffding's inequality, we have $\Pr[\xi \geq 6] \geq 1 - e^{-\frac{72z^2}{z/\varepsilon^2}} = 1 - e^{-72\varepsilon^2 z} \geq 1/2$ and $z = \lfloor n / \lceil 5/\varepsilon \rceil \rfloor \geq \varepsilon n/6$, as $1/n^{1/3} \leq \varepsilon < 2^{-48}$. Hence, $\Pr\left[\sum_j \xi_j \geq \varepsilon n\right] \geq \Pr[\xi \geq 6] \geq 1/2$.

It remains to show that conditioned on $r = 1$, the indel distance between u and L is at least $\sum_j \xi_j$. As each word of form bw_jd, where $w_j \in \{a, c\}^* \setminus L_0$ requires at least one insertion or deletion to become a factor of a word in L, the bound follows by construction. Hence, $\Pr[\delta(u, L) \geq \varepsilon n] \geq \Pr[r = 1] \cdot \Pr\left[\sum_j \xi_j \geq \varepsilon n\right] \geq 1/4$. ◀

▶ **Lemma 17.** *Let q_j be the number of symbols queried by A in the j-th interval $[a_j, b_j]$. Consider an input u sampled w.r.t. $\mathcal{D}$. If for all j such that $\tau_j > 0$ we have $q_j < 2^{\tau_j}$, then A must accept u.*

Proof. Assume that A rejects u. If $u \in L$, we immediately get a contradiction. If u is a negative instance, then for all j such that $\tau_j > 0$ we have $u[a_j, b_j] = (bw_jd)^{\lceil 2^{-\tau_j}/\varepsilon \rceil} aa \ldots a$, where w_j is a word of length 2^{τ_j} not in L_0. Since A queries less than 2^{τ_j} symbols in $u[a_j, b_j]$, there is a symbol of w_j that A does not query. Hence, there is a word in L_0 that we denote by w'_j that has length 2^{τ_j} and does not differ from w_j on the queried symbols. We replace all copies of w_j by w'_j. Let u' denote the resulting word. Notice that $u' \in L$, and all the symbols for u and u' in the queried positions are equal and therefore A rejects it as well. The probability of u' under the distribution $\mathcal{D}$ is non-zero, and therefore there is a non-zero probability that A rejects a positive instance, a contradiction. ◀

Proof of Theorem 15. We finally derive that if A queries $\sum_j q_j < 1/(3 \cdot 192\varepsilon) \log(1/\varepsilon)$ symbols, then it errs with probability $> 1/9$, which implies the theorem.

Consider an input u of length n generated according to the distribution $\mathcal{D}$. Let $A(u)$ denote the output of A on u: 1 for accepting, 0 for rejecting. Let F denote the event "$\delta(u, L) \geq \varepsilon n$". The probability that A rejects inputs that are far from L is given by:

$$\begin{aligned}
\Pr\left[A(u) = 0 \mid F\right] = \Pr\left[A(u) = 0 \wedge F\right] / \Pr\left[F\right] &\leq \Pr\left[A(u) = 0\right] / \Pr\left[F\right] \\
&\leq 4 \cdot \Pr\left[A(u) = 0\right] \;\; \text{(Lemma 16)} \\
&\leq 4 \cdot \Pr\left[\exists j : q_j \geq 2^{\tau_j} \wedge \tau_j > 0\right] \;\; \text{(Lemma 17)} \\
&\leq 4 \cdot \sum_j \Pr\left[q_j \geq 2^{\tau_j} \wedge \tau_j > 0\right] \;\; \text{(union bound)} \\
&\leq 4 \cdot \sum_j \sum_{t=1}^{\lceil \log q_j \rceil} p_t \leq 4 \sum_j \frac{12\varepsilon}{\log(1/\varepsilon)} \sum_{t=1}^{\lceil \log q_j \rceil} 2^t \leq \frac{192\varepsilon}{\log(1/\varepsilon)} \sum_j q_j < 1/3.
\end{aligned}$$

The probability that A gives an incorrect answer is at least the probability that it accepts a word that is ε-far. In other words, this probability is equal to $\Pr\left[A(u) = 1 \wedge F\right] = \Pr\left[A(u) = 1 \mid F\right] \cdot \Pr\left[F\right] > (2/3) \cdot (1/4) > 1/9$, concluding the proof. ◀

4 Streaming property testing of VPLs

Let us start by reminding the formal definitions of visibly pushdown languages and streaming property testers. Let $\Sigma = \Sigma_+ \sqcup \Sigma_= \sqcup \Sigma_-$. We refer to the symbols in Σ_+ as *push* symbols, and the symbols in Σ_- as *pop* symbols.

▶ **Definition 18** (Visibly pushdown automaton). *A visibly pushdown automaton (VPA) $\mathcal{A}$ over Σ is a tuple $(\Sigma, \Gamma, Q, Q_{in}, Q_f, \Delta)$, where*

- *Γ is a finite set of stack symbols,*
- *Q is a finite set of states, $Q_{in} \subseteq Q$ of initial states, $Q_f \subseteq Q$ of final states,*
- *$\Delta \subseteq (Q \times \Sigma_+ \times Q \times \Gamma) \cup (Q \times \Sigma_= \times Q) \cup (Q \times \Sigma_- \times \Gamma \times Q)$ is the transition relation.*

When running the automaton on a word $u \in \Sigma^n$, we maintain a stack. Let $\perp \notin \Gamma$ be a special symbol to denote the bottom of the stack. A configuration of a VPA $\mathcal{A}$ is a tuple $(\sigma, q) \in (\perp \cdot \Gamma^*) \times Q$. For $a \in \Sigma$, there is a transition from a configuration (σ, q) to (σ', q'), denoted $(\sigma, q) \xrightarrow{a}_\Delta (\sigma', q')$, in the following cases:

- *if $a \in \Sigma_+, \sigma' = \sigma \cdot \gamma, \gamma \in \Gamma$ and $(q, a, q', \gamma) \in \Delta$ (we write $q \xrightarrow{a}_\Delta (q', push(\gamma))$),*
- *if $a \in \Sigma_-, \sigma = \sigma' \cdot \gamma, \gamma \in \Gamma$ and $(q, a, \gamma, q') \in \Delta$ (we write $(q, pop(\gamma)) \xrightarrow{a}_\Delta q'$),*
- *if $a \in \Sigma_=$ and $(q, a, q') \in \Delta$ (we write $q \xrightarrow{a}_\Delta q'$).*

For a word $u \in \Sigma^n$, if for all $1 \leq i \leq n, (\sigma_{i-1}, q_{i-1}) \xrightarrow{u[i]}_\Delta (\sigma_i, q_i)$, we write $(\sigma_0, q_0) \xrightarrow{u}_\Delta (\sigma_n, q_n)$. A word u is accepted by an automaton $\mathcal{A}$ if there exists a initial state $q_{in} \in Q_{in}$ and a final state $q_f \in Q_f$ such that $(\perp, q_{in}) \xrightarrow{u}_\Delta (\perp, q_f)$. We denote the language of all words accepted by $L(\mathcal{A})$. A language $L \subseteq \Sigma^*$ is a visibly pushdown language *(VPL) if $L = L(\mathcal{A})$ for some VPA $\mathcal{A}$.*

▶ **Definition 19** (Streaming property testing algorithm). *A **streaming ε-property testing algorithm** for a language L under distance d is an algorithm that given streaming access to a word u:*

- *accepts if $u \in L$ with probability 1,*
- *rejects with probability at least p if u is ε-far from L w.r.t. d,*
- *accepts or rejects otherwise.*

If $p = 1$, we say that a streaming ε-property testing algorithm is deterministic. *If $p = 2/3$, we say that it is randomised. The* space complexity *of the algorithm is defined to be the total space used (in bits)* including *the space needed to store any information about the input.*

4.1 Upper bound

François et al. [5] showed that streaming property testing of visibly pushdown languages can be reduced to the problem of (approximately) encoding words as relationships in finite automata[3]. Consider a (non-deterministic) finite automaton $A = (\widehat{\Sigma}, \widehat{Q}, \widehat{Q}_{in}, \widehat{Q}_f, \widehat{\Delta})$. For $\widehat{\Sigma}' \subseteq \widehat{\Sigma}$, define a distance function $\mathsf{wed}_{\widehat{\Sigma}'}$ as the weighted edit distance where the insertions are restricted to symbols in $\widehat{\Sigma}'$. For a word u, let $R_u = \{(p,q) \mid p,q \in \widehat{Q}, p \xrightarrow{u}_{\widehat{\Delta}} q\}$.

▶ **Definition 20** (ξ-approximation). *A relation $R \subseteq Q^2$ is an (ξ, Σ')-approximation of R_u if the following two conditions are satisfied:*

- *For all p, q such that $p \xrightarrow{u}_{\widehat{\Delta}} q$, $(p,q) \in R$,*
- *If $(p,q) \in R$, then there exists a word v such that $\mathsf{wed}_{\widehat{\Sigma}'}(u,v) \leq \xi \cdot \mathsf{weight}(u)$ and $p \xrightarrow{v}_{\widehat{\Delta}} q$.*

We call A $\widehat{\Sigma}'$-*closed* if $p \xrightarrow{u}_{\widehat{Q}} q$ for some $u \in (\widehat{\Sigma})^*$ iff $p \xrightarrow{u'}_{\widehat{Q}} q$ for some $u' \in (\widehat{\Sigma}')^*$. The $\widehat{\Sigma}'$-*diameter* of A, denoted by d, is the maximum over all pairs of states p, q of $\min\{|u| : u \in (\widehat{\Sigma}')^*, p \xrightarrow{u}_{\widehat{\Delta}} q\}$, whenever this minimum is not over the empty set. Finally, for a fragment F and $S \subseteq \widehat{Q}$, let $\mathsf{reach}_{\widehat{\Sigma}'}(F, S)$ denote the set of states that can be reached from a state of S when following a run labelled by some word v that contains F as a fragment and such that all symbols in $v \setminus F$ belong to $\widehat{\Sigma}'$. By extending our property tester for regular languages, we obtain:

▶ **Lemma 21.** *Let A be $\widehat{\Sigma}'$-closed. Given a query access to a word u, Algorithm 2 computes a $(\xi, \widehat{\Sigma}')$-approximation of $R_u = \{(p,q) \mid p,q \in \widehat{Q}, p \xrightarrow{u}_{\widehat{Q}} q\}$ correctly with probability $\geq 1 - \mu$.*

Proof. For every $p, q \in \widehat{Q}$ and every fragment F of u, if $p \xrightarrow{u}_{\widehat{Q}} q$, then $q \in \mathsf{reach}_{\widehat{\Sigma}'}(F, \{p\})$. Therefore, the algorithm can err only if there exist $p, q \in \widehat{Q}$ such that $(p,q) \in R$ and for every word w such that $p \xrightarrow{w}_{\widehat{Q}} q$ there is $\mathsf{wed}_{\widehat{\Sigma}'}(w,u) \geq \xi \cdot \mathsf{weight}(u)$. This is equivalent to saying that $\mathsf{wed}_{\widehat{\Sigma}'}(u, L(A_{p,q})) \geq \xi \cdot \mathsf{weight}(u)$, where $A_{p,q}$ is the finite automaton A with a unique initial state p and a unique final state q. As A is $\widehat{\Sigma}'$-closed, an argument analogous to Theorem 5 shows that the algorithm errs for p, q with probability at most μ/m^2. The claim follows by the union bound. ◀

By plugging this result into the framework of François et al. [5], we obtain:

▶ **Corollary 22.** *Let $\varepsilon > 0$ be a constant, $\mathcal{A} = (Q, \Sigma, \Gamma, Q_{in}, Q_f, \Delta)$ be a VPA of constant size over $\Sigma = \Sigma_+ \sqcup \Sigma_= \sqcup \Sigma_-$, and $L = L(\mathcal{A})$. There is a randomised streaming property tester for L that uses $\mathcal{O}(\varepsilon^{-3} \log^5 n \log\log n)$ space.*

Proof. François et al. showed that property testing of visibly pushdown languages can be solved by running $\mathcal{O}(\varepsilon^{-1} \log^2 n)$ instances of the approximation algorithm of Lemma 21 with $\xi = \varepsilon/(6 \log n)$ and $\mu = 2/3n$, on an NFA of constant size [5, Theorem 5.4]. Furthermore, sampling factors of the input strings according to the ℓ-factor distribution can be imitated in streaming at an expense of $\mathcal{O}(\ell \cdot (\ell + \log n))$ space per sample [5, Fact 4.6]. We implement $\mathsf{reach}_{\widehat{\Sigma}'}$ similar to Lemma 13 using constant extra space. Therefore, an instance of the approximation algorithm of Lemma 21 requires space

$$\mathcal{O}(\xi^{-1} \log(1/\mu) + \sum_t r_t \ell_t \cdot (\ell_t + \log n)) =$$
$$= \mathcal{O}(\xi^{-2} \log^2(1/\mu) + \xi^{-1} \ln(1/\mu) \log(\xi^{-1} \ln(1/\mu)) \log n) = \mathcal{O}(\varepsilon^{-2} \log^3 n \log\log n).$$

The bound follows. ◀

[3] In this section, we refer to a more complete arXiv version of the paper.

Algorithm 2 Building an $(\xi, \widehat{\Sigma}')$-approximation R of R_u. Here k is the number of strongly connected components of A, and $m = |\widehat{Q}|$.

```
$\beta \leftarrow \xi/(6m)$, $\gamma \leftarrow 2/\beta$
if $\mathsf{weight}(u) \le 6kmd/\xi$ then
    Read the whole input $u$
    $R = \{(p,q) \mid p \in \widehat{Q}, q \in \mathsf{reach}_{\widehat{\Sigma}'}(u, \{p\})\}$
else
    Query $\tau = \lceil 2\ln(3k2^k m^2/\mu)/\beta \rceil$ 1-factors of $u$
    for $t = 0$ to $T = \lceil \log(2\gamma) \rceil$ do
        $\ell_t \leftarrow 2^t, r_t \leftarrow \lceil 2\ln(3k2^k m^2/\mu)\gamma/\ell_t \rceil$
        Query $u[1, 2\ell_t]$
        Query $r_t$ $2\ell_t$-factors of $u$ according to the $2\ell_t$-factor distribution
    $R = \{(p,q) \mid p \in \widehat{Q}, q \in \mathsf{reach}_{\widehat{\Sigma}'}(F, \{p\}\}$
return $R$
```

4.2 Lower bound

▶ **Theorem 23.** *There exist a VPL L and a constant ε_0 such that for any $1/n \le \varepsilon < \varepsilon_0$ any deterministic streaming ε-property tester for L under the edit distance uses space $\bar{\Omega}(n(1 - 16\varepsilon\log(1/\varepsilon)))$. Any randomised streaming ε-property tester for VPLs under the edit distance uses space $\bar{\Omega}(\max(\varepsilon^{-1}, \log n))$.*

Proof of Theorem 23. We first prove the deterministic bound, and then use it to derive the randomised one. Let L_{mirror} be a VPL over $\Sigma = \{0, 1, \bar{0}, \bar{1}\}$ defined as $L_{mirror} = \{w\bar{w}, w \in \{0,1\}\}$, where $\bar{w} = \overline{w[n]} \dots \overline{w[1]}$. For example, if $w = 1101$ then $\bar{w} = \bar{1}\bar{0}\bar{1}\bar{1}$. Recall that the indel distance $\delta(u, v)$ between two words u, v is equal to the minimum number of insertions and deletions needed to transform u into v and that a lower bound for the indel distance gives an asymptotically equal lower bound for the edit distance.

▶ **Lemma 24.** *Assume n is even. Let $u, v \in \Sigma^{n/2}$. If $\delta(u,v) > 2\varepsilon n$, then $\delta(u\bar{v}, L_{mirror}) > \varepsilon n$.*

Proof. We prove the claim by contrapositive. We assume that $\delta(u\bar{v}, L_{mirror}) \le \varepsilon n$ and show that $\delta(u,v) \le 2\varepsilon n$. Let $w\bar{w}$ be a word of L_{mirror} such that $\delta(u\bar{v}, L_{mirror}) = \delta(u\bar{v}, w\bar{w})$. By the triangle inequality, we have $\delta(u,v) \le \delta(u,w) + \delta(w,v) = \delta(u,w) + \delta(\bar{w}, \bar{v})$.

Let us start with an auxiliary claim. Consider $y \in \Sigma^k, z \in \Sigma^l$. Let y_1 (resp. y_2) denote $y[1, \lceil k/2 \rceil]$ (resp. $y[\lceil k/2 \rceil + 1, k]$), and similarly for z. We show that if $\delta(y,z) = 2$, then $\delta(y_1, z_1) + \delta(y_2, z_2) \le 4$.

We have either $k = l$ or $|k - l| = 2$. If $k = l$, then $|y_1| = |z_1|$, $|y_2| = |z_2|$, and the two edits are one insertion and one deletion. If the two edits occur in y_1, we have $\delta(y_1, z_1) \le 2$ and $\delta(y_2, z_2) = 0$. The case when the two edits occur in y_2 is symmetric. If one edit occurs in y_1 and the other in y_2, then y_1 is transformed into a prefix of z of length $|y_1| - 1$ or $|y_1| + 1$, and y_2 into a suffix of z of length $|y_2| + 1$ or $|y_2| + 1$, respectively. Therefore, $\delta(y_1, z_1) \le 2$ and $\delta(y_2, z_2) \le 2$. Assume now $|k - l| = 2$. W.l.o.g., $k = l + 2$, and the two edit operations are deletions. Since $k = l + 2$, we have $|y_1| = |z_1| + 1$, $|y_2| = |z_2| + 1$. Consider two cases:

- One deletion occurs in y_1, and one deletion occurs in y_2. In this case, y_1 is transformed into z_1, and y_2 into z_2. Hence, $\delta(y_1, z_1) + \delta(y_2, z_2) = 2$.
- The two deletions occur in y_1 (the proof for y_2 is symmetrical). In this case, y_1 is transformed into a prefix of v of length $|y_1| - 2 = |z_1| - 1$, and y_2 is equal to the suffix of v of length $|y_2|$. Therefore, $\delta(y_1, z_1) \le 3$ and $\delta(y_2, z_2) = 1$, and the claim follows.

For simplicity, let $T = \delta(u\bar{v}, w\bar{w})$. Note that T is even, as both words have even length. For every $0 \leq t \leq T$, let x_t be the word obtained by applying the first t edit operations to $u\bar{v}$ in the sequence that transforms $u\bar{v}$ into $w\bar{w}$, and let $x_1^t = x^t[1, \lceil |x^t|/2 \rceil]$ and $x_2^t = x_t[\lceil |x^t|/2 \rceil + 1, |x^t|]$. For all $t \leq T-2$, $\delta(x^t, x^{t+2}) = 2$, and therefore $\delta(x_1^t, x_1^{t+2}) + \delta(x_2^t, x_2^{t+2}) \leq 4$. Finally, we obtain

$$\begin{aligned}\delta(u,w) + \delta(\bar{v}, \bar{w}) = \delta(x_1^0, x_1^T) + \delta(x_2^0, x_2^T) &\leq \sum_{t=0}^{T/2} \delta(x_1^{2t}, x_1^{2t+2}) + \sum_{t=0}^{T/2} \delta(x_2^{2t}, x_2^{2t+2}) \\ &\leq \sum_{t=0}^{T/2} \left(\delta(x_2^{2t}, x_2^{2t+2}) + \delta(x_2^{2t}, x_2^{2t+2}) \right) \leq 4T/2 \leq 2\varepsilon n\end{aligned}$$

In short, we have $\delta(u,w) + \delta(\bar{v}, \bar{w}) \leq 2\varepsilon n$, which completes the proof. ◀

▷ Claim 25. Let $u \in \{0,1\}^n$ and $B(u,k,n) = \{v \in \{0,1\}^n \mid \delta(u,v) \leq k\}$. For all $2/n \leq \alpha < 1$, we have $|B(u, \alpha n, n)| \leq 2^{\alpha n(1 + 2\log 8e/\alpha)/2}$.

Proof. Let $2t$ be the largest even number smaller or equal to αn. Let $v \in B(u,k,n)$. Since $|u| = |v| = n$, we can obtain v from u using t insertions and t deletions. (If $\delta(u,v) < 2t$, we can insert a symbol $(2t - \delta(u,v))/2$ times, and then delete it $(2t - \delta(u,v))/2$ times.)

W.l.o.g., assume that the insertions occur before the deletions. After t insertions, we obtain a word of size $n+t$. The number of possible ways to delete t symbols in this word is therefore $\binom{n+t}{t}$. We give an upper bound on the number of words that can be reached with t insertions from a word of length n the following way: in a word of length $n+t$, there are t symbols that have been inserted, and there are two options for each symbol. Therefore, there are at most $2^t\binom{n+t}{t}$ such words. The number of words in $B(u,k,n)$ is at most the number of words reached after a sequence of t insertions, multiplied by the number of sequences of t deletions. Overall, we obtain:

$$\begin{aligned}|B(u, \alpha n, n)| &\leq 2^t \binom{n+t}{t}^2 \leq 2^t \left(\frac{e(n+t)}{t} \right)^{2t} \leq 2^{(1+2\log e)t} (\frac{n}{t} + 1)^{2t} \\ &\leq 2^{(1+2\log e)t} (8/\alpha)^{2t} \leq 2^{\alpha n(1 + 2\log e + 2\log(8/\alpha))/2} \\ &\leq 2^{\alpha n(1 + 2\log(8e/\alpha))/2}\end{aligned}$$

◁

▶ **Corollary 26.** *There exists a constant ε_0 such that for any $1/n \leq \varepsilon < \varepsilon_0$, any deterministic streaming ε-property tester for L_{mirror} under the indel distance uses space $\tilde{\Omega}(n(1 - 16\varepsilon \log(1/\varepsilon)))$.*

Proof. Assume n is even. Consider the memory state of a tester after reading two words $u, v \in \{0,1\}^{n/2}$. Suppose that they give the same memory state. We can then continue the streams with $\bar{u}$, and since the algorithm must accept $u\bar{u} \in L_{mirror}$, it must also accept $v\bar{u}$. By the definition of a streaming ε-property tester and Lemma 24, we have $\delta(u,v) \leq 2\varepsilon n$. Therefore, the number of distinct memory states of the tester after reading $n/2$ symbols is at least $2^{n/2}/|B(u, 2\varepsilon n, n/2)|$. The space $s(n)$ used by the tester is at least the logarithm of the number of memory states. Therefore,

$$\begin{aligned}s(n) &\geq \log(2^{n/2}/|B(u, 2\varepsilon n, n/2)|) \\ &\geq \log(2^{n/2}/2^{\varepsilon n(1+2\log(2e/\varepsilon))}) \text{ (applying Claim 25 for } \alpha = 4\varepsilon \text{ and size } n/2) \\ &\geq n/2 - \varepsilon n(1 + 2\log(2e/\varepsilon)) \geq \frac{n}{2}(1 - 16\varepsilon\log(1/\varepsilon))\end{aligned}$$

◀

The deterministic bound for the edit distance follows immediately. We now derive the randomised lower bound. We show $\bar{\Omega}(\log n)$ and $\bar{\Omega}(1/\varepsilon)$ space lower bounds separately, and then combine them to yield the theorem.

▶ **Corollary 27.** *Any randomised streaming ε-property tester for L_{mirror} under the edit distance uses space $\bar{\Omega}(\log n)$.*

Proof. A streaming ε-property tester for a VPL L with an input of length n can be viewed as an automaton A_n whose states are the memory states of the algorithm. Deterministic algorithms are deterministic automata, and randomised algorithms are probabilistic automata (see Rabin [22] for definitions).

Consider a randomised streaming ε-property testing algorithm A_n for L_{mirror} on inputs of length n. Let $L(A_n) = \{u \in \{0, 1, \bar{0}, \bar{1}\}^n \mid A_n$ accepts u with probability $\geq 2/3\}$. By definition, a deterministic streaming algorithm that recognizes $L(A_n)$ is a streaming ε-property tester for L_{mirror}. By [7, Lemma 6.4], the space complexity of A_n is at least the logarithm of the space complexity of the deterministic one, that is $\bar{\Omega}(\log(n(1 - 16\varepsilon \log(1/\varepsilon)))) = \bar{\Omega}(\log n)$. ◀

We now show the $\bar{\Omega}(1/\varepsilon)$ bound. Consider a VPL $L_{disj} = \{x\bar{y} \mid x, y \in \{0, 1\}^n$ and $\forall 1 \leq i \leq n, x[i] \cdot y[i] = 0\}$.

▷ Claim 28. Let $\alpha = \lfloor 1/\varepsilon \rfloor$ and assume that n is a multiple of α. Define a morphism $\phi : \Sigma^* \to \Sigma^*$ such that for any $a \in \Sigma$ we have $\phi(a) = a^{6n/\alpha}$. Consider a word $u = x\bar{y}$, where $x, y \in \Sigma^\alpha$. If $u \in L_{disj}$, then $\mathsf{ed}(\phi(u), L_{disj}) = 0$, and otherwise $\mathsf{ed}(\phi(u), L_{disj}) > \varepsilon n$.

Proof. The first part of the claim is obvious. The rest of the proof is devoted to the case $u \notin L_{disj}$. Assume by contradiction that $\mathsf{ed}(\phi(u), L_{disj}) \leq \varepsilon n$, in other words, that there exists a sequence of at most εn edits such that when applied to $\phi(u)$, we obtain a word in L_{disj}. W.l.o.g. assume that the edits are applied only to the first half of $\phi(u)$, i.e. to $\phi(x)$. Since $u \notin L_{disj}$, there exists i such that $x[i] \cdot y[i] = 1$. Consider the middle part of $\phi(x[i])$, i.e. the factor $w = \phi(u)[(i-1) \cdot (6n/\alpha) + 2n/\alpha + 1, (i-1) \cdot (6n/\alpha) + 4n/\alpha]$. After we apply the at most $\varepsilon n \leq n/\alpha$ edits to $\phi(x)$, there is at least one symbol of w that does not change and therefore is equal to 1, let it be $w[j]$. Moreover, the index of this symbol in the resulting word is between $(i-1) \cdot (6n/\alpha) + n/\alpha + 1$ and $(i-1) \cdot (6n/\alpha) + 5n/\alpha$. On the other hand, $\phi(y[i])$ can be shifted by at most εn positions to the left or to the right. Therefore, $w[j]$ will be aligned with a symbol in $\phi(y[i])$ equal to 1 as well, a contradiction. ◁

The $\bar{\Omega}(1/\varepsilon)$ bound follows immediately from the linear space lower bound for randomised streaming 0-property testing algorithms for L_{disj} [5, 15]. ◀

References

1 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, 2001. doi:10.1137/S0097539700366528.

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

3 Eldar Fischer, Frédéric Magniez, and Michael de Rougemont. Approximate satisfiability and equivalence. *SIAM Journal on Computing*, 39(6):2251–2281, 2010.

4 Eldar Fischer, Frédéric Magniez, and Tatiana Starikovskaya. Improved bounds for testing Dyck languages. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1529–1544. SIAM, 2018. doi:10.1137/1.9781611975031.100.

5 Nathanaël François, Frédéric Magniez, Michel De Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. *CoRR*, abs/1505.03334, 2015. arXiv:1505.03334.

6 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proceedings of the 24th Annual European Symposium on Algorithms*, volume 57 of *LIPIcs*, pages 43:1–43:17, 2016. doi:10.4230/LIPIcs.ESA.2016.43.

7 Moses Ganardi. *Language recognition in the sliding window model.* PhD thesis, University of Siegen, Germany, 2019.

8 Moses Ganardi. Visibly pushdown languages over sliding windows. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science*, volume 126 of *LIPIcs*, pages 29:1–29:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.29.

9 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *Proceedings of 35th International Symposium on Theoretical Aspects of Computer Science*, volume 96 of *LIPIcs*, pages 31:1–31:14, 2018. doi:10.4230/LIPIcs.STACS.2018.31.

10 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 65 of *LIPIcs*, pages 18:1–18:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.18.

11 Moses Ganardi, Danny Hucke, and Markus Lohrey. Randomized sliding window algorithms for regular languages. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs*, pages 127:1–127:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.127.

12 Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. In *Proceedings of the 30th International Symposium on Algorithms and Computation*, volume 149 of *LIPIcs*, pages 6:1–6:13, 2019. doi:10.4230/LIPIcs.ISAAC.2019.6.

13 Moses Ganardi, Artur Jeż, and Markus Lohrey. Sliding windows over context-free languages. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science*, volume 117 of *LIPIcs*, pages 15:1–15:15, 2018. doi:10.4230/LIPIcs.MFCS.2018.15.

14 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.

15 Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

16 Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming algorithms for recognizing nearly well-parenthesized expressions. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science*, volume 6907 of *LNCS*, pages 412–423, 2011.

17 Frédéric Magniez and Michel de Rougemont. Property testing of regular tree languages. *Algorithmica*, 49(2):127–146, 2007. doi:10.1007/s00453-007-9028-3.

18 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM J. Comput.*, 43(6):1880–1905, 2014. doi:10.1137/130926122.

19 Antoine Ndione, Aurélien Lemay, and Joachim Niehren. Approximate membership for regular languages modulo the edit distance. *Theor. Comput. Sci.*, 487:37–49, 2013. doi:10.1016/j.tcs.2013.03.004.

20 Antoine Ndione, Aurélien Lemay, and Joachim Niehren. Sublinear DTD validity. In *International Conference on Language and Automata Theory and Applications*, volume 8977 of *LNCS*, pages 739–751, 2015. doi:10.1007/978-3-319-15579-1_58.

21 Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Testing membership in parenthesis languages. *Random Struct. Algorithms*, 22(1):98–138, 2003. doi:10.1002/rsa.10067.

22 Michael O. Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.

23 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.