Streaming Regular Expression Membership and Pattern Matching

Bartlomiej Dudek, Pawel Gawrychowski, Garance Gourdel, Tatiana Starikovskaya

Streaming Regular Expression Membership and Pattern Matching

Bartłomiej Dudek* Paweł Gawrychowski† Garance Gourdel‡ Tatiana Starikovskaya§

Abstract

Regular expression search is a key primitive in myriads of applications, from web scrapping to bioinformatics. A regular expression is a formalism for compactly describing a set of strings, built recursively from single characters using three operators: concatenation, union, and Kleene star. Two basic algorithmic problems concerning such expressions are membership and pattern matching. In the regular expression membership problem, we are given a regular expression R and a string T of length n, and must decide whether T matches R. In the regular expression pattern matching problem, the task to find the substrings of T that match R.

By now we have a good understanding of the complexity of regular expression membership and pattern matching in the classical setting. However, only some special cases have been considered in the practically relevant streaming setting: dictionary matching and wildcard pattern matching. In the dictionary matching problem, we are given a dictionary of d strings of length at most m and a string T, and must find substrings of T that match one of the dictionary strings. In the wildcard pattern matching problem, we are given a string P of length m that contains d wildcards, where a wildcard is a special symbol that matches any character of the alphabet, and a string T, and must find all substrings of T that match P. Both problems can be solved in the streaming model by a randomised Monte Carlo algorithm that uses $\mathcal{O}(d \log m)$ space [Golan and Porat (ESA 2017), Golan, Kopelowitz and Porat (Algorithmica 2019)].

In the general case, we cannot hope for a streaming algorithm with space complexity smaller than the length of R for either variant of regular expression search. The main contribution of this paper is that we identify the number of unions and Kleene stars, denoted by d, as the parameter that allows for an efficient streaming algorithm. This parameter has been previously considered in the classical setting, and it has been observed that in practice it is significantly smaller than the length of R. We design general randomised Monte Carlo algorithms for both problems that use $\mathcal{O}(d^3 \operatorname{polylog} n)$ space in the streaming setting.

A crucial technical ingredient of our algorithms is an adaptation of the general framework for evaluating a circuit with addition and convolution gates in a space-efficient manner [Lokshtanov and Nederlof (STOC 2010), Bringmann (SODA 2017)], initially designed as a key component of a pseudopolynomial time algorithm for the subset sum problem. We show how to replace the Extended Generalised Riemann Hypothesis in [Bringmann (SODA 2017)] by an application of the Bombieri–Vinogradov theorem to achieve the same bounds (but unconditionally), which might be of independent interest.

1 Introduction

The fundamental notion of regular expressions was introduced back in the 1951 by Kleene [45]. Regular expression search is one of the key primitives in diverse areas of large scale data analysis: computer networks [47], databases and data mining [32,49,54], human-computer interaction [44], internet traffic analysis [42,64], protein search [56], and many others. As such, this primitive is often the main computational bottleneck in these areas and in the pursuit for efficiency has been implemented in many programming languages: Perl, Python, JavaScript, Ruby, AWK, Tcl and Google RE2, to name just a few.

A regular expression R is a sequence containing characters of a specified alphabet Σ and three special symbols (operators): concatenation ($\cdot$), union ($|$), and Kleene star ($*$), and it describes a set of strings $L(R)$ on Σ. For example, a regular expression $R = (a|b)^*c$ specifies a set of strings $L(R)$ on the alphabet $\Sigma = \{a, b, c\}$ such that their last character equals c, and all other characters are equal to a or b. (See formal definition in Section 2). In this work, we consider two classical formalisations of regular expressions search, regular expression membership and pattern matching. In the regular expression membership problem, we are given a string T of length n, and

*Institute of Computer Science, University of Wrocław, Poland. Partially supported by the National Science Centre, Poland, under grant number 2017/27/N/ST6/02719 and by the Foundation for Polish Science (FNP).

†Institute of Computer Science, University of Wrocław, Poland. Partially supported by the Bekker programme of the Polish National Agency for Academic Exchange (PPN/BEK/2020/1/00444) and the grant ANR-20-CE48-0001 from the French National Research Agency (ANR).

‡DI/ENS, PSL Research University, IRISA Inria Rennes, France. Partially supported by the grant ANR-20-CE48-0001 from the French National Research Agency (ANR).

§DI/ENS, PSL Research University, France. Partially supported by the grant ANR-20-CE48-0001 from the French National Research Agency (ANR).

must decide whether $T \in L(R)$ for a given regular expression R. In the regular expression pattern matching problem, we must find all positions $1 \le r \le n$ such that for some $1 \le \ell \le r$, the substring $T[\ell \,..\, r] \in L(R)$.

Assume that T is read-only, and let m be the length of the regular expression. The classical algorithm by Thompson [62] allows to solve both problems in $\mathcal{O}(nm)$ time and $\mathcal{O}(m)$ space by constructing a non-deterministic finite automaton accepting $L(R)$. Galil [25] noted that while the space bound of Thompson's algorithm is optimal in the deterministic setting, the time bound could probably be improved. Since then, the effort has been mainly focused on improving the time complexity of regular expression search. The first breakthrough was achieved by Myers [55], who showed that both problems can be solved in $\mathcal{O}(mn/\log n + (n+m)\log n)$ time and $\mathcal{O}(mn/\log n)$ space. Bille and Farach-Colton [7] reduced the space complexity down to $\mathcal{O}(n^\varepsilon + m)$, for an arbitrary constant $\varepsilon > 0$. This result was further improved by Bille and Thorup [8] who showed an algorithm with running time $\mathcal{O}(nm(\log\log n)/\log^{3/2} n + n + m)$ time that uses $\mathcal{O}(n^\varepsilon + m)$ space. The idea of the algorithms by Myers [55], Bille and Farach-Colton [7], and Bille and Thorup [8] is to decompose Thompson's automaton into small non-deterministic finite automata and tabulate information to speed up simulating the behaviour of the original automaton when reading T. A slightly different approach was taken by Bille [6] who showed that the small non-deterministic finite automata can be simulated directly using the parallelism built-in in the Word RAM model. For w being the size of the machine word, Bille showed $\mathcal{O}(m)$-space algorithms with running times $\mathcal{O}(n\frac{m\log w}{w} + m\log w)$ for $m > w$, $\mathcal{O}(n\log m + m\log m)$ for $\sqrt{w} < m \le w$, and $\mathcal{O}(\min\{n + m^2, n\log m + m\log m\})$ for $m \le \sqrt{w}$. Finally, Bille and Thorup [9] identified a new parameter affecting the complexity of regular expression search, which is particularly relevant to this paper. Namely, they noticed that in practice a regular expression contains $d \ll m$ occurrences of the union symbol and Kleene stars, and showed that regular expression membership and pattern matching can be solved in $\mathcal{O}(m)$ space and $\mathcal{O}(n \cdot (\frac{d\log w}{w} + \log d))$ time[1].

It is easy to see, however, that in the general case the time complexity of all the algorithms above remains close to "rectangular", with some polylogarithmic factors shaved. Recently, fine-grained complexity provided an explanation for this. Backurs and Indyk [4] followed by Bringmann, Grønlund, and Larsen [13] considered a subclass of regular expressions which they refer to as "homogeneous". Intuitively, a regular expression is homogeneous, if the operators at the same level of the expression are equal. Assume that the alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$. To give a few examples, the following regular expressions are homogeneous: $R_1 = (P_1|P_2|\ldots|P_d)$, $R_2 = P_1(1|2|\ldots|\sigma)P_2(1|2|\ldots|\sigma)\ldots(1|2|\ldots|\sigma)P_d$, and $R_3 = (P_1|P_2|\ldots|P_d)^*$, where P_i, $1 \le i \le d$, are strings on Σ, i.e. concatenations of characters in Σ. [4,13] considered both the membership and the pattern matching problems. A careful reader might notice that in the pattern matching setting the expression R_1 corresponds to the famous dictionary matching problem [2] and R_2 to pattern matching with wildcards (don't cares) [14,18,23,41,43]. In the membership setting, R_3 corresponds to the Word Break problem [48,63]. As such, a seemingly simple class of homogeneous regular expressions covers many classical problems in stringology. The authors of [4,13] provided a complete dichotomy of the time complexities for homogeneous regular expressions in both settings. Namely, they showed that in both settings, every regular expression either allows a solution in near-linear time, or requires $\Omega((nm)^{1-\alpha})$ time, conditioned on the Strong Exponential Time Hypothesis [40]. The only exception is the Word Break problem in the membership setting, for which [13] showed an $\mathcal{O}(n(m\log m)^{1/3} + m)$-time algorithm and a matching combinatorial lower bound (up to polylogarithmic factors). Later, Abboud and Bringmann [1] took an even more fine-grained approach and showed that in general, regular expression pattern matching and membership cannot be solved in time $\mathcal{O}(nm/\log^{7+\alpha} n)$ for any constant $\alpha > 0$ under the Formula-SAT Hypothesis. Schepper [60] extended their result by revisiting the dichotomy for homogeneous regular expressions, and showed an $\mathcal{O}(nm/2^{\Omega(\sqrt{\log\min\{n,m\}})})$ time bound for some regular expressions, and for the remaining ones an improved lower bound of $\Omega(nm/\operatorname{polylog} n)$.

By now we seem to have a rather good understanding of the time complexity of regular expression membership and pattern matching. However, in multiple practical applications one needs to work with the input arriving as a stream, one character at a time, without the possibility of going back and retrieving any of the previous characters on demand. This motivates studying both problems in the streaming model of computation. In this model, we mostly focus on designing algorithms with small space complexity, and need to account for storing any information about the input. On the other hand, we allow for randomised algorithms, more specifically Monte Carlo algorithm returning correct answers with high probability (with respect to the length to the input). The

[1] Formally, they consider a parameter k equal to the number of strings in R, but it is not hard to see that $k = \Theta(d)$.

field of streaming algorithms for string processing is relatively recent but, because of its practical interest, quickly developing. It started with a seminal paper of Porat and Porat [57], who showed streaming algorithms for exact pattern matching and for the k-mismatches problem. This was followed by a series of works on streaming pattern matching [11, 15–17, 35–37, 39, 46, 58, 61], search of repetitions in streams [19–21, 33, 34, 52, 53], and recognising formal languages in streams [3, 5, 24, 26–31, 51].

For a general regular expression membership and pattern matching, it is not hard to see that $\Omega(m)$ bits of space are required by a reduction from Set Intersection. However, there are at least two interesting special cases of regular expression pattern matching that admit better streaming algorithms. In the dictionary matching, we are given a dictionary of d strings of length at most m over an alphabet Σ and for each position r in T must decide whether there is a position $\ell \leq r$ such that $T[\ell\,..\,r]$ matches a dictionary string. A series of work [11,15,37,39,57] showed that this problem can be solved by a randomised Monte Carlo algorithm in $\mathcal{O}(d \log m)$ space and $\mathcal{O}(\log\log|\Sigma|)$ time per character of the text. In the $(d-1)$-wildcard pattern matching the expression is $R = P_1(1|2|\ldots|\sigma)P_2(1|2|\ldots|\sigma)\ldots(1|2|\ldots|\sigma)P_d$, where P_i, $1 \leq i \leq d$ are strings of total length at most m over an alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$. Golan, Kopelowitz, and Porat [38] showed that this problem can be solved by a randomised Monte Carlo algorithm in $\mathcal{O}(d \log m)$ space and $\mathcal{O}(d + \log m)$ time per character. The d-wildcard problem is a special case of the k-mismatch problem which asks to compute Hamming distances between a pattern and all its alignments to a text for which the Hamming distance does not exceed the given threshold k. The most space efficient algorithm for the d-mismatch problem is by Clifford, Kociumaka and Porat [17] and implies an algorithm that uses $\mathcal{O}(d \log \frac{m}{d})$ words of space and spends $\mathcal{O}(\log \frac{m}{d}(\sqrt{d \log d} + \log^3 m))$ time per character which is also the most efficient for the d-wildcard problem.

In a related work, Ganardi et al. [26–29] considered a variant of the regular expression membership problem, where the automaton describing the regular expression has constant size, and one must tell, for each position r of T, whether $T[r-\ell+1\,..\,r] \in L(R)$, where ℓ is an integer specified in advance ("window" size). As a culmination of their work, they showed that any randomised Monte Carlo algorithm for this variant of the regular expression membership problem takes either constant, or $\Theta(\log\log \ell)$, or $\Theta(\log \ell)$, or $\Theta(\ell)$ bits of space, and provided descriptions of these complexity classes.

This brings the challenge of identifying a structural parameter of a regular expression that determines whether it admits better streaming algorithms. As mentioned earlier, Bille and Thorup [9] observed that in practice the number d of occurrences of the union symbol and Kleene stars is significantly smaller than the size m of the expression R. Furthermore, both the dictionary matching and the wildcard pattern matching can be casted as instances of the regular expression pattern matching, and streaming algorithms with space complexity of the form $\mathrm{poly}(d, \log n)$ are known. The main goal of this paper is to investigate whether this is also the case for the general regular expression membership and pattern matching.

1.1 Our results We consider the space complexity of regular expression membership and pattern matching in the streaming model of computation. As by now traditional in streaming string processing, we assume that we receive R and n first, preprocess them, and then receive the string T character by character. We do not account neither for the time nor for the space used during the preprocessing stage. In the membership problem, we must output the answer after having read T entirely, whereas in the pattern matching problem we must decide whether there is a substring $T[\ell\,..\,r] \in L(R)$ at the moment when we receive the character $T[r]$.

Our main conceptual contribution is that we identify the small number d of occurrences of the union symbol and Kleene stars in R as allowing for space-efficient streaming algorithms for regular expression membership and pattern matching. More specifically, we design randomised Monte Carlo algorithms that solve both problems using $\mathcal{O}(d^3 \operatorname{polylog} n)$ space and $\mathcal{O}(nd^5 \operatorname{polylog} n)$ time per character of the text (Theorem 4.14). While it was known that the value of d determines the space complexity in the two special cases of streaming dictionary matching and wildcard pattern matching, our approach works for any regular expression. We leave it as an open problem to obtain algorithms with $\mathrm{poly}(d, \log n)$ space complexity and $\mathrm{poly}(d, \log n)$ time complexity.

On a very high-level, our approach is based on storing carefully chosen subsets of occurrences of the strings appearing in R. As usual in the area, this is easier when the strings are not periodic, that is, any two occurrences of a string S in T must be more than $|S|/2$ characters apart. Of course, this is not always the case, and the usual remedy is to treat periodic and aperiodic strings separately (more specifically, in streaming pattern matching algorithms one applies this reasoning on every prefix of length being a power of 2). The technical novelty of our algorithms is that we apply this reasoning on $\mathcal{O}(\log n)$ levels, thus obtaining a hierarchical decomposition of a

periodic string. Next, because not all occurrences are stored we need to recover the omitted information. Very informally, we need to decide whether a substring of T sandwiched between two occurrences of strings A_1, A_2 is a label of some run from A_1 to A_2 in the compact Thompson automaton for R, where the period of the substring is equal to the period of some prefix of length 2^k of one of the strings. The difficulty is that, while the substring has a simple structure, it could be very long, and it is not clear how to implement this computation in a space-efficient manner. We overcome this difficulty by recasting the problem in the language of evaluating a circuit with addition and convolution gates. This technique was introduced by Lokshtanov and Nederlof [50] for designing a space-efficient solution for the subset sum problem. Later, Bringmann [12] replaced complex numbers with computation modulo a prime number p to obtain a tighter bound on the time and space complexity. In more detail, he designed two solutions, one using the Extended Riemann Hypothesis and the other unconditional but with polynomially higher time and space. We revisit his approach and show that, in fact, one can replace the Extended Riemann Hypothesis by an application of the Bombieri–Vinogradov theorem to achieve the same bounds. We believe that this might be of independent interest. As a consequence of our improvement, we obtain an efficient randomised Monte Carlo algorithm for the following classical problem: given a directed multigraph G with non-negative integer weights on edges, its two nodes v_1, v_2, and a number x, decide whether there is a walk from v_1 to v_2 of total weight x. Our algorithm requires $x \cdot \mathrm{poly}(|G|, \log x)$ time and $\mathrm{poly}(|G|, \log x)$ space.

The rest of the paper is organised as follows. We first remind the necessary definitions in Section 2, and in Section 3 we give an overview of the main technical ideas we introduced in this paper. We describe the new algorithms for regular expression membership and pattern matching in Section 4.2. Finally, in Section 5 we describe how to replace the Extended Riemann Hypothesis with an application of the Bombieri–Vinogradov theorem in Bringmann's framework and design a space-efficient algorithm for checking if there is a walk of specified weight between two nodes of a directed multigraph.

2 Preliminaries

We assume an integer alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$ with σ *characters*. A *string* Y is a sequence of characters numbered from 1 to $n = |Y|$. For $1 \le i \le n$, we denote the i-th character of Y by $Y[i]$. For $1 \le i \le j \le n$, we define $Y[i\,..\,j]$ to be equal to $Y[i] \ldots Y[j]$, called a *fragment* of Y. We call a fragment $Y[1] \ldots Y[j]$ *a prefix* of Y and use a simplified notation $Y[..\,j]$, and a fragment $Y[i] \ldots Y[n]$ *a suffix* of Y denoted by $Y[i\,..]$. We say that a fragment $Y[i\,..\,j]$ contains a position k if $i \le k \le j$. We denote by ε the empty string.

We say that X is a *substring* of Y if $X = Y[i\,..\,j]$ for some $1 \le i \le j \le n$. The fragment $Y[i\,..\,j]$ is called an *occurrence* of X. We say that an integer p is a period of Y if for each $1 \le i \le |Y| - p$, $Y[i] = Y[i + p]$. The smallest period of Y is referred to as *the period* of Y. We say that Y is *periodic* with period ρ if ρ is the period of Y and $\rho \le |Y|/2$. For the period ρ of Y, we define the string period of Y to be equal to $Y[1\,..\,\rho]$.

For an integer k, we denote the concatenation of k copies of Y by Y^k. We say that a string X is *primitive* if $X \neq Y^k$ for any string $Y \neq X$ and any integer k. Note that the string period of a string is always primitive.

Definition 2.1. (Regular expression) *We define regular expressions over Σ as well as the languages they match recursively. Let $L(R)$ be the language matched by a regular expression R.*

- *Any $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression and $L(a) = \{a\}$.*

For two regular expressions A and B, we can form a new expression using one of the three symbols $\cdot$ (concatenation), $|$ (union), or * (Kleene star):

- *$A \cdot B$ is a regular expression and $L(A \cdot B) = \{XY,\ \text{for } X \in L(A) \text{and } Y \in L(B)\}$;*
- *$A \mid B$ is a regular expression and $L(A \mid B) = L(A) \cup L(B)$;*
- *A^* is a regular expression and $L(A^*) = \bigcup_{k \ge 0} \{X_1 X_2 \ldots X_k,\ \text{where } X_i \in L(A) \text{ for } 1 \le i \le k\}$.*

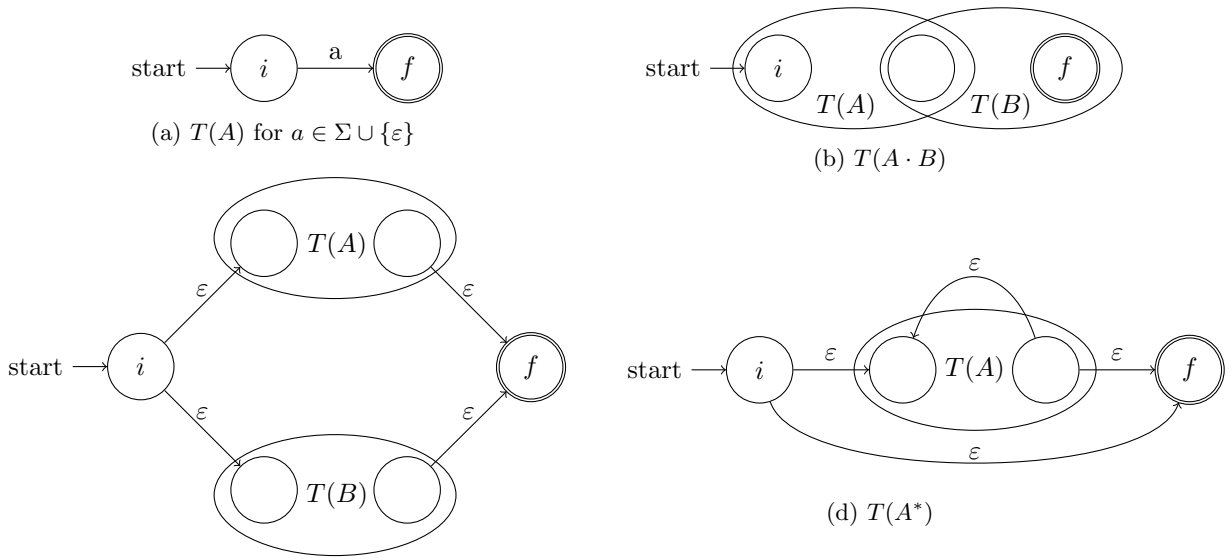
Definition 2.2. (Thompson automaton [62]) *For a regular expression R we define the Thompson automaton of R, $T(R)$, recursively. This non-deterministic finite automaton (NFA) accepts all strings $s \in L(R)$.*

- *If $R = a \in \Sigma \cup \{\varepsilon\}$, $T(R)$ is constructed as in Figure 1a;*
- *If $R = A \cdot B$, $T(R)$ is constructed as in Figure 1b. Namely, the initial state of $T(A)$ becomes the initial state of $T(R)$, the final state of $T(A)$ becomes the initial state of $T(B)$, and the final state of $T(B)$ becomes the final state of $T(R)$;*

- *If $R = A|B$, $T(R)$ is constructed as in Figure 1c. Namely, the initial state of $T(R)$ goes via ε-transitions both to the initial state of $T(A)$ and to the initial state of $T(B)$, and the final states of $T(A)$ and $T(B)$ go via ε-transitions to the final state of $T(R)$;*
- *If $R = A^*$, $T(R)$ is constructed as in Figure 1d. Namely, the initial state of $T(R)$ and the final state of $T(A)$ go via ε-transitions both to the initial state of $T(A)$, and to the final state of $T(R)$.*

DEFINITION 2.3. (COMPACT THOMPSON AUTOMATON) *Given a Thompson automaton $T(R)$, we define the compact Thompson automaton $T_C(R)$ as the automaton obtained from $T(R)$ by replacing every maximal path of transitions labelled by $a_1, a_2, \ldots, a_k \in \Sigma$ by a single transition labelled by $a_1 a_2 \ldots a_k$. The non-empty labels of $T_C(R)$ are called* atomic strings*, and the size of the (multiset) of the atomic strings is defined to be the* size *of R.*

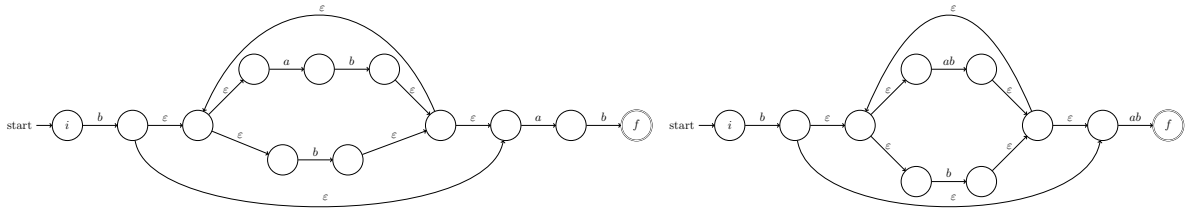
Figure 2 gives an example of the Thompson automata for $R = b(ab|b)^*ab$. We note that in general the size of a regular expression is much smaller than the total number of characters in it and is bounded by twice the number of union and Kleene star symbols plus two. The size of a regular expression measures its "complexity".



(a) $T(A)$ for $a \in \Sigma \cup \{\varepsilon\}$ (b) $T(A \cdot B)$

(c) $T(A|B)$ (d) $T(A^*)$

Figure 1: Thompson automaton. In each automaton, i and f are the initial and final states, respectively.



(a) $T(b(ab|b)^*ab)$ (b) $T_C(b(ab|b)^*ab)$

Figure 2: The Thompson automatons of the regular expression $b(ab|b)^*ab$.

DEFINITION 2.4. (OCCURRENCE OF A REGULAR EXPRESSION) *We say that a fragment $S[i\,..\,j]$ of a string S, where $1 \le i \le j \le |S|$, is an* occurrence *of a regular expression R, if $S[i\,..\,j] \in L(R)$, or in other words if there is a walk from the initial state of $T_C(R)$ to the final state of $T_C(R)$ such that the concatenation of the labels of the transitions in this walk equals $S[i\,..\,j]$.*

We will also need a notion of a partial occurrence of R. Intuitively, $S[i\,..\,j]$ is a partial occurrence of R if it is a prefix of a string in $L(R)$, but we will need a more precise definition.

DEFINITION 2.5. (PARTIAL OCCURRENCE OF A REGULAR EXPRESSION) *We say that a fragment $S[i\,..\,j]$ of a string S, where $1 \leq i \leq j \leq |S|$, is a* partial occurrence *of a regular expression R ending with a prefix P of an atomic string A, if there is a walk from the initial state of $T_C(R)$ to the endpoint of the transition corresponding to A such that the concatenation of the labels of the transitions in this walk equals $S[i\,..\,j]A[|P|+1\,..]$.*

3 Technical Overview

In this section, we give an overview of the main technical ideas we introduced in this paper.

Statement of the problems and the model of computation. Let us start by giving the precise formulation of the regular expression membership and pattern matching problems and reminding the definition of the streaming model of computation.

REGULAR EXPRESSION MEMBERSHIP AND PATTERN MATCHING
Given a string T of length n over an alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$, where $\sigma = n^{\mathcal{O}(1)}$, and a regular expression R over Σ of size d. In the regular expression membership problem, we must decide whether $T \in L(R)$. In the regular expression pattern matching problem, we must find all positions $1 \leq r \leq n$, such that there exists a position $1 \leq \ell \leq r$ such that $T[\ell\,..\,r] \in L(R)$.

We work in the streaming model of computation. As it is now standard in the streaming string processing algorithms, we assume to receive n and R first. We do not account neither for the time nor for the space we need to preprocess R. After having preprocessed R, we receive T as a stream, character by character. At the moment we receive the first character of T, the main phase of the algorithm starts. During the main phase, we account for *all* the space and time used.

Definitions and tools. Let $A_1, A_2, \ldots, A_d$ be the atomic strings of the regular expression R. We define $\Pi = \{A_i[1\,..\,\min\{2^j, |A_i|\}] : 1 \leq i \leq d, 0 \leq j \leq \lceil \log |A_i| \rceil\}$. The prefixes of A_i's that belong to Π are called *canonical*.

We can assume that all atomic strings have length at most n, otherwise they never appear in the text and we can ignore them. Formally, during the preprocessing phase we delete all transitions (u, v) from $T_C(R)$ that are labelled by atomic strings of lengths larger than n. We also assume that $d \leq n$, otherwise we can use the following solution:

CLAIM 3.1. *Given a streaming text T of length n and a regular expression of size $d \geq n$. Assume that all atomic strings have length at most n each. There is a deterministic algorithm that solves the membership and the pattern matching problems for T and R in $\mathcal{O}(d^2)$ space and $\mathcal{O}(d^3)$ time per character of T.*

Proof. First note that we can afford storing T in full. Second, we build a compact trie on the reverses of the atomic strings of R. The trie occupies $\mathcal{O}(dn) = \mathcal{O}(d^2)$ space. Finally, let $\mathcal{F}$ contain all atomic strings A such that there is an ε-transitions path from the endpoint of the transition labelled by A to the final state of $T_C(R)$.

Define an array D of length $n + 1 = \mathcal{O}(d)$ such that $D[0]$ contains a singleton set consisting of the starting state of $T_C(R)$ and $D[r]$, $1 \leq r \leq n$, stores all states u such that u is the end of some transition labelled by an atomic string and $T[1\,..\,r]$ equals the concatenation of the labels of the transitions in some walk from the starting state of $T_C(R)$ to u. Assume that we have constructed $D[1\,..\,r]$. To compute $D[r+1]$, we use the trie to find the atomic strings $A_1, A_2, \ldots, A_q$ equal to $D[1\,..\,r+1], D[2\,..\,r+1], \ldots$ or $D[r+1]$ in $\mathcal{O}(r+q)$ time. Note that $q \leq d$. For each atomic string A_i, $1 \leq i \leq q$, labelling a transition (v, w), we add w to $D[r+1]$ if there is a state u in $D[r+1-|A_i|]$ such that there is an ε-transition path from u to v, which can be checked in $\mathcal{O}(d)$ time and space. In total, the algorithm spends $\mathcal{O}(d^3)$ time to process a character of T ($q = \mathcal{O}(d)$, and for each $1 \leq i \leq q$ the set $D[r+1-|A_i|]$ contains $\mathcal{O}(d)$ states). The algorithm reports that $T \in L(R)$ if $D[n]$ contains a state v, which is an endpoint of a transition labelled by some $A \in \mathcal{F}$.

In the regular expression pattern matching problem, we define an array D in the following way. As before, $D[0]$ contains a singleton set consisting of the starting state of $T_C(R)$. For every $1 \leq r \leq n$, $D[r]$ stores the starting state of $T_C(R)$ and all states u such that u is the end of some transition labelled by an atomic string and $T[\ell\,..\,r]$, for some $\ell \leq r$, equals the concatenation of the labels of the edges in some walk from the starting state of $T_C(R)$

to u. Assume that we have constructed $D[1\,..\,r]$. To compute $D[r+1]$, we use the trie to find the atomic strings $A_1, A_2, \ldots, A_q$ equal to $D[1\,..\,r+1], D[2\,..\,r+1], \ldots$ or $D[r+1]$ in $\mathcal{O}(r+q) = \mathcal{O}(d)$ time. For each atomic string A_i, $1 \le i \le q$, labelling a transition (v, w), we add w to $D[r+1]$ if there is a state u in $D[r+1-|A_i|]$ such that there is an ε-transition path from u to v, which can be checked in $\mathcal{O}(d)$ time and space. In total, the algorithm spends $\mathcal{O}(d^3)$ time to process a character of T. We report all positions r such that $D[r]$ contains a state v, which is an endpoint of a transition labelled by some $A \in \mathcal{F}$. □

From now on, we assume that all atomic strings have length at most n, and that $d \le n$. For a string P and a text T, denote by $\mathsf{occ}(P, T)$ the set of the ending positions of the occurrences of P in T. Our solutions for streaming regular expression membership and pattern matching are very similar, the main difference is how we define a witness:

Definition 3.2. (Witness (Membership)) *Let P be a canonical prefix of an atomic string, and $r \in \mathsf{occ}(P, T)$. We say that r is a* witness *if $T[1\,..\,r]$ is a partial occurrence of R ending with P.*

Definition 3.3. (Witness (Pattern matching)) *Let P be a canonical prefix of an atomic string, and $r \in \mathsf{occ}(P, T)$. We say that r is a* witness *if there exists a position $1 \le \ell \le r$ such that $T[\ell\,..\,r]$ is a partial occurrence of R ending with P.*

We exploit the following algorithm, which we refer to as the pattern matching algorithm[2]:

Theorem 3.4. (cf. [57, Theorem 2]) *Given a pattern of length at most n and a text T of length n over an alphabet of size $n^{\mathcal{O}(1)}$. There exists a randomised Monte Carlo streaming algorithm that uses $\mathcal{O}(\log n)$ space and $\mathcal{O}(\log n)$ time per character of the text. When it receives $T[i]$, it says whether $i \in \mathsf{occ}(P, T)$. The algorithm is correct with high probability.*[3]

We also make use of the following well-known fact:

Fact 3.5. (Fine and Wilf's periodicity lemma [22]) *If a string X has two periods of length p and q and $p + q \le |X|$, then X also has a period of length $\gcd(p, q)$.*

Intuition: non-periodic case. To give intuition behind our solutions, consider a very simple case when every canonical prefix is not periodic. We start with the following simple observation:

Observation 3.6. *By Fact 3.5, if P is not periodic, there can be at most two occurrences of P in a string of length $\le 2|P|$.*

Therefore, if none of the strings in Π is periodic, we can use the following approach. For each $P \in \Pi$ and T, we run the pattern matching algorithm and at any moment store the two most recent witnesses for P discovered by the algorithm (for membership, witness are defined as in Definition 3.2, and for pattern matching as in Definition 3.3). When the algorithm discovers a new position $r \in \mathsf{occ}(P, T)$, we must decide whether it is a witness. Let $P = A[1\,..\,\min\{2^k, |A|\}]$, where A is an atomic string.

If $k = 0$, we consider the starting node u of the transition in the compact Thompson automaton $T_C(R)$ labelled by A. Suppose that there is an ε-transitions path from the endpoints of the transitions labelled by atomic strings $A_{i_1}, A_{i_2}, \ldots, A_{i_j}$ to u. We then check if $(r-1)$ is a witness for at least one of $A_{i_1}, A_{i_2}, \ldots, A_{i_j}$. If it is, then r is a witness. Importantly, if $r-1$ is a witness for $A_{i_{j'}}$, $1 \le j' \le j$, it is the most recent one and is stored in the memory of the instance of the pattern matching algorithm for $A_{i_{j'}}$ and T. Suppose now that $k \ge 1$. We then must check whether $(r - 2^{k-1})$ is a witness for $A[1\,..\,2^{k-1}]$. If it is, then r is a witness for P. Note that by Observation 3.6, if $(r - 2^{k-1})$ is a witness for $A[1\,..\,2^{k-1}]$, it is one of the two most recent ones and will be stored by the pattern matching algorithm for $A[1\,..\,2^{k-1}]$.

Let $\mathcal{F}$ contain all atomic strings A such that there is an ε-transitions path from the endpoint of the transition labelled by A to the final state of $T_C(R)$. In the regular expression pattern matching problem, we report

[2]One could also use one of the streaming dictionary matching algorithms (see the introduction), but this does not change the final complexity and makes the description of the algorithm more complex.

[3]With high probability means with probability at least $1 - 1/n^c$ for any predefined constant $c > 1$.

all positions r such that r is a witness in $\mathsf{occ}(A,T)$ for some $A \in \mathcal{F}$. In the regular expression membership problem, $T \in L(R)$ if n is a witness for $\mathsf{occ}(A,T)$, for some $A \in \mathcal{F}$.

We do not provide the formal analysis of the algorithm, as we only give it for intuition, but it is easy to see that it uses $\mathcal{O}(d^2 \log^2 n)$ space and $\mathcal{O}(d \log^2 n)$ time per character of the text (recall that we do not account for the time spent during the preprocessing phase). As all atomic strings have length at most n and $d \leq n$, the algorithm is correct with high probability by Theorem 3.4.

General case: main technical contributions. In general, unfortunately, some of the canonical prefixes are periodic we can no longer use Observation 3.6. However, the following generalisation holds:

Observation 3.7. *By Fact 3.5, if for a string P and a string X, $X \leq 2|P|$, we have $|\mathsf{occ}(P,X)| > 2$, then P is periodic and the set $\mathsf{occ}(P,X)$ can be represented as an arithmetic progression with difference ρ, where ρ is the period of P.* [4]

Observation 3.7 gives the idea behind our approach for the general case. By this observation, we obtain that every $r \in \mathsf{occ}(P,T)$, where P is a canonical prefix of some atomic string periodic with period ρ, belongs to a fragment of form $(\Delta(P))^k$, where $\Delta(P) = P[|P| - \rho + 1\,..]$ and k is an integer. Instead of storing the last two witnesses for each canonical prefix, we would like to store the witnesses in the last two fragments of form $(\Delta(P))^k$. However, the number of such witnesses can be large. Our main technical novelty is a compressed representation of such witnesses. We give a high-level overview of the approach we use for the membership problem, our solution to the regular expression pattern matching problem is similar. We show that for each fragment of form $(\Delta(P))^k$ it suffices to store a small, carefully selected subset of witnesses that belong to this fragment. The remaining ones can be restored in small space at request.

Consider a witness $r \in \mathsf{occ}(P,T)$, where $P \in \Pi$. By definition, there is a partition $T[1\,..\,r] = T[\ell_1\,..\,r_1]T[\ell_2\,..\,r_2]\dots T[\ell_m\,..\,r_m]$ such that each fragment in the partition, except for the last one, is an atomic string, and the last one equals P. Furthermore, by Observation 3.7, r must belong to some fragment $F = T[i\,..\,i+k\rho-1] = (\Delta(P))^k$. Let m' be the index of the first fragment such that $r_{m'} \geq i$. Consider the fragment $W = T[\ell_{m''}\,..\,r_{m''}]$, $m' \leq m'' \leq m$, containing a position $i + 2\rho - 1$ (we call this position an "anchor"). Note that W is a canonical prefix of some atomic string and $r_{m''} \in \mathsf{occ}(W,T)$ is a witness. If there are a few witnesses $t \in \mathsf{occ}(W,T)$ such that $T[t-|W|+1,t]$ contains the anchor $i+2\rho-1$, we can store them explicitly. Otherwise, there is a periodic fragment containing $i+2\rho-1$, and we can recurse for it by choosing a new anchor close to its starting point. We choose the definition of anchors (see Section 4.1) so that the recursion stops in a logarithmic number of steps and for some of the anchors there is a witness that we store explicitly for this anchor.

To summarize, the idea of the compact representation of witnesses that belong to a fragment of form $(\Delta(P))^k$ is to choose a logarithmic set of anchors close to the starting point of the fragment, and for each of these anchors to store a constant number of witnesses for each canonical prefix in Π. Suppose now that $r \in \mathsf{occ}(P,T)$, where P is a canonical prefix of an atomic string A, r belongs to a fragment $F = T[i\,..\,i+k\rho-1] = (\Delta(P))^k$. To decide whether it is a witness we use the following approach. From above we know that r is a witness iff there is a witness $r' \in \mathsf{occ}(A',T)$, where A' is an atomic string, that we store in the compact representation of witness in F, and there is a path in $T_C(R)$ from the ending node of the transition labelled by A' and to the starting node of the transition labelled by A such that the concatenation of the strings on the edges of the path equals $T[r'+1\,..\,r-|A|]$ (which is a substring of $(\Delta(P))^k$). Unfortunately, it is not clear how to verify this condition in a straightforward way as we do not have random access neither to $\Delta(P)$, nor to the strings on the edges of $T_C(R)$. Instead, using anchors again, we show that verifying this condition can be reduced to the following question, where G is a graph of size $\mathrm{poly}(d, \log n)$ (see Lemma 4.12 for details):

> Walks in a weighted graph
> Given a directed multigraph G with non-negative integer weights on edges, two nodes and a number x, decide if there is a walk from the first node to the second one of total weight x.

Walks in a weighted graph and circuits. In Section 5, we show the following theorem:

[4] Note that when $|\mathsf{occ}(P,X)| \leq 2$, we can represent $\mathsf{occ}(P,X)$ as at most two (degenerate) arithmetic progressions of length 1, we will use this fact to simplify the description of the algorithms.

THEOREM 3.8. *There exists an algorithm which, given a directed multigraph G with non-negative integer weights on edges, its two nodes v_1 and v_2 and a number x, decides if there is a walk from v_1 to v_2 of total weight x in $\mathcal{O}((|E(G)|+|V(G)|^3)x \operatorname{polylog} x)$ time and $\mathcal{O}((|E(G)|+|V(G)|^3)\operatorname{polylog} x)$ space and succeeds with probability at least $1/2$.*

Let $N = |V(G)|$. For the simpler case when the graph is unweighted, we could use a folklore approach and compute the x-th power of the adjacency matrix in $\mathcal{O}(N^3 \log x)$ time and $\mathcal{O}(N^2)$ space. In order to handle arbitrary weights of edges, we compute the arrays C_k of bit-vectors of length $x+1$, where $C_k[u,v][d]$ stores a bit indicator of whether there exists a walk from u to v in G of at most 2^k edges of total weight exactly d. The following formula holds:

$$C_k[u,v][d] = \bigvee_{\substack{w \in V(G) \\ i \in \{0,\ldots,d\}}} C_{k-1}[u,w][i] \wedge C_{k-1}[w,v][d-i]$$

Using the fast Fourier transform to compute the convolutions, we obtain an algorithm with time $\mathcal{O}(N^3 x \log^2 x)$ and space $\mathcal{O}(N^2 x)$.

In our application, x can be equal to n, and the approach above uses $\Omega(n)$ space, which is prohibitive. In order to improve the space complexity, we represent the above computations as a circuit with binary OR and CONVOLUTION$_x$ gates operating on bit-vectors of length $x+1$. Every element $C_k[u,v]$ requires a separate gate and while computing its value we need to perform N convolutions, for every possible intermediate node w, so in total there are $\mathcal{O}(N^3 \log x)$ gates. The CONVOLUTION$_x$ gates store only the first $x+1$ bits of the results, as we never need paths of total weight larger than x. We are interested only in a single bit of output of the circuit, namely $C_{\lceil \log x \rceil}[v_1,v_2][x]$. If there were only OR gates in the circuit, we could store only the x-th element at each gate. In order to handle also CONVOLUTION$_x$ gates, we use the discrete Fourier transform over a suitably chosen ring.

We use the technique introduced by Lokshtanov and Nederlof [50] and then modified Bringmann [12] to work with numbers modulo p instead of complex numbers. Informally, they show that if we operate on $\mathbb{Z}_p^t$ (vectors of length t with elements in $\mathbb{Z}_p$ for suitably chosen p and t) instead of the bit-vectors, we can compute $out(C)[x]$, the x-th element of the output of the circuit C in $\mathcal{O}(|C|t \operatorname{polylog} p)$ time and $\mathcal{O}(|C| \log p)$ space (see details in Theorem 5.1). However, there are technical difficulties that we need to overcome to apply their technique to our solution. The approach of Bringmann [12] requires that $t > x$ and $\mathbb{Z}_p$ contains a t-th root of the unity. The main difficulty is to choose these numbers as small as possible as they directly affect the complexity of the algorithm. This question was also faced by Bringmann [12], who showed two variants of the framework, one using the Extended Riemann Hypothesis and the other unconditional but with polynomially higher time and space, which is not good enough for our streaming application. By using Bombieri–Vinogradov theorem (see details in Theorem 5.6) and facts about counting primes in arithmetic progressions, we obtain an unconditional time bound comparable to that of Bringmann that assumes the Extended Riemann Hypothesis.

4 Regular Expression Membership and Pattern Matching

In this section, we address the problems of regular expression membership and pattern matching in the streaming model of computation. In Section 4.1, we introduce a notion of *anchors* that is the key to achieving the desired space complexity. In Section 4.2, we describe the algorithms.

4.1 Anchors

In this section we define the notion of anchors that will allow us to store all occurrences (partial or not) of regular expressions detected by the algorithm efficiently.

DEFINITION 4.1. (ANCHORS) *Consider a periodic string W with period ρ. Set the anchor $a_0 = 2\rho$ and the period $\rho_0 = \rho$. Suppose that $(a_r, \rho_r)_{r=0}^{q-1}$ are defined. We define (a_q, ρ_q) recursively. Consider the set S of fragments $W[i\,..\,j]$ of W satisfying the following properties:*

1. *$W[i\,..\,j]$ contains a_{q-1} and is periodic with period $\pi < \rho_{q-1}$;*
2. *$i + 4\pi - 1 \le a_{q-1}$ (there are at least four repetitions of the string period of $W[i\,..\,j]$ before a_{q-1});*
3. *$a_{q-1} \le j - 4\pi - r$, where $r = (j-i+1) \pmod{\pi}$ (there are at least four repetitions of the string period of $W[i\,..\,j]$ after a_{q-1}).*

Let $i_q = \min\{i : W[i\,..\,j] \in S\}$ and $j_q = \max\{j : W[i_q\,..\,j] \in S\}$. If $W[i_q\,..\,j_q]$ is undefined, recursion stops. Otherwise, ρ_q is defined to be the period of $W[i_q\,..\,j_q]$ and $a_q := i_q + 2\rho_q - 1$.

Let $\mathcal{A}(W) = \{a_0, a_1, \ldots, a_Q\}$, where (a_Q, ρ_Q) is the last defined anchor-period pair. We call $\mathcal{A}$ the generator set of anchors *of W. Define $\mathcal{A}^*(W) = \left[\bigcup_{\Delta\in\mathbb{Z}_+}(\mathcal{A}(W) + \Delta\cdot\rho)\right] \cap [1, |W| - 2\rho]$. We refer to $\mathcal{A}^*$ simply as the* set of anchors *of W. For an illustration, see Fig. 3.*

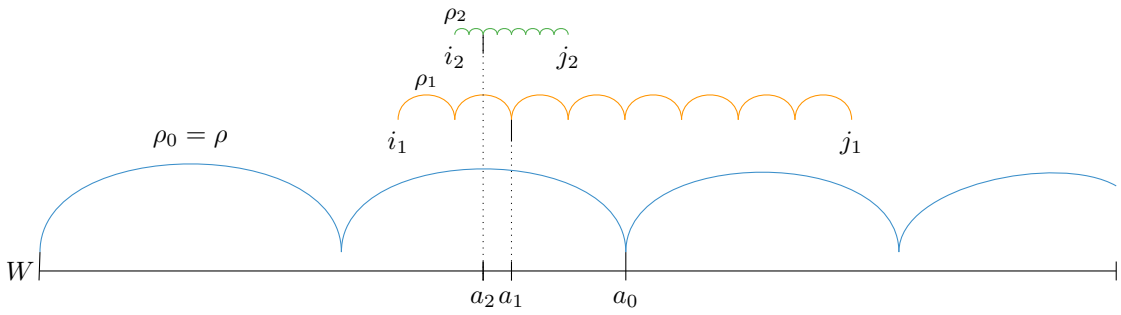


Figure 3: Anchors of a periodic string W with period ρ.

(In the next section we slightly abuse notation and extend the notion of the set of anchors to infinite strings in a natural way, i.e. for an infinite string W with period ρ the set $\mathcal{A}^*(W) = \left[\bigcup_{\Delta\in\mathbb{Z}_+}(\mathcal{A}(W) + \Delta\cdot\rho)\right]$.) We first show that the generator set of anchors has logarithmic size:

LEMMA 4.2. *Let W be a periodic string with period ρ. We have $|\mathcal{A}(W)| = \mathcal{O}(\log\rho)$.*

Proof. Let $\mathcal{A}(W) = \{a_0, a_1, \ldots, a_Q\}$. For each $0 \le q \le Q$, let $W[i_q\,..\,j_q]$ be the fragment associated with a_q, and ρ_q be its period. (In particular, for $q = 0$ we have $i_0 = 1$, $j_0 = |W|$, $\rho_0 = \rho$.) We show that for each $1 \le q \le Q$ we have $|W[i_q\,..\,j_q]| \le 2\rho_{q-1}$. This implies, in particular, that $|W[i_q\,..\,a_{q-1}]| \le 2\rho_{q-1}$ and therefore $\rho_q \le \rho_{q-1}/2$. The lemma follows immediately.

Fix $0 \le q \le Q$. Assume by contradiction that $|W[i_q\,..\,j_q]| > 2\rho_{q-1}$. We then have that $W[i_q\,..\,j_q]$ has periods ρ_{q-1} and ρ_q and $\rho_{q-1} + \rho_q < |W[i_q\,..\,j_q]|$. Hence, $W[i_q\,..\,j_q]$ is periodic with period $\pi = \gcd(\rho_{q-1}, \rho_q)$ by Fact 3.5. The substring $W[i_q\,..\,j_q]$ contains a full copy of $W[i_{q-1}\,..\,i_{q-1} + \rho_{q-1} - 1]$. Therefore, $W[i_{q-1}\,..\,i_{q-1} + \rho_{q-1} - 1]$ has a period π, which implies that it equals $(W[i_{q-1}\,..\,i_{q-1} + \pi - 1])^{\rho_{q-1}/\pi}$, i.e. the string period of $W[i_{q-1}\,..\,j_{q-1}]$ is not primitive, a contradiction. ∎

DEFINITION 4.3. *Let W be a periodic string with period ρ. We say that a fragment $F = W[i\,..\,j]$ is anchored by an anchor $a \in \mathcal{A}^*(W)$ if $i \le a \le j$ and for any strings $U \in \Sigma^*, V \in \Sigma^\rho$ such that $V \ne W[1\,..\,\rho]$ there are at most eight occurrences of F in $UV(W[1\,..\,j])$ containing the anchor (i.e., containing $|U| + |V| + a$).*

LEMMA 4.4. *Let W be a periodic string with period ρ. Consider a fragment $W[\ell\,..\,r]$ of length at least 4ρ and a partitioning $W[\ell\,..\,r] = W[\ell_1\,..\,r_1]W[\ell_2\,..\,r_2]\ldots W[\ell_k\,..\,r_k]$.*

(a) There exists $1 \le k' \le k$ such that $W[\ell_{k'}\,..\,r_{k'}]$ is anchored by an anchor $a \in \mathcal{A}^*(W) \cap [r - 4\rho + 1, r]$.

(b) If, in addition, $1 \le \ell \le 2\rho$, there exists $1 \le k' \le k$ such that $W[\ell_{k'}\,..\,r_{k'}]$ is anchored by an anchor $a \in \mathcal{A}^*(W) \cap [1, 4\rho]$.

Proof. The high-level idea of the proof of (a) and (b) is as follows. Let $\mathcal{A}(W) = \{a_0, a_1, \ldots a_Q\}$. For every $0 \le q \le Q$ and an integer $\Delta > 0$ to be determined later, define $a_q^\Delta := a_q + \Delta\cdot\rho$. Let $F_q = W[\ell_{k_q}, r_{k_q}]$ be the fragment that contains a_q^Δ. We show that either F_q is anchored by a_q^Δ or $q < Q$, which yields the lemma. We exploit two auxiliary claims:

CLAIM 4.5. *Let π be the period of F_q, $0 \le q \le Q$. If F_q is not anchored by a_q^Δ, then there exist $U \in \Sigma^*, V \in \Sigma^\rho$ with $V \ne W[1\,..\,\rho]$ such that there is a fragment $S[p\,..\,t]$ of the string $S = UV(W[..\,r_q])$ satisfying the following properties:*

1. $p \le |U| + |V| + a_q^\Delta \le t$ *(the fragment contains the anchor);*
2. $S[p\,..\,t]$ *is periodic with period* π*;*
3. $p + 7 \cdot \pi \le |U| + |V| + a_q^\Delta$ *(there are at least seven repetitions of the string period of the fragment before the anchor);*
4. $|U| + |V| + a_q^\Delta \le t - 6\pi - r$*, where* $r = t - p + 1 \pmod{\pi}$ *(there are at least six repetitions of the string period of the fragment after the anchor).*

Proof. Let $U \in \Sigma^*, V \in \Sigma^\rho$ with $V \neq W[1\,..\,\rho]$ such that there are least eight occurrences of F_q in $S = UV(W[1\,..\,r_q])$ containing $a := |U| + |V| + a_q^\Delta$. Let the last eight occurrences be $S[p_k\,..\,t_k]$, $1 \le k \le 8$. As all occurrences contain a, the length of $S[p_1\,..\,t_8]$ is at most $2|F_q|$. By Observation 3.7, we obtain that $S[p_1\,..\,t_8]$ is periodic with period π, $p_k = p_1 + (k-1)\pi$, and $t_k = t_1 + (k-1)\pi$. As $p_8 = p_1 + 7\pi \le a$, we have $S[p_1\,..\,a] \ge 7\pi$. On the other hand, $t_2 - r + 1 \ge t_1 + 1 \ge a + 1$. Therefore, $|S[a+1\,..\,t_8 - r]| \ge |S[t_2 - r + 1\,..\,t_8 - r]| \ge 6\pi$. By taking $p = p_1$ and $t = t_8$, we obtain the claim. For an illustration, see Fig. 4. ∎

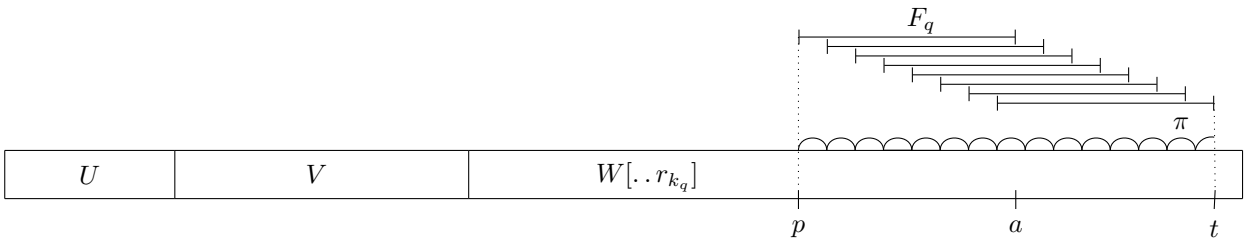


Figure 4: Illustration of Claim 4.5.

Claim 4.6. *Assume that* $q \ge 0$ *and that the period of* F_q *is* $\pi < \rho_q$*. If* F_q *is not anchored by* a_q^Δ*, then* $q < Q$*.*

Proof. By Claim 4.5, there exist $U \in \Sigma^*, V \in \Sigma^\rho$ with $V \neq W[1\,..\,\rho]$ such that there is a fragment $S[p\,..\,t]$ of the string $S = UV(W[..\,r_q])$ periodic with period π that contains $a := |U| + |V| + a_q^\Delta$ and such that there are at least six repetitions of the string period before and after a.

Let us show that $a - 2\rho_q + 1 \le p < t \le a + 2\rho_q$. Suppose by contradiction that $p < a - 2\rho_q + 1$. We have that $S[a - 2\rho_q + 1\,..\,a] = W[a_q^\Delta - 2\rho_q + 1\,..\,a_q^\Delta]$ (note that by definition $a_q^\Delta \ge 2\rho_q$ for any Δ). Furthermore, $W[a_q^\Delta - 2\rho_q + 1\,..\,a_q^\Delta]$ has periods ρ_q (by definition of i_q and a_q^Δ) and π (by the assumption). By Fact 3.5, $W[a_q^\Delta - 2\rho_q + 1\,..\,a_q^\Delta]$ has a period $\gcd(\rho_q, \pi) < \rho_q$. As $W[a_q^\Delta - 2\rho_q + 1\,..\,a_q^\Delta]$ contains a full copy of the string period of $W[i_q\,..\,j_q]$, we obtain that it is not primitive, a contradiction. (See Fig. 5). To show that $t \le a + 2\rho_q$, note that $S[a+1\,..\,a+2\rho_q] = W[a_q^\Delta + 1\,..\,a_q^\Delta + 2\rho_q]$, $a_0^\Delta + 2\rho \le |W|$ and, for $q \ge 1$, $a_q^\Delta + 2\rho_q \le a_{q-1}^\Delta$ by definition of i_q and a_q. Therefore, for all $q \ge 0$, $W[a_q^\Delta + 1\,..\,a_q^\Delta + 2\rho_q]$ is periodic with period ρ_q. The rest of the argument is analogous.

From $a - 2\rho_q + 1 \le p < t \le a + 2\rho_q$ and the fact that $S[p\,..\,t]$ contains at least six repetitions of its string period before and after a, we obtain that i_{q+1}, j_{q+1} and hence a_{q+1}, ρ_{q+1} are well-defined, which completes the proof of the claim. ∎

We are now ready to show (a) and (b).

(a) Let $\Delta \le \lfloor |W|/\rho \rfloor - 2$ be the smallest integer such that $a_0 + \Delta \cdot \rho \ge r - 4\rho$. Note that Δ is well-defined. Consider an anchor $a_0^\Delta = a_0 + \Delta \cdot \rho \in \mathcal{A}^*(W) \cap [r - 4\rho + 1, r]$. Let $F_0 = W[\ell_{k_0}\,..\,r_{k_0}]$ be the fragment that contains a_0^Δ and π be its period. If F_0 is anchored by a_0^Δ, we are done. Otherwise, by Claim 4.5, there exist $U \in \Sigma^*, V \in \Sigma^\rho$ with $V \neq W[1\,..\,\rho]$ such that there is a fragment $S[p\,..\,t]$ of the string $S = UV(W[..\,r_{k_0}])$ periodic with period π that contains $a := |U| + |V| + a_0^\Delta$ and such that there are at least six repetitions of the string period of F_0 before and after a. As we have $r_{k_0} - a_0^\Delta + 1 \le r - a_0^\Delta + 1 \le 4\rho$, there is $\pi \le 2\rho/3$. It follows that i_1, j_1 and hence a_1, ρ_1 are well-defined, i.e. $0 < Q$.

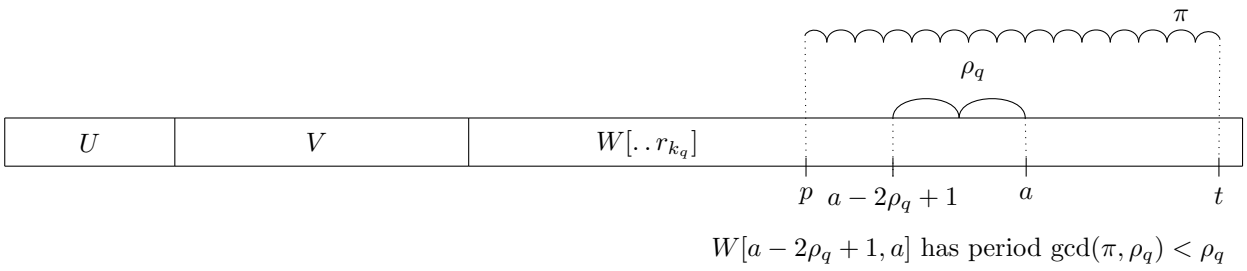


Figure 5: Illustration of the proof of Claim 4.6, case $p < a - 2\rho_q + 1$.

We now show that for arbitrary $q \geq 1$ either $a_q^\Delta = a_q + \Delta \cdot \rho$ anchors F_q or the period π of F_q is smaller than ρ_q, which by Claim 4.6 implies that $q < Q$. By our choice of Δ, F_q is well-defined. If F_q is anchored by a_q^Δ, we are done. Otherwise, by Claim 4.5, there exist $U \in \Sigma^*, V \in \Sigma^\rho$ with $V \neq W[1\,..\,\rho]$ such that there is a fragment $S[p\,..\,t]$ of the string $S = UV(W[..\,r_{k_q}])$ periodic with period π that contains $|U| + |V| + a_q^\Delta$ and such that there are at least six repetitions of the string period of $S[p\,..\,t]$ before and after $|U| + |V| + a_q^\Delta$. Recall that $a_q^\Delta = (i_q + 2\rho_q) + \Delta \cdot \rho$.

First, we have that $\pi \neq \rho_q$, otherwise we could have extended $W[i_q\,..\,j_q]$ to the left. Second, let us show that the case $\pi > \rho_q$ is impossible. Suppose otherwise. If $t - 4\pi + 1 \leq |U| + |V| + a_{q-1}^\Delta$, then $W[a_q^\Delta + 1\,..\,a_{q-1}^\Delta]$ contains a copy of $S[1\,..\,2\pi]$, and therefore $S[1\,..\,2\pi]$ has a period ρ_q. By Fact 3.5, the string period $S[1\,..\,\pi]$ of S is not primitive, a contradiction. (See Fig. 6a.) Otherwise, $|U| + |V| + a_{q-1}^\Delta$ is contained in $S[p\,..\,t]$, which has period $\pi > \rho_{q-1}$. In addition, there are at least four repetitions of the string period of $S[p\,..\,t]$ before and after $|U| + |V| + a_{q-1}^\Delta$, and $p < |U| + |V| + a_q^\Delta - 2\rho_q \leq |U| + |V| + i_q$, a contradiction with the choice of $W[i_q\,..\,j_q]$. (See Fig. 6b.) It finally follows that $\pi < \rho_q$ and therefore by Claim 4.6, $q < Q$.

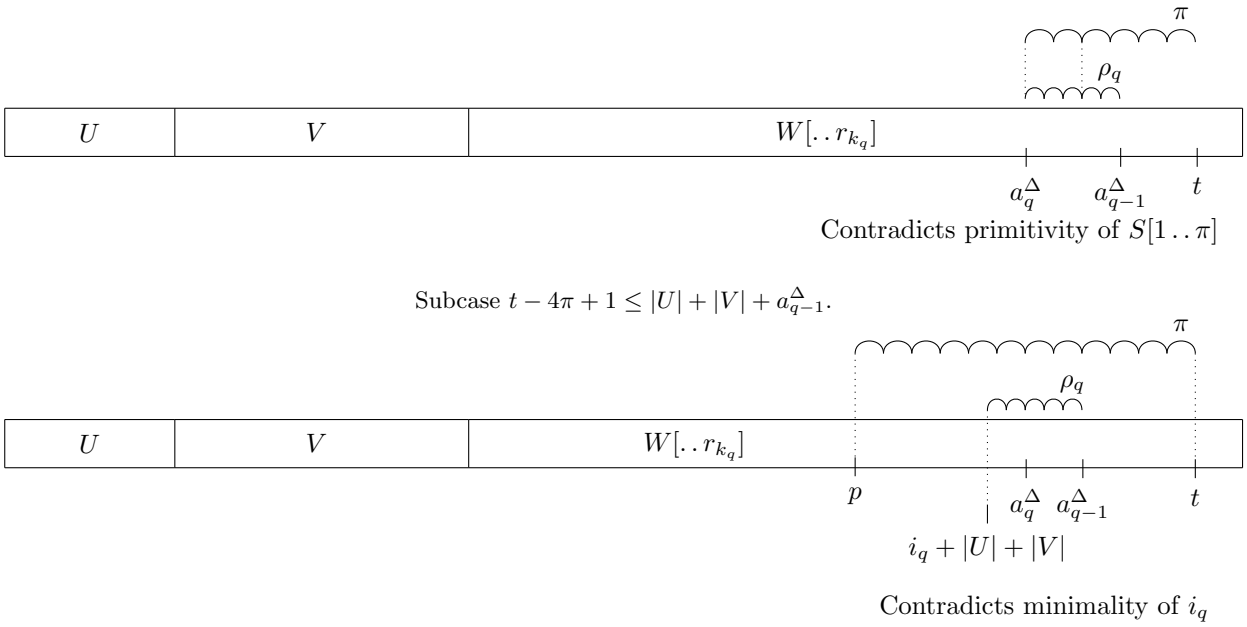


Figure 6: Illustration of the proof of Lemma 4.4(a), case $\pi > \rho_q$.

(b) Let Δ be the smallest integer such that $a_0 + (\Delta - 2)\rho \geq \ell$ (note that $\Delta = 1, 2$). Consider an anchor $a_0^\Delta = a_0 + \Delta \cdot \rho \in A^*(W) \cap [1, 4\rho]$. We first show that either F_0 is caught by a_0^Δ, or $0 < Q$.

If F_0 is anchored by a_0^{Δ}, we are done. Otherwise, let π be the period of F_0. We claim that $\pi < \rho = \rho_0$. As F_0 is not anchored, by Claim 4.5 there exist $U \in \Sigma^*, V \in \Sigma^\rho$ with $V \neq W[1\,..\,\rho]$ such that there is a fragment $S[p\,..\,t]$ of the string $S = UV(W[..\,r_{k_0}])$ periodic with period π that contains $a := |U| + |V| + a_0^{\Delta}$ and such that there are at least six repetitions of the string period of S before and after a. We can immediately rule out the case $\pi = \rho$ as $a \leq 4\rho$ and $V \neq W[1\,..\,\rho]$. Consider now the case $\pi > \rho_0$. Consider the suffix $S[a+1\,..] = W[a_0^{\Delta}+1\,..\,r_{k_0}]$. It contains an occurrence of $S[1\,..\,2\pi]$. By Fact 3.5, $S[1\,..\,2\pi]$ has a period $\gcd(\pi,\rho)$, and therefore the string period of S is not primitive, a contradiction. For $q \geq 1$, F_q is well-defined by our choice of Δ. The rest of the argument repeats the argument in the proof of (a).

This concludes the proof of Lemma 4.4. ∎

4.2 Algorithms In this section, we give a description of our new streaming algorithms for regular expression membership and pattern matching. The structure of the algorithms is very similar, the main difference is how we define a witness (see Definitions 3.2 and 3.3). For this reason, we describe the algorithms in parallel. Recall that T is a string of length n over an alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$, where $\sigma = n^{\mathcal{O}(1)}$, and $A_1, A_2, \ldots, A_d$ are the atomic strings for the regular expression R. Recall also that Π is the set of canonical prefixes of the atomic strings, defined as $\Pi = \{A_i[1\,..\,\min\{2^j, |A_i|\}] : 1 \leq i \leq d, 0 \leq j \leq \lceil \log |A_i| \rceil\}$.

We make use of the following corollary of Fact 3.5:

Corollary 4.7. (Of Fact 3.5) *For a primitive string X of length x, a string $D = XX$ can contain only two occurrences of string X, $D[1\,..\,x]$ and $D[x+1\,..\,2x]$.*

Preprocessing. We start by deleting all transitions (u, v) from $T_C(R)$ that are labelled by atomic strings of lengths larger than n.

Let $\mathcal{F}$ contain all atomic strings A such that there is an ε-transitions path from the endpoint of the transition labelled by A to the final state of $T_C(R)$. In addition, for each atomic string A, compute the subset $A_{i_1}, A_{i_2}, \ldots, A_{i_j}$ of atomic strings such that there is an ε-transitions path from the endpoint of the transition labelled by $A_{i_{j'}}$, $1 \leq j' \leq j$, to the starting point of the transition labelled by A.

For each periodic $P \in \Pi$, consider a string $\Delta(P) = P[|P| - \rho + 1\,..]$, where ρ is the period of P. During the preprocessing step, the algorithm computes the generator set of anchors $\mathcal{A}$ (Definition 4.1) for the string $W = (\Delta(P))^\infty$.

Define the *overlap* of two strings X and Y as the maximal length of a suffix of X that equals a prefix of Y, $\Pi(P)$ to be the set of all canonical prefixes such that their overlap with W is at least 2ρ, and $\mathsf{Overlap}(P) = \bigcup_{P' \in \Pi(P)} \{\ell : \ell \text{ is the overlap of } P' \text{ and } W\}$. The algorithm computes $\Pi(P)$ and $\mathsf{Overlap}(P)$ during the preprocessing step as well.

For regular expression pattern matching, it also computes the smallest integer $\mu(P)$ such that $p = \mu(P) \cdot \rho \in \mathsf{occ}(P, W)$ and p is a witness for P in W (in the sense of Definition 3.3).

Finally, the algorithm creates a directed graph $G(P) = (V, E)$ from the compact Thompson automaton $T_C(R)$. Consider again the string $W = (\Delta(P))^\infty$. For each canonical prefix $P' \in \Pi(P)$, the algorithm creates a node $v \in V$ corresponding to a pair (P', r), where r is the remainder of the overlap of P' and W modulo ρ. Additionally, for every fragment $W[i\,..\,j] = P'' \in \Pi$ which is anchored by an anchor $a \in \mathcal{A}^*(W)$, it creates a node $v \in V$ corresponding to a pair $(P'', j \pmod \rho)$ (for two identical prefix-remainder pairs, it creates just one node).

Consider two nodes $v', v'' \in V$. Suppose that v' corresponds to (P', r') and v'' to (P'', r''), where P', P'' are canonical prefixes of atomic strings A', A'', respectively, and r', r'' are remainders modulo ρ. For $0 < \ell \leq 10\rho$, the algorithm adds an edge (v', v'') of length ℓ to E if there is a walk in $T_C(R)$ from the ending state of the transition labelled by A' to the ending state of the transition labelled by A'' such that the concatenation of the labels in this walk equals to a string $L = \Delta(P)[r'\,..]\Delta(P)^\alpha\Delta(P)[..\,r'']$, where the integer power α is chosen so that $|L| = \ell$ (in other words, the concatenation equals to a fragment of W with the offsets defined by v' and v' and of an appropriate length, if such a fragment does not exist, the algorithm does not create the edge).

It might seem that the resulting graph is infinite, but as we show below, this is not the case.

Claim 4.8. *Consider all occurrences $W[\ell_i\,..\,r_i]$, $i \in Z_+$, of a string X in W. The size of the set $\{r_i \pmod \rho : W[\ell_i\,..\,r_i]$ is anchored by $a \in \mathcal{A}^*(W), i \in Z_+\}$ is $\mathcal{O}(\log \rho)$.*

Proof. We consider two cases: $|X| \geq 2\rho$ and $|X| < 2\rho$. In the first case, by Fact 3.5, if there is at least one occurrence of X in W, then X is periodic with period ρ. By Corollary 4.7, we have $r_i = q \pmod{\rho}$ for some fixed q and all $i \in Z_+$. The claim follows from Lemma 4.2.

In the second case, for all $a \in \mathcal{A}^*(W)$ such that $a \geq 2\rho$, and for all strings $U \in \Sigma^*, V \in \Sigma^\rho$ such that $V \neq W[1\,..\,\rho]$, all occurrences of X that contain $|U| + |V| + a$ are contained in $(UVW)[|U| + |V| + a - 2\rho + 1\,..\,|U| + |V| + a - 2\rho - 1] = W[a - 2\rho + 1\,..\,a - 2\rho - 1]$. It follows that for all $a, a' \in \mathcal{A}^*(W)$ such that $a, a' \geq 2\rho$ and $a = a' \pmod{\rho}$, the sets $\{r_i \pmod{\rho} : W[\ell_i\,..\,r_i]$ is anchored by $a, i \in Z_+\}$ and $\{r_i \pmod{\rho} : W[\ell_i\,..\,r_i]$ is anchored by $a', i \in Z_+\}$ are equal. Moreover, each of them contains only a constant number of elements. Therefore, the size of the set $\{r_i \pmod{\rho} : W[\ell_i\,..\,r_i]$ is anchored by $a \in \mathcal{A}^*(W), a \geq 2\rho, i \in Z_+\}$ is $\mathcal{O}(\log \rho)$ by Lemma 4.2. It remains to estimate the size of the analogous sets for anchors smaller than 2ρ. The number of such anchors is $\mathcal{O}(\log \rho)$ by Lemma 4.2, and each of them can anchor only a constant number of occurrences of X. The claim follows. ∎

COROLLARY 4.9. *$G(P)$ contains $|V| = \mathcal{O}(d \log^2 n)$ nodes and $|E| = \mathcal{O}(d^2 \log^4 n)$ edges, and can be constructed in $\mathcal{O}(\rho \cdot d^3 \log^4 n)$ time.*

Proof. The size of Π, and consequently $\Pi(P)$, is $\mathcal{O}(d \log n)$. For each string $P' \in \Pi(P)$, the remainder r of the overlap of P' and W modulo ρ is defined in a unique way. By Claim 4.8, for each string $P'' \in \Pi$ there are $\mathcal{O}(\log \rho)$ different remainders r modulo ρ such that there is an anchored occurrence of P'' ending at a position $p = r \pmod{\rho}$. Thus $|V| = \mathcal{O}(d \log^2 n)$.

Observe that the interval $[0, 10\rho]$ can contain only a constant number of values ℓ such that the power α is an integer. Hence for each pair of nodes we have only a constant number of possible edges, so $|E| = \mathcal{O}(d^2 \log^4 n)$. For each edge, we can check whether it exists in time $\mathcal{O}(\rho \cdot |T(R)|) = \mathcal{O}(|\rho| \cdot d)$. ∎

COROLLARY 4.10. (OF THEOREM 3.8) *There exists an algorithm which, given the graph $G(P)$, its two nodes v_1 and v_2 and a number $x \leq n$, decides if there is a walk from v_1 to v_2 of total weight x in $\mathcal{O}(xd^3 \operatorname{polylog} n)$ time and $\mathcal{O}(d^3 \operatorname{polylog} n)$ space and succeeds with high probability.*

Proof. Recall that $\rho \leq n$. We substitute the bounds from Corollary 4.9 into Theorem 3.8. Then we repeat the algorithm of Theorem 3.8 $2c\lceil \log n \rceil$ times and output the majority answer to obtain a success probability of at least $1 - 1/n^c$. ∎

Main phase. During the main phase of the algorithm, we run the pattern matching algorithm (Theorem 3.4) for every $P \in \Pi$. For every non-periodic $P \in \Pi$, we store (at most) two latest witnesses. For every periodic $P \in \Pi$, we run the pattern matching algorithm for $\Delta(P) = P[|P| - \rho + 1\,..]$ and T, where ρ is the period of P.

DEFINITION 4.11. *We say that a fragment $T[i\,..\,j]$ is a* streak *of a string X if $T[i\,..\,j] = X^k$ for some integer $k \geq 1$ and it is maximal, i.e. it cannot be extended neither to the left nor to the right.*

The pattern matching algorithm detects streaks of $\Delta(P)$ in the arrived prefix of T. Every witness r for P belongs to $\mathsf{occ}(P, T)$ and by Observation 3.7 ends in such a streak. At any moment of the algorithm, we store (at most) two latest streaks and a compact representation of witnesses in $\mathsf{occ}(P, T)$ that end in it. For membership testing we assume that the witnesses are defined as in Definition 3.2, and for pattern matching as in Definition 3.3. Perhaps a bit counter-intuitively, the representation contains witnesses from $\mathsf{occ}(P, T)$ and witnesses for other canonical prefixes as well, the reason for it will become clear later. Formally, the representation of a streak $S = T[i\,..\,j]$ consists of the following elements, where $\rho = |\Delta(P)|$:

1. For each $P' \in \Pi(P)$ and its overlap ℓ with W, the representation contains $p = i + \ell - 1$ if $p \in \mathsf{occ}(P', T)$ and is a witness;
2. All witnesses in $\mathsf{occ}(P, T)$ that belong to the interval $[i\,..\,i + 12\rho - 1]$;
3. For every $\ell \in \mathsf{Overlap}(P)$, all witnesses in $\mathsf{occ}(P, T)$ that belong to the interval $[i + \ell\,..\,(i + \ell - 1) + 8\rho]$;
4. For every $P' \in \Pi$, all witnesses $r \in \mathsf{occ}(P', T)$ such that $T[r - |P'| + 1\,..\,r]$ is a fragment of S and is anchored by an anchor in $\mathcal{A}^*(F) \cap [i, i + 4\rho]$.

By Observation 3.7 and Lemma 4.2, the compact representation of witnesses in a streak has size $\mathcal{O}(d \log^2 n)$. The algorithm uses the compact representations of witnesses to decide whether a newly detected occurrence $r \in \mathsf{occ}(P, T)$ for some $P \in \Pi$ is a witness:

LEMMA 4.12. *Let $d < n$. Assume that $T[r]$ is the latest arrived character of the text and that $r \in \mathsf{occ}(P, T)$. Assume that for each non-periodic $P' \in \Pi$ we store two latest witnesses in $\mathsf{occ}(P', T)$, and for each periodic $P' \in \Pi$ we store the two latest streaks of $\Delta(P')$ and compact representations of the witnesses in them. In addition, assume that we store the set of all atomic strings A such that $(r-1)$ is a witness for $\mathsf{occ}(A, T)$. One can decide whether r is a witness for P in $\mathcal{O}(nd^4 \operatorname{polylog} n)$ time and $\mathcal{O}(d^3 \operatorname{polylog} n)$ (extra) space with high probability.*

Proof. Let $P = A[1 \mathinner{..} \min\{2^k, |A|\}]$, where A is an atomic string. Consider two cases: $k = 0$ and $k \geq 1$.

Case 1: $k = 0$. If $k = 0$, let $A_{i_1}, A_{i_2}, \ldots, A_{i_j}$ be the atomic strings such that there is an ε-transitions path from the endpoint of the transition labelled by $A_{i'_j}$, $1 \leq j' \leq j$, to the starting point of the transition labelled by A. The position r is a witness for P iff for some $1 \leq j' \leq j$, the position $(r-1)$ is a witness for $A_{i_{j'}}$. We can decide whether this holds in $\mathcal{O}(d)$ time.

Case 2: $k \geq 1$. If $k \geq 1$, the position r is a witness for P iff $r - 2^{k-1}$ is a witness for $P[1 \mathinner{..} 2^{k-1}]$. For brevity, denote $r' = r - 2^{k-1}$, $P' = P[1 \mathinner{..} 2^{k-1}]$, and $\rho = |\Delta(P')|$. If P' is non-periodic and $r' \in \mathsf{occ}(P', T)$, the algorithm stores it explicitly by Observation 3.6. Otherwise, by Observation 3.7, r' belongs to one of the two latest streaks of $\Delta(P')$, let it be a fragment $S = T[i \mathinner{..} i + \ell \cdot \rho - 1]$. Suppose that r' is a witness for P'. Let us first explain the solution for the regular expression membership problem, and then we will show how to modify it for the regular expression pattern matching problem.

In the membership problem, if r' is a witness for P', then $T[1 \mathinner{..} r']P[2^{k-1} + 1 \mathinner{..}]$ is a partial occurrence of R and there is a partition of $T[1 \mathinner{..} r'] = T[\ell_1 \mathinner{..} r_1]T[\ell_2 \mathinner{..} r_2] \ldots T[\ell_m \mathinner{..} r_m]$, where each $T[\ell_{m'} \mathinner{..} r_{m'}]$, $1 \leq m' < m$, is an atomic string, and $T[\ell_m \mathinner{..} r_m] = P'$. Let $T[\ell_{m'} \mathinner{..} r_{m'}]$ be the fragment containing i. We consider two subcases: $r_{m'} - i + 1 > 2\rho$ and $r_{m'} - i + 1 \leq 2\rho$.

Case 2(a): $r_{m'} - i + 1 > 2\rho$. We claim that in this subcase $r_{m'} - i + 1$ equals the overlap ℓ of $T[\ell_{m'} \mathinner{..} r_{m'}]$ and $W = (\Delta(P'))^\infty$. By definition, $r_{m'} - i + 1 \leq \ell$. Suppose that $r_{m'} - i + 1 < \ell$. If $\ell - (r_{m'} - i + 1)$ is a multiple of ρ, then we obtain that $T[r_{m'} - \rho \mathinner{..} r_{m'} - 1] = \Delta(P')$, a contradiction with the definition of S. If $\ell - (r_{m'} - i + 1)$ is not a multiple of ρ, then there is an occurrence of $\Delta(P')$ in the prefix $(\Delta(P'))^2$ of S that does not end at positions ρ or 2ρ. By Corollary 4.7, we obtain a contradiction. Therefore, $r_{m'} - i + 1 = \ell$ and the compact representation of witnesses in S stores $r_{m'} \in \mathsf{occ}(T[\ell_{m'} \mathinner{..} r_{m'}], T)$.

If $m' = m$ or $r_m - r_{m'} \leq 8\rho$, then we are done: if r' is a witness, it must be stored explicitly, and we can check whether it is the case in $\mathcal{O}(d \log^2 n)$ time. Otherwise, we use the following claim:

CLAIM 4.13. *There is a sequence $m' = m_0 < m_1 < m_2 < \cdots < m_q = m$ such that each $T[\ell_{m_{q'}} \mathinner{..} r_{m_{q'}}]$, $1 \leq q' < q$, is either anchored by an anchor $a \in \mathcal{A}^*(S)$, or has length at least 2ρ, and for each $1 \leq q' \leq q$, $\ell_{m_{q'}} - r_{m_{q'-1}} \leq 10\rho$.*

Proof. The sequence is built as follows. Let $m_0 = m'$ and $m_{q'}$ be the latest index added to the sequence. If there is an index m'' such that $T[\ell_{m''} \mathinner{..} r_{m''}]$ has length at least 2ρ or $m'' = m$ and $\ell_{m''} - r_{q'} \leq 10\rho$, then set $m_{q'+1} = m''$ and continue. Otherwise, let m'' be the smallest index such that $\ell_{m''} - r_{m_{q'}} \geq 8\rho$. Note that we also have $\ell_{m''} - r_{m_{q'}} \leq 10\rho$ (otherwise, the length of $T[\ell_{m''-1} \mathinner{..} r_{m''-1}]$ would have been larger than 2ρ). By Lemma 4.2, there is $m_{q'} < \tilde{m} \leq m''$ such that $T[\ell_{\tilde{m}} \mathinner{..} r_{\tilde{m}}]$ is anchored by an anchor $a \in \mathcal{A}^*(S) \cap [r_{m''} - 4\rho + 1, r_{m''}]$. We set $m_{q'+1} = \tilde{m}$ and continue. ∎

Let v' be the node in $G(P)$ corresponding to $(T[\ell_{m'} \mathinner{..} r_{m'}], r_{m'} - i + 1 \pmod{\rho})$, and v be the node corresponding to $(P', r_m - i + 1 \pmod{\rho})$. We have that j' is a witness iff there exists the sequence $m' = m_0 < m_1 < m_2 < \cdots < m_q = m$ as above iff there is a walk from v to v' of length $|T[r_{m'} + 1 \mathinner{..} r_m]| \leq n$, which we can check in $\mathcal{O}(nd^4 \operatorname{polylog} n)$time and $\mathcal{O}(d^3 \operatorname{polylog} n)$ extra space with high probability via Corollary 4.10 (we must check whether this condition is verified for each of the $\mathcal{O}(d \log^2 n)$ witnesses stored in the compact representation of S).

Case 2(b): $r_{m'} - i + 1 \leq 2\rho$. Consider now the second subcase. If $r_m \leq 12\rho$, then we are done: if r' is a witness, it must belong to the compact representation of witnesses in S, which can be verified in $\mathcal{O}(d \log^2 n)$ time. Otherwise, $r_m - r_{m'} \geq 8\rho$. By Lemma 4.4(a) there is p, $m' < p \leq m$, such that $T[\ell_p \mathinner{..} r_p]$ is anchored by an anchor $\mathcal{A}^*(F) \cap [i, i + 4\rho - 1]$, and therefore $T[\ell_p \mathinner{..} r_p]$ is stored in the compact representation of witnesses

in S. Analogously to Case 2(a), we can show equivalence of the following conditions: r' is a witness; there is a walk in $G(P)$ from the node corresponding to $(T[\ell_p \mathinner{..} r_p], r_p - i + 1 \pmod{\rho})$ to the node corresponding to $(T[\ell_m \mathinner{..} r_m], r_m - i + 1 \pmod{\rho})$ of length $|T[r_p + 1 \mathinner{..} r_{m-1}]|$. We can therefore decide whether r' is a witness via Corollary 4.10 in $\mathcal{O}(nd^4 \operatorname{polylog} n)$ time and $\mathcal{O}(d^3 \operatorname{polylog} n)$ space with high probability.

We now explain how to modify the argument so that it can be used for regular expression pattern matching. Note that in pattern matching, if r' is a witness for P', then there is some position ℓ', $1 \leq \ell' \leq r'$ such that $r' \in \mathsf{occ}(P', T)$ is a witness. The position ℓ' can be inside the streak S, i.e. m' can be undefined, making it impossible to apply the argument above. However, we can easily check if this is the case using the integer $\mu(P')$ we computed during the preprocessing step: if $\mu(P') \cdot \rho \geq (r' - i + 1)$, then r' is a witness and we are done, and otherwise $\ell' \leq i$ (if r' is a witness), m' is defined, and we can apply the argument above. ∎

THEOREM 4.14. *Given a streaming text T of length n and a regular expression R of size d. There is a randomised algorithm that solves the membership and the pattern matching problems for T and R in $\mathcal{O}(d^3 \operatorname{polylog} n)$ space and $\mathcal{O}(nd^5 \operatorname{polylog} n)$ time per character of the text. The algorithm succeeds with high probability.*

Proof. If $d \geq n$, we can use Claim 3.1. Below we assume that $d < n$. Recall that we do not account for the time used during the preprocessing step (but one can note that it is polynomial in d and the total length of the atomic strings of R). The information computed during this step, including the graphs $G(P)$ for each $P \in \Pi$, takes $\mathcal{O}(d^3 \log^5 n)$ space.

During the main step, we use Lemma 4.12 to maintain the compact representations of the streaks of $\Delta(P)$ for each $P \in \Pi$, and to decide, eventually, whether T matches the regular expression R. Whenever an instance of the pattern matching algorithm detects an occurrence of $\Delta(P)$, we decide in constant time whether this occurrence extends the latest streak of $\Delta(P)$ or starts a new one. If the number of streaks becomes equal to three, we discard the oldest streak.

When an instance of the pattern matching algorithm detects $r \in \mathsf{occ}(P, T)$ for some $P \in \Pi$, we must decide whether r is a witness and whether we must store it in the compact representation of the streaks containing r. We apply Lemma 4.12 to decide whether r is a witness in $\mathcal{O}(nd^4 \operatorname{polylog} n)$ total time and $\mathcal{O}(d^3 \operatorname{polylog} n)$ space and then in $\mathcal{O}(d \log n)$ time whether r must be added to the compact representations of the streaks containing r. Note that a position r can belong to $\mathsf{occ}(P, T)$ for $\mathcal{O}(d \log n)$ canonical prefixes $P \in \Pi$, and therefore in the worst case we spend $\mathcal{O}(nd^5 \operatorname{polylog} n)$ to process r. The compact representations of the streaks take $\mathcal{O}(d^2 \log^3 n)$ space.

Recall that $\mathcal{F}$ contains all atomic strings A such that there is an ε-transitions path from the endpoint of the transition labelled by A to the final state of $T_C(R)$. In the regular expression pattern matching problem, we report all positions r such that r is a witness in $\mathsf{occ}(A, T)$ for some $A \in \mathcal{F}$. In the regular expression membership problem, $T \in L(R)$ if n is a witness for $\mathsf{occ}(A, T)$ for some $A \in \mathcal{F}$. ∎

5 Proof of Theorem 3.8

An important tool in our proof is the framework that allows computing output of a circuit time- and space-efficiently. Before we describe the framework in detail, we provide some notation following [12]. A circuit is a directed acyclic graph with nodes of in-degree 0 or 2. Degree-0 nodes are called inputs and degree-2 nodes are gates. In our application, every node corresponds to a vector from $\mathbb{Z}_p^t$ (i.e. a vector of length t with values in $\mathbb{Z}_p$) indexed from 0 to $t - 1$, for some values of p and t that will be specified later. A vector in $\mathbb{Z}_p^t$ is a singleton if it has at most one non-zero entry.

There are two types of gates: $\boxplus$ and $\boxtimes$ that denote respectively the pointwise addition and vector convolution binary gates, that is $(a \boxplus b)[i] = a[i] + b[i]$ and $(a \boxtimes b)[i] = \sum_{j=0}^{i} a[j] \cdot b[i-j]$. Every gate corresponds to the result of its underlying operation applied to its incoming nodes. We say that a convolution gate with input $a, b \in \mathbb{Z}_p^t$ does not overflow, if for all $i \geq t$, $(a \boxtimes b)[i] = 0$. An element ω is a t-th root of unity in $\mathbb{Z}_p$ iff $\omega^t \equiv 1 \mod p$ but $\omega^s \not\equiv 1 \mod p$ for all $0 < s < t$.

With the definitions at hand, we are ready to state the technique introduced by Lokshtanov and Nederlof [50] for complex numbers and its modular variant discussed by Bringmann [12]:

THEOREM 5.1. (CF. [12, THEOREM 4.2]) *Let p be a prime, $t \geq 1$, and suppose that $\mathbb{Z}_p$ contains a t-th root of the unity, ω. Let C be a circuit over $(\mathbb{Z}_p^t, \boxplus, \boxtimes)$ which takes as an input only singleton constants and outputs a vector $out(C) \in \mathbb{Z}_p^t$. Suppose that no convolution gate overflows. Then given p, t, ω, and $0 \leq x < t$ we can compute $out(C)[x]$ in time $\mathcal{O}(|C| t \operatorname{polylog} p)$ and space $\mathcal{O}(|C| \log p)$.*

For Bringmann's framework to be efficient, one must provide a method to choose p and ω. Bringmann [12] showed two different methods, one requiring the Extended Riemann Hypothesis and the other one resulting in an additional t^ε factor in both time and space [12, Lemma 4.4]. Below we show that one can achieve the bounds of the former method unconditionally.

5.1 Finding primes The goal of this section is to prove the following theorem:

THEOREM 5.2. *There is a procedure that, given y, finds in $\mathcal{O}(y \operatorname{polylog} y)$ arithmetic operations and $\mathcal{O}(\log y)$ space a prime p and ω such that for every $N \le 2^{\mathcal{O}(y \log y)}$, with probability at least $1/2$ the following holds:*

1. *ω is a t-th primitive root of unity in $\mathbb{Z}_p$, for some t satisfying $y \le t = \mathcal{O}(y \operatorname{polylog} y)$;*
2. *$p = \mathcal{O}(y^2 \operatorname{polylog} y)$;*
3. *$p \nmid N$.*

Let B be a constant to be determined later. To compute numbers p and ω we run the following procedure:

1. Set x to be the smallest number such that $\frac{1}{2}\sqrt{x}\log^{-B} x \ge y$ (using binary search);
2. Choose a random $q \in [\frac{1}{2}, 1] \cdot \sqrt{x}\log^{-B} x$;
3. Find a prime p such that $p \le x$ and $p \equiv 1 \bmod q$ (by guessing candidate p and checking all numbers up to $\sqrt{p}$ if they divide p or not);
4. Find a generator g of $\mathbb{Z}_p^*$ (by guessing candidate g and checking if $g^{\frac{p-1}{p'}} \not\equiv 1 \bmod p$ for all prime divisors p' of $p-1$);
5. Set $t = q$ and $\omega = g^{\frac{p-1}{t}}$.

Clearly, $t \mid p-1$ and $\omega = g^{\frac{p-1}{t}}$ is well-defined. As g is a generator, we have that ω is a t-th primitive root of unity in $\mathbb{Z}_p^*$. By the choice of x and p, we have $p \le x = \mathcal{O}(y^2 \operatorname{polylog} y)$ and hence $y \le q = t = \mathcal{O}(y \operatorname{polylog} y)$. In the following lemma we show that with probability at least $7/8$, there are many primes p satisfying $p \le x$ and $p \equiv 1 \bmod q$ and hence we can efficiently find one. Finally, we show how to find the generator g efficiently. Let $\pi(x; q, a) = |\{p \le x : p \equiv a \pmod q\}|$ and $\phi(n) = |\{1 \le a \le n : \gcd(a, n) = 1\}|$.

LEMMA 5.3. *Let q be chosen uniformly at random from $[\frac{1}{2}, 1] \cdot \sqrt{x}\log^{-B} x$. With probability at least $7/8$ we have $\pi(x; q, 1) = \Omega(\sqrt{x}(\log x)^{B-1})$.*

Before we prove the lemma, we remind some number-theory notation and facts related to counting primes. The reader familiar with this area can skip this part. We first remind the definitions of von Mangoldt function $\Lambda(n)$ and Chebyshev functions ψ and ϑ:

$$\psi(x; q, a) = \sum_{\substack{n \le x \\ n \equiv a \pmod q}} \Lambda(n), \qquad \text{where } \Lambda(n) = \begin{cases} \log p & \text{if } n = p^k \text{ for some prime } p \text{ and } k \in Z_+, \\ 0 & \text{otherwise.} \end{cases}$$

$$\vartheta(x; q, a) = \sum_{\substack{p \le x \\ p \equiv a \pmod q}} \log p, \qquad \text{where the summation is over prime numbers } p.$$

By skipping the last two arguments we denote $\vartheta(x) = \sum_{0 \le a < q} \vartheta(x; q, a)$ and similarly for $\psi(x)$.

FACT 5.4. (BY DEFINITION) $\vartheta(x; q, a) \le \pi(x; q, a) \cdot \log x$.

FACT 5.5. (CF. [59, THEOREM 13]) $\psi(x; q, a) \le \vartheta(x; q, a) + \mathcal{O}(\sqrt{x})$.

Proof. Theorem 13 of [59] states that $\psi(x) - \vartheta(x) = \mathcal{O}(\sqrt{x})$, so for completeness we show an adaptation of this property to numbers forming an arithmetic progression.

$$\begin{aligned}\psi(x;q,a) - \vartheta(x;q,a) &= \sum_{\substack{n\le x\\ n\equiv a \bmod q\\ n=p^k,k\ge 1}} \log p - \sum_{\substack{p\le x\\ p\equiv a \bmod q}} \log p\\ &\le \sum_{\substack{n\le x\\ n=p^2}} \log p + \sum_{\substack{n\le x\\ n=p^k,k\ge 3}} \log p\\ &\le \vartheta(\sqrt{x}) + \mathcal{O}(\sqrt[3]{x}\log^2 x) = \mathcal{O}(\sqrt{x})\end{aligned}$$

as $\sum_{\substack{n\le x\\ n=p^2}} \log p = \sum_{p\le\sqrt{x}} \log p = \vartheta(\sqrt{x})$ and finally $\vartheta(x) = x + o(x)$ by [59, (2.29)]. ∎

THEOREM 5.6. (BOMBIERI–VINOGRADOV THEOREM [10]) *For every $A > 0$ there exists $B = B(A) > 0$ such that for every x:*

$$\sum_{q\le\sqrt{x}(\log x)^{-B}} \max_{y\le x} \max_{\gcd(a,q)=1} \left|\psi(y;q,a) - \frac{y}{\phi(q)}\right| = \mathcal{O}(x\log^{-A} x).$$

With all the notation at hand we are ready to prove Lemma 5.3.

Proof. [Proof of Lemma 5.3]

Let $\mathcal{R} = [\frac{1}{2}, 1]\cdot\sqrt{x}\log^{-B} x$ be the range from which we draw q. By choosing $y = x$ and $a = 1$ and summing only over $q \in \mathcal{R}$ we lower bound the left-hand side of Bombieri–Vinogradov theorem obtaining that, for every $A > 0$ there exists $B = B(A) > 0$ such that

$$\sum_{q\in\mathcal{R}} \left|\psi(x;q,1) - \frac{x}{\phi(q)}\right| = \mathcal{O}(x\log^{-A} x). \tag{5.1}$$

Similarly to Markov's inequality, for q chosen uniformly at random from $\mathcal{R}$ we have with probability at least $7/8$:

$$\left|\psi(x;q,1) - \frac{x}{\phi(q)}\right| = \mathcal{O}(\sqrt{x}(\log x)^{-A+B}) \tag{5.2}$$

Indeed, there can be at most $|\mathcal{R}|/8$ numbers q in $\mathcal{R}$ such that $\psi(x;q,1) \ge \frac{8\cdot\Omega(x\log^{-A} x)}{|\mathcal{R}|} = \Omega(\sqrt{x}(\log x)^{-A+B})$ in order not to exceed the right-hand side of (5.1). Let B be the constant from Theorem 5.6 for $A = 1$. Without loss of generality, we assume $B \ge 3 > A = 1$. Now we rewrite (5.2) using properties of $\phi(n), \vartheta(n)$ and $\psi(n)$ and obtain:

$$\pi(x;q,1) = \Omega(\sqrt{x}(\log x)^{B-1})$$

In more detail:

$$\begin{aligned}\frac{x}{\phi(q)} - \psi(x;q,1) &= \mathcal{O}(\sqrt{x}(\log x)^{-A+B}) && \text{from (5.2)}\\ \frac{x}{q} &\le \vartheta(x;q,1) + \mathcal{O}(\sqrt{x}) + \mathcal{O}(\sqrt{x}(\log x)^{-A+B}) && \text{from Fact 5.5 and } \phi(q)\le q\\ \frac{x}{q} &\le \pi(x;q,1)\cdot\log x + \mathcal{O}(\sqrt{x}(\log x)^{-A+B}) && \text{from Fact 5.4 and } B > A\\ \pi(x;q,1) &\ge \sqrt{x}(\log x)^{B-1} - \mathcal{O}(\sqrt{x}(\log x)^{-A+B-1}) && \text{as } q\in\mathcal{R}\\ \pi(x;q,1) &= \Omega(\sqrt{x}(\log x)^{B-1}) && \text{as } A = 1\end{aligned}$$

∎

LEMMA 5.7. *Suppose* $\pi(x;q,1) = \Omega(\sqrt{x}(\log x)^{B-1})$. *With probability at least* 7/8, *in* $\mathcal{O}(\sqrt{x}\log x)$ *arithmetic operations and* $\mathcal{O}(\log x)$ *space we can find such a prime* $p \le x$ *such that* $p \equiv 1 \bmod q$.

Proof. Clearly, we can check if a number n is prime by iterating through all numbers $2, 3, \ldots, \sqrt{n}$ and checking if they are a divisor of n. Let $\mathcal{Q} = \{n \le x : n = 1 \bmod q\}$. Observe that the probability that a number chosen uniformly at random from $\mathcal{Q}$ is prime is at least:

$$\frac{\pi(x;q,1)}{|\mathcal{Q}|} = \frac{\pi(x;q,1)}{x/q} = \Omega\left(\frac{\sqrt{x}(\log x)^{B-1}\sqrt{x}\log^{-B} x}{x}\right) = \Omega\left(\frac{1}{\log x}\right)$$

Hence by checking $\Theta(\log x)$ numbers from $\mathcal{Q}$ we find a prime with probability at least 7/8. ∎

LEMMA 5.8. *With probability at least* 7/8, *we can find a generator* g *of* $\mathbb{Z}_p^*$ *in* $\mathcal{O}(\sqrt{p})$ *arithmetic operations and* $\mathcal{O}(\log p)$ *space.*

Proof. First, we generate the set of all divisors of $p-1$ in $\mathcal{O}(\sqrt{p})$ time by iterating through $2, 3, \ldots, \sqrt{p-1}$ and checking if they are a divisor of $p-1$. By using an auxiliary accumulator we can restrict only to prime divisors, we call this set $\mathcal{D}$. Now we can check if a number g is a generator of $\mathbb{Z}_p^*$ by checking if for every $p' \in \mathcal{D}$, a prime divisor of $p-1$, we have $g^{(p-1)/p'} \not\equiv 1 \bmod p$. Using exponentiating by squaring, this runs in total $\mathcal{O}(\text{polylog}\, p)$ time.

The probability of a random $g \in \{0, \ldots, p-2\}$ to be a generator is $\phi(p-1)/(p-1) = \Omega(1/\log\log p)$, as $\phi(n) = \Omega(n/\log\log n)$ [59, Theorem 15]. Hence by checking $\Theta(\log\log p)$ numbers from $\mathbb{Z}_p^*$ we find a generator with probability at least 7/8. ∎

Finally, as $x \ge y^2$ and $\pi(x;q,1) = \Omega(\sqrt{x}(\log x)^{B-1})$ and $B \ge 3$ we have $\pi(x;q,1) \ge y\log^2 y \ge 8\log N$, as $N \le 2^{\mathcal{O}(y\log y)}$. Because N has at most $\log N$ prime divisors, the probability that the chosen prime p is one of them is at most 1/8. Summing up, there are four events due to which our algorithm can fail:

1. The number q does not satisfy $\pi(x;q,1) = \Omega(\sqrt{x}(\log x)^{B-1})$;
2. We did not find p in the planned number of iterations;
3. The chosen p divides N;
4. We did not find g in the planned number of iterations.

We note that we do not have access to the value of N during the algorithm, so we cannot spot immediately that the chosen p is wrong. When we fail to find g or p in the planned number of iterations we terminate. However, if q is chosen wrongly, we cannot detect it immediately, but then the subsequent steps (choosing p or g) will have a larger probability of failure. To conclude, the overall probability of a failure is at most 1/2 and the running time of the whole procedure is $\mathcal{O}(\sqrt{x}\,\text{polylog}(x)) = \mathcal{O}(y\,\text{polylog}\, y)$. This concludes the proof of Theorem 5.2.

5.2 Walks in a weighted graph. We can finally prove Theorem 3.8. We first describe an algorithm that uses significantly much more time and space than desired, and then improve it. We compute arrays C_k for $k \in \{0, \ldots, \lceil \log x \rceil\}$ indexed by nodes $u, v \in V(G)$, where $C_k[u,v]$ is a bit-vector of length $x+1$ such that:

1. $C_k[u,v][d] = 1$ implies that there exists a walk of weight d from u to v;
2. For every $d \le 2^k$, if there exists a walk of weight d from u to v in G, we have $C_k[u,v][d] = 1$.

In other words, C_k contains the information about all walks of weight at most 2^k in G and possibly some other walks of weight at most x. We initialize the array C_0 in the following way: $\forall_{u \in V(G)} C_0[u,u][0] = 1$ and $C_0[u,v][d] = 1$ if there is an edge from u to v of weight $0 \le d \le x$. If there are 0-weight edges in G, we first need to compute their transitive closure in G in $\mathcal{O}(|V(G)|^3)$ time and mark in C_0 all walks of total weight 0 or 1 in G. We define $(\text{OR}, \text{CONVOLUTION}_x)$-product of matrices consisting of bit-vectors, truncated to the first $x+1$ positions:

$$\forall_{\substack{u,v\in V(G)\\ d\in\{0,\ldots,x\}}} (A\odot B)[u,v][d] := \bigvee_{\substack{w\in V(G)\\ i\in\{0,\ldots,d\}}} A[u,w][i]\wedge B[w,v][d-i]$$

Now we compute the consecutive arrays C_k by repeated application of $(\text{OR}, \text{CONVOLUTION}_x)$-product as follows:

$$C_{k+1} := C_k \odot C_0 \odot C_k$$

Both invariants for the array C_k follow by inductive reasoning, as every walk of weight d can be split into three parts of weights d_1, d_2, d_3 where $d_1, d_3 \leq d/2$ and the middle part consists of a single edge (recall that for each edge (u, v) of weight $0 \leq d \leq x$ we have $C_0[u, v][d] = 1$). Then, for the given nodes v_1, v_2 we can return the entry $C_{\lceil \log x\rceil}[v_1, v_2][x]$.

This approach runs in $\mathcal{O}(|V(G)|^2 x)$ space and $\mathcal{O}(|V(G)|^3 + |V(G)|^2 x \operatorname{polylog} x)$ time when we use the fast Fourier transform at every step. Observe that this complexity matches the time and space bounds stated in Theorem 3.8 for the case when $x = \mathcal{O}(|V(G)|)$. Hence, we focus on the case when $x = \Omega(|V(G)|)$.

Saving space with circuits. To save both time and space, we will use circuits and the framework of Theorem 5.1. In order to use this framework, we need to modify our algorithm in various aspects. First, in $\mathcal{O}(x \operatorname{polylog} x)$ time we find the appropriate values of p, t, ω using Theorem 5.2 from Section 5.1 for $y = \Theta(x)$ that will be defined precisely later. Then $t = \mathcal{O}(x \operatorname{polylog} x)$. Instead of bit-vectors as entries of the array C_k, we operate on vectors from $\mathbb{Z}_p^t$ over $(\mathbb{Z}_p, +, \cdot)$. In other words we use $\boxplus$ (addition in $\mathbb{Z}_p^t$) instead of Boolean OR and $\boxtimes$ (the standard $(+, \cdot)$-convolution modulo p) instead of the Boolean $(\vee, \wedge)$-convolution. Then the $\odot$ product between $A \odot B$ becomes $(A \odot B)[u, v] = \boxplus_{w\in V(G)} A[u, w] \boxtimes B[w, v]$. With $\odot$ defined this way, $C_k[u, v][d]$ counts modulo p walks from u to v of weight d, possibly counting one walk more than once — we analyse these values in detail at the end of the proof.

Now we describe the construction of the circuit. To simplify the presentation, we work with multi-ary addition gates $\boxplus^*$ which can be replaced with binary gates $\boxplus$ at the expense of doubling the total size of the circuit.

1. For every $k \in \{0, \ldots, \lceil \log x\rceil\}$ and $u, v \in V(G)$ we create a $\boxplus^*$ gate $C_k[u, v]$;
2. For every node $v \in V(G)$ we create a singleton constant V_v with only the 0-th entry set to 1, connected to the $\boxplus^*$ gate $C_0[v, v]$;
3. For every edge $(u, v, d) \in E(G)$ from node u to v of weight d, we create a singleton constant $E_{u,v,d}$ with only the d-th entry set to 1, connected to the $\boxplus^*$ gate $C_0[u, v]$;
4. As $(A \odot B)[u, v] = \boxplus^*_{w\in V(G)} A[u, w] \boxtimes B[w, v]$, we can implement every product $X = A \odot B$ with $|V(G)|^3$ gates $X^w[u, v] := A[u, w] \boxtimes B[w, v]$ and $|V(G)|^2$ gates $X[u, v] := \boxplus^*_{w\in V(G)} X^w[u, v]$. For every $k > 0$, it holds $C_k = C_{k-1} \odot C_0 \odot C_{k-1}$, so we need an intermediate product $C'_k := C_{k-1} \odot C_0$ and then $C_k := C'_k \odot C_{k-1}$.

The above construction gives a circuit on $\mathcal{O}(|E(G)| + |V(G)|^3 \log x)$ gates with singleton constants, out of which we need to output if $C_{\lceil \log x\rceil}[v_1, v_2][x] > 0$. However, we still cannot use the framework from Theorem 5.1, as we cannot guarantee that there are no convolution gate overflows. Indeed, if there are edges of weight almost x, we would obtain walks of weight x^2. In the following paragraph we show a refined construction in which we have more control on the maximum weight of walks considered in the k-th step of the algorithm.

Refined construction. Let ε be a value to be determined precisely later. Instead of the arrays C_k, we will compute arrays D_k that, informally, describe all walks of total weight at most $(1 + \varepsilon)^k$, some walks of weight $d \leq (1 + \varepsilon)^k \cdot (1 + \varepsilon)^{2k\cdot\log(1+\varepsilon)}$ and no longer walks. As we operate on values modulo p, let $D'_k[u, v][d]$ be the value of $D_k[u, v][d]$ if computed exactly, without taking modulo p at every step. Formally, for every $k = \{0, \ldots, \lceil \log_{1+\varepsilon} x\rceil\}$ we have:

1. $D'_k[u, v][d] > 0$ implies that there exists a walk of weight d from u to v;
2. For each $d \leq (1 + \varepsilon)^k$, if there exists a walk of weight d from u to v in G, we have $D'_k[u, v][d] > 0$;
3. $D_k[u, v][d] = 0$ for all $d > (1 + \varepsilon)^k \cdot (1 + \varepsilon)^{2k\cdot\log(1+\varepsilon)}$.

The array D_0 stores all walks of total weight at most 1, that is $D_0[u,v][d] = 1$ iff $d \in \{0,1\}$ and there is a walk from u to v of total weight d. It can be computed in $\mathcal{O}(|V(G)|^3)$ time as C_0, by first computing all pairs of nodes connected by a walk of 0-weight edges. Now we show how to obtain the array D_k. Again, every walk of weight d can be cut into three parts of total weights d_1, d_2, d_3 where $d_1, d_3 \leq d/2$ and the middle part consist of a single edge. We need to control the total weight of the walk, so we will iterate over all possible base-$(1+\varepsilon)$ logarithms of weights of the three parts k_1, k_2, k_3. For all possible values d_1, d_2, d_3 such that $d_1 + d_2 + d_3 \leq (1+\varepsilon)^k$, we process the triple k_1, k_2, k_3 where $\forall_{i \in \{1,2,3\}} (1+\varepsilon)^{k_i - 1} < d_i \leq (1+\varepsilon)^{k_i}$. Then, from the definition of arrays D_k, every walk of weight d_i will be included in D_{k_i}. For single edges of particular weight, let B_k describe all pairs of nodes connected by an edge of weight at most $(1+\varepsilon)^k$: $B_k[u,v][d] = 1$ iff $d \leq (1+\varepsilon)^k$ and there is an edge of weight d from u to v. Note that $B_k = B_{k-1} \boxplus F_k$ where F_k describes all edges of weight from $\left((1+\varepsilon)^{k-1}, (1+\varepsilon)^k\right]$. We restrict only to triples k_1, k_2, k_3 satisfying both:

$$(a) \quad (1+\varepsilon)^{k_1 - 1} + (1+\varepsilon)^{k_2 - 1} + (1+\varepsilon)^{k_3 - 1} \leq (1+\varepsilon)^k \tag{5.3}$$

$$(b) \quad 2 \cdot (1+\varepsilon)^{\max\{k_1, k_3\} - 1} \leq (1+\varepsilon)^k \tag{5.4}$$

and call such triples *k-good*. Then we compute D_k in the following way:

$$D_k := \boxplus^*_{k\text{-good } k_1,k_2,k_3} D_{k_1} \odot B_{k_2} \odot D_{k_3} \tag{5.5}$$

Now we show that all the invariants about D_k are satisfied. Clearly there are no false-positive entries in the array. We never miss a walk of weight at most $(1+\varepsilon)^k$, as the condition (a) filters out the triples k_i contributing only the walks of total weight larger than $(1+\varepsilon)^k$. The condition (b) guarantees that the first and third part of the walk have weight at most $\frac{1}{2}(1+\varepsilon)^k$. In the following lemma we show that we also never construct walks of too large weight.

LEMMA 5.9. *For every k and every k-good triple k_1, k_2, k_3, the largest weight of a walk in $D_{k_1} \odot B_{k_2} \odot D_{k_3}$ is at most $(1+\varepsilon)^k \cdot (1+\varepsilon)^{2k \cdot \log(1+\varepsilon)}$.*

Proof. Induction on k. Without loss of generality assume $k_1 \geq k_3$ and then the walks in $D_{k_1} \odot B_{k_2} \odot D_{k_3}$ have total weight at most:

$$\begin{aligned}
&\leq (1+\varepsilon)^{k_1} \cdot (1+\varepsilon)^{2k_1 \cdot \log(1+\varepsilon)} + (1+\varepsilon)^{k_2} + (1+\varepsilon)^{k_3} \cdot (1+\varepsilon)^{2k_3 \cdot \log(1+\varepsilon)} \\
&\leq [(1+\varepsilon)^{k_1} + (1+\varepsilon)^{k_2} + (1+\varepsilon)^{k_3}] \cdot (1+\varepsilon)^{2k_1 \cdot \log(1+\varepsilon)} \\
&\leq (1+\varepsilon)^{k+1} \cdot (1+\varepsilon)^{2k_1 \cdot \log(1+\varepsilon)} \text{ (from the condition (a))} \qquad (5.6)
\end{aligned}$$

Now we use the condition (b) of a good k-triple:

$$\begin{aligned}
(1+\varepsilon)^k &\geq 2 \cdot (1+\varepsilon)^{k_1 - 1} && \text{apply } \log_{1+\varepsilon}(\cdot) \text{ and rearrange} \\
k - k_1 &\geq \log_{1+\varepsilon} 2 - 1 && \text{multiply both sides by } 2 \cdot \log_2(1+\varepsilon) \\
2 \cdot \log_2(1+\varepsilon) \cdot (k - k_1) &\geq 2 \cdot (1 - \log_2(1+\varepsilon)) > 1 && \text{as } 1 + \varepsilon < \sqrt{2} \\
2k \cdot \log_2(1+\varepsilon) &\geq 2k_1 \cdot \log_2(1+\varepsilon) + 1
\end{aligned}$$

Applying the above inequality to (5.6) concludes the inductive step. ∎

Setting $\varepsilon = 1/\log x$, as $\log(1+\varepsilon) > \varepsilon$ we obtain that there are

$$r = \lceil \log_{1+\varepsilon} x \rceil \leq 1 + \frac{\log x}{\log(1+\varepsilon)} = \mathcal{O}\left(\frac{\log x}{\varepsilon}\right) = \mathcal{O}(\log^2 x)$$

arrays D_k to compute. As $\log(1+\varepsilon) < 2\varepsilon$, the largest possible weight is bounded from above by

$$\begin{aligned}
&(1+\varepsilon)^r \cdot (1+\varepsilon)^{2r \cdot \log(1+\varepsilon)} \leq (1+\varepsilon)^{r \cdot (4\varepsilon + 1)} \leq (1+\varepsilon)^{(\log_{1+\varepsilon} x + 1) \cdot (4\varepsilon + 1)} \\
&\leq x \cdot x^{4\varepsilon} \cdot ((1+\varepsilon)^{\varepsilon})^4 \cdot 2 = \mathcal{O}(x \cdot 2^{4 \cdot \log x \cdot \frac{1}{\log x}}) = \mathcal{O}(x).
\end{aligned}$$

Hence, the appropriate choice of $y = \Theta(x)$ guarantees that no convolution gate overflows.

Now we estimate the size of the constructed circuit. We compute $r = \mathcal{O}(\log^2 x)$ arrays D_k. For each of them we process $\mathcal{O}(r^3)$ k-good triples which perform two $\odot$ products each. Every $\odot$ product introduces $\mathcal{O}(|V(G)|^3)$ $\boxplus$ and $\boxtimes$ gates. Hence the total number of gates is $\mathcal{O}(|E(G)| + |V(G)|^3 \cdot \operatorname{polylog} x)$.

Finally we discuss the properties of the values computed in D_r and all the intermediate gates. Recall that $D'_k[u, v][d]$ is the value of $D_k[u, v][d]$ if computed exactly, without taking modulo p at every step. For each $d \le (1 + \varepsilon)^k$, every walk between u and v of weight d contributes at least 1 to $D'_k[u, v][d]$. Notice that such walk may contribute more than 1, as it can be cut into three parts in many ways, for different triples k_1, k_2, k_3. As no walks of weight different from d contribute to $D'_k[u, v][d]$, there is a walk from u and v of weight d iff $D'_k[u, v][d] > 0$. However, while computing the arrays D_k we operate in $\mathbb{Z}_p$, so we might have false negative error if $p \mid D'_k[u, v][d]$. In Theorem 5.2 we include such situations in the probability of failure (we fail if $p \mid N$, where $N = D'_r[u, v][d]$), but we need to ensure that $D'_r[u, v][d]$ never exceeds $2^{\mathcal{O}(t \log t)}$.

LEMMA 5.10. *Suppose we execute the above algorithm up to the r-th matrix D_r in $\mathbb{Z}$, not applying modulo p in every gate. Then all the obtained values are bounded by $2^{\mathcal{O}(x \log x)}$.*

Proof. Recall that $\varepsilon = \frac{1}{\log x}$, $r = \lceil \log_{1+\varepsilon} x \rceil$, we operate on vectors with $t = \mathcal{O}(x \operatorname{polylog} x)$ entries, and the convolutions do not overflow as the result always fits in the first $y = \mathcal{O}(x)$ elements of the vectors. Let $f(k)$ be a monotonous function that upper bounds the values in D'_k and $g = |V(G)|$. Observe that a single product $A \odot B$ of matrices with entries bounded by respectively $a_{\max}$ and $b_{\max}$ results in a matrix with entries bounded by $g \cdot y \cdot a_{\max} \cdot b_{\max}$. Hence, as values in B_{k_2} are 0 or 1, from Equation (5.5) we have the following bound:

$$f(k) \le \sum_{k-\text{good } k_1,k_2,k_3} f(k_1) \cdot (y \cdot g)^2 \cdot f(k_3) \le k^3 (y \cdot g)^2 \cdot f^2(k_{\max}) \le W f^2(k_{\max})$$

where $k_{\max}$ is the largest possible value of k_i that can be a part of a k-good triple and $W = r^3 \cdot (y \cdot g)^2 = \mathcal{O}(x^5)$ as $y = \mathcal{O}(x)$ and we consider the case when $x = \Omega(g)$. From Equation (5.4) we have:

$$\begin{aligned} 2 \cdot (1 + \varepsilon)^{k_{\max} - 1} &< (1 + \varepsilon)^k \\ k_{\max} + \log_{1+\varepsilon} 2 - 1 &< k \end{aligned}$$

As arguments of f are integers, it would be more convenient to write $k_{\max} \le k - c$ where $c = \lceil \log_{1+\varepsilon} 2 - 1 \rceil \ge \log_{1+\varepsilon} 2 - 1$. As f is monotonous, we have:

$$f(k) \le \begin{cases} W \cdot f^2(k - c), & k \ge c \\ W, & k < c \end{cases}$$

which solves by induction to $f(k) < W^{2^{\lceil \frac{k}{c} \rceil + 1} - 1}$. Then, as $W = \mathcal{O}(x^5)$ we have:

$$f(r) < W^{2^{\lceil \frac{r}{c} \rceil + 1} - 1} < W^{\mathcal{O}(2^{r/c})} < 2^{\mathcal{O}(2^{r/c} \log x)}$$

Finally, we show that $r/c \le \log x + \mathcal{O}(1)$.

$$\begin{aligned} \frac{r}{c} &\le \frac{\log_{1+\varepsilon} x + 1}{\log_{1+\varepsilon} 2 - 1} = \frac{\frac{\log x}{\log(1+\varepsilon)} + 1}{\frac{\log 2}{\log(1+\varepsilon)} - 1} = \frac{\log x + \log(1 + \varepsilon)}{1 - \log(1 + \varepsilon)} \le \frac{\log x + 2\varepsilon}{1 - 2\varepsilon} \quad (\text{as } \log(1 + \varepsilon) < 2\varepsilon) \\ &= \frac{\log^2 x + 2}{\log x - 2} = \log x + \mathcal{O}(1) \end{aligned}$$

Combining that with the above bound on $f(r)$ we obtain:

$$f(r) < 2^{\mathcal{O}(2^{r/c} \log x)} \le 2^{\mathcal{O}(2^{\log x + \mathcal{O}(1)} \log x)} = 2^{\mathcal{O}(x \log x)}$$

which gives the desired bound on the obtained values. ∎

Hence we can apply Theorem 5.1 to the circuit computing $D_r[u, v]$ for the values p, t, ω from Theorem 5.2. The running time of the algorithm is $\mathcal{O}(|C|t \operatorname{polylog} p) = \mathcal{O}((|E(G)| + |V(G)|^3)x \operatorname{polylog} x)$ as $|C| = \mathcal{O}(|E(G)| + |V(G)|^3 \cdot \operatorname{polylog} x)$, $t = \mathcal{O}(x \operatorname{polylog} x)$, $p = \mathcal{O}(y^2 \operatorname{polylog} y)$ and $y = \Theta(x)$. The space complexity is bounded by $\mathcal{O}(|C| \log p) = \mathcal{O}((|E(G)| + |V(G)|^3) \operatorname{polylog} x)$. This concludes the proof of Theorem 3.8.

References

[1] Amir Abboud and Karl Bringmann. Tighter connections between formula-SAT and shaving logs. In *2018 International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.8.

[2] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.

[3] Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013. doi:10.1016/j.tcs.2012.12.028.

[4] Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *2016 IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 457–466. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.56.

[5] Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *2021 International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 119:1–119:17, 2021. doi:10.4230/LIPIcs.ICALP.2021.119.

[6] Philip Bille. New algorithms for regular expression matching. In *2006 International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 4051 of *Lecture Notes in Computer Science*, pages 643–654. Springer, 2006. doi:10.1007/11786986_56.

[7] Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(3):486–496, 2008. doi:10.1016/j.tcs.2008.08.042.

[8] Philip Bille and Mikkel Thorup. Faster regular expression matching. In *2009 International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 171–182. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-02927-1_16.

[9] Philip Bille and Mikkel Thorup. Regular expression matching with multi-strings and intervals. In *2010 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1297–1308, 2010. doi:10.1137/1.9781611973075.104.

[10] Enrico Bombieri. *Le grand crible dans la théorie analytique des nombres*. Number 18 in Astérisque. Société mathématique de France, 1974.

[11] Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Transaction on Algorithms*, 10(4):22:1–22:12, 2014. doi:10.1145/2635814.

[12] Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *2017 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1073–1084. SIAM, 2017. doi:10.1137/1.9781611974782.69.

[13] Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *2017 IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 307–318, 2017. doi:10.1109/FOCS.2017.36.

[14] Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Information Processing Letters*, 101(2):53–54, 2007. doi:https://doi.org/10.1016/j.ipl.2006.08.002.

[15] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. Dictionary matching in a stream. In *2015 European Symposium on Algorithms (ESA)*, volume 9294 of *LNCS*, pages 361–372, 2015. doi:10.1007/978-3-662-48350-3_31.

[16] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k-mismatch problem revisited. In *2016 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2039–2052, 2016. doi:10.1137/1.9781611974331.ch142.

[17] Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. In *2019 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1106–1125, 2019. doi:10.1137/1.9781611975482.68.

[18] Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *2002 ACM Symposium on Theory of Computing (STOC)*, page 592–601, 2002. doi:10.1145/509907.509992.

[19] Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Streaming periodicity with mismatches. In *2017 International Conference on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 42:1–42:21, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.42.

[20] Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Periodicity in data streams with wildcards. In *2018 International Computer Science Symposium in Russia (CSR)*, pages 90–105, 2018. doi:10.1007/978-3-319-90530-3_9.

[21] Funda Ergün, Hossein Jowhari, and Mert Sağlam. Periodicity in streams. In *2010 International Conference on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 545–559, 2010. doi:10.1007/978-3-642-15369-3_41.

[22] Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16:109–114, 1965.

[23] Michael J. Fischer and Michael S. Paterson. String-matching and other products. Technical report, Massachusetts Institute of Technology, USA, 1974.

[24] N. François, F. Magniez, M. de Rougemont, and O. Serre. Streaming property testing of visibly pushdown languages. In *2016 European Symposium on Algorithms (ESA)*, volume 57 of *LIPIcs*, pages 43:1–43:17, 2016. doi:10.4230/LIPIcs.ESA.2016.43.

[25] Zvi Galil. Open problems in stringology. In *Combinatorial Algorithms on Words*, pages 1–8, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

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

[27] Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *2016 Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 65 of *LIPIcs*, pages 18:1–18:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.18.

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

[29] Moses Ganardi, Danny Hucke, and Markus Lohrey. Sliding window algorithms for regular languages. In *2018 International Conference on Language and Automata Theory and Applications (LATA)*, volume 10792, pages 26–35, 2018. doi:10.1007/978-3-319-77313-1_2.

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

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

[32] Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Mining sequential patterns with regular expression constraints. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(3):530–552, 2002. doi:10.1109/TKDE.2002.1000341.

[33] Paweł Gawrychowski, Oleg Merkurev, Arseny M. Shur, and Przemysław Uznański. Tight tradeoffs for real-time approximation of longest palindromes in streams. *Algorithmica*, 81(9):3630–3654, 2019. doi:10.1007/s00453-019-00591-8.

[34] Paweł Gawrychowski, Jakub Radoszewski, and Tatiana Starikovskaya. Quasi-periodicity in streams. In *2019 Symposium on Combinatorial Pattern Matching (CPM)*, volume 128 of *LIPIcs*, pages 22:1–22:14, 2019. doi:10.4230/LIPIcs.CPM.2019.22.

[35] Paweł Gawrychowski and Tatiana Starikovskaya. Streaming dictionary matching with mismatches. In *2019 Symposium on Combinatorial Pattern Matching (CPM)*, volume 128 of *LIPIcs*, pages 21:1–21:15, 2019. doi:10.4230/LIPIcs.CPM.2019.21.

[36] Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. The streaming k-mismatch problem: tradeoffs between space and total time. In *2020 Symposium on Combinatorial Pattern Matching (CPM)*, volume 161 of *LIPIcs*, pages 15:1–15:15, 2020. doi:10.4230/LIPIcs.CPM.2020.15.

[37] Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams. In *2018 International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 65:1–65:16, 2018. doi:10.4230/LIPIcs.ICALP.2018.65.

[38] Shay Golan, Tsvi Kopelowitz, and Ely Porat. Streaming pattern matching with d wildcards. *Algorithmica*, 81(5):1988–2015, 2019. doi:10.1007/s00453-018-0521-7.

[39] Shay Golan and Ely Porat. Real-time streaming multi-pattern search for constant alphabet. In *2017 European Symposium on Algorithms (ESA)*, volume 107 of *LIPIcs*, pages 41:1–41:15, 2017. doi:10.4230/LIPIcs.ESA.2017.41.

[40] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:https://doi.org/10.1006/jcss.2000.1727.

[41] Piotr Indyk. Faster algorithms for string matching problems: matching the convolution bound. In *1998 IEE Symposium on Foundations of Computer Science (FOCS)*, page 166, 1998. doi:10.1109/SFCS.1998.743440.

[42] Theodore Johnson, Shan Muthu Muthukrishnan, and Irina Rozenbaum. Monitoring regular zxpressions on out-of-order streams. In *2007 IEEE International Conference on Data Engineering (ICDE)*, pages 1315–1319, 2007.

doi:10.1109/ICDE.2007.369001.

[43] Adam Kalai. Efficient pattern-matching with don't cares. In *2002 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, page 655–656, 2002.

[44] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. Proton: multitouch gestures as regular expressions. In *2012 Conference on Human Factors in Computing Systems (CHI)*, page 2885–2894, 2012. doi:10.1145/2207676.2208694.

[45] Stephen C. Kleene. *Representation of events in nerve nets and finite automata.* RAND Corporation, Santa Monica, CA, 1951.

[46] Tomasz Kociumaka, Ely Porat, and Tatiana Starikovskaya. Small space and streaming pattern matching with k edits. *CoRR*, abs/2106.06037, 2021. To appear in *2021 IEEE Symposium on Foundations of Computer Science (FOCS).*

[47] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan S. Turner. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, page 339–350, 2006. doi:10.1145/1159913.1159952.*

[48] LeetCode. Problem 139. Word break. https://leetcode.com/problems/word-break/.

[49] Quanzhong Li and Bongki Moon. Indexing and querying XML data for regular path expressions. In 2001 International Conference on Very Large Data Bases (VLDB)*, page 361–370, 2001.*

[50] Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In ACM Symposium on Theory of Computing (STOC)*, pages 321–330, 2010. doi:10.1145/1806689.1806735.*

[51] Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. SIAM Journal on Computing*, 43(6):1880–1905, 2014. doi:10.1137/130926122.*

[52] Oleg Merkurev and Arseny M. Shur. Searching long repeats in streams. In 2019 Symposium on Combinatorial Pattern Matching (CPM)*, volume 128 of* LIPIcs*, pages 31:1–31:14, 2019. doi:10.4230/LIPIcs.CPM.2019.31.*

[53] Oleg Merkurev and Arseny M. Shur. Searching runs in streams. In 2019 Symposium on String Processing and Information Retrieval (SPIRE)*, volume 11811 of* LNCS*, pages 203–220, 2019. doi:10.1007/978-3-030-32686-9_15.*

[54] Makoto Murata. Extended path expressions of XML. In 2001 ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, page 126–137, 2001. doi:10.1145/375551.375569.*

[55] Eugene W. Myers. A four Russians algorithm for regular expression pattern matching. Journal of the ACM*, 39(2):432–448, April 1992. doi:10.1145/128749.128755.*

[56] Gonzalo Navarro and Mathieu Raffinot. Fast and simple character classes and bounded gaps pattern matching, with application to protein searching. In 2001 International Conference on Computational Biology (RECOMB)*, page 231–240, 2001. doi:10.1145/369133.369220.*

[57] Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In 2009 Symposium on Foundations of Computer Science (FOCS)*, pages 315–323, 2009. doi:10.1109/FOCS.2009.11.*

[58] Jakub Radoszewski and Tatiana Starikovskaya. Streaming k*-mismatch with error correcting and applications.* Journal of Information and Computation*, 271:104513, 2020. doi:10.1016/j.ic.2019.104513.*

[59] John Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. Illinois Journal of Mathematics*, 6(1):64 – 94, 1962. doi:10.1215/ijm/1255631807.*

[60] Philipp Schepper. Fine-grained complexity of regular expression pattern matching and membership. In 2020 European Symposium on Algorithms (ESA)*, volume 173 of* Leibniz International Proceedings in Informatics (LIPIcs)*, pages 80:1–80:20, 2020. doi:10.4230/LIPIcs.ESA.2020.80.*

[61] Tatiana Starikovskaya. Communication and streaming complexity of approximate pattern matching. In 2017 Symposium on Combinatorial Pattern Matching (CPM)*, volume 78 of* LIPIcs*, pages 13:1–13:11, 2017. doi:10.4230/LIPIcs.CPM.2017.13.*

[62] Ken Thompson. Programming techniques: regular expression search algorithm. Communication of ACM*, 11(6):419–422, 1968. doi:10.1145/363347.363387.*

[63] Daniel Tunkelang. Retiring a great interview problem. https://thenoisychannel.com/2011/08/08/retiring-a-great-interview-problem/, 2011.

[64] Fang Yu, Zhifeng Chen, Yanlei Diao, T. V. Lakshman, and Randy H. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In 2006 Symposium on Architecture For Networking And Communications Systems*, pages 93–102, 2006. doi:10.1145/1185347.1185360.*