Suffix Random Access via Function Inversion: A Key for Asymmetric Streaming String Algorithms

Panagiotis Charalampopoulos
King's College London, UK

Taha El Ghazi
ENS Paris, Université PSL, France

Jonas Ellert
CWI Amsterdam, The Netherlands

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland

Tatiana Starikovskaya
ENS Paris, Université PSL, France

Abstract

Many string processing problems can be phrased in the streaming setting, where the input arrives symbol by symbol and we have sublinear working space. The area of streaming algorithms for string processing has flourished since the seminal work of Porat and Porat [FOCS 2009].

Unfortunately, problems with efficient solutions in the classical setting often do not admit efficient solutions in the streaming setting. As a bridge between these two settings, Saks and Seshadhri [SODA 2013] introduced the *asymmetric streaming model* (see also [Andoni, Krauthgamer, and Onak; FOCS 2010]). Here, one is given read-only access to a (typically short) reference string R of length m, while a (typically long) text T arrives as a stream.

We provide a generic technique to reduce fundamental string problems in the asymmetric streaming model to the online read-only model, lifting several existing algorithms and generally improving upon the state of the art. Most notably, we obtain asymmetric streaming algorithms for *exact* and *approximate pattern matching* (under both the Hamming and edit distances), and for *relative Lempel–Ziv compression*, a popular scheme for measuring and exploiting redundancy in repetitive text collections.

At the heart of our approach lies a novel tool that facilitates efficient computation in the asymmetric streaming model: the *suffix random access data structure*. In its simplest variant, it maintains constant-time random access to the longest suffix of (the seen prefix of) T that occurs in R. Let τ be a parameter that denotes the size of the data structure. A straightforward approach maintains the data structure in $\mathcal{O}(m/\tau)$ time per arriving symbol of T.

We drastically improve this tradeoff and reveal fundamental barriers via a bidirectional reduction between suffix random access and *function inversion*, a central problem in cryptography:

- By leveraging Fiat and Naor's function inversion data structure [SIAM J. Comput. 2000], we achieve $\tilde{\mathcal{O}}(1 + m^3/\tau^6)$ update time.[1] In particular, for $\tau = \sqrt{m}$, we obtain $\tilde{\mathcal{O}}(1)$ update time, improving over the $\Omega(\sqrt{m})$ bound of the straightforward solution.
- We establish an unconditional $\tilde{\Omega}(m/\tau^3)$ lower bound on the update time. Additionally, we show that achieving update time $o(m^3/\tau^7)$ would imply a breakthrough in function inversion.

On the way to our upper bound, we propose a variant of the string synchronizing sets ([Kempa and Kociumaka; STOC 2019]) with a local sparsity condition that, as we show, admits an efficient streaming construction algorithm. We believe that our framework and techniques will find broad applications in the development of small-space string algorithms.

[1] The $\tilde{\mathcal{O}}(\cdot)$ and $\tilde{\Omega}(\cdot)$ notations suppress, respectively, $\log^{\mathcal{O}(1)} N$ and $1/\log^{\mathcal{O}(1)} N$ factors, where N is the input-size.

53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026).
Editors: Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis;
Article No. 55; pp. 55:1–55:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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

Keywords and phrases streaming algorithms, function inversion, string algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.55

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2604.19371

Funding *Taha El Ghazi, Jonas Ellert, Tatiana Starikovskaya* were partially funded by grant ANR20-CE48-0001 from the French National Research Agency. Research visits during which some of the presented ideas were conceived were funded by a Royal Society International Exchanges Award.
Jonas Ellert: Partially supported by the ERCIM "Alain Bensoussan" Fellowship Programme.
Paweł Gawrychowski: Partially supported by the Polish National Science Centre grant number 2023/51/B/ST6/01505.

1 Introduction

The streaming model is designed to capture problems in which we need to efficiently process very large amounts of incoming data. The common assumption is that we do not have enough space to store the input. Ideally, we would like to use space polylogarithmic in the size of the data, perhaps at the expense of introducing randomization or returning approximate results (especially if it is provably impossible to return exact results). Many problems concerning strings (sequences of symbols) can be naturally expressed in the streaming model. While string algorithms in the classical setting have been studied intensively for many decades, a systematic study of algorithmic problems on strings in the streaming setting was initiated relatively recently by a seminal work of Porat and Porat [88] on exact and approximate pattern matching. It was followed by a flurry of results for pattern matching [13, 16, 26–28, 35, 58, 60–63, 76, 90, 95], repetition detection [37–39, 56, 57, 59, 83, 84], and formal language recognition [6, 8, 9, 42, 48–54, 81] in the streaming model.

However, the streaming model is very restrictive, and some natural string processing problems do not admit sublinear-space streaming algorithms. For example, Bathie, Charalampopoulos, and Starikovskaya [7] showed that this is the case for longest common substring and circular pattern matching. On the other hand, both of these problems do admit efficient solutions in the read-only setting [7, 78], where one is given constant-time random access to the input and does not need to account for the space required to store it.

This suggests that one should seek a computational model (i) relevant to applications in which we need to process long sequences of incoming symbols, and (ii) powerful enough to allow for non-trivial space-efficient solutions. A promising option is the *asymmetric streaming* model proposed by Saks and Seshadhri [94] (see also the work of Andoni, Krauthgamer, and Onak [3]). In this model, the input typically consists of a short read-only string that we are allowed to preprocess and a long streaming string that arrives symbol by symbol. The algorithm must only account for any extra space used while processing T, that is, the space required to store the read-only string is not counted toward the space complexity. So far, existing results in this model consider edit distance and longest common subsequence computation [24, 80, 93, 94]. We thus ask:

Which other fundamental string processing problems admit efficient solutions in the asymmetric streaming model?

Towards answering this question, we introduce a novel tool for string algorithms in the asymmetric streaming model: the *suffix random access data structure*. We demonstrate that this data structure serves as a unifying primitive for asymmetric streaming string processing. In particular, we show that:

(a) It enables a generic reduction that transforms online read-only algorithms to asymmetric streaming algorithms for a wide class of pattern matching problems. In particular, this reduction yields the first *deterministic online* algorithms for exact and approximate pattern matching (under both the Hamming and edit distances) with strongly sublinear space usage on top of read-only access to P (without random access to T).

(b) It leads to the first efficient asymmetric streaming algorithm for relative Lempel–Ziv factorization [79], a popular compression scheme (especially for repetitive text collections).

This allows deterministically processing a single reference string (for pattern matching or compression) and d concurrently arriving text streams in $o(md)$ space, which was previously not possible.

1.1 Suffix Random Access Data Structures

In asymmetric streaming, we have read-only access to a *reference* string R of length m. Then, a *text* string T of length n arrives as a stream. Ideally, we would like to provide access to the last $\Theta(m)$ symbols of the text: this would immediately facilitate the design of asymmetric streaming algorithms for an abundance of problems. For example, if we want to locate exact occurrences of the reference in the text, then we indeed only need to access the last m symbols to determine if R occurs as a suffix of (the seen prefix of) T. However, maintaining m arbitrary symbols is impossible in $o(m)$ space. To overcome this, we observe that, in many applications, it suffices to have access to the longest suffix of the text that is "similar" to the reference string. A trivial example is the case in which the alphabets of the reference and the text are disjoint: for exact pattern matching, it would then suffice to store the empty suffix.

Formally, we introduce the notion of a $(k-1)$-*error suffix random access data structure*, parametrized by an integer $k \geq 1$, that supports random access to a suffix of T at least as long as the longest suffix of T that can be decomposed into $k-1$ single symbols and k substrings of R.

Upper Bounds for Suffix Random Access

As a warm-up, we use read-only constant-space pattern matching [30] to obtain a simple 0-error suffix random access data structure that uses $\mathcal{O}(\tau)$ space, has update time[2] $\mathcal{O}(m/\tau)$, and has constant query time:

▶ **Lemma 1.** *Let $\tau \in [1\,..\,m]$ be a parameter. There is a streaming 0-error random access data structure with space complexity $\mathcal{O}(\tau)$, worst-case update time $\mathcal{O}(m/\tau)$, and query time $\mathcal{O}(1)$. The data structure is deterministic and does not require any preprocessing.*

Note that the product of the space and the update time is linear in m. This raises a natural question about 0-error suffix random access data structures:

Is there a data structure with sublinear product of space and update time that has (near-)constant query time?

[2] Throughout this work, we use *update time* for the time spent per arriving symbol of T and *query time* for the time required for randomly accessing symbols in the supported suffix of the text.

We answer this question affirmatively: for every constant $\epsilon > 0$ and for every positive integer $\tau \geq m^{2/5+\epsilon}$, we design a data structure with space usage $\tilde{\mathcal{O}}(\tau)$ and a strongly sublinear product of space and update time. Notably, for $\tau = \lfloor\sqrt{m}\rfloor$, we achieve near-constant update time, and hence the product is $\tilde{\mathcal{O}}(\sqrt{m})$. Our main result for maintaining a suffix random access data structure in asymmetric streaming is as follows:

▶ **Theorem 2** (simplified). *There is an algorithm that, for any reference $R \in \Sigma^m$ and positive integer parameters τ and $k = \tilde{\mathcal{O}}(\tau)$, maintains a $(k-1)$-error streaming suffix random access data structure with space complexity $\tilde{\mathcal{O}}(\tau)$, worst-case update time $\tilde{\mathcal{O}}(1+k^3m^3/\tau^6)$, and query time $\mathcal{O}(1+\log k)$. The preprocessing succeeds without error in $\mathcal{O}(\mathrm{poly}(m))$ expected time.*

A pattern-matching task. A natural approach to designing a suffix random access data structure would be to construct a text indexing data structure for R, which then allows locating an occurrence of any given query string in R. By periodically locating recently received substrings of T within R, we can represent the maintained suffix of T as a list of fragments of R.

A textbook index is the *suffix array* of R, which requires $\mathcal{O}(m)$ space and allows searching for any length-ℓ string in $\mathcal{O}(\ell+\log m)$ time. To decrease space usage, one can store the *sparse suffix array* (see, e.g., [5] and references therein), which can efficiently locate every string of length at least ℓ and uses only roughly $\mathcal{O}(m/\ell)$ space, for a parameter ℓ of our choice. However, this requires fixing the value of ℓ, and it provides a small-space data structure only when ℓ is large.

Interestingly, preprocessing a string of length m in small space for detecting an occurrence of a query string of fixed length $\ell = \Theta(\log m)$ has been studied in another context under the name of *systematic substring search* [29, 47]. In particular, Corrigan-Gibbs and Kogan [29] showed that systematic substring search is equivalent to a well-known problem in cryptography: function inversion. The idea of their reduction is to define a function that maps a position of R to a length-ℓ substring starting at that position. By combining this reduction with the seminal data structure for function inversion devised by Fiat and Naor [40], they achieve a trade-off $\tau^3 q = \tilde{\mathcal{O}}(m^3)$, where τ and q respectively denote the index size and the query time. However, we would like to emphasize that systematic substring search (as well as the reduction to function inversion from [29]) only works for $\ell = \Theta(\log m)$, when the substrings are essentially integers from a polynomial domain.

The construction of a random access data structure may require computing occurrences of substrings of T of arbitrary lengths (in $[1\,..\,m]$) in R. Hence, both existing approaches described above are inadequate for the task at hand.

Our approach. We obtain Theorem 2 via a reduction to function inversion. From the discussion above, one might assume that the reduction is an adaptation of the one used for systematic substring search in [29]. While our problem bears a conceptual resemblance to systematic substring search, the definition of systematic substring search is fairly restrictive, and their straightforward reduction is not applicable in our case. To nonetheless use function inversion, we present two novel and highly non-trivial ideas of independent interest:

1. We design a data structure that can be seen as a relaxation of text indexing on R. Given a query string $Q[1\,..\,3q]$, it either locates an occurrence of the central fragment $Q(q\,..\,2q]$ in R, or reports that the entire Q has no occurrence in R; we call this query a *core-matching query*.

2. We enrich *string synchronizing sets* [72], a tool for locally consistent sampling of substrings with numerous applications in stringology (e.g., [17, 20, 21, 36, 74, 75, 77, 91]), with a local sparsity condition. We then show a space-efficient algorithm for constructing such sets.[3]

By combining these ideas, we ultimately reduce the maintenance of a suffix random access data structure to inverting a function over a domain of size $\tilde{\mathcal{O}}(m/\tau)$. We next provide further details on each of these two ideas and clarify the challenges they address.

Our data structure for core-matching queries is of size $\tilde{\mathcal{O}}(\tau)$ and supports queries for strings of the form $Q[1\,..\,3q]$ with $q = 2^j \cdot \tau$, for integer parameters $j \geq 0$ and $\tau \geq 1$. It is based on the following strategy. Consider covering the reference R with blocks of length $2q$, starting at positions that are multiples of q. Then, every occurrence of $Q[1\,..\,3q]$ in R fully contains one of the blocks, and this block, in turn, contains the central fragment $Q(q\,..\,2q]$ of the occurrence. Therefore, to answer the query, it suffices to (attempt to) locate every length-$2q$ substring of Q among the $\mathcal{O}(m/q)$ blocks. Hence, rather than considering every position of the reference as a potential occurrence, we only have to consider $\mathcal{O}(m/q)$ positions. In our core-matching solution using function inversion, this translates to reducing the domain of the function to $\mathcal{O}(m/q) \subseteq \mathcal{O}(m/\tau)$.

However, by designing a core-matching data structure this way, we inadvertently create another challenge: to answer a query, in the worst case, we have to try to locate *every* length-$2q$ substring of the query string among the blocks. To avoid this, we use a variant of q-synchronizing sets [72]. Introduced by Kempa and Kociumaka [72], they provide a consistent sampling mechanism for length-$2q$ substrings of R and Q such that, broadly speaking, within every length-$3q$ substring of R and Q, at least one length-$2q$ substring is sampled. Crucially, the total number of sampled substrings in R is $\mathcal{O}(m/\tau)$. Hence, instead of the evenly-spaced blocks of length $2q$, we can use the sampled substrings. Thus, rather than having to locate all length-$2q$ substrings of Q, we only have to locate the first sampled length-$2q$ substring of Q, accelerating queries by a factor q.

To actually implement this idea, we crucially need to control the local sparsity of the sampling scheme, i.e., we not only need the total number of sampled substrings in R to be $\mathcal{O}(m/\tau)$, but we also need the number of samples within any length-$\mathcal{O}(q)$ substring to be $\tilde{\mathcal{O}}(1)$. (Otherwise, in the reduction to function inversion, there will be a blow-up in the size of the domain.) Thus, we design a method for sampling a synchronizing set with this local sparsity property that, in addition, can be implemented efficiently in the streaming model, using $\tilde{\mathcal{O}}(q)$ space.

Lower Bounds for Suffix Random Access

We complement our main result with a reverse reduction from the function inversion problem to the problem of maintaining a suffix random access data structure. This implies each of the following results (for a suffix random access data structure whose query time does not exceed its update time):

1. An $\tilde{\mathcal{O}}(\tau)$-space suffix random access data structure with update time faster than ours by a factor $\gg \tau$ would imply a breakthrough in function inversion, improving over Fiat and Naor's solution [40]. This gives a conditional update time lower bound of $\Omega(m^3/\tau^7)$ (Corollary 19).

[3] A different locally sparse variant of string synchronizing set has been proposed by Kempa and Kociumaka in [73]; however the authors of this work do not provide a space-efficient construction algorithm.

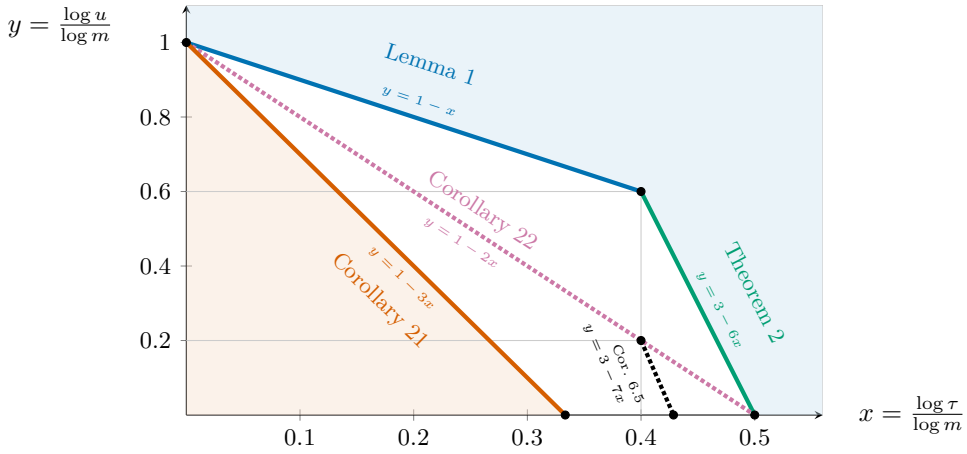

Figure 1 Trade-offs for suffix random access data structures in a doubly-logarithmic scale. Each line segment corresponds to a relation of the exponents of the space τ and the update time u of such a data structure as functions of m, that is, $\tau = m^x$ and $u = m^y$. We consider only data structures where the query time is at most equal to the update time. The blue line segment corresponds to the upper bound of the warm-up data structure (see Lemma 1). The green line segment corresponds to the upper bound of our main suffix access random data structure (see Theorem 2). The orange line segment corresponds to our unconditional lower bound (see Corollary 21). Any improvement of the lower bound beyond the pink dotted line segment would imply a breakthrough in the long-standing unconditional lower bound for function inversion [32,98] (see Corollary 22). Finally, any improvement in the upper bound beyond the black dotted line segment would yield an improvement over the Fiat–Naor function inversion data structure [40] (see Corollary 19).

2. An $\tilde{\mathcal{O}}(\tau)$-space suffix random access data structure cannot have update time better than $\tilde{\mathcal{O}}(m/\tau^3)$, up to subpolynomial factors (Corollary 21). We stress that this is an *unconditional lower bound*.
3. An unconditional lower bound higher than ours by a τ factor would imply a breakthrough in the long-standing unconditional lower bound for function inversion [32, 98]. Hence, there is little hope of turning our conditional lower bound into an unconditional one (Corollary 22).

See Figure 1 for an illustration of our upper and lower bounds for suffix random access data structures.

1.2 Applications

We demonstrate that our suffix random data structure is a powerful tool by using it to obtain asymmetric streaming algorithms for pattern matching and compression.

Generic Pattern Matching

We show that our suffix random data structure allows a unifying reduction for a large family of pattern matching problems. Broadly speaking, if such a problem has an online algorithm in the read-only setting, then our data structure immediately yields an algorithm with similar guarantees in the asymmetric streaming model. In a pattern matching problem, we are given a pattern $P[1\,..\,m]$ and a text $T[1\,..\,n]$, and we are looking for (the ending positions of) fragments of T, called *occurrences*, that *match* P under some matching relation. We introduce the following class of *generic pattern matching* problems:

▶ **Definition 3** (Generic pattern matching). *For an integer $z \geq 0$, a pattern matching problem is* z-generic *if it satisfies the following property. For a pattern $P[1\,..\,m]$ and a text $T[1\,..\,n]$, every occurrence of P in T is a fragment of T that equals a concatenation of at most $z+1$ substrings of P and at most z symbols.*

The generic pattern matching framework captures several by now classical pattern matching problems. To name just a few examples, standard pattern matching, circular pattern matching [7, 18, 19, 22, 23, 44, 66, 67, 70, 96], a variant of elastic-degenerate string matching [4, 10–12, 46, 55, 64, 69, 87, 89], and string matching with variable-length gaps [1, 2, 14, 31, 43, 45, 65, 68, 85, 92] are all $\mathcal{O}(1)$-generic. Additionally, their k-approximate variants under each of the Hamming distance and the edit distance are $\mathcal{O}(k)$-generic pattern matching problems. We show that a suffix random access data structure can be used to transform any online read-only algorithm for a k-generic pattern matching problem into an asymmetric streaming algorithm at a small overhead cost:

▶ **Theorem 4.** *Consider integers $z \geq 0$ and $\tau > 0$. Suppose that, for a z-generic pattern matching problem with pattern $P[1\,..\,m]$, there exists an online read-only algorithm $\mathcal{A}$ that uses $\mathcal{O}(\tau)$ additional space and processes a text $T[1\,..\,n]$ in $g(m,\tau)$ worst-case (or amortized) time per symbol, where f and g are functions that are non-decreasing in their parameters.*

Suppose that there exists an $\mathcal{O}(\tau)$-space streaming z-error random access data structure with worst-case update time $c(m,\tau)$ and worst-case query time $q(m,\tau)$.

Then, there is an asymmetric streaming algorithm that, given τ, random access to P, and streaming access to T, solves the given z-generic pattern matching problem for P and T using $\mathcal{O}(\tau)$ additional space and $\mathcal{O}(c(m,\tau) + g(m,\tau)\cdot q(m,\tau))$ time per symbol of T; this time complexity is worst-case (amortized) if $\mathcal{A}$ processes each symbol in $g(m,\tau)$ worst-case (resp. amortized) time.

In particular, if one allows $\tau = \tilde{\mathcal{O}}(\sqrt{mk})$ extra space, then an online read-only algorithm implies an asymmetric streaming algorithm without asymptotic time overhead. Despite its generality, our reduction is conceptually simple: we maintain a suffix random access structure and use it to simulate the online read-only algorithm through symbol queries. The definition of the class of generic pattern matching problems guarantees that all necessary symbols are available.

Our reduction applies to the entire class of z-generic pattern matching problems, thus providing a general framework for transforming read-only online algorithms into asymmetric streaming ones. Below, we give an example of an application of the reduction to standard and circular pattern matching problems.

▶ **Corollary 5.** *Suppose that we are given positive integers τ and k, read-only access to a pattern P of length m, and a streaming text T of length n. After preprocessing P, we can deterministically*

1. *compute all exact occurrences of P in T using space $\mathcal{O}(\tau)$ and worst-case time $\tilde{\mathcal{O}}(1+m^3/\tau^6)$ per symbol of T, and*
2. *compute all k-approximate occurrences of P in T under the Hamming distance using space $\mathcal{O}(k\log m+\tau)$ and worst-case time $\tilde{\mathcal{O}}(k+k^3m^3/\tau^6)$ per symbol of T, and*
3. *compute all k-approximate occurrences of P in T under the edit distance using space $\tilde{\mathcal{O}}(k^4+\tau)$ and amortized time $\tilde{\mathcal{O}}(k^4+k^3m^3/\tau^6)$ per symbol of T.*

We summarize the results for exact and approximate (standard) pattern matching in Table 1.[4] We note that the bounds for the fully streaming model carry straightforwardly to the asymmetric streaming one. All streaming algorithms, however, critically rely on Monte Carlo randomization. In contrast, the output of our algorithms for the asymmetric streaming model is always correct and only the preprocessing of the pattern is Las Vegas randomized; for $\tau \geq m^{0.4}$ and $k = m^{o(1)}$, our algorithms' time-space product is sublinear. This poses the challenge of exploring the trade-off between algorithmic efficiency and randomization in asymmetric streaming pattern matching.

Table 1 Complexities of exact and approximate (standard) pattern matching for a pattern of length m. Here, τ is an integer parameter and time complexities are given per symbol of the text. The complexities in the asymmetric streaming model (marked with †) are shown in this work and are *deterministic* (apart from Las Vegas randomization in the preprocessing of the pattern). In contrast, complexities in the streaming setting are *Monte Carlo randomized*.

	Online read-only (deterministic, space used on top of read-only P *and* T)	**Asym. streaming** (deterministic, space used on top of read-only P)	**Streaming** (randomized)
Exact	$\mathcal{O}(1)$ space $\mathcal{O}(1)$ time [15]	$\mathcal{O}(\tau)$ space $\tilde{\mathcal{O}}(1 + m^3/\tau^6)$ time†	$\mathcal{O}(\log m)$ space $\mathcal{O}(\log m)$ time [88]
k-mism.	$\mathcal{O}(k \log m)$ space $\mathcal{O}(k)$ time [8]	$\mathcal{O}(k \log m + \tau)$ space $\tilde{\mathcal{O}}(k + k^3m^3/\tau^6)$ time†	$\mathcal{O}(k \log m)$ space $\tilde{\mathcal{O}}(\sqrt{k})$ time [28]
k-edit	$\tilde{\mathcal{O}}(k^4)$ space $\tilde{\mathcal{O}}(k^4)$ (amort.) time [8]	$\tilde{\mathcal{O}}(k^4 + \tau)$ space $\tilde{\mathcal{O}}(k^4 + k^3m^3/\tau^6)$ (amort.) time†	$\tilde{\mathcal{O}}(k^2)$ space $\tilde{\mathcal{O}}(k^2)$ time [13]

Our reduction also has consequences for circular pattern matching. Bathie, Charalampopoulos, and Starikovskaya [7], showed that, for every $\tau \in [\sqrt{m} \log m (\log \log m)^3 \,..\, m]$, there exists an asymmetric streaming algorithm that solves this problem in $\mathcal{O}(\tau)$ space and $\tilde{\mathcal{O}}(m/\tau)$ time per symbol. We next obtain a significantly better trade-off; notably, for $\tau \geq m^{0.4}$, the time-space product of the algorithm encapsulated in Corollary 6 is sublinear.

▶ **Corollary 6.** *Let $\tau > 0$ be an integer parameter. There is a deterministic asymmetric streaming algorithm solving the exact circular pattern matching problem on a read-only pattern of length m and a streaming text of length n in $\tilde{\mathcal{O}}(\tau)$ space and $\tilde{\mathcal{O}}(m^3/\tau^6 + 1)$ time per symbol of the text.*

Relative Lempel–Ziv Factorization

Relative Lempel–Ziv factorization is a compression scheme in which a text T of length n is greedily parsed into lengthwise maximal substrings of a reference R of length m. This method is especially effective for compressing collections of strings that are highly similar to a reference, and it has inspired a small-space text index for databases containing full genome sequences of individuals of the same species [33, 34, 79, 86, 97]. Nevertheless, no small-space algorithm for computing this factorization is known. Formally, the relative Lempel–Ziv factorization is defined as follows:

[4] There is an unpublished work [82] on asymmetric streaming pattern matching. However, this work considers the problem where one is given random access to the text while the pattern arrives as a stream. Hence, this contribution is incomparable to ours.

▶ **Definition 7** ([79]). *Let strings $T, R \in \Sigma^*$. We say that $T = f_1 f_2 \dots f_z$ is the Lempel–Ziv factorization of T relative to R, $\mathsf{rlz}(T, R)$, if for every $1 \le i \le z$, f_i is either a symbol that does not occur in R, or the longest substring of R that is a prefix of $f_i f_{i+1} \dots f_z$. We call $f_1, f_2, \dots, f_z$ the factors of the factorization.*

We address this gap by leveraging our suffix random access data structure to design the first asymmetric streaming algorithm for relative Lempel–Ziv factorization. We call a factorization of a string T into phrases that are symbols and fragments of a string R an *R-factorization of T*.

▶ **Theorem 8.** *Consider a read-only reference string $R[1\,..\,m]$, a streaming string $T[1\,..\,n]$, a positive integer $\tau \in [\lceil m^{0.4} \rceil / \epsilon\,..\,m]$, and a real $\epsilon \in (0, 1]$. Let $z = |\mathsf{rlz}(T, R)|$. We can preprocess R in $\mathrm{poly}(m)$ expected time to construct an $\tilde{\mathcal{O}}(\tau)$-space data structure, so that we can then process T in total time $\tilde{\mathcal{O}}(z \cdot (m/\tau)^{5/3} + z \cdot m/(\epsilon\tau) + n \cdot (1 + m^3/\tau^6) + m)$ total time using $\tilde{\mathcal{O}}(\tau)$ extra space, outputting an R-factorization of T of size at most $(1+\epsilon)z$ as a stream.*

The preprocessing can be implemented in $\tilde{\mathcal{O}}(m \cdot \mathrm{poly}(\lambda))$ time with success probability at least $1 - 2^{-\lambda}$ at the expense of increasing the space complexity and time required for processing T by an $\mathcal{O}(\mathrm{poly}(\lambda))$ multiplicative factor, for any integer $\lambda \ge \log_2(m\sigma)$.

The algorithm starts by constructing a factorization of size $\mathcal{O}(\log^2 m) \cdot z$ using core-matching queries. The factorization is then refined using ideas similar to those in a work of Fischer, Gagie, Gawrychowski, and Kociumaka [41], who showed an efficient read-only algorithm for constructing the *standard* Lempel–Ziv factorization of a text in small space. For both steps, we exploit random access on the relevant suffix of T.

On the lower bound side, we show that any τ-space $\tilde{\mathcal{O}}(1)$-approximation asymmetric streaming algorithm for relative Lempel–Ziv factorization must use $\tilde{\Omega}(z(m/\tau))$ time:

▶ **Corollary 9.** *Fix positive integers n, m, z, φ such that $100z\varphi \le n \le zm$. On a word RAM of word-width $\mathcal{O}(\log m)$, consider an algorithm that preprocesses a read-only reference $R[1\,..\,m]$ into a data structure of size $\tau = o(m/\varphi^2)$ bits. Then, given text $T[1\,..\,n]$ satisfying $|\mathsf{rlz}(R, T)| \in [z, 100z\varphi]$, it outputs an R-factorization of T of size at most $\varphi \cdot |\mathsf{rlz}(R, T)|$ as a stream in left-to-right order (each phrase as a fragment of T without providing an occurrence in R), using τ bits of additional space.*

If the preprocessing succeeds with probability $\ge 1/100$ such that afterwards the text processing succeeds for every text, then the text processing takes $\Omega(zm/(\varphi^2\tau))$ time in the worst-case.

We obtain this lower bound via a reduction from (a version of) the set-complementation problem, for which we show a new lower bound using R-way branching programs.

2 Technical Overview of the Suffix Random Access Data Structure

In this section, we give a technical overview of the main results of this work related to the suffix random access data structure. (Below, we often refer to it as random access data structure for brevity.) Intuitively, we are given a short read-only string R and a long streaming string T, and would like to maintain random access to the longest suffix of T that is close to a substring of R using little extra space. We study upper and lower bounds for random access data structures via bi-directional reductions to function inversion.

We assume the word RAM model with words of width w, i.e., we can perform basic arithmetic operations on integers from $[0\,..\,2^{\mathcal{O}(w)}]$ in constant time. We further assume that the input strings are over a finite integer alphabet $\Sigma = [1\,..\,\sigma]$ with $\sigma \in n^{\mathcal{O}(1)}$, and make the

common assumption that $\log n = \mathcal{O}(w)$. We develop classic offline read-only algorithms and asymmetric streaming algorithms, introduced by Andoni et al. [3] and Saks and Seshadhri [94]. In the former setting, one can access symbols of the input in constant time, and does not account for the space required to store the input in the space complexity of an algorithm. In the asymmetric streaming setting, the input consists of a read-only string $R \in \Sigma^*$, called a *reference*, and a streaming string $T \in \Sigma^*$. Here, we receive T symbol by symbol and must account for the space required to store any information about T in the space complexity of an algorithm (whereas the space required to store R is not accounted for). Unless explicitly said otherwise, we measure the space complexity in words (including the working space used while answering queries).

2.1 Upper Bounds for Random Access Data Structures

We formalize the problem of maintaining a random access data structure as follows.

▶ **Definition 10** (k-Error Random Access Data Structure)**.** *For an integer $k \geq 0$, a string $R \in \Sigma^m$ (called the* reference*), and a string $T \in \Sigma^n$ (called the* text*), a* k-error random access data structure

- *explicitly stores its* support-length $h \in [0\,..\,n]$*, which satisfies the following property: if $h < n$, then $T[n-h\,..\,n]$ cannot be factorized into k symbols and $k+1$ substrings of R, and*
- *can return $T[i]$ upon receiving a* random access query $i \in (n-h\,..\,n]$*, possibly accessing the reference R, but not accessing the text T.*

We introduce different algorithms for constructing such data structures in the read-only model and in the asymmetric streaming model. In both models, we assume that the data structure is defined by a given parameter τ. Also in both models, we have read-only access to the reference, and we may preprocess it into a data structure of size $\tilde{\mathcal{O}}(\tau)$. Then, in the read-only model, the text is given as a read-only string, and we have to construct the random access data structure using the precomputed information and random access on R and T. We call this version of the data structure *offline*. In the asymmetric streaming model, we instead receive the text as a stream. Whenever we receive some symbol $T[i]$, we must update the maintained data structure such that it is a random access data structure for the reference R and a text $T[1\,..\,i]$. We call this version of the data structure *streaming*. In either model, the total space occupied at any time after preprocessing (apart from R) may not exceed $\tilde{\mathcal{O}}(\tau)$. To summarize, apart from the $\tilde{\mathcal{O}}(\tau)$ space complexity of the data structure, we are interested in the preprocessing time (spent processing R before the first access to T), the query time for random access queries, and either the construction time after preprocessing (for the read-only model) or the update time per arriving symbol (for the asymmetric streaming model).

Reduction to the 0-Error Offline Setting

We start by showing that an efficient offline construction algorithm immediately implies an efficient streaming algorithm. This is done by cutting T into blocks of geometrically increasing sizes on $\mathcal{O}(\log m)$ levels, and then constructing the offline data structure for each block. The cost of running the offline construction algorithm can be de-amortized over the arriving symbols. This is similar to the black box offline to online reduction for approximate pattern matching, see [25]. Unlike [25], we cannot afford $\mathcal{O}(m)$ additional space, and we have to ensure that we already support random access to a block while constructing its data structure. While constructing the offline data structure for a block of size 2^ℓ, we use the random access data structures already constructed for the two blocks of size $2^{\ell-1}$ that make up the larger block.

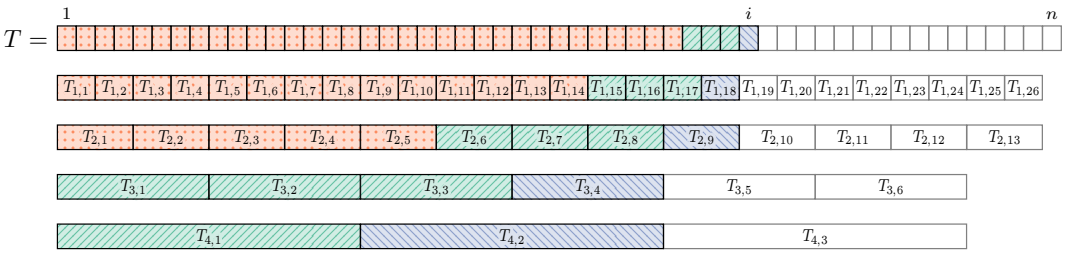

Figure 2 Text partitioning for the proof of Corollary 11. After $T[i]$ arrives, the streaming data structure consists of the offline data structures of the blocks marked ▨. The data structures of previous blocks, marked ▨, have already been deleted. Blocks marked ▨ have already been received, but their offline data structures have not been constructed yet. The offline data structure of block $T_{3,4}$ will be constructed while $T_{2,10}$ arrives. As soon as the last symbol of $T_{2,10}$ arrives, the offline data structure of $T_{3,1}$ will be deleted.

For the offline construction, we further observe that, instead of constructing a k-error data structure, we can chain together $\mathcal{O}(k)$ 0-error data structures. A random access query can then be answered by binary searching in $\mathcal{O}(\log k)$ time for the data structure responsible for the query.

▶ **Corollary 11.** *Assume that there is an offline 0-error random access data structure with preprocessing time $\mathcal{O}(p(m))$, construction time $\mathcal{O}(n \cdot c(m))$, query time $\mathcal{O}(q(m))$, and space complexity $\mathcal{O}(s(m))$. For every positive integer k, there is a streaming k-error random access data structure with preprocessing time $\mathcal{O}(p(m))$, worst-case update time $\mathcal{O}(c(m) \cdot (q(m) + \log k) \cdot \log m)$, query time $\mathcal{O}(q(m) + \log k)$, and space complexity $\mathcal{O}(k \cdot s(m) \cdot \log m)$.*

Reduction to Core-matching Queries

In the 0-error offline setting, we further isolate the main computational challenge by showing that a simple data structure of size $\tilde{\mathcal{O}}(\tau)$ with constant query time can be constructed by answering a series of $\tilde{\mathcal{O}}(1)$ so-called *core-matching queries* on R. Each query consists of a string $T'[1\,..\,3n']$ (a substring of T) with $n' = 2^\ell \cdot \tau$ for some integer $\ell \geq 0$, and the result of the query is some position i such that $R[i\,..\,i+n') = T'(n'\,..\,2n']$, i.e., we must locate the central length-n' fragment of T' in R. We are allowed to fail if T' does not occur in R.

To understand the idea of the reduction, consider a particularly simple case when T is of length $n = 3 \cdot 2^L \cdot \tau$ and the entire T is a substring of R. In this case, the random access data structure consists of a copy of the length-τ prefix and suffix of T, as well as pointers to $\mathcal{O}(\log n)$ fragments $r_1, \ldots, r_{\mathcal{O}(\log n)}$ of R obtained by answering core-matching queries for all prefixes and suffixes of T that are of length $3 \cdot 2^\ell \cdot \tau$ for integer $\ell \geq 0$. This is visualized in Figure 3.

Figure 3 Reduction to core-matching queries. Here, ▨ indicates the length-τ prefix and suffix of T, and ⊢r_i⊣ indicates the core-matching query used to find r_i.

With simple techniques to generalize for a string T of arbitrary length that is not a substring of R, this leads to the following result.

▶ **Lemma 12** (simplified). *Let $\tau > 0$ be an integer parameter. Computing an offline 0-error random access data structure with constant query time and space complexity $\mathcal{O}(\tau + \log m)$ for reference $R \in \Sigma^m$ can be reduced to $\mathcal{O}(\log^2 m)$ core-matching queries on R. The reduction takes $\mathcal{O}(\tau + \log^2 m)$ time and $\mathcal{O}(\tau + \log m)$ additional working space.*

Core-matching Queries via Function Inversion

As a warm-up, we first show a simple solution to core-matching queries based on the following observation: If $T' = R[i\,..\,i + 3n')$ for some $i \in [1\,..\,m - 3n' + 1]$, then this occurrence of T' fully contains the fragment $R(bn' - 2n'\,..\,bn']$ with $b = \lfloor \frac{i+3n'-1}{n'} \rfloor$. For this, we use Karp–Rabin fingerprints that are composable and have low collision probability.

▶ **Definition 13** (Karp–Rabin fingerprint [71, 88]). *For a prime number p and an integer $r \in [0\,..\,p)$, the Karp–Rabin fingerprint of $X \in \Sigma^n$ is $\varphi(X) = \sum_{i=1}^{n} X[i] \cdot r^{n-i} \pmod p$.*

We precompute the Karp–Rabin fingerprints [71] $\varphi(U)$ of all fragments U of the form $R(bn' - 2n'\,..\,bn']$ with $b = \lfloor \frac{i+3n'-1}{n'} \rfloor$ in $\mathcal{O}(m/n')$ space and store them in a balanced binary search tree. At query time, we compute the fingerprints of all length-$2n'$ substrings of T', and use them to find j such that $T'[j\,..\,j + 2n') = R(bn' - 2n'\,..\,bn']$. We can then simply output $R(bn' - n' - j + 1\,..\,bn' - j + 1] = T(n'\,..\,2n']$. Combined with the reductions above, this already gives a k-error random access data structure with a non-trivial trade-off:

▶ **Corollary 14** (simplified). *There is a streaming k-error random access data structure with space complexity $\tilde{\mathcal{O}}((k+1) \cdot \sqrt{m})$, worst-case update time $\tilde{\mathcal{O}}(1)$, and query time $\mathcal{O}(1 + \log(k+1))$. The preprocessing takes $\mathcal{O}(\mathrm{poly}(m))$ time and $\tilde{\mathcal{O}}(\sqrt{m})$ space deterministically.*

In a pursuit of a better trade-off, we replace the binary search tree with a function inversion data structure. The idea is to use a function $f(x) = \varphi(R(xn' - 2n'\,..\,xn'])$ with domain $[N]$, where $N = \lfloor m/n' \rfloor - 1$. Rather than locating the fingerprint y of every length-$2n'$ substring $T'[j\,..\,j + 2n')$ of T' in the binary search tree, we instead invert the function and obtain $x \in f^{-1}(y)$, which implies $T'[j\,..\,j + 2n) = R(xn' - 2n'\,..\,xn']$. The best-known trade-off for function inversion is due to Fiat and Naor [40], who showed that $\tilde{\mathcal{O}}(N^3/\tau^3)$ inversion time can be achieved in $\tilde{\mathcal{O}}(\tau)$ space:

▶ **Fact 15** ([40]). *Let $f : [N] \to [N]$ be a function that can be evaluated at any point in constant time. For any integer parameter $\tau > 0$, we can construct in $\tilde{\mathcal{O}}(N)$ time and with success probability $1 - 1/N$ a data structure that is capable of inverting f at any point in $\tilde{\mathcal{O}}(N^3/\tau^3)$ time. The data structure can be constructed, stored, and queried in $\tilde{\mathcal{O}}(\tau)$ space.*

For core-matching queries of length $3n' = 3\tau$, we can achieve $\tilde{\mathcal{O}}(\tau)$ space and $\tilde{\mathcal{O}}(m^3/\tau^4)$ time: the domain of f is of size $N = \mathcal{O}(m/\tau)$, but we have to multiply the $\tilde{\mathcal{O}}(N^3/\tau^3)$ inversion time of Fact 15 with the number of inversion queries and the evaluation time of f, both of which are roughly τ. To reduce the time, we use a variant of the string synchronizing sets [72] to consistently and regularly select synchronizing positions in both R and T. This allows us to obtain a better function f that retains the size of the domain and the evaluation time, but only needs to be inverted (close to) a constant number of times in order to answer a core-matching query. In the definition below, we use the original consistency and density conditions from [72], but also add a sparsity condition, ensuring that, within any length-τ fragment of R, we select a small number of positions. For a string T, we denote by $\mathsf{per}(T)$ the smallest period of T, that is, the smallest integer p such that $T[i] = T[i + p]$ for all $i \in [1\,..\,n - p]$.

▶ **Definition 16.** *Let k, τ be positive integers. For a string $R \in \Sigma^m$, a set $S \subseteq [1\,..\,m-2\tau+1]$ is a k-sparse τ-synchronizing set if it satisfies the following conditions:*

(Consistency) *For $i, j \in [1\,..\,m-2\tau+1]$, if $R[i\,..\,i+2\tau) = R[j\,..\,j+2\tau)$ then $i \in S \iff j \in S$.*

(Density) *For $i \in [1\,..\,m-3\tau+2]$, it holds $S \cap [i\,..\,i+\tau) = \emptyset$ if and only if $\mathsf{per}(R[i\,..\,i+3\tau-2]) \leq \frac{\tau}{3}$.*

(Sparsity) *For $i \in [1\,..\,m-3\tau+2]$, it holds $|S \cap [i\,..\,i+\tau)| \leq k$.*

During construction, we enforce the sparsity condition by sampling every distinct substring with probability around k/τ. Namely, we define a function $\mathsf{sync} : \Sigma^{2\tau} \to \{0,1\}$ that identifies synchronizing substrings. Crucially, the function can be stored in $\tilde{\mathcal{O}}(1)$ space, and can be evaluated efficiently in a rolling manner. This way, we can produce the elements of a string synchronizing set as a stream.

▶ **Theorem 17.** *Let $R \in \Sigma^m$ be a string, and let $\tau \in [1\,..\,\lfloor \frac{m}{2} \rfloor]$ and $\lambda \geq \log_2(m\sigma)$ be integers. In $\mathcal{O}(\mathrm{poly}(\lambda))$ time and space, and with success probability $1 - 2^{-\lambda}$, we can construct a function $\mathsf{sync} : \Sigma^{2\tau} \to \{0,1\}$ encoded in $\mathcal{O}(\lambda)$ space with the following properties:*

1. *$S = \{i \in [1\,..\,m-2\tau+1] \mid \mathsf{sync}(R[i\,..\,i+2\tau)) = 1\}$ is an $\mathcal{O}(\lambda)$-sparse τ-synchronizing set.*
2. *Given $R' \in \Sigma^{m'}$ of arbitrary length, the set $\{i \in [1\,..\,m'-2\tau+1] \mid \mathsf{sync}(R'[i\,..\,i+2\tau)) = 1\}$ can be produced as a stream in increasing order, in $\mathcal{O}(m'\lambda)$ time and $\mathcal{O}(\lambda)$ working space.*

We are now ready to explain our final solution for core-matching queries. To solve a query $T'[1\,..\,3n']$, we use an $\tilde{\mathcal{O}}(1)$-sparse n'-synchronizing set S of R. We find the minimal $j \in [1\,..\,n'+1]$ such that $\mathsf{sync}(T'[j\,..\,j+2n')) = 1$, which takes $\mathcal{O}(n')$ time with the algorithm from Theorem 17(2). For intuition, assume that j is well-defined. (If j does not exist, then T' is periodic and the solution only becomes simpler.) We compute $y = \varphi(T'[j\,..\,j+2n'))$ and use Fact 15 to obtain some $x \in f^{-1}(y)$, where this time $f(x) = \varphi(R[x\,..\,x+2n'))$ is defined not for domain $[\lfloor \frac{m}{n'} \rfloor - 1]$ but for domain S, which is still of size $\tilde{\mathcal{O}}(\frac{m}{n'})$. While Fact 15 requires that the domain is an integer range, the sparsity of S allows us to obtain an efficient mapping between $[|S|]$ and S. Hence we can still invert f, and, if we find $x \in f^{-1}(y)$, then it holds $T'[j\,..\,j+2n) = R[x\,..\,x+2n')$.

For core-matching queries of length $3n' = 3\tau$, using synchronizing sets improves the time complexity to $\tilde{\mathcal{O}}(m^3/\tau^5)$. We plug this solution for core-matching queries into Lemma 12 to obtain a 0-error offline data structure, and then into Corollary 11 (with a slight modification that improves the dependency on k) to obtain our main result:

▶ **Theorem 2** (simplified)**.** *There is an algorithm that, for any reference $R \in \Sigma^m$ and positive integer parameters τ and $k = \tilde{\mathcal{O}}(\tau)$, maintains a $(k-1)$-error streaming suffix random access data structure with space complexity $\tilde{\mathcal{O}}(\tau)$, worst-case update time $\tilde{\mathcal{O}}(1+k^3m^3/\tau^6)$, and query time $\mathcal{O}(1+\log k)$. The preprocessing succeeds without error in $\mathcal{O}(\mathrm{poly}(m))$ expected time.*

2.2 Lower Bound for Random Access Data Structures

In computational complexity, a fundamental difference arises between *uniform* and *non-uniform* computation. In the former case, we assume that a single algorithm is used to solve input instances of all possible sizes (i.e., the size is part of the input). In the latter case, we are allowed to design a family of algorithms, using a different algorithm for each input size. We say *non-uniform algorithm* to refer to the entire family, and *non-uniform data structure* whenever a data structure is constructed, updated, or queried using non-uniform algorithms. Crucially, a non-uniform algorithm may embed an arbitrary amount of *advice* that depends only on the size of the input.

To show a lower bound for random access data structures, we show how to obtain a non-uniform function inversion protocol from any offline random access data structure. Assume that the text T is a substring of the reference R. Intuitively, the construction algorithm for a random access data structure must find an occurrence of T in R. Using this observation, we show that a random access data structure implies a function inversion protocol: we encode the image of a function in a reference R, and to invert an element of the image, we encode and feed it as a text T. As a result, we obtain the following:

▶ **Corollary 18.** *Let $\chi, \delta > 0$ with $\chi < 1$ be constant. Assume that there is a (possibly non-uniform) offline random access data structure with space complexity $\mathcal{O}(m^\chi)$ bits, amortized query time $\tilde{\mathcal{O}}(m^\delta)$, and construction time $\tilde{\mathcal{O}}(m^{\chi+\delta})$ for a text of length at most $m^\chi \log m$. Then there is a non-uniform function inversion data structure that, for functions with domain and co-domain $[N]$, has query time $\tilde{\mathcal{O}}(N^{(\chi+\delta)/(1-\chi)})$ and can be stored and queried in $\tilde{\mathcal{O}}(N^{\chi/(1-\chi)})$ space.*

This reduction directly yields a conditional lower bound for random access data structures:

▶ **Corollary 19.** *Let $\chi, \delta > 0$ with $2/5 \leq \chi < 1$ be constant. Assume that there is a (possibly non-uniform) offline random access data structure with space complexity $\mathcal{O}(m^\chi)$ bits, amortized query time $\tilde{\mathcal{O}}(m^\delta)$, and construction time $\tilde{\mathcal{O}}(m^{\chi+\delta})$ for a text of length at most $m^\chi \log m$. If $\delta < 3 - 7\chi$, then there is a non-uniform function inversion data structure that improves over the time-space trade-off of Fiat and Naor.*

Using well-known lower bounds on the time-space trade-off of permutation inversion [32,98], we also derive the following unconditional lower bound:

▶ **Corollary 20.** *Let $\chi, \delta > 0$ with $\chi < 1$ be constant. Assume that there is a (possibly non-uniform) offline random access data structure with space complexity $\mathcal{O}(m^\chi)$ bits, amortized query time $\tilde{\mathcal{O}}(m^\delta)$, and construction time $\tilde{\mathcal{O}}(m^{\chi+\delta})$ for a text of length at most $m^\chi \log m$. Then $\delta \geq 1 - 3\chi$.*

The above results hold only for relatively short texts. However, in the streaming setting, we can show that similar lower bounds hold for texts of arbitrary length.

▶ **Corollary 21.** *Let ϵ, χ with $0 < \chi < 1$ be constant. Assume that there is a streaming random access data structure with space complexity $\mathcal{O}(m^\chi)$. For all $n \geq m^\chi \log m$ and all constant $\varepsilon > 0$, there is a length-n text T such that both of the following hold:*

1. *If the amortized query time is $\mathcal{O}(Q/m^\varepsilon)$, where $Q = m^{3-7\chi}$ and $0.4 \leq \chi < 1$, then the worst-case update time is $\tilde{\Omega}(Q)$, unless there is a non-uniform function inversion data structure that improves over the time-space trade-off of Fiat and Noar.*
2. *If the amortized query time is $\mathcal{O}(Q/m^\varepsilon)$, where $Q = m^{1-3\chi}$ and $0 < \chi < 1$, then the worst-case update time is $\tilde{\Omega}(Q)$.*

Finally, via our reduction from function inversion to suffix random access, we show that an unconditional lower bound higher than ours by a factor of τ would imply a breakthrough in the long-standing unconditional lower bound for function inversion [32, 98].

▶ **Corollary 22.** *Let $0 < \epsilon < 1$ and let $\tau > 0$ be an integer parameter. Assume that, for every 0-error streaming random access data structure with space complexity $\tilde{\mathcal{O}}(\tau)$ bits and constant query time, the worst-case update time is $\tilde{\Omega}(m/\tau^{2-\epsilon})$. Then, every τ-space function inversion data structure for a constant-time computable function $f : [N] \to [N]$ must have worst-case query time $\tilde{\Omega}(N/\tau^{1-\epsilon})$, and hence the product of space and query time is $\tilde{\Omega}(N\tau^\epsilon)$.*

3 Discussion and Open Problems

The main open problem that stems from our work is closing the gap between the obtained upper and lower bounds for suffix random access data structures. Let us briefly discuss some of the main obstacles toward this goal.

The main challenge in improving our lower bound from Corollary 21 to match the $\tilde{\Omega}(m/\tau)$ space-time product of Corollary 22 is the following difference in the nature of the two considered problems: a function inversion data structure becomes active just after construction, while a random access data structure can essentially afford to wait for τ symbols (as it can simply store them explicitly). In our lower bound construction, we naturally encode a function f into the reference and use the encoding of the image $f(x)$ of a domain element x as the text. If this text is shorter than τ, then a (naive) random access data structure can explicitly store a copy. Therefore, we enforce the text length to be significantly greater than τ by encoding $f(x)$ in $\gg \tau$ symbols, for each x in the domain. Consequently, the size of the domain is limited by $\mathcal{O}(m/\tau)$. While we can indeed show that processing the encoding of $f(x)$ as the text allows us to invert f at position $f(x)$, we only obtain a lower bound $\Omega(m/\tau^3)$ on the update time: we have to divide the $\Omega(m/\tau^2)$ lower bound for inverting a function with domain of size m/τ by the length of the text.

The main reason for the extra τ factor in the upper bound of Theorem 2 compared to Corollary 19 is that the function we invert (Karp–Rabin fingerprints) cannot be evaluated in constant time. Essentially, we can afford to wait for at most τ symbols to arrive, but then we intuitively have to perform a task of a pattern-matching flavor for a length-τ substring of T. Using string synchronizing sets, we reduce this task to inverting a function over a domain of size roughly m/τ; this can be done in $\tilde{\mathcal{O}}(m^3/\tau^6)$ time using the Fiat–Naor function inversion data structure [40]. We only need to invert once every $\mathcal{O}(\tau)$ symbols arrive, so we need to pay for $\tilde{\mathcal{O}}(m^3/\tau^7)$ function evaluations per symbol (as part of the inversions). However, evaluating Karp–Rabin fingerprints of fragments of T requires $\Omega(\tau)$ time (given our space constraints). We believe that overcoming this challenge will require a fundamentally new idea.

References

1 Amihood Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B. Riva Shalom. Mind the gap! - online dictionary matching with one gap. *Algorithmica*, 81(6):2123–2157, 2019. doi:10.1007/S00453-018-0526-2.

2 Amihood Amir, Avivit Levy, Ely Porat, and B. Riva Shalom. Dictionary matching with a few gaps. *Theor. Comput. Sci*, 589:34–46, 2015. doi:10.1016/j.tcs.2015.04.011.

3 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proc. of FOCS*, pages 377–386, 2010. doi:10.1109/FOCS.2010.43.

4 Kotaro Aoyama, Yuto Nakashima, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster Online Elastic Degenerate String Matching. In *Proc. of CPM*, pages 9:1–9:10, Dagstuhl, Germany, 2018. doi:10.4230/LIPIcs.CPM.2018.9.

5 Lorraine A. K. Ayad, Grigorios Loukides, Solon P. Pissis, and Hilde Verbeek. Sparse suffix and LCP array: Simple, direct, small, and fast. In *Proc. of LATIN (1)*, pages 162–177, 2024. doi:10.1007/978-3-031-55598-5_11.

6 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theor. Comput. Sci.*, 494:13–23, 2013. doi:10.1016/J.TCS.2012.12.028.

7 Gabriel Bathie, Panagiotis Charalampopoulos, and Tatiana Starikovskaya. Internal pattern matching in small space and applications. In *Proc. of CPM*, pages 4:1–4:20, 2024. doi:10.4230/LIPIcs.CPM.2024.4.

8 Gabriel Bathie, Tomasz Kociumaka, and Tatiana Starikovskaya. Small-space algorithms for the online language distance problem for palindromes and squares. In *Proc. of ISAAC*, pages 10:1–10:17, 2023. doi:10.4230/LIPIcs.ISAAC.2023.10.

9 Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *Proc. of ICALP*, pages 119:1–119:17, 2021. doi:10.4230/LIPIcs.ICALP.2021.119.

10 Giulia Bernardini, Estéban Gabory, Solon P. Pissis, Leen Stougie, Michelle Sweering, and Wiktor Zuba. Elastic-degenerate string matching with 1 error or mismatch. *Theory Comput. Syst.*, 68(5):1442–1467, 2024. doi:10.1007/S00224-024-10194-8.

11 Giulia Bernardini, Paweł Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Elastic-degenerate string matching via fast matrix multiplication. *SIAM J. Comput*, 51(3):549–576, 2022. doi:10.1137/20M1368033.

12 Giulia Bernardini, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Approximate pattern matching on elastic-degenerate text. *Theor. Comput. Sci*, 812:109–122, 2020. In memoriam Danny Breslauer (1968-2017). doi:10.1016/j.tcs.2019.08.012.

13 Sudatta Bhattacharya and Michal Koucký. Streaming k-edit approximate pattern matching via string decomposition. In *Proc. of ICALP*, pages 22:1–22:14, 2023. doi:10.4230/LIPIcs.ICALP.2023.22.

14 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and David Kofoed Wind. String matching with variable length gaps. *Theor. Comput. Sci*, 443:25–34, 2012. doi:10.1016/j.tcs.2012.03.029.

15 Dany Breslauer, Roberto Grossi, and Filippo Mignosi. Simple real-time constant-space string matching. *Theor. Comput. Sci.*, 483:2–9, 2013. doi:10.1016/J.TCS.2012.11.040.

16 Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern hamming distances. In *Proceedings of STOC 2020*, pages 643–656, 2020. doi:10.1145/3357713.3384266.

17 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Faster algorithms for longest common substring. In *Proc. of ESA*, pages 30:1–30:17, 2021. doi:10.4230/LIPIcs.ESA.2021.30.

18 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba. Circular pattern matching with k mismatches. *J. Comput. Syst. Sci.*, 115:73–85, 2021. doi:10.1016/J.JCSS.2020.07.003.

19 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Walen, and Wiktor Zuba. Approximate circular pattern matching. In *Proc. of ESA*, pages 35:1–35:19, 2022. Full version: https://doi.org/10.48550/arXiv.2208.08915. doi:10.4230/LIPIcs.ESA.2022.35.

20 Panagiotis Charalampopoulos, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, Tomasz Walen, and Wiktor Zuba. Counting distinct square substrings in sublinear time. In *Proc. of MFCS*, pages 36:1–36:19, 2025. doi:10.4230/LIPIcs.MFCS.2025.36.

21 Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest palindromic substring in sublinear time. In *Proc. of CPM*, pages 20:1–20:9, 2022. doi:10.4230/LIPIcs.CPM.2022.20.

22 Panagiotis Charalampopoulos, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Tomasz Walen, and Wiktor Zuba. Approximate circular pattern matching under edit distance. In *Proc. of STACS*, pages 24:1–24:22, 2024. doi:10.4230/LIPIcs.STACS.2024.24.

23 Kuei-Hao Chen, Guan-Shieng Huang, and Richard Chia-Tung Lee. Bit-Parallel Algorithms for Exact Circular String Matching. *Comput. J.*, 57(5):731–743, March 2013. doi:10.1093/comjnl/bxt023.

24 Kuan Cheng, Alireza Farhadi, MohammadTaghi Hajiaghayi, Zhengzhong Jin, Xin Li, Aviad Rubinstein, Saeed Seddighin, and Yu Zheng. Streaming and small space approximation algorithms for edit distance and longest common subsequence. In *Proc. of ICALP*, pages 54:1–54:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.54.

25 Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. *Inf. Comput.*, 209(4):731–736, 2011. doi:10.1016/J.IC.2010.12.007.

26 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. Dictionary matching in a stream. In *Proc. of ESA*, pages 361–372, 2015. doi:10.1007/978-3-662-48350-3_31.

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

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

29 Henry Corrigan-Gibbs and Dmitry Kogan. The function-inversion problem: Barriers and opportunities. In *Proc. of TCC (1)*, volume 11891, pages 393–421. Springer, 2019. doi:10.1007/978-3-030-36030-6_16.

30 Maxime Crochemore. String-matching on ordered alphabets. *Theor. Comput. Sci.*, 92(1):33–47, 1992. doi:10.1016/0304-3975(92)90134-2.

31 Maxime Crochemore, Costas Iliopoulos, Christos Makris, Wojciech Rytter, Athanasios Tsakalidis, and Kostas Tsichlas. Approximate string matching with gaps. *Nordic J. of Computing*, 9(1):54–65, 2002.

32 Anindya De, Luca Trevisan, and Madhur Tulsiani. Non-uniform attacks against one-way functions and PRGs. *Electron. Colloquium Comput. Complex.*, TR09-113, 2009. URL: https://eccc.weizmann.ac.il/report/2009/113.

33 Sebastian Deorowicz, Agnieszka Danek, and Szymon Grabowski. Genome compression: a novel approach for large collections. *Bioinformatics*, 29(20):2572–2578, 2013. doi:10.1093/BIOINFORMATICS/BTT460.

34 Sebastian Deorowicz and Szymon Grabowski. Robust relative compression of genomes with random access. *Bioinformatics*, 27(21):2979–2986, 2011. doi:10.1093/bioinformatics/btr505.

35 Bartlomiej Dudek, Paweł Gawrychowski, Garance Gourdel, and Tatiana Starikovskaya. Streaming regular expression membership and pattern matching. In *Proc. of SODA*, pages 670–694, 2022. doi:10.1137/1.9781611977073.30.

36 Jonas Ellert. Sublinear time Lempel-Ziv (LZ77) factorization. In *Proc. of SPIRE*, pages 171–187, 2023. doi:10.1007/978-3-031-43980-3_14.

37 Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Streaming periodicity with mismatches. In *Proc. of APPROX-RANDOM*, pages 42:1–42:21, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.42.

38 Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Periodicity in data streams with wildcards. In *Proc. of CSR*, pages 90–105, 2018. doi:10.1007/978-3-319-90530-3_9.

39 Funda Ergün, Hossein Jowhari, and Mert Saglam. Periodicity in streams. In *Proc. of APPROX-RANDOM*, volume 6302, pages 545–559. Springer, 2010. doi:10.1007/978-3-642-15369-3_41.

40 Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 2000. doi:10.1137/S0097539795280512.

41 Johannes Fischer, Travis Gagie, Paweł Gawrychowski, and Tomasz Kociumaka. Approximating LZ77 via small-space multiple-pattern matching. In *Proc. of ESA*, pages 533–544, 2015. Full version available at arXiv:1504.06647. doi:10.1007/978-3-662-48350-3_45.

42 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proc. of ESA*, pages 43:1–43:17, 2016. doi:10.4230/LIPIcs.ESA.2016.43.

43 Kimmo Fredriksson and Szymon Grabowski. Efficient algorithms for pattern matching with general gaps, character classes, and transposition invariance. *Inf. Retr.*, 11(4):335–357, 2008. doi:10.1007/S10791-008-9054-Z.

44 Kimmo Fredriksson and Szymon Grabowski. Average-optimal string matching. *J. Discrete Algorithms*, 7(4):579–594, 2009. doi:10.1016/j.jda.2008.09.001.

45 Kimmo Fredriksson and Szymon Grabowski. Nested counters in bit-parallel string matching. In *Proc. of LATA*, volume 5457, pages 338–349. Springer, 2009. doi:10.1007/978-3-642-00982-2_29.

46 Estéban Gabory, Moses Njagi Mwaniki, Nadia Pisanti, Solon P. Pissis, Jakub Radoszewski, Michelle Sweering, and Wiktor Zuba. Elastic-degenerate string comparison. *Inf. Comput.*, 304:105296, 2025. doi:10.1016/j.ic.2025.105296.

47 Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. *Theor. Comput. Sci.*, 379(3):405–417, 2007. doi:10.1016/J.TCS.2007.02.047.

48 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *Proc. of STACS*, pages 31:1–31:14, 2018. doi:10.4230/LIPIcs.STACS.2018.31.

49 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *Proc. of FST TCS*, pages 18:1–18:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.18.

50 Moses Ganardi, Danny Hucke, and Markus Lohrey. Randomized sliding window algorithms for regular languages. In *Proc. of ICALP*, pages 127:1–127:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.127.

51 Moses Ganardi, Danny Hucke, and Markus Lohrey. Sliding window algorithms for regular languages. In *Proc. of LATA*, pages 26–35, 2018. doi:10.1007/978-3-319-77313-1_2.

52 Moses Ganardi, Danny Hucke, Markus Lohrey, Konstantinos Mamouras, and Tatiana Starikovskaya. Regular languages in the sliding window model. *TheoretiCS*, 4, 2025. doi:10.46298/THEORETICS.25.8.

53 Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. In *Proc. of ISAAC*, pages 6:1–6:13, 2019. doi:10.4230/LIPIcs.ISAAC.2019.6.

54 Moses Ganardi, Artur Jeż, and Markus Lohrey. Sliding windows over context-free languages. In *Proc. of MFCS*, pages 15:1–15:15, 2018. doi:10.4230/LIPIcs.MFCS.2018.15.

55 Paweł Gawrychowski, Adam Górkiewicz, Pola Marciniak, Solon P. Pissis, and Karol Pokorski. Faster Approximate Elastic-Degenerate String Matching - Part B. In *Proc. of CPM*, pages 29:1–29:21, 2025. doi:10.4230/LIPIcs.CPM.2025.29.

56 Paweł Gawrychowski, Oleg Merkurev, Arseny M. Shur, and Przemyslaw 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.

57 Paweł Gawrychowski, Jakub Radoszewski, and Tatiana Starikovskaya. Quasi-periodicity in streams. In *Proc. of CPM*, pages 22:1–22:14, 2019. doi:10.4230/LIPIcs.CPM.2019.22.

58 Paweł Gawrychowski and Tatiana Starikovskaya. Streaming dictionary matching with mismatches. *Algorithmica*, 84(4):896–916, 2022. doi:10.1007/S00453-021-00876-X.

59 Taha El Ghazi and Tatiana Starikovskaya. Streaming periodicity with mismatches, wildcards, and edits. In *Proc. of ISAAC*, pages 36:1–36:20, 2025. doi:10.4230/LIPIcs.ISAAC.2025.36.

60 Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. The streaming k-mismatch problem: Tradeoffs between space and total time. In *Proc. of CPM*, pages 15:1–15:15, 2020. doi:10.4230/LIPIcs.CPM.2020.15.

61 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams. In *Proc. of ICALP*, pages 65:1–65:16, 2018. doi:10.4230/LIPIcs.ICALP.2018.65.

62 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.

63 Shay Golan and Ely Porat. Real-time streaming multi-pattern search for constant alphabet. In *Proc. of ESA*, pages 41:1–41:15, 2017. doi:10.4230/LIPIcs.ESA.2017.41.

64 Roberto Grossi, Costas S. Iliopoulos, Chang Liu, Nadia Pisanti, Solon P. Pissis, Ahmad Retha, Giovanna Rosone, Fatima Vayani, and Luca Versari. On-Line Pattern Matching on Similar Texts. In *Proc. of CPM*, volume 78, pages 9:1–9:14, 2017. doi:10.4230/LIPIcs.CPM.2017.9.

65 Tuukka Haapasalo, Panu Silvasti, Seppo Sippu, and Eljas Soisalon-Soininen. Online dictionary matching with variable-length gaps. In *Proc. of SEA*, pages 76–87, 2011. doi:10.1007/978-3-642-20662-7_7.

66 Wing-Kai Hon, Tsung-Han Ku, Rahul Shah, and Sharma V. Thankachan. Space-efficient construction algorithm for the circular suffix tree. In *Proc. of CPM*, pages 142–152, 2013. doi:10.1007/978-3-642-38905-4_15.

67 Wing-Kai Hon, Chen-Hua Lu, Rahul Shah, and Sharma V. Thankachan. Succinct indexes for circular patterns. In *Proc. of ISAAC*, pages 673–682, 2011. doi:10.1007/978-3-642-25591-5_69.

68 I.Lee, A.Apostolico, C.S.Iliopoulos, and K.Park. Finding approximate occurrences of a pattern that contains gaps. In *Proc. of AWOCA*, pages 89–100, 2003.

69 Costas S. Iliopoulos, Ritu Kundu, and Solon P. Pissis. Efficient pattern matching in elastic-degenerate strings. *Inf. Comput.*, 279:104616, 2021. doi:10.1016/j.ic.2020.104616.

70 Costas S. Iliopoulos, Solon P. Pissis, and M. Sohel Rahman. Searching and indexing circular patterns. In *Algorithms for Next-Generation Sequencing Data: Techniques, Approaches, and Applications*, pages 77–90. Springer, 2017. doi:10.1007/978-3-319-59826-0_3.

71 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987. doi:10.1147/RD.312.0249.

72 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *Proc. of STOC*, pages 756–767, 2019. doi:10.1145/3313276.3316368.

73 Dominik Kempa and Tomasz Kociumaka. Dynamic suffix array with polylogarithmic queries and updates. In *Proc. of STOC*, pages 1657–1670, 2022. doi:10.1145/3519935.3520061.

74 Dominik Kempa and Tomasz Kociumaka. Breaking the $O(n)$-barrier in the construction of compressed suffix arrays and suffix trees. In *Proc. of SODA*, pages 5122–5202, 2023. doi:10.1137/1.9781611977554.CH187.

75 Dominik Kempa and Tomasz Kociumaka. Lempel-Ziv (LZ77) factorization in sublinear time. In *Proc. of FOCS*, pages 2045–2055, 2024. doi:10.1109/FOCS61266.2024.00122.

76 Tomasz Kociumaka, Ely Porat, and Tatiana Starikovskaya. Small-space and streaming pattern matching with k edits. In *Proc. of FOCS*, pages 885–896, 2021. doi:10.1109/FOCS52979.2021.00090.

77 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Internal pattern matching queries in a text and applications. *SIAM J. Comput.*, 53(5):1524–1577, 2024. doi:10.1137/23M1567618.

78 Tomasz Kociumaka, Tatiana Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear space algorithms for the longest common substring problem. In *Proc. of ESA*, pages 605–617, 2014. doi:10.1007/978-3-662-44777-2_50.

79 Shanika Kuruppu, Simon J. Puglisi, and Justin Zobel. Relative lempel-ziv compression of genomes for large-scale storage and retrieval. In *Proc. of SPIRE*, pages 201–206, 2010. doi:10.1007/978-3-642-16321-0_20.

80 Xin Li and Yu Zheng. Lower bounds and improved algorithms for asymmetric streaming edit distance and longest common subsequence. In *Proc. of FST TCS*, pages 27:1–27:23, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.27.

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

82 Tung Mai, Anup Rao, Ryan A. Rossi, and Saeed Seddighin. Optimal space and time for streaming pattern matching, 2021. arXiv:2107.04660.

83 Oleg Merkurev and Arseny M. Shur. Searching long repeats in streams. In *Proc. of CPM*, pages 31:1–31:14, 2019. doi:10.4230/LIPIcs.CPM.2019.31.

84 Oleg Merkurev and Arseny M. Shur. Searching runs in streams. In *Proc. of SPIRE*, pages 203–220, 2019. doi:10.1007/978-3-030-32686-9_15.

85 Michele Morgante, Alberto Policriti, Nicola Vitacolonna, and Andrea Zuccolo. Structured motifs search. *J. Comput. Biol.*, 12(8):1065–1082, 2005. doi:10.1089/CMB.2005.12.1065.

86 Gonzalo Navarro, Victor Sepulveda, Mauricio Marín, and Senén González. Compressed filesystem for managing large genome collections. *Bioinformatics*, 35(20):4120–4128, 2019. doi:10.1093/BIOINFORMATICS/BTZ192.

87 Solon P. Pissis, Jakub Radoszewski, and Wiktor Zuba. Faster Approximate Elastic-Degenerate String Matching - Part A. In *Proc. of CPM*, pages 28:1–28:19, 2025. doi:10.4230/LIPIcs.CPM.2025.28.

88 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *Proc. of FOCS*, pages 315–323, 2009. doi:10.1109/FOCS.2009.11.

89 Petr Procházka, Ondrej Cvacho, Lubos Krcál, and Jan Holub. Backward pattern matching on elastic degenerate strings. In *Proc. of BIOSTEC, Volume 3: BIOINFORMATICS, Online Streaming*, pages 50–59. SCITEPRESS, 2021. doi:10.5220/0010243600500059.

90 Jakub Radoszewski and Tatiana Starikovskaya. Streaming k-mismatch with error correcting and applications. *Inf. Comput.*, 271:104513, 2020. doi:10.1016/j.ic.2019.104513.

91 Jakub Radoszewski and Wiktor Zuba. Computing string covers in sublinear time. In *Proc. of SPIRE*, pages 272–288, 2024. doi:10.1007/978-3-031-72200-4_21.

92 M. Sohel Rahman, Costas S. Iliopoulos, Inbok Lee, Manal Mohamed, and William F. Smyth. Finding patterns with variable length gaps or don't cares. In *Proc. of COCOON*, pages 146–155, 2006. doi:10.1007/11809678_17.

93 Barna Saha. Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems. In *Proc. of FOCS*, pages 118–135, 2015. doi:10.1109/FOCS.2015.17.

94 Michael E. Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proc. of SODA*, pages 1698–1709, 2013. doi:10.1137/1.9781611973105.122.

95 Tatiana Starikovskaya. Communication and streaming complexity of approximate pattern matching. In *Proc. of CPM*, pages 13:1–13:11, 2017. doi:10.4230/LIPIcs.CPM.2017.13.

96 Robert Susik, Szymon Grabowski, and Sebastian Deorowicz. Fast and simple circular pattern matching. In *Proc. of ICMMI (Man-Machine Interactions 3)*, pages 537–544, 2013. doi:10.1007/978-3-319-02309-0_59.

97 Daniel Valenzuela, Tuukka Norri, Niko Välimäki, Esa Pitkänen, and Veli Mäkinen. Towards pan-genome read alignment to improve variation calling. *BMC Genom.*, 19(S2), 2018. doi:10.1186/S12864-018-4465-8.

98 Andrew Chi-Chih Yao. Coherent functions and program checkers (extended abstract). In *Proc. of STOC*, pages 84–94, 1990. doi:10.1145/100216.100226.