On the Communication Complexity of Approximate Pattern Matching[*]

Tomasz Kociumaka
tomasz.kociumaka@mpi-inf.mpg.de
Max Planck Institute for Informatics,
Saarland Informatics Campus
Saarbrücken, Germany

Jakob Nogler
jakob.nogler@inf.ethz.ch
ETH Zurich
Zürich, Switzerland

Philip Wellnitz
wellnitz@mpi-inf.mpg.de
Max Planck Institute for Informatics,
Saarland Informatics Campus
Saarbrücken, Germany

ABSTRACT

The decades-old *Pattern Matching with Edits* problem, given a length-n string T (the text), a length-m string P (the pattern), and a positive integer k (the threshold), asks to list all fragments of T that are at edit distance at most k from P. The one-way communication complexity of this problem is the minimum amount of space needed to encode the answer so that it can be retrieved without accessing the input strings P and T.

The closely related Pattern Matching with Mismatches problem (defined in terms of the Hamming distance instead of the edit distance) is already well understood from the communication complexity perspective: Clifford, Kociumaka, and Porat [SODA 2019] proved that $\Omega(n/m \cdot k \log(m/k))$ bits are necessary and $O(n/m \cdot k \log(m|\Sigma|/k))$ bits are sufficient; the upper bound allows encoding not only the occurrences of P in T with at most k mismatches but also the substitutions needed to make each k-mismatch occurrence exact.

Despite recent improvements in the running time [Charalampopoulos, Kociumaka, and Wellnitz; FOCS 2020 and 2022], the communication complexity of Pattern Matching with Edits remained unexplored, with a lower bound of $\Omega(n/m \cdot k \log(m/k))$ bits and an upper bound of $O(n/m \cdot k^3 \log m)$ bits stemming from previous research. In this work, we prove an upper bound of $O(n/m \cdot k \log^2 m)$ bits, thus establishing the optimal communication complexity up to logarithmic factors. We also show that $O(n/m \cdot k \log m \log(m|\Sigma|))$ bits allow encoding, for each k-error occurrence of P in T, the shortest sequence of edits needed to make the occurrence exact. Our result further emphasizes the close relationship between Pattern Matching with Mismatches and Pattern Matching with Edits.

We leverage the techniques behind our new result on the communication complexity to obtain quantum algorithms for Pattern Matching with Edits: we demonstrate a quantum algorithm that uses $O(n^{1+o(1)}/m \cdot \sqrt{km})$ queries and $O(n^{1+o(1)}/m \cdot (\sqrt{km}+k^{3.5}))$ quantum time. Moreover, when determining the existence of at least one occurrence, the algorithm uses $O(\sqrt{n^{1+o(1)}/m} \cdot \sqrt{km})$ queries and $O(\sqrt{n^{1+o(1)}/m} \cdot (\sqrt{km} + k^{3.5}))$ time. For both cases, we establish corresponding lower bounds to demonstrate that the query complexity is optimal up to sub-polynomial factors.

CCS CONCEPTS

• **Theory of computation** → **Design and analysis of algorithms**; **Communication complexity**; **Pattern matching**; *Quantum complexity theory*.

KEYWORDS

Pattern Matching with Edits, Communication Complexity, Quantum Algorithms

ACM Reference Format:
Tomasz Kociumaka, Jakob Nogler, and Philip Wellnitz. 2024. On the Communication Complexity of Approximate Pattern Matching. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3618260.3649604

1 INTRODUCTION

While a *string* is perhaps the most basic way to represent data, this fact makes *algorithms* working on strings more applicable and powerful. Arguably, the very first thing to do with any kind of data is to find *patterns* in it. The *Pattern Matching* problem for strings and its variations are thus perhaps among the most fundamental problems that Theoretical Computer Science has to offer.

In this paper, we study the practically relevant *Pattern Matching with Edits* variation [33]. Given a text string T of length n, a pattern string P of length m, and a threshold k, the aim is to calculate the set $\mathrm{Occ}^E_k(P,T)$ consisting of (the starting positions of) all the fragments of T that are at most k edits away from the pattern P. In other words, we compute the set of k-error occurrences of P in T, more formally defined as

$$\mathrm{Occ}^E_k(P,T) \coloneqq \{i \in [\,0\,.\,.\,n\,] : \exists_{j\in[\,i\,.\,.\,n\,]} \delta_E(P, T[\,i\,.\,.\,j\,) \le k)\},$$

where we utilize the classical edit distance δ_E (also referred to as the Levenshtein distance) [32] as the distance measure. Here, an edit is either an insertion, a deletion, or a substitution of a single character.

Pattern Matching with Edits
Input: a pattern P of length m, a text T of length n, and an integer threshold $k > 0$.
Output: the set $\mathrm{Occ}^E_k(P,T)$.

Even though the Pattern Matching with Edits problem is almost as classical as it can get, with key algorithmic advances (from $O(mn)$ time down to $O(kn)$ time) dating back to the early and late

[*]The full version of the paper is available at https://arxiv.org/abs/2403.18812.

1980s [30, 31, 33], major progress has been made even very recently, when Charalampopoulos, Kociumaka, and Wellnitz [16] obtained an $\tilde{O}(n + k^{3.5}n/m)$-time[1] solution and thereby broke through the 20-years-old barrier of the $O(n + k^4 n/m)$-time algorithm by Cole and Hariharan [20]. And the journey is far from over yet: the celebrated Orthogonal-Vectors-based lower bound for edit distance [5] rules out only $O(n + k^{2-\Omega(1)}n/m)$-time algorithms (also consult [16] for details), leaving open a wide area of uncharted algorithmic territory. In this paper, we provide tools and structural insights that—we believe—will aid the exploration of the said territory.

We add to the picture a powerful new finding that sheds new light on the solution structure of the Pattern Matching with Edits problem—similar structural results [11, 15] form the backbone of the aforementioned breakthrough [16]. Specifically, we investigate how much space is needed to store all k-error occurrences of P in T. We know from [15] that $O(n/m \cdot k^3 \log m)$ bits suffice since one may report the occurrences as $O(k^3)$ arithmetic progressions if $n = O(m)$. However, such complexity is likely incompatible with algorithms running faster than $\tilde{O}(n + k^3 n/m)$. In this paper, we show that, indeed, $O(n/m \cdot k \log^2 m)$ bits suffice to represent the set $\mathrm{Occ}^E_k(P, T)$.

Formally, the communication complexity of Pattern Matching with Edits measures the space needed to encode the output so that it can be retrieved without accessing the input. We may interpret this setting as a two-party game: Alice is given an instance of the problem and constructs a message for Bob, who must be able to produce the output of the problem given Alice's message. Since Bob does not have any input, it suffices to consider one-way single-round communication protocols.

Main Theorem 1. *The Pattern Matching with Edits problem admits a one-way deterministic communication protocol that sends $O(n/m \cdot k \log^2 m)$ bits. Within the same communication complexity, one can also encode the family of all fragments of $T[\,i\,..\,j\,)$ that satisfy $\delta_E(P, T[\,i\,..\,j\,)) \le k$, as well as all optimal alignments $P \rightsquigarrow T[\,i\,..\,j\,)$ for each of these fragments. Further, increasing the communication complexity to $O(n/m \cdot k \log m \log(m|\Sigma|))$, where Σ denotes the input alphabet, one can also retrieve the edit information for each optimal alignment.*

Observe that our encoding scheme suffices to retrieve not only the set $\mathrm{Occ}^E_k(P, T)$ (which contains only starting positions of the k-error occurrences) but also the fragments of T with edit distance at most k from P. In other words, it allows retrieving all pairs $0 \le i \le j \le n$ such that $\delta_E(P, T[\,i\,..\,j\,)) \le k$.

We complement Main Theorem 1 with a simple lower bound that shows that our result is tight (essentially up to one logarithmic factor).

Main Theorem 2. *Fix integers n, m, k such that $n/2 \ge m > k > 0$. Every communication protocol for the Pattern Matching with Edits problem uses $\Omega(n/m \cdot k \log(m/k))$ bits for $P = \mathtt{0}^m$ and some $T \in \{\mathtt{0}, \mathtt{1}\}^n$.*

Observe that our lower bound holds for the very simple case that the pattern is the all-zeros string and only the text contains nonzero characters. In this case, the edit distance of the pattern and another string depends only on the length and the number of nonzero characters in the other string, and we can thus easily compute the edit distance in linear time.

From Structural Insights to Better Algorithms: A Success Story. Let us take a step back and review how structural results aided the development of approximate-pattern-matching algorithms in the recent past.

First, let us review the key insight of [15] that led to the breakthrough of [16]. Crucially, the authors use that, for any pair of strings P and T with $|T| \le {}^3\!/_2 \cdot |P|$ and threshold $k \ge 1$, either (a) P has at most $O(k^2)$ occurrences with at most k edits in T, or (b) P and the relevant part of T are at edit distance $O(k)$ to periodic strings with the same period. This insight helps as follows: First, one may derive that, indeed, all k-error occurrences of P in T form $O(k^3)$ arithmetic progressions. Second, it gives a blueprint for an algorithm: one has to tackle just two important cases: an easy *non-periodic* case, where P and T are highly unstructured and k-error occurrences are rare, and a not-so-easy periodic case, where P and T are highly repetitive and occurrences are frequent but appear in a structured manner.

The structural insights of [15] have found widespread other applications. For example, they readily yielded algorithms for differentially private approximate pattern matching [35], approximate circular pattern matching problems [13, 14, 17], and they even played a key role in obtaining small-space algorithms for (online) language distance problems [6], among others.

Interestingly, an insight similar to the one of [15] was first obtained in [11] for the much easier problem of Pattern Matching with Mismatches (where we allow neither insertions nor deletions) before being tightened and ported to Pattern Matching with Edits in [15]. Similarly, in this paper, we port a known communication complexity bound from Pattern Matching with Mismatches to Pattern Matching with Edits; albeit with a much more involved proof. As proved in [19], Pattern Matching with Mismatches problem admits a one-way deterministic $O(k \log(m|\Sigma|/k))$-bit communication protocol. While we discuss later (in the Technical Overview) the result of [19] as well as the challenges in porting it to Pattern Matching with Edits, let us highlight here that their result was crucial for obtaining an essentially optimal *streaming* algorithm for Pattern Matching with Mismatches.

Finally, let us discuss the future potential of our new structural results. First, as a natural generalization of [19], $\hat{O}(k)$-space algorithms for Pattern Matching with Edits should be plausible in the semi-streaming and (more ambitiously) streaming models, because $\hat{O}(k)$-size edit distance sketches have been developed in parallel to this work [29]. Nevertheless, such results would also require $\hat{O}(k)$-space algorithms constructing sketches and recovering the edit distance from the two sketches, and [29] does not provide such space-efficient algorithms. Second, our result sheds more light on the structure of the non-periodic case of [15]: as it turns out, when relaxing the notion of periodicity even further, we obtain a periodic structure also for patterns with just a (sufficiently large) constant number of k-error occurrences. This opens up a perspective for classical Pattern Matching with Edits algorithms that are even faster than $\tilde{O}(n/m + k^3)$.

[1]The $\tilde{O}(\cdot)$ and $\hat{O}(\cdot)$ notations suppress factors poly-logarithmic and sub-polynomial in the input size $n + m$, respectively.

Application of our Main Result: Quantum Pattern Matching with Edits. As a fundamental problem, Pattern Matching with Edits has been studied in a plethora of settings, including the compressed setting [9, 15, 23, 36], the dynamic setting [15], and the streaming setting [8, 28, 34], among others. However, so far, the *quantum setting* remains vastly unexplored. While quantum algorithms have been developed for Exact Pattern Matching [26], Pattern Matching with Mismatches [27], Longest Common Factor (Substring) [2, 22, 27], Lempel–Ziv factorization [24], as well as other fundamental string problems [1, 4, 10, 18, 37], no quantum algorithm for Pattern Matching with Edits has been known so far. The challenge posed by Pattern Matching with Edits, in comparison to Pattern Matching with Mismatches, arises already from the fact that, while the computation of Hamming distance between two strings can be easily accelerated in the quantum setting, the same is not straightforward for the edit distance case. Only very recently, Gibney, Jin, Kociumaka, and Thankachan [24] demonstrated a quantum edit-distance algorithm with the optimal query complexity of $\tilde{O}(\sqrt{kn})$ and the time complexity of $\tilde{O}(\sqrt{kn}+k^2)$.

We follow the long line of research on quantum algorithms on strings and employ our new structural results (combined with the structural results from [15]) to obtain the following quantum algorithms for the Pattern Matching with Edits problem.

Main Theorem 3. *Let P denote a pattern of length m, let T denote a text of length n, and let $k > 0$ denote an integer threshold.*

(1) *There is a quantum algorithm that solves the Pattern Matching with Edits problem. The algorithm uses $\hat{O}(n/m\cdot\sqrt{km})$ queries and $\hat{O}(n/m\cdot(\sqrt{km}+k^{3.5}))$ time.*
(2) *There is a quantum algorithm deciding whether $\mathrm{Occ}^E_k(P,T)\neq\varnothing$. The algorithm uses $\hat{O}(\sqrt{n/m}\cdot\sqrt{km})$ queries and $\hat{O}(\sqrt{n/m}\cdot(\sqrt{km}+k^{3.5}))$ time.*

Surprisingly, for $n = O(m)$, we achieve the same query complexity as quantum algorithms for computing the (bounded) edit distance [24] and even the bounded Hamming distance of strings (a simple application of Grover search yields an $\tilde{O}(\sqrt{kn})$ upper bound; a matching $\Omega(\sqrt{kn})$ lower bound is also known [7]). While we did not optimize the time complexity of our algorithms (reasonably, one could expect a time complexity of $\tilde{O}(n/m\cdot(\sqrt{km}+k^{3.5}))$ based on our structural insights and [16]), we show that our query complexity is essentially optimal by proving a matching lower bound.

Main Theorem 4. *Let us fix integers $n \geq m > k > 0$.*

(1) *Every quantum algorithm that solves the Pattern Matching with Edits problem uses $\Omega(n/m\cdot\sqrt{k(m-k)})$ queries for $P = \mathtt{0}^m$ and some $T\in\{\mathtt{0},\mathtt{1}\}^n$.*
(2) *Every quantum algorithm that decides whether $\mathrm{Occ}^E_k(P,T)\neq\varnothing$ uses $\Omega(\sqrt{n/m}\cdot\sqrt{k(m-k)})$ queries for $P = \mathtt{0}^m$ and some $T\in\{\mathtt{0},\mathtt{1}\}^n$.*

Again, our lower bounds hold already for the case when the pattern is the all-zeroes string and just the text contains nonzero entries.

2 TECHNICAL OVERVIEW

In this section, we describe the technical contributions behind our positive results: Main Theorems 1 and 3. We assume that $n\leq {}^3\!/_2\, m$ (if the text is longer, one may split the text into $O(n/m)$ overlapping pieces of length $O(m)$ each) and that $k = o(m)$ (for $k = \Theta(m)$, our results trivialize). Due to space constraints, we defer the proofs and the technical details to the full version.

2.1 Communication Complexity of Pattern Matching with Mismatches

Before we tackle Main Theorem 1, it is instructive to learn how to prove an analogous result for Pattern Matching with Mismatches. Compared to the original approach of Clifford, Kociumaka, and Porat [19], we neither optimize logarithmic factors nor provide an efficient decoding algorithm; this enables significant simplifications. Recall that our goal is to encode the set $\mathrm{Occ}^H_k(P,T)$, which is the Hamming-distance analog of the set $\mathrm{Occ}^E_k(P,T)$. Formally, we set

$$\mathrm{Occ}^H_k(P,T) \coloneqq \{i\in[\,0\,..\,n-m\,] : \delta_H(P,T[\,i\,..\,i+m\,)) \leq k\}.$$

Without loss of generality, we assume that $\{0,n-m\}\subseteq\mathrm{Occ}^H_k(P,T)$, that is, P has k-mismatch occurrences both as a prefix and as a suffix of T. Otherwise, either we have $\mathrm{Occ}^H_k(P,T)=\varnothing$ (which can be encoded trivially), or we can crop T by removing the characters to the left of the leftmost k-mismatch occurrence and to the right of the rightmost k-mismatch occurrence.

Encoding All k-Mismatch Occurrences. First, if $k=0$, as a famous consequence of the Periodicity Lemma [21], the set

$$\mathrm{Occ}^H_0(P,T)=\mathrm{Occ}(P,T)$$

is guaranteed to form a single arithmetic progression (recall that $n\leq {}^3\!/_2\, m$), and thus it can be encoded using $O(\log m)$ bits. Consult Figure 1 for a visualization of an example.

If $k>0$, the set $\mathrm{Occ}^H_k(P,T)$ does not necessarily form an arithmetic progression. Still, we may consider the smallest arithmetic progression that contains $\mathrm{Occ}^H_k(P,T)$ as a subset. Since we have $0\in\mathrm{Occ}^H_k(P,T)$, the difference of this progression can be expressed as $g\coloneqq\gcd(\mathrm{Occ}^H_k(P,T))$.

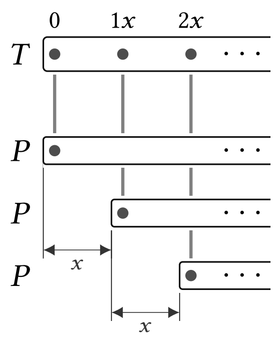
A crucial property of the $\gcd(\cdot)$ function is that, as we add elements to a set maintaining its greatest common divisor g, each insertion either does not change g (if the inserted element is already a multiple of g) or results in the value g decreasing by a factor of at least 2 (otherwise). Consequently, there is a set $\{0,n-m\}\subseteq S\subseteq\mathrm{Occ}^H_k(P,T)$ of size $|S|=O(\log m)$ such that $\gcd(S)=\gcd(\mathrm{Occ}^H_k(P,T))=g$.

The encoding that Alice produces consists of the set S with each k-mismatch occurrences $i\in S$ augmented with the *mismatch information* for P and $T[\,i\,..\,i+m\,)$, that is, a set

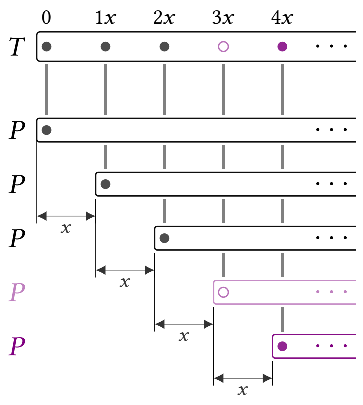
$$\{(j,P[j],T[i+j]) : j\in[\,0\,..\,m\,) \text{ such that } P[j]\neq T[i+j]\}.$$

For a single k-mismatch occurrence, the mismatch information can be encoded in $O(k\log(m|\Sigma|))$ bits, where Σ is the alphabet of P and T. Due to $|S|=O(\log m)$, the overall encoding size is $O(k\log m\log(m|\Sigma|))$.

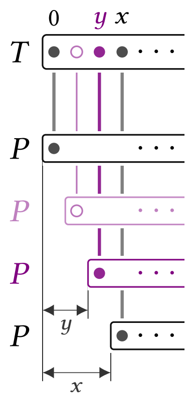
Recovering the k-Mismatch Occurrences. It remains to argue that the encoding is sufficient for Bob to recover $\mathrm{Occ}^H_k(P,T)$. To that end, consider a graph $\mathbf{G}_S$ whose vertices correspond to characters in P and T. For every $i\in S$ and $j\in[\,0\,..\,m\,)$, the graph $\mathbf{G}_S$ contains an edge between $P[j]$ and $T[i+j]$. If $P[j]=T[i+j]$, then the edge



(a) The pattern P occurs in T starting at the positions 0, x, and $2x$; these starting positions form the arithmetic progression $(ix)_{0\le i\le 2}$.



(b) Suppose that we were to identify an additional occurrence of P in T starting at position $4x$. Now, since occurrences start at $0, 2x$, and $4x$ (which in particular implies that $T[\,0\,..\,2x+|P|\,) = T[\,2x\,..\,4x+|P|\,)$), as well as at position x, we directly obtain that there is also an occurrence that starts at position $3x$ in T; which means that the arithmetic progression from Figure 1a is extended to $(ix)_{0\le i\le 4}$. More generally, one may prove that any additional occurrence at a position ix extends the existing arithmetic progression in a similar fashion.



(c) Suppose that we were to identify an additional occurrence of P in T starting at position $0 < y < x$. Now, similarly to Figure 1b, we can argue that there is also an occurrence that starts at every position of the form $i\gcd(x, y)$ (this is a consequence of the famous Periodicity Lemma due to [21])—again an arithmetic progression. Crucially, the difference of the arithmetic progression obtained in this fashion decreased by a factor of at least two compared to the initial arithmetic progression.

Figure 1: The structure of occurrences of exact pattern matching is easy: either all exact occurrences of P in T form an arithmetic progression or there is just one such occurrence (which we may also view as a degenerate arithmetic progression). Depicted is a text T and exact occurrences starting at the positions denoted above the text; we may assume that there is an occurrence that starts at position 0 and that there is an occurrence that ends at position $|T| - 1$.

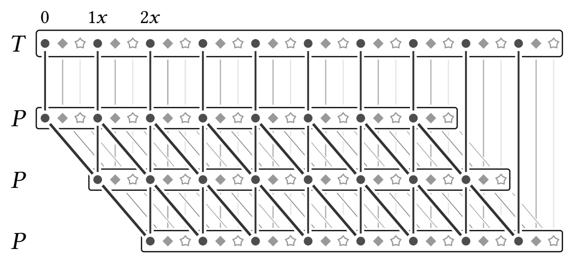
is *black*; otherwise, the edge is *red* and annotated with the values $P[j] \ne T[i + j]$. Observe that Bob can reconstruct $\mathbf{G}_S$ using the set S and the mismatch information for the k-mismatch occurrences at positions $i \in S$.

Next, we focus on the connected components of the graph $\mathbf{G}_S$. We say that a component is black if all of its edges are black and red if it contains at least one red edge. Observe that Bob can reconstruct the values of all characters in red components: the annotations already provide this information for vertices incident to red edges, and since black edges connect matching characters, the values can be propagated along black edges, ultimately covering all vertices in red components. The values of characters in black components remain unknown, but each black component is guaranteed to be *uniform*, meaning that every two characters in a single black component match.

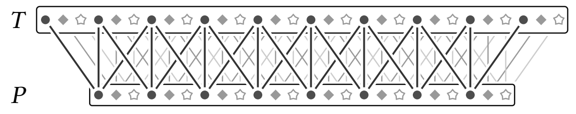
The last crucial observation is that the connected components of $\mathbf{G}_S$ are very structured: for every remainder $c \in [\,0\,..\,g\,)$ modulo g, there is a connected component consisting of all vertices $P[i]$ and $T[i]$ with $i \equiv_g c$. This can be seen as a consequence of the Periodicity Lemma [21] applied to strings obtained from P and T by replacing each character with a unique identifier of its connected component. Consult Figure 2 for an illustration of an example for the special case if there are no mismatches and consult Figure 3 for a visualization of an example with mismatches.

Testing if an Occurrences Starts at a Given Position. With these ingredients, we are now ready to explain how Bob tests whether a given position $i \in [\,0\,..\,n-m\,]$ belongs to $\mathrm{Occ}^H_k(P,T)$. If i is not divisible by g, then for sure $i \notin \mathrm{Occ}^H_k(P,T)$. Otherwise, for every $j \in [\,0\,..\,m\,)$, the characters $P[j]$ and $T[i+j]$ belong to the same connected component. If this component is red, then Bob knows the values of $P[j]$ and $T[i+j]$, so he can simply check if the characters match. Otherwise, the component is black, meaning that $P[j]$ and $T[i + j]$ are guaranteed to match. As a result, Bob can compute the Hamming distance $\delta_H(P, T[\,i\,..\,i+m\,))$ and check if it does not exceed k. In either case (as long as i is divisible by g), he can even retrieve the underlying mismatch information.

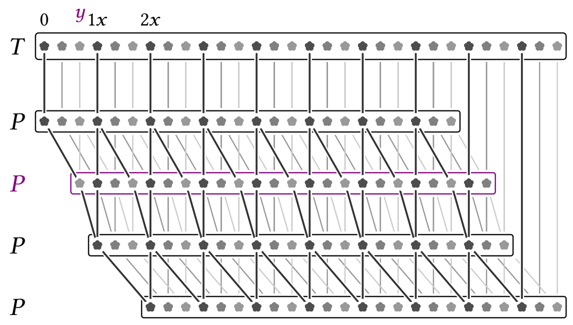
A convenient way of capturing Bob's knowledge about P and T is to construct auxiliary strings $P^{\#}$ and $T^{\#}$ obtained from P and T, respectively, by replacing all characters in each black component with a sentinel character (unique for the component). Then, $\mathrm{Occ}^H_k(P,T) = \mathrm{Occ}^H_k(P^{\#},T^{\#})$ and the mismatch information is preserved for the k-mismatch occurrences.



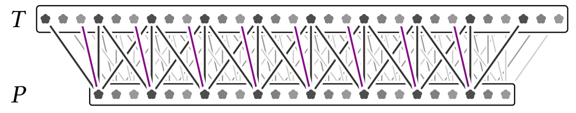
(a) Compare Figure 1a. So far, we identified three occurrences of P in T; each occurrence is an exact occurrence. Correspondingly, we have $S = \{(0,\varnothing),(x,\varnothing),(2x,\varnothing)\}$.
With this set S, we obtain three different black components, which we depict with a circle, a diamond, or a star.



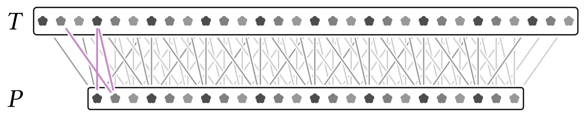
(b) The graph G_S that corresponds to Figure 2a: observe how we collapsed the different patterns from Figure 2a into a single pattern P.
In the example, we have three black components, that is, $\mathrm{bc}(G_S) = 3$.



(c) Suppose that we were to identify an additional occurrence of P in T starting at position $0 < y < x$ (highlighted in purple). From Figure 1c, we know how the set of all occurrences changes, but—and this is the crucial point— we do not add all of these implicitly found occurrences to S, but just y.
In our example, we observe that the black components collapse into a single black component, which we depict with a cloud.



(d) The graph G_S that corresponds to Figure 2c: observe how we collapsed the different patterns from Figure 2c into a single pattern P. Highlighted in purple are some of the edges that we added due to the new occurrence that we added to S.
In the example, we have one black components, that is, $\mathrm{bc}(G_S) = 1$.



(e) Recovering an occurrence in G_S from Figure 2d that starts at position $\gcd(x, y)$, illustrated for the first character of the pattern.

Figure 2: Compare Figure 1: we fully understand the easy structure of exact pattern matching. In this figure, we reinterpret our knowledge in terms of the encoding scheme of Alice for Pattern Matching with Mismatches (in particular we show just the occurrences included in the set S) and showcase how the corresponding graph G_S and its black components evolve.
We connect the same positions in P, as well as pairs of positions that are aligned by an occurrence of P in T. As there are no mismatches, every such line implies that the connected characters are equal.
For each connected component of the resulting graph (a black component), we know that all involved positions in P and T must have the same symbol. For illustrative purposes, we assume that $x = 3$ and we replace each character of a black component with a sentinel character (unique to that component), that is, we depict the strings $P^\#$ and $T^\#$.

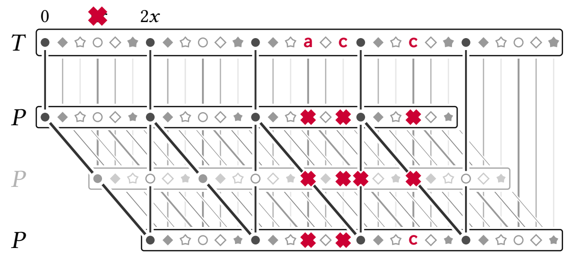
2.2 Communication Complexity of Pattern Matching with Edits

On a very high level, our encoding for Pattern Matching with Edits builds upon the approach for Pattern Matching with Mismatches presented above:

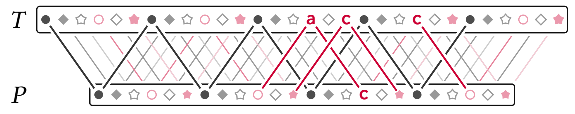
- Alice still constructs an appropriate size-$O(\log m)$ set S of k-error occurrences of P in T, including a prefix and a suffix of T.
- Bob uses the edit information for the occurrences in S to construct a graph G_S and strings $P^\#$ and $T^\#$, obtained from P and T by replacing characters in some components with sentinel characters so that $\mathrm{Occ}^E_k(P,T) = \mathrm{Occ}^E_k(P^\#,T^\#)$.

At the same time, the edit distance brings new challenges, so we also deviate from the original strategy:

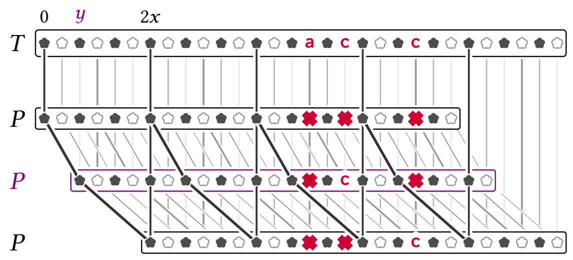
- Connected components of G_S do not have a simple periodic structure, so $g = \gcd(S)$ loses its meaning. Nevertheless, we



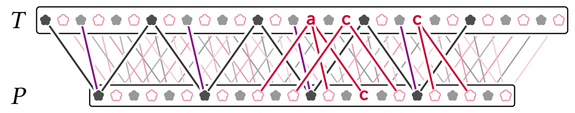
(a) Compare Figure 2a. We depict mismatched characters in an alignment of P to T by placing a cross over the corresponding character in P.
If we allow at most 3 mismatches, we now do not have an occurrence starting at position x anymore; hence we obtain six black components.



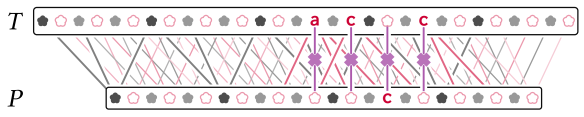
(b) The graph G_S that correspond to Figure 3a. We make explicit characters that are different from the "default" character of a component; the corresponding red edges (that are highlighted) are exactly the mismatch information that is stored in S. For the remaining edges, the color depicts the color of the connected component that they belong to.
In the example, we have four black components, that is, $\mathrm{bc}(G_S) = 4$. (Observe that contrary to what the image might make you believe, not every "non-default" character needs to end in a highlighted red edge.)



(c) Compare Figure 2c. We are still able to identify an additional occurrence of P in T starting at position $0 < y < x$ (highlighted in purple). Now, as before, connected components of G_S merge; this time, this also means that some characters that were previously part of a black component now become part of a red component (but crucially never vice-versa).
In the example, this means that we now have just a single black component, that is, $\mathrm{bc}(G_S) = 1$.



(d) The graph G_S for the situation in Figure 3c. Again, we make explicit characters that are different from the "default" character of a component; the corresponding red edges (that are highlighted) are exactly the mismatch information that is stored in S. For the remaining edges, the color depicts the color of the connected component that they belong to (where purple highlights some of the black edges added due to the new occurrence).



(e) Checking for an occurrence at position $2\gcd(x, y)$ (which would be an occurrence were it not for mismatched characters). We check two things, first that the black component aligns; and second, for the red component where we know all characters, we compute exactly the Hamming distance (which is 4 in the example, meaning that there is no occurrence at the position in question).

Figure 3: Compared to Figure 2, we now have characters in P and T that mismatch. Again, we showcase how the corresponding graph G_S and its black components evolve; in the example, we allow for up to $k = 3$ mismatches.
Again, for illustrative purposes, we assume that $x = 3$ and we replace each character of a black component with a sentinel character (unique to that component), that is, we depict the strings $P^{\#}$ and $T^{\#}$.

prove that black components still behave in a structured way, and thus the number of black components, denoted $\mathrm{bc}(G_S)$, can be used instead.

- The value $\mathrm{bc}(G_S)$ is not as easy to compute as $\gcd(S)$, so we grow the set $S \subseteq \mathrm{Occ}^E_k(P, T)$ iteratively. In each step, either we add a single k-error occurrence so that $\mathrm{bc}(G_S)$ decreases by a factor of at least 2, or we realize that the information related to the alignments already included in S suffices to retrieve all k-error occurrences of P in T.
- Once this process terminates, there may unfortunately remain k-error occurrences whose addition to S would decrease $\mathrm{bc}(G_S)$—yet, only very slightly. In other words, such k-error occurrences generally obey the structure of black components, but may occasionally violate it. We need to

understand where the latter may happen and learn the characters behind the black components involved so that they are not masked out in $P^{\#}$ and $T^{\#}$. This is the most involved part of our construction, where we use recent insights relating edit distance to compressibility [12, 24] and store compressed representations of certain fragments of T.

2.2.1 General Setup. Technically, the set S that Alice constructs contains, instead of k-error occurrences $T[\,t\,..\,t'\,)$, specific alignments $P \rightsquigarrow T[\,t\,..\,t'\,)$ of cost at most k. Every such alignment describes a sequence of (at most k) edits that transform P onto $T[\,t\,..\,t'\,)$; see the full version for details. In the message that Alice constructs, each alignment is augmented with *edit information*, which specifies the positions and values of the edited characters; again, see the full version for details. For a single alignment of cost k, this information takes $O(k\log(m|\Sigma|))$ bits, where Σ is the alphabet of P and T.

Just like for Pattern Matching with Mismatches, we can assume without loss of generality that P has k-error occurrences both as a prefix or as a suffix of T. Consequently, we always assume that S contains an alignment $\mathcal{X}_{\text{pref}}$ that aligns P with a prefix of T and an alignment $\mathcal{X}_{\text{suf}}$ that aligns P with a suffix of T.

The graph $\mathbf{G}_S$ is constructed similarly as for mismatches: the vertices are characters of P and T, whereas the edges correspond to pairs of characters aligned by any alignment in S. Matched pairs of characters correspond to black edges, whereas substitutions correspond to red edges, annotated with the values of the mismatching characters. Insertions and deletions are also captured by red edges; see the full version for details.

Again, we classify connected components of $\mathbf{G}_S$ into black (with black edges only) and red (with at least one red edge). Observe that Bob can reconstruct the graph $\mathbf{G}_S$ and the values of all characters in red components and that black components remain *uniform*, that is, every two characters in a single black component match. Consult Figure 4 for a visualization of an example.

Finally, we define $\text{bc}(\mathbf{G}_S)$ to be the number of black components in $\mathbf{G}_S$. If $\text{bc}(\mathbf{G}_S) = 0$, then Bob can reconstruct the whole strings P and T, so we henceforth assume $\text{bc}(\mathbf{G}_S) > 0$.

First Insights into $\mathbf{G}_S$. Our first notable insight is that black components exhibit periodic structure. To that end, write $P_{|S}$ for the subsequence of P that contains all characters of P that are contained in a black component in $\mathbf{G}_S$ and write $T_{|S}$ for the subsequence of T that contains all characters of T that are contained in a black component in $\mathbf{G}_S$. Then, for every $c \in [\,0\,..\,\text{bc}(\mathbf{G}_S)\,)$, there is a component consisting of all characters $P_{|S}[i]$ and $T_{|S}[i]$ such that $i \equiv_{\text{bc}(\mathbf{G}_S)} c$; for a formal statement and proof, consult the full version. Also consult Figure 4c for an illustration of an example.

Next, we denote the positions in P and T of the subsequent characters of $P_{|S}$ and $T_{|S}$ belonging to a specific component $c \in [\,0\,..\,\text{bc}(\mathbf{G}_S)\,)$ as $\pi_0^c, \pi_1^c, \ldots$ and $\tau_0^c, \tau_1^c, \ldots$, respectively. The characterization of the black components presented above implies that $\pi_j^c < \pi_{j'}^{c'}$ if and only if either $j < j'$ or $j = j'$ and $c < c'$ (analogously for $\tau_j^c < \tau_{j'}^{c'}$). We assume that the cth black component contains m_c characters of P and n_c characters in T; note that $m_c \in \{m_0, m_0 - 1\}$ and $n_c \in \{n_0, n_0 - 1\}$.

2.2.2 Extra Information to Capture Close Alignments. By definition of the graph $\mathbf{G}_S$, the alignments in S obey the structure of the black components. Specifically, for every $\mathcal{X} \in S$, there is a shift $i \in [\,0\,..\,n_0 - m_0\,]$ such that $\mathcal{X}$ matches $P[\pi_j^c]$ with $T[\tau_{i+j}^c]$ for every $c \in [\,0\,..\,\text{bc}(\mathbf{G}_S)\,)$ and $j \in [\,0\,..\,m_c\,)$. The quasi-periodic structure of P and T suggests that we should expect further shifts $i \in [\,0\,..\,n_0 - m_0\,]$ with low-cost alignments matching $P[\pi_j^c]$ with $T[\tau_{i+j}^c]$ for every $c \in [\,0\,..\,\text{bc}(\mathbf{G}_S)\,)$ and $j \in [\,0\,..\,m_c\,)$. Unfortunately, even if an optimum alignment $\mathcal{X} : P \rightsquigarrow T[\,t\,..\,t'\,)$ matches $P[\pi_j^c]$ with $T[\tau_{i+j}^c]$, there is no guarantee that it also matches $P[\pi_{j'}^{c'}]$ with $T[\tau_{i+j'}^{c'}]$ for other values $c' \in [\,0\,..\,\text{bc}(\mathbf{G}_S)\,)$ and $j' \in [\,0\,..\,m_{c'}\,)$. Even worse, it is possible that no optimal alignment $P[\,\pi_j^{c-1}\,..\,\pi_j^{c+1}\,) \rightsquigarrow T[\,\tau_{i+j}^{c-1}\,..\,\tau_{i+j}^{c+1}\,)$ matches $P[\pi_j^c]$ with $T[\tau_{i+j}^c]$. The reason behind this phenomenon is that the composition of optimal edit-distance alignments is not necessarily optimal (more generally, the edit information of optimal alignments $X \rightsquigarrow Y$ and $Y \rightsquigarrow Z$ is insufficient to recover $\delta_E(X, Z)$).

In these circumstances, our workaround is to identify a set $C_S \subseteq [\,0\,..\,\text{bc}(\mathbf{G}_S)\,)$ such that the underlying characters can be encoded in $\tilde{O}(k|S|)$ space and every alignment $\mathcal{X} : P \rightsquigarrow T[\,t\,..\,t'\,)$ that we need to capture matches $P[\pi_j^c]$ with $T[\tau_{i+j}^c]$ for every $c \in [\,0\,..\,\text{bc}(\mathbf{G}_S)\,) \setminus C_S$ and $j \in [\,0\,..\,m_c\,)$. For this, we investigate how an optimal alignment $\mathcal{X} : P \rightsquigarrow T[\,t\,..\,t'\,)$ may differ from a canonical alignment $\mathcal{A} : P \rightsquigarrow T[\,t\,..\,t'\,)$ that matches $P[\pi_j^c]$ with $T[\tau_{i+j}^c]$ for all $c \in [\,0\,..\,\text{bc}(\mathbf{G}_S)\,)$ and $j \in [\,0\,..\,m_c\,)$. Following recent insights from [12, 24], we observe that the fragments of P on which $\mathcal{A}$ and $\mathcal{X}$ are disjoint can be compressed into $O(\delta_E^{\mathcal{A}}(P, T[\,t\,..\,t'\,)))$ space (using Lempel–Ziv factorization [38], for example). Moreover, the compressed size of each of these fragments is at most proportional to the cost of $\mathcal{A}$ on the fragment. Consequently, our goal is to understand where $\mathcal{A}$ makes edits and learn all the fragments of P (and T) with a sufficiently high density of edits compared to the compressed size. Due to the quasi-periodic nature of P and T, for each $c \in [\,0\,..\,\text{bc}(\mathbf{G}_S)\,)$, all characters in the cth black component are equal to $T[\tau_0^c]$, so we can focus on learning fragments of $T[\,\tau_0^0\,..\,\tau_0^{\text{bc}(\mathbf{G}_S)-1}\,]$.

The bulk of the alignment $\mathcal{A}$ can be decomposed into pieces that align $P[\,\pi_j^c\,..\,\pi_j^{c+1}\,)$ onto $T[\,\tau_{i+j}^c\,..\,\tau_{i+j}^{c+1}\,)$. In the full version, we prove that $\delta_E(P[\,\pi_j^c\,..\,\pi_j^{c+1}\,), T[\,\tau_{i+j}^c\,..\,\tau_{i+j}^{c+1}\,)) \le \text{w}_S(c)$, where $\text{w}_S(c)$ is the total cost incurred by alignments in S on all fragments $P[\,\pi_{j'}^c\,..\,\pi_{j'}^{c+1}\,)$ for $j' \in [\,0\,..\,m_c\,)$. Intuitively, this is because the path from $P[\pi_j^c]$ to $T[\tau_{i+j}^c]$ in $\mathbf{G}_S$ allows us to obtain an alignment

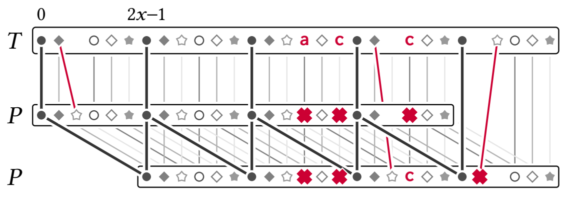
$$P[\,\pi_j^c\,..\,\pi_j^{c+1}\,) \rightsquigarrow T[\,\tau_{i+j}^c\,..\,\tau_{i+j}^{c+1}\,)$$

as a composition of pieces of alignments in S and their inverses. Every component $c \in [\,0\,..\,\text{bc}(\mathbf{G}_S)\,)$ uses distinct pieces, so the total weight $w := \sum_c \text{w}_S(c)$ does not exceed $k \cdot |S|$.

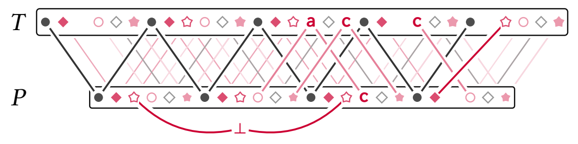
The weight function $\text{w}_S(c)$ governs which characters of

$$T[\,\tau_0^0\,..\,\tau_0^{\text{bc}(\mathbf{G}_S)-1}\,]$$

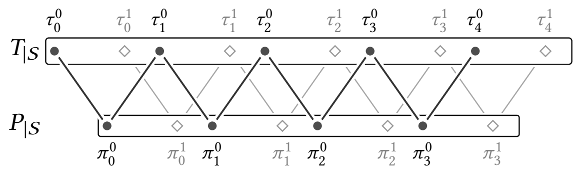
we need to learn. In the full version, we formalize this with a notion of a *period cover* $C_S \subseteq [\,0\,..\,\text{bc}(\mathbf{G}_S)\,)$. Most importantly, we require that $[\,a\,..\,b\,] \subseteq C_S$ holds whenever the compressed size of $T[\,\tau_0^a\,..\,\tau_0^b\,]$ is smaller than the total weight $\sum_{c=a-1}^{b} \text{w}_S(c)$ (scaled



(a) Compare Figure 3a. In addition to mismatched characters, we now also have missing characters in P and T (depicted by a white space). Further, as alignments for occurrences are no longer unique, we have to choose an alignment for each occurrence in the set S (which can fortunately be stored efficiently).



(b) The graph $\mathbf{G}_S$ that corresponds to the situation in Figure 3a. Observe that now, we also have a sentinel vertex $\perp$ to represent that an insertion or deletion happened. Observe further that due to insertions and deletions, the last empty star character of T now belongs to the component of filled diamonds.
In the example, we have two black components, that is, $\mathrm{bc}(\mathbf{G}_S) = 2$.



(c) An illustration of the additional notation that we use to analyze $\mathbf{G}_S$. Removing every character involved in a red component, we obtain the strings $T_{|S}$ and $P_{|S}$. For each black component, we number the corresponding characters in P and T from left to right.

Figure 4: Compare Figures 2 and 3. In addition to mismatches, we now also allow character insertions or deletions. In the example, we depict occurrence with at most $k = 4$ edits.

up by an appropriate constant factor). Additionally, to handle corner cases, we also learn the longest prefix and the longest suffix of $T[\,\tau_0^0 \mathinner{.\,.} \tau_0^{\mathrm{bc}(\mathbf{G}_S)-1}\,]$ of compressed size $O(w+k)$. As we prove in the full version, the set $\{(c, T[\tau_0^c]) : c \in C_S\}$ can be encoded in $O((w+k)\log(m|\Sigma|)) = O(k|S|\log(m|\Sigma|))$ bits on top of the graph $\mathbf{G}_S$ (which can be recovered from the edit information for alignments in S).

Following the aforementioned strategy of comparing the regions where $\mathcal{X} : P \rightsquigarrow T[\,t \mathinner{.\,.} t'\,)$ is disjoint with the canonical alignment $\mathcal{A} : P \rightsquigarrow T[\,t \mathinner{.\,.} t'\,)$, we prove the following result. Due to corner cases arising at the endpoints of $T[\,t \mathinner{.\,.} t'\,)$ and between subsequent fragments $T[\,\tau_{i+j}^0 \mathinner{.\,.} \tau_{i+j}^{\mathrm{bc}(\mathbf{G}_S)-1}\,]$ and $T[\,\tau_{i+j+1}^0 \mathinner{.\,.} \tau_{i+j+1}^{\mathrm{bc}(\mathbf{G}_S)-1}\,]$, the proof is rather complicated.

Proposition 2.1. *Let $\mathcal{X} : P \rightsquigarrow T[\,t \mathinner{.\,.} t'\,)$ be an optimal alignment of P onto a fragment $T[\,t \mathinner{.\,.} t'\,)$ such that $\delta_E(P, T[\,t \mathinner{.\,.} t'\,)) \le k$. If there exists $i \in [\,0 \mathinner{.\,.} n_0 - m_0\,]$ such that $|\tau_i^0 - t - \pi_0^0| \le w + 3k$, then the following holds for every $c \in [\,0 \mathinner{.\,.} \mathrm{bc}(\mathbf{G}_S)\,) \setminus C_S$:*

(1) $\mathcal{X}$ aligns $P[\pi_j^c]$ to $T[\tau_{i+j}^c]$ for every $j \in [\,0 \mathinner{.\,.} m_c\,)$, and
(2) $\tau_{i'}^c \notin [\,t \mathinner{.\,.} t'\,)$ for every $i' \in [\,0 \mathinner{.\,.} n_c\,) \setminus [\,i \mathinner{.\,.} i + m_c\,)$.

2.2.3 Extending S with Uncaptured Alignments. Proposition 2.1 indicates that S *captures* all k-error occurrences $T[\,t \mathinner{.\,.} t'\,)$ such that $|\tau_i^0 - t - \pi_0^0| \le w + 3k$ holds for some $i \in [\,0 \mathinner{.\,.} n_0 - m_0\,]$. As long as S does not capture some k-error occurrence $T[\,t \mathinner{.\,.} t'\,)$, we add an underlying optimal alignment $\mathcal{X} : P \rightsquigarrow T[\,t \mathinner{.\,.} t'\,)$ to the set S. In the full version, we prove that $\mathrm{bc}(\mathbf{G}_{S\cup\{\mathcal{X}\}}) \le \mathrm{bc}(\mathbf{G}_S)/2$ holds for such an alignment $\mathcal{X}$. For this, we first eliminate the possibility of $t + \pi_0^0 \gg \tau_{n_0-m_0}^0$ (using $\mathcal{X}_{\mathrm{suf}} \in S$, which matches $P[\pi_0^0]$ with $T[\tau_{n_0-m_0}^0]$). If $|\tau_i^0 - t - \pi_0^0| > w + 3k$ holds for every $i \in [\,0 \mathinner{.\,.} n_0\,)$, on the other hand, then there is no $c \in [\,0 \mathinner{.\,.} \mathrm{bc}(\mathbf{G}_S)\,)$ such that $P[\pi_0^c]$ can be matched with any character in the cth connected component. Consequently, each black component becomes red or gets merged with another black component, resulting in the claimed inequality $\mathrm{bc}(\mathbf{G}_{S\cup\{\mathcal{X}\}}) \le \mathrm{bc}(\mathbf{G}_S)/2$.

From $\mathrm{bc}(\mathbf{G}_{S\cup\{\mathcal{X}\}}) \le \mathrm{bc}(\mathbf{G}_S)/2$ and since $\mathrm{bc}(\mathbf{G}_S) \le m$ holds when we begin with $|S| = 2$, the total size $|S|$ does not exceed $O(\log m)$ before we either arrive at $\mathrm{bc}(\mathbf{G}_S) = 0$, in which case the whole input can be encoded in $O(k|S|\log(m|\Sigma|))$ bits, or S captures all k-error occurrences. In the latter case, the encoding consists of the edit information for all alignments in S, as well as the set $\{(c, T[\tau_0^c]) : c \in C_S\}$ which we know how to encode in $O(k|S|\log(m\Sigma))$ bits on top of the graph $\mathbf{G}_S$ (as we prove in the full version).

Based on this encoding, we can construct strings $P^\#$ and $T^\#$ obtained from P and T, respectively, by replacing with $\#_c$ every character in the cth connected component for every $c \in [\,0 \mathinner{.\,.} \mathrm{bc}(\mathbf{G}_S)\,) \setminus C_S$. As a relatively straightforward consequence of Proposition 2.1, we then prove that $\mathrm{Occ}_k^E(P, T) = \mathrm{Occ}_k^E(P^\#, T^\#)$ and that the edit information is preserved for every optimal alignment $P \rightsquigarrow T[\,t \mathinner{.\,.} t'\,)$ of cost at most k.

2.3 Quantum Query Complexity of Pattern Matching with Edits

As an illustration of the applicability of the combinatorial insights behind our communication complexity result (Main Theorem 1), we study quantum algorithms for Pattern Matching with Edits. As indicated in Main Theorems 3 and 4, the query complexity we achieve is only a sub-polynomial factor away from the unconditional lower bounds, both for the decision version of the problem (where we only need to decide whether $\mathrm{Occ}^E_k(P,T)$ is empty or not) and for the standard version asking to report $\mathrm{Occ}^E_k(P,T)$.

Our lower bounds (in Main Theorem 4) are relatively direct applications of the adversary method of Ambainins [3], so this overview is solely dedicated to the much more challenging upper bounds. Just like for the communication complexity above, we assume that $n \le {}^3/_2\, m$ and $k = o(m)$. In this case, our goal is to achieve the query complexity of $\hat{O}(\sqrt{km})$.

Our solution incorporates four main tools:

- the approximate pattern matching algorithm of [15],
- the recent quantum algorithm for computing (bounded) edit distance [24],
- the novel combinatorial insights behind Main Theorem 1,
- a new quantum $n^{o(1)}$-factor approximation algorithm for edit distance that uses $\hat{O}(\sqrt{n})$ queries and is an adaptation of a classic sublinear-time algorithm of [25].

2.3.1 Baseline Algorithm. We set the stage by describing a relatively simple algorithm that relies only on the first two of the aforementioned four tools. This algorithm makes $\tilde{O}(\sqrt{k^3 m})$ quantum queries to decide whether $\mathrm{Occ}^E_k(P,T) = \emptyset$.

The findings of [15] outline two distinct scenarios: either there are *few k-error occurrences of P in T* or the pattern is *approximately periodic*. In the former case, the set $\mathrm{Occ}^E_k(P,T)$ is of size $O(k^2)$, and it is contained in a union of $O(k)$ intervals of length $O(k)$ each. In the latter case, a primitive *approximate period* Q of small length $|Q| = O(m/k)$ exists such that P and the relevant portion of T (excluding the characters to the left of the leftmost k-error occurrence and to the right of the rightmost k-error occurrence) are at edit distance $O(k)$ to substrings of Q^∞. It is solely the pattern that determines which of these two cases holds: the initial two options in the following lemma correspond to the *non-periodic* case, where there are few k-error occurrences of P in T, whereas the third option indicates the (approximately) *periodic* case, where the pattern admits a short approximate period Q. Here, $\delta_E(S, {}^*Q^*)$ denotes the minimum edit distance between S and any substring of Q^∞.

Lemma 2.2 ([15, Lemma 5.19]). *Let P denote a string of length m and let $k \le m$ denote a positive integer. Then, at least one of the following holds:*

(a) The string P contains $2k$ disjoint fragments $B_1, \ldots, B_{2k}$ (called breaks*) each having period* $\mathrm{per}(B_i) > m/128k$ *and length* $|B_i| = \lfloor m/8k \rfloor$.

(b) The string P contains disjoint repetitive regions $R_1, \ldots, R_r$ *of total length $\sum_{i=1}^r |R_i| \ge 3/8 \cdot m$ such that each region R_i satisfies $|R_i| \ge m/8k$ and has a primitive* approximate period Q_i *with $|Q_i| \le m/128k$ and $\delta_E(R_i, {}^*Q_i^*) = \lceil 8k/m \cdot |R_i| \rceil$.*

(c) The string P has a primitive approximate period Q *with $|Q| \le m/128k$ and $\delta_E(P, {}^*Q^*) < 8k$.*

The proof of Lemma 2.2 is constructive, providing a classical algorithm that performs the necessary decomposition and identifies the specific case. The analogous procedure for Pattern Matching with Mismatches also admits an efficient quantum implementation [27] using $\tilde{O}(\sqrt{km})$ queries and time. As our first technical contribution, we adapt the decomposition algorithm for the edit case to the quantum setting so that it uses $\tilde{O}(\sqrt{km})$ queries and $\tilde{O}(\sqrt{km} + k^2)$ time.

Compared to the classic implementation in [15] and the mismatch version in [27], it is not so easy to efficiently construct repetitive regions. In this context, we are given a length-$\lfloor m/8k \rfloor$ fragment with exact period Q_i and the task is to extend it to R_i so that $k_i := \delta_E(R_i, {}^*Q_i^*)$ reaches $\lceil 8k/m \cdot |R_i| \rceil$. Previous algorithms use Longest Common Extension queries and gradually grow R_i, increasing k_i by one unit each time; this can be seen as an online implementation of the Landau–Vishkin algorithm for the bounded edit distance problem [30]. Unfortunately, the near-optimal quantum algorithm for bounded edit distance [24] is much more involved and does not seem amenable to an online implementation. To circumvent this issue, we apply exponential search (just like in Newton's root-finding method, this is possible even though the sign of $\lceil 8k/m \cdot |R_i| \rceil - \delta_E(R_i, {}^*Q_i^*)$ may change many times). At each step, we apply a slightly extended version of the algorithm of [24] that allows simultaneously computing the edit distance between R_i and multiple substrings of Q_i^∞; see the full version for details.

Once the decomposition has been computed, the next step is to apply the structure of the pattern in various cases to find the k-error occurrences. The fundamental building block needed here is a subroutine that *verifies* an interval I of $O(k)$ positive integers, that is, computes $\mathrm{Occ}^E_k(P,T) \cap I$. The aforementioned extension of the bounded edit distance algorithm of [24] allows implementing this operation using $\tilde{O}(\sqrt{km})$ quantum queries and $\tilde{O}(\sqrt{km} + k^2)$ time.

By directly following the approach of [15], computing $\mathrm{Occ}^E_k(P,T)$ can be reduced to verification of $O(k^2)$ intervals (the periodic case constitutes the bottleneck for the number of intervals), which yields total a query complexity of $\tilde{O}(\sqrt{k^5 m})$. If we only aim to decide whether $\mathrm{Occ}^E_k(P,T)$, we can apply Grover's search on top of the verification algorithm, reducing the query complexity to $\tilde{O}(\sqrt{k^3 m})$. One can also hope for further speed-ups based on the more recent results of [16], where the number of intervals is effectively reduced to $\tilde{O}(k^{1.5})$. Nevertheless, already in the non-periodic case, where the number of intervals is $O(k)$, this approach does not provide any hope of reaching query complexity beyond $\tilde{O}(\sqrt{k^2 m})$ for the decision version and $\tilde{O}(\sqrt{k^3 m})$ for the reporting version of Pattern Matching with Edits.

2.3.2 How to Efficiently Verify $\boldsymbol{O(k)}$ Candidate Intervals? As indicated above, the main bottleneck that we need to overcome to achieve the near-optimal query complexity is to verify $O(k)$ intervals using $\hat{O}(\sqrt{km})$ queries. Notably, an unconditional lower bound for bounded edit distance indicates that $\Omega(\sqrt{km})$ queries are already needed to verify a length-1 interval.

A ray of hope stemming from our insights behind Main Theorem 1 is that, as described in Section 2.2, already a careful selection of just $O(\log m)$ among the k-error occurrences reveals a lot of structure that can be ultimately used to recover the whole set $\mathrm{Occ}^E_k(P,T)$. To illustrate how to use this observation, let us initially make an unrealistic assumption that every candidate interval I contains a K-error occurrence for some $K = \hat{O}(k)$. Such occurrences can be detected using the existing verification procedure using $\tilde{O}(\sqrt{Km}) = \hat{O}(\sqrt{km})$ queries.

First, we verify the leftmost and the rightmost intervals. This allows finding the leftmost and the rightmost K-error occurrences of P in T. We henceforth assume that text T is cropped so that these two K-error occurrences constitute a prefix and a suffix of T, respectively. The underlying alignments are the initial elements of the set S that we maintain using the insights of Section 2.2. Even though these two alignments have cost at most K, for technical reasons, we subsequently allow adding to S alignments of cost up to $K' = K + O(k)$. Using the edit information for alignments $\mathcal{X} \in S$, we build the graph $\mathbf{G}_S$, calculate its connected components, and classify them as red and black components.

If there are no black components, that is, $\mathrm{bc}(\mathbf{G}_S) = 0$, then the edit information for the alignments $\mathcal{X} \in S$ allows recovering the whole input strings P and T. Thus, no further quantum queries are needed, and we complete the computation using a classical verification algorithm in $O(m + k^3)$ time.

If there are black components, we retrieve the positions

$$\pi^0_0, \ldots, \pi^0_{m_0-1} \quad \text{and} \quad \tau^0_0, \ldots, \tau^0_{n_0-1}$$

contained in the 0-th black component. Based on these positions, we can classify K'-error occurrences $T[\, t \,..\, t' \,)$ into those that are *captured* by S (for which $|\tau^0_i - \pi^0_0 - t|$ is small for some $i \in [\, 0 \,..\, n_0 - m_0 \,]$) and those which are not captured by S. Although we do not know K'-error occurrences other than those contained in S, the test of comparing $|\tau^0_i - \pi^0_0 - t|$ against a given threshold (which is $O(K'|S|)$) can be performed for any position t, and thus we can classify arbitrary positions $t \in [\, 0 \,..\, |T| \,]$ into those that are captured by S and those that are not.

If any of the candidate intervals I contains a position $t \in I$ that is not captured by S, we verify that interval and, based on our assumption, obtain a K-error occurrence of P in T that starts somewhere within I. Furthermore, we can derive an optimal alignment $\mathcal{X} : P \rightsquigarrow T[\, t \,..\, t' \,)$ whose cost does not exceed $K + |I| \le K'$ because $|I| = O(k)$. This K'-error occurrence is not captured by S, so we can add $\mathcal{X}$ to S and, as a result, the number of black components decreases at least twofold.

The remaining possibility is that S captures all positions t contained in the candidate intervals I. In this case, our goal is to construct strings $P^\#$ and $T^\#$, which are guaranteed to satisfy

$$\mathrm{Occ}^E_k(P, T) \cap I = \mathrm{Occ}^E_k(P^\#, T^\#) \cap I$$

for each candidate interval I because $k \le K'$. For this, we need to build a period cover C_S (that has the aforementioned properties; again see the full version for details), which requires retrieving certain compressible substrings of T. The minimum period cover C_S utilized in our encoding does not seem to admit an efficient quantum construction procedure, so we build a slightly larger period cover whose encoding incurs a logarithmic-factor overhead. The key subroutine that we repeatedly use while constructing this period cover asks to compute the longest fragment of T (or of the reverse text $\overline{T}$) that starts at a given position and admits a Lempel–Ziv factorization [38] of size bounded by a given threshold. For this, we use exponential search combined with the recent quantum LZ factorization algorithm [24]. Based on the computed period cover, we can construct the strings $P^\#$ and $T^\#$ and resort to a classic verification algorithm (that performs no quantum queries) to process all $O(k)$ intervals I in time $O(m + k^3)$.

The next step is to drop the unrealistic assumption that every candidate interval I contains a K-error occurrence of P. The natural approach is to test each of the candidate intervals using an approximation algorithm that either reports that $\mathrm{Occ}_k(P, T) \cap I = \varnothing$ (in which case we can drop the interval since we are ultimately looking for k-error occurrences) or that $\mathrm{Occ}_K(P, T) \cap I \neq \varnothing$ (in which case the interval satisfies our assumption). Given that $|I|$ is much smaller than K, it is enough to approximate $\delta_E(P, T[\, t \,..\, t + m \,))$ for an arbitrary single position $t \in I$ (distinguishing between distances at most $O(k)$ and at least $K - O(k)$). Although the quantum complexity of approximating edit distance has not been studied yet, we observe that the recent sublinear-time algorithm of Goldenberg, Kociumaka, Krauthgamer, and Saha [25] is easy to adapt to the quantum setting, resulting in a query complexity of $\hat{O}(\sqrt{n})$ and an approximation ratio of $n^{o(1)} = \hat{O}(1)$; see the full version for details.

Unfortunately, we cannot afford to run this approximation algorithm for every candidate interval: that would require $\hat{O}(k\sqrt{m})$ queries. Our final trick is to use Grover's search on top: given a subset of the $O(k)$ candidate intervals, using just $\hat{O}(\sqrt{km})$ queries, we can either learn that none of them contains any k-error occurrence (in this case, we can discard all of them) or identify one that contains a K-error occurrence. Combined with binary search, this approach allows discarding some candidate intervals so that the leftmost and the rightmost among the remaining ones contain K-error occurrences. The underlying alignments (constructed using the exact quantum bounded edit distance algorithm of [24]) are used to initialize the set S. At each step of growing S, on the other hand, we apply our approximation algorithm to the set of all candidate intervals that are not yet (fully) captured by S. Either none of these intervals contain k-error occurrences (and the construction of S may stop), or we get one that is guaranteed to contain a K-error occurrence. In this case, we construct an appropriate low-cost alignment $\mathcal{X}$ using the exact algorithm and extend the set S with $\mathcal{X}$. Thus, the unrealistic assumption is not needed to construct the set S and the strings $P^\#$ and $T^\#$ using $\hat{O}(\sqrt{km})$ queries.

2.3.3 *Handling the Approximately Periodic Case.*

Verifying $O(k)$ candidate intervals was the only bottleneck of the non-periodic case of Pattern Matching with Edits. In the approximately periodic case, on the other hand, we may have $O(k^2)$ candidate intervals, so a direct application of the approach presented above only yields an $\hat{O}(\sqrt{k^2 m})$-query algorithm.

Fortunately, a closer inspection of the candidate intervals constructed in [15] reveals that they satisfy the unrealistic assumption that we made above: each of them contains an $O(k)$-error occurrence of P. This is because both P and the relevant part of T are at edit distance $O(k)$ from substrings of Q^∞ and each of the intervals contains a position that allows aligning P into T via the substrings of Q^∞ (so that perfect copies of Q are matched with no edits). Consequently, the set S of $O(\log m)$ alignments covering all candidate intervals can be constructed using $\tilde{O}(\sqrt{km})$ queries. Moreover, once we construct the strings $P^\#$ and $T^\#$, instead of verifying all $O(k^2)$ candidate intervals, which takes $O(m + k^4)$ time, we can use the classic $\tilde{O}(m + k^{3.5})$-time algorithm of [16] to construct the entire set $\mathrm{Occ}^E_k(P^\#, T^\#) = \mathrm{Occ}^E_k(P, T)$.

ACKNOWLEDGEMENTS

The work of Jakob Nogler has been mostly carried out during a summer internship at the Max Planck Institute for Informatics.

REFERENCES

[1] Scott Aaronson, Daniel Grier, and Luke Schaeffer. 2019. A Quantum Query Complexity Trichotomy for Regular Languages. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*. IEEE Computer Society, 942–965. https://doi.org/10.1109/FOCS.2019.00061

[2] Shyan Akmal and Ce Jin. 2023. Near-Optimal Quantum Algorithms for String Problems. *Algorithmica* 85, 8 (2023), 2260–2317. https://doi.org/10.1007/S00453-022-01092-X

[3] Andris Ambainis. 2002. Quantum Lower Bounds by Quantum Arguments. *J. Comput. System Sci.* 64, 4 (2002), 750–767. https://doi.org/10.1006/JCSS.2002.1826

[4] Andris Ambainis, Kaspars Balodis, Jānis Iraids, Kamil Khadiev, Vladislavs Kļevickis, Krišjānis Prūsis, Yixin Shen, Juris Smotrovs, and Jevgēnijs Vihrovs. 2020. Quantum Lower and Upper Bounds for 2D-Grid and Dyck Language. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020 (LIPIcs, Vol. 170)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 8:1–8:14. https://doi.org/10.4230/LIPIcs.MFCS.2020.8

[5] Arturs Backurs and Piotr Indyk. 2018. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.* 47, 3 (2018), 1087–1097. https://doi.org/10.1137/15M1053128

[6] Gabriel Bathie, Tomasz Kociumaka, and Tatiana Starikovskaya. 2023. Small-Space Algorithms for the Online Language Distance Problem for Palindromes and Squares. In *34th International Symposium on Algorithms and Computation, ISAAC 2023 (LIPIcs, Vol. 283)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 10:1–10:17. https://doi.org/10.4230/LIPIcs.ISAAC.2023.10

[7] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. 2001. Quantum lower bounds by polynomials. *J. ACM* 48, 4 (2001), 778–797. https://doi.org/10.1145/502090.502097

[8] Sudatta Bhattacharya and Michal Koucký. 2023. Streaming k-edit approximate pattern matching via string decomposition. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023 (LIPIcs, Vol. 261)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 22:1–22:14. https://doi.org/10.4230/LIPIcs.ICALP.2023.22

[9] Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. 2015. Random Access to Grammar-Compressed Strings and Trees. *SIAM J. Comput.* 44, 3 (2015), 513–539. https://doi.org/10.1137/130936889

[10] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. 2021. Approximating edit distance in truly subquadratic time: Quantum and MapReduce. *J. ACM* 68, 3 (2021), 19:1–19:41. https://doi.org/10.1145/3456807

[11] Karl Bringmann, Marvin Künnemann, and Philip Wellnitz. 2019. Few Matches or Almost Periodicity: Faster Pattern Matching with Mismatches in Compressed Texts. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*. SIAM, 1126–1145. https://doi.org/10.1137/1.9781611975482.69

[12] Alejandro Cassis, Tomasz Kociumaka, and Philip Wellnitz. 2023. Optimal Algorithms for Bounded Weighted Edit Distance. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*. IEEE, 2177–2187. https://doi.org/10.1109/FOCS57990.2023.00135

[13] Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, and Wiktor Zuba. 2021. Circular pattern matching with k mismatches. *J. Comput. System Sci.* 115 (2021), 73–85. https://doi.org/10.1016/J.JCSS.2020.07.003

[14] Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. 2022. Approximate Circular Pattern Matching. In *30th Annual European Symposium on Algorithms, ESA 2022 (LIPIcs, Vol. 244)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 35:1–35:19. https://doi.org/10.4230/LIPIcs.ESA.2022.35

[15] Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. 2020. Faster Approximate Pattern Matching: A Unified Approach. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 978–989. https://doi.org/10.1109/FOCS46700.2020.00095

[16] Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. 2022. Faster Pattern Matching under Edit Distance: A Reduction to Dynamic Puzzle Matching and the Seaweed Monoid of Permutation Matrices. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*. IEEE, 698–707. https://doi.org/10.1109/FOCS54457.2022.00072

[17] Panagiotis Charalampopoulos, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. 2024. Approximate Circular Pattern Matching under Edit Distance. In *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024 (LIPIcs)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. arXiv:2402.14550

[18] Andrew M. Childs, Robin Kothari, Matt Kovacs-Deak, Aarthi Sundaram, and Daochen Wang. 2022. Quantum divide and conquer. arXiv:2210.06419

[19] Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. 2019. The streaming k-mismatch problem. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*. SIAM, 1106–1125. https://doi.org/10.1137/1.9781611975482.68

[20] Richard Cole and Ramesh Hariharan. 2002. Approximate String Matching: A Simpler Faster Algorithm. *SIAM J. Comput.* 31, 6 (2002), 1761–1782. https://doi.org/10.1137/S0097539700370527

[21] Nathan J. Fine and Herbert S. Wilf. 1965. Uniqueness Theorems for Periodic Functions. *Proc. Amer. Math. Soc.* 16, 1 (1965), 109–114. https://doi.org/10.1090/S0002-9939-1965-0174934-9

[22] François Le Gall and Saeed Seddighin. 2023. Quantum Meets Fine-Grained Complexity: Sublinear Time Quantum Algorithms for String Problems. *Algorithmica* 85, 5 (2023), 1251–1286. https://doi.org/10.1007/S00453-022-01066-Z

[23] Paweł Gawrychowski and Damian Straszak. 2013. Beating $O(nm)$ in Approximate LZW-Compressed Pattern Matching. In *24th International Symposium on Algorithms and Computation, ISAAC 2013 (LNCS, Vol. 8283)*. Springer, 78–88. https://doi.org/10.1007/978-3-642-45030-3_8

[24] Daniel Gibney, Ce Jin, Tomasz Kociumaka, and Sharma V. Thankachan. 2024. Near-Optimal Quantum Algorithms for Bounded Edit Distance and Lempel-Ziv Factorization. In *35th ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*. SIAM, 3302–3332. https://doi.org/10.1137/1.9781611977912.11

[25] Elazar Goldenberg, Tomasz Kociumaka, Robert Krauthgamer, and Barna Saha. 2022. Gap Edit Distance via Non-Adaptive Queries: Simple and Optimal. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*. IEEE, 674–685. https://doi.org/10.1109/FOCS54457.2022.00070

[26] Ramesh Hariharan and V. Vinay. 2003. String matching in $\tilde{O}(\sqrt{n}+\sqrt{m})$ quantum time. *Journal of Discrete Algorithms* 1, 1 (2003), 103–110. https://doi.org/10.1016/S1570-8667(03)00010-8

[27] Ce Jin and Jakob Nogler. 2023. Quantum Speed-ups for String Synchronizing Sets, Longest Common Substring, and k-mismatch Matching. In *34th ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*. SIAM, 5090–5121. https://doi.org/10.1137/1.9781611977554.CH186

[28] Tomasz Kociumaka, Ely Porat, and Tatiana Starikovskaya. 2021. Small-space and streaming pattern matching with k edits. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*. IEEE, 885–896. https://doi.org/10.1109/FOCS52979.2021.00090

[29] Michal Koucký and Michael Saks. 2024. Almost Linear Size Edit Distance Sketch. In *56th Annual ACM Symposium on the Theory of Computing; STOC 2024*.

[30] Gad M. Landau and Uzi Vishkin. 1988. Fast String Matching with k Differences. *J. Comput. System Sci.* 37, 1 (1988), 63–78. https://doi.org/10.1016/0022-0000(88)90045-1

[31] Gad M. Landau and Uzi Vishkin. 1989. Fast Parallel and Serial Approximate String Matching. *Journal of Algorithms* 10, 2 (1989), 157–169. https://doi.org/10.1016/0196-6774(89)90010-2

[32] Vladimir Iosifovich Levenshtein. 1965. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR* 163, 4 (1965), 845–848. http://mi.mathnet.ru/eng/dan31411

[33] Peter H. Sellers. 1980. The Theory and Computation of Evolutionary Distances: Pattern Recognition. *Journal of Algorithms* 1, 4 (1980), 359–373. https://doi.org/10.1016/0196-6774(80)90016-4

[34] Tatiana Starikovskaya. 2017. Communication and Streaming Complexity of Approximate Pattern Matching. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017 (LIPIcs, Vol. 78)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 13:1–13:11. https://doi.org/10.4230/LIPIcs.CPM.2017.13

[35] Teresa Anna Steiner. 2024. Differentially Private Approximate Pattern Matching. In *15th Innovations in Theoretical Computer Science Conference, ITCS 2024 (LIPIcs, Vol. 287)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 94:1–94:18. https://doi.org/doi.org/10.4230/LIPIcs.ITCS.2024.94

[36] Alexander Tiskin. 2014. Threshold Approximate Matching in Grammar-Compressed Strings. In *Prague Stringology Conference, PSC 2014*, Jan Holub and Jan Žďárek (Eds.). 124–138. http://www.stringology.org/event/2014/p12.html

[37] Qisheng Wang and Mingsheng Ying. 2024. Quantum Algorithm for Lexicographically Minimal String Rotation. *Theory of Computing Systems* 68, 1 (2024), 29–74. https://doi.org/10.1007/S00224-023-10146-8

[38] Jacob Ziv and Abraham Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 23, 3 (1977), 337–343. https://doi.org/10.1109/TIT.1977.1055714